



ON WEIGHTLESS GRAPH CLASSIFICATION

Raul Bezerra Barbosa

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Diego Leonel Cadette Dutra
Diego Moreira de Araujo
Carvalho
Felipe Maia Galvão França

Rio de Janeiro
Fevereiro de 2026

ON WEIGHTLESS GRAPH CLASSIFICATION

Raul Bezerra Barbosa

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Diego Leonel Cadette Dutra
Diego Moreira de Araujo Carvalho
Felipe Maia Galvão França

Aprovada por: Prof. Diego Leonel Cadette Dutra
Prof. Diego Moreira de Araujo Carvalho
Prof. Priscila Machado Vieira Lima
Prof. Claudio Miceli de Farias
Prof. Valmir Carneiro Barbosa
Prof. Leandro Santiago de Araújo
Prof. Fabio Protti

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2026

Bezerra Barbosa, Raul

ON WEIGHTLESS GRAPH CLASSIFICATION/Raul
Bezerra Barbosa. – Rio de Janeiro: UFRJ/COPPE, 2026.
XV, 116 p.: il.; 29, 7cm.

Orientadores: Diego Leonel Cadette Dutra

Diego Moreira de Araujo Carvalho

Felipe Maia Galvão França

Tese (doutorado) – UFRJ/COPPE/Programa de
Engenharia de Sistemas e Computação, 2026.

Referências Bibliográficas: p. 92 – 108.

1. weightless neural networks. 2. graph classification.
3. machine learning. I. Leonel Cadette Dutra, Diego
et al. II. Universidade Federal do Rio de Janeiro, COPPE,
Programa de Engenharia de Sistemas e Computação. III.
Título.

...to our little star...

Acknowledgements

Primeiramente, gostaria de repetir as palavras de alguns anos atrás e de agradecer a Deus, se existir, por me manter vivo, apesar de todas as minhas tentativas de jogar minha vida fora. Eu ainda não entendo se existe uma aleatoriedade, ou se foi uma decisão de algum ser supremo, mas fato é que ainda estou aqui e eu duvidava que fosse difícil passar por mais dificuldades ainda do que passei durante o mestrado.

Gostaria de agradecer aos meus pais, Lourdes e Barbosa, por todo o apoio dado ao longo da minha vida, à minha companheira Ana Teresa, testemunha ocular e apoio firme dos muitos dos tropeços que dei ao longo desta caminhada. Ao meu irmão, Rodolpho, minha cunhada, Andreza, e meus três sobrinhos que eu tanto amo, Ana, Helena e Pedro. Aos meus sogros, Conceição e Antônio, por me colocarem como mais um filho em sua família. Aos meus professores de dia a dia: prof. Diego Carvalho, prof. Diego Dutra, prof. Felipe França e prof. Priscila Lima, por me estenderem a mão ao longo deste longo processo e tanto contribuírem para que o trabalho chegasse até aqui. A todos os professores de quem pude ser aluno, em especial ao prof. João Carlos da Silva e ao prof. Severino Collier. Aos professores e membros da banca, pelo esforço árduo de avaliação deste trabalho. Aos responsáveis pelo NACAD, onde pude realizar quase todos os experimentos desta tese. Aos servidores e terceirizados da UFRJ e do CEFET que fazem sempre muito com pouco e muitas vezes passam despercebidos, mas que ajudaram a prover toda a estrutura necessária, em especial à D. Lourdes e ao Irmão.

Esta tese nunca existiria sem a participação da minha chefe e amiga, Teresa Marino, a quem não tenho palavras nem ações que consigam dimensionar o quanto sou eternamente grato. Também agradeço aos meus chefes superiores do IBGE, Bianca Sotelo e Arnaldo Lyrio por depositarem a confiança e abraçarem este doutorado. Aos amigos do IBGE, em especial ao Diego Felipe e Edson Junior, por segurarem as pontas na minha ausência. Ao meu psicólogo, Ramon Reis, que me ajudou a enxergar caminhos no meio de muitas nuvens. Aos amigos dessa louca vida, sem os quais não posso ficar, e que dividiram preciosos momentos durante este longo ciclo e não quero distinguir para não cometer a injustiça de esquecer alguém.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

ON WEIGHTLESS GRAPH CLASSIFICATION

Raul Bezerra Barbosa

Fevereiro/2026

Orientadores: Diego Leonel Cadette Dutra
Diego Moreira de Araujo Carvalho
Felipe Maia Galvão França

Programa: Engenharia de Sistemas e Computação

Em um contexto onde redes neurais para grafos (GNN) aparecem como o padrão para aprendizado de máquina com grafos, este trabalho propõe uma alternativa de aprendizado em grafos com o foco na tarefa de classificação de grafos baseada em redes neurais sem pesos. Dessa forma, tendo as redes sem peso como guia, uma nova representação para aprendizado raso é desenvolvida e avaliada em conjuntos de dados já amplamente utilizados pela comunidade. Os resultados demonstram que com a representação proposta e o uso de redes neurais sem peso é possível atingir melhores resultados que os observados com o uso de redes profundas utilizando-se de menos recursos computacionais. Ainda, resultados corroboram que outros classificadores e métodos de aprendizado raso podem se beneficiar da representação proposta.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

ON WEIGHTLESS GRAPH CLASSIFICATION

Raul Bezerra Barbosa

February/2026

Advisors: Diego Leonel Cadette Dutra
Diego Moreira de Araujo Carvalho
Felipe Maia Galvão França

Department: Systems Engineering and Computer Science

In a context where graph neural networks (GNN) emerge as a standard in graph machine learning, this work proposes a contrasting weightless graph learning architecture focused on the task of graph classification. Weightless neural networks are leveraged as design guides to a new representation for shallow graph learning. This new architecture is also evaluated in widely adopted datasets. Results demonstrate that using this representation and weightless models is possible to achieve better results than those observed when using extra-large graph neural networks or heavy-weight kernel methods while using less computer and memory requirements. Results also show that other classifiers can benefit from the proposed representation.

Contents

List of Figures	xi
List of Tables	xiii
List of Algorithms	xv
1 Introduction	1
1.1 Motivation	1
1.2 Contributions & Research Goals	2
1.3 Related Works	4
1.4 Thesis Outline	5
2 Weightless Neural Networks	6
2.1 Neural Networks and Machine Learning Concepts	6
2.1.1 Neural Networks	6
2.1.2 Machine Learning Concepts	8
2.2 WiSARD - an n-Tuple Classifier	10
2.3 WiSARD Extensions	12
2.3.1 Counter & Bleaching Strategies	12
2.3.2 ClusWiSARD	14
2.3.3 Memory-Efficient WiSARD	15
2.3.4 Recent Models	16
2.4 Binarization Strategies	18
2.4.1 Thermometers	18
2.4.2 KernelCanvas	20
2.4.3 KDTree Binarization	22
2.4.4 Other Encodings	23
3 Graphs & Graph Learning	26
3.1 Preliminaries	26
3.2 Machine Learning on Graphs	27
3.3 Graph Statistics and Bag-of-Nodes	28

3.4	Graph Kernels	30
3.5	Graph Embeddings & Manifold Learning	30
3.6	Graph Neural Networks	32
4	<i>K</i>-Means Thermometer	34
4.1	The <i>K</i> -Means Thermometer	34
4.2	Experimental Setup & Results	36
4.2.1	Thermometers and the WiSARD	36
4.2.2	One-Hot Counterparts for the WiSARD	37
4.2.3	Thermometers and the DWN	38
4.2.4	One-hot Counterparts for the DWN	39
4.2.5	On the Effects of Thermometers in Non-WNN Models	39
4.2.6	One-Hot Counterparts in Non-WNN Models	40
4.2.7	Other Encodings	40
4.2.8	Visualizing the Thresholds	42
4.3	Conclusion	43
5	Weightless Primers to Graph Classification	45
5.1	Leveraging Graph Embeddings for Weightless Graph Classification	45
5.1.1	A Vanilla Architecture Using Graph Embeddings	46
5.1.2	Experimental Setup & Results	46
5.2	A Bag of Nodes Primer on Weightless Graph Classification	49
5.2.1	On Whole Distribution Binarization	50
5.2.2	Experimental Setup & Results	51
5.3	Conclusions	55
6	KernelCanvas++ for Graph Classification	56
6.1	Extending the Node Primer	56
6.1.1	An Updated Kernel Canvas and Histogram Procedure	56
6.1.2	The Final Representation	59
6.1.3	Experimental Setup & Results	60
6.2	KCVQ and DKC++	70
6.2.1	KernelCanvas-VQ	71
6.2.2	Distributive KernelCanvas++	72
6.2.3	First Experimental Setup & Results	74
6.2.4	Larger Scale Experiment	77
6.2.5	Sampled Input Size and Activation Rate	79
6.3	Leveraging Embeddings with KC++ and DKC++	84
6.3.1	Architecture for Attributed Graphs and KC Investigation	84
6.3.2	Experimental Setup & Results	85

6.4	Conclusions	87
7	Conclusion & Future Works	88
7.1	Conclusions	88
7.1.1	Paper Contributions	90
7.2	Future Works	90
	References	92
A	Hyperparameter Settings	109
A.1	Hyperparameter Settings for the <i>K</i> -Means Thermometer Experiments	109
A.2	Hyperparameter Settings for the Initial Graph Classification Architectures	109
A.3	Hyperparameter Settings for the KCpp Experiments	113
B	Full Experiment Results	115
B.1	Full Experiment Results - <i>K</i> -Means Thermometer	115

List of Figures

2.1	Example of an MLP	8
2.2	Training a WiSARD classifier	11
2.3	Classification using a WiSARD classifier	12
2.4	Classification using Bleaching	13
2.5	Classification using ClusWiSARD	15
2.6	BloomWiSARD illustrated	16
2.7	The thermometer encoding	19
2.8	Linear Thermometer example	19
2.9	Gaussian Thermometer example	20
2.10	Distributive Thermometer example	21
2.11	KernelCanvas example	23
2.12	KDTree Binarization example	24
3.1	Example of a graph and its adjacency matrix	26
4.1	K -Means thermometer example	35
4.2	Histogram plot of distributions and their division by each thermometer strategy	43
5.1	Global big picture of the graph embeddings classification framework	46
5.2	Global big picture of a shallow classification framework	49
5.3	Example of ECCDF plot	50
5.4	Plot Binarization example	51
5.5	Histogram Binarization example	52
5.6	Top-ten F1-score results - Initial experiment	54
6.1	Final Representation Example	60
6.2	The KernelCanvas procedure applied to two different graphs	61
6.3	Trade-offs between structural and non-structural attributes in the best result for all datasets	66
6.4	Accuracy of different KernelCanvas strategies on the COLLAB dataset	69

6.5	Accuracy of different KernelCanvas strategies on the ENZYMES dataset	70
6.6	Accuracy of different KernelCanvas strategies on the IMDB-M dataset	70
6.7	Accuracy of different KernelCanvas strategies on the MUTAG dataset	71
6.8	Accuracy of different KernelCanvas strategies on the NCI1 dataset . .	71
6.9	Accuracy of different KernelCanvas strategies on the PROTEINS dataset	72

List of Tables

4.1	Accuracy results for all evaluated thermometers in all datasets - WiS-ARD classifier	37
4.2	Accuracy results for evaluated one-hot counterparts in all datasets - WiSARD classifier	38
4.3	Accuracy results for all evaluated thermometers in all datasets - DWN	38
4.4	Accuracy results for evaluated one-hot counterparts in all datasets - DWN	39
4.5	Accuracy results for all evaluated thermometers in all datasets - RandomForest	40
4.6	Accuracy results for all evaluated thermometers in all datasets - NaiveBayes	41
4.7	Accuracy results for all evaluated counterparts - RandomForest	41
4.8	Accuracy results for all evaluated counterparts - NaiveBayes	42
4.9	Accuracy results for CMAC and Minchinton Cells - WNN	42
4.10	Accuracy results for CMAC and Minchinton Cells - Non-WNN	43
5.1	Statistics of Selected TUDatasets	47
5.2	Attributes of Selected TUDatasets	47
5.3	Accuracy results for graph embeddings - IMDB-M and COLLAB datasets	48
5.4	Dataset Statistics	52
5.5	Hypothesis set for initial architecture	53
6.1	Accuracy on TUDatasets	63
6.2	Accuracy using different vector input sizes - COLLAB dataset	64
6.3	Accuracy using different vector input sizes - ENZYMES dataset	64
6.4	Accuracy using different vector input sizes - IMDB-M dataset	64
6.5	Accuracy using different vector input sizes - MUTAG dataset	65
6.6	Accuracy using different vector input sizes - NCI1 dataset	65
6.7	Accuracy using different vector input sizes - PROTEINS dataset	65
6.8	Average energy consumption on TUDatasets	67

6.9	Average final model size on TUDatasets	67
6.10	Average virtual memory usage on TUDatasets	68
6.11	Time results - TUDatasets	68
6.12	Accuracy results in TUDatasets - KCpp vs DKCcpp	75
6.13	Accuracy results per KCcpp and DKCcpp strategies - TUDatasets . . .	76
6.14	Energy consumption per KCcpp and DKCcpp strategies - TUDatasets .	76
6.15	Virtual memory usage per KCcpp and DKCcpp strategies - TUDatasets	77
6.16	Time results per KCcpp and DKCcpp strategies - TUDatasets	77
6.17	OGB-molhiv dataset statistics	78
6.18	OGB-molhiv dataset attributes	78
6.19	ROC-AUC for the OGB-molhiv datasets	79
6.20	Accuracy results per strategy - COLLAB dataset	81
6.21	Accuracy results per strategy - ENZYMES dataset	82
6.22	Accuracy results per strategy - IMDB-M dataset	82
6.23	Accuracy results per strategy - MUTAG dataset	83
6.24	Accuracy results per strategy - NCI1 dataset	83
6.25	Accuracy results per strategy - PROTEINS dataset	84
6.26	ROC-AUC for the OGB-molhiv dataset	85
6.27	Accuracy results for attributed variations - TUDatasets	86
A.1	Hyperparameter settings for the <i>K</i> -means thermometer experiment .	110
A.2	Hyperparameter settings for the initial architecture	111
A.3	Hyperparameter settings for the initial architecture	112
A.4	Hyperparameter settings for the KCcpp experiments	114
B.1	Full Results - Thermometer Review	116

List of Algorithms

1	K-Means Algorithm	10
2	KernelCanvas Main Algorithm	21
3	KCpp Algorithm	58
4	KCVQ Algorithm	73

Chapter 1

Introduction

1.1 Motivation

Life is about relationships and nothing is more important than life. Science is nothing but a structured way of learning about life. So, if one wants to understand life, understanding relationships is necessary, because there is no life or world without relationships. One of the most fundamental ways of modeling and studying relationships is using graphs, and there may lie the reason why graphs are such an active research topic, especially these days [1–4].

Every second, trillions of data are generated by devices, social media interactions, chemical or biological reactions, natural phenomena sensing, and so are graphs. Yet, it is not limited to that: from molecules to galaxies, code to complex systems, simple logic propositions to complete knowledge bases, it may be better to say that graphs can model everything, being a building block for any kind of learning.

Graphs are composed of vertices, which represent the entities, and edges, the connections between entities. But graphs are only a mathematical abstraction and data. There must be knowledge produced from these data. Moreover, this knowledge has to be applied to increase human perception of the world and enhance its experience here. Mankind has discovered a way of extracting knowledge from this abundant data and applying it to its daily problems - machine learning, and machine learning is changing how work and life are perceived.

Machine learning on graphs has been the subject of many works [3, 5–7] and the vast literature agrees that there are at least three main tasks: node classification, link prediction and graph classification. This work has focused on graph classification problems, where the problem is posed as a whole-graph label or property prediction. Graph classification has motivation in various fields, such as biology, chemistry, computer science, and physics [8–11]. For example, a graph classification application predicts whether a molecule, represented as a graph, is carcinogenic [12].

The function call graph of a code base can be used to detect malwares [13]. The identification of several mental disorders has also been studied as a graph classification problem, where the brain is modeled as a graph of regions of interest [14].

Graph machine learning has become a trending topic due to the rise of so-called graph neural networks (GNN) as a consecutive advance of deep neural networks (DNN). Several GNN models have been published [15–17], deriving from the message-passing framework [18] or earlier methods [19, 20] with promising results. Nevertheless, it remains an open problem [21]. Apart from the inner problems of GNN, such as their limited expressiveness [16, 22], graph neural networks are still deep neural networks. These are known powerful hardware and energy consumers [23, 24], thus expensive and red artificial intelligence models [25] that, at some point, must be avoided.

Hence, this work takes inspiration from Occam’s razor principle, focusing on one central question: what can be learned from graphs in a more resource-friendly way. Graph learning is not limited to graph (deep) neural networks [26, 27]. Thus, this is the exact point where lies the motivation of this work: to develop a feasible model and, if does not exceed in accuracy, at least is easier and cheaper to carry on with comparable results. This is where weightless neural networks (WNN) come into play.

WNNs have both a past and present of consistent reduced training time and low computational demand through an architecture with few hyperparameters to train. Special attention must be given to the WiSARD [28] model and some of its derivatives, considering their lightweight design [29], easiness of implementation [30], explainability of decisions [31, 32], and robustness [33]. Even with these benefits, no other work has come to join both worlds of graph learning and weightless neural networks, except for [34], which uses it in a very specific context. Here, since no context is predefined, the whole framework can be applied to a broader audience.

1.2 Contributions & Research Goals

The main goal of this work was to develop a viable approach to graph learning, more specifically the graph classification problem. The initial belief is that what is done using complex structures can certainly be completed using simpler ones, using less energy, and taking less time. The idea was to build a bridge from the deep and heavy island of graph neural networks to the tiny and lighter beach of weightless neural networks. However, since a bridge cannot be built without its bricks, several other problems have arisen during the development of this work, leading to the exploration of different paths related to data pre-processing, representation learning, high-performance computing, and model development and assessment. Hence, the

main contributions of this work can be summarized as follows:

1. Efficient graph classification models are proposed, implemented, and evaluated using novel representations as building blocks. These proposed models are capable of surpassing the GNN and SVM models in several datasets, both in terms of accuracy and resource consumption. GNN models are still dominant in inference time, model size, and, to the extent of this work, in large-scale scenarios where greater models can be developed and trained. In addition, some of the proposed architectures incur much more training time and energy consumption than simple thermometer approaches. In spite of this fact, the developed WNN models result in viable alternatives to graph classification;
2. A complete redesign of the KernelCanvas is proposed to approach graph classification problems. This enables the use of all node and edge attributes and enhances performance in general. Several extensions to the proposed representation are discussed and evaluated, improving the accuracy in different scenarios. This fact is observed for both WNN and non-WNN models, such as RandomForest;
3. Different encoding and binarization methods are stressed in different datasets. This establishes the Distributive Thermometer as the standard encoding for WNN. The results also indicate that thermometer-based approaches are better suited for WNN models than one-hot representations. Despite this fact, there is still a gap to fill when binarization is performed, mostly corroborated by the results when using non-WNN classifiers;
4. A novel encoding technique called K-Means Thermometer is presented and explored, and results show that its performance can exceed the Distributive Thermometer in some datasets.
5. Distribution-based encodings are presented and used to compose graph input vectors, achieving scores that are on-par with graph kernels with support vector machines. Extensions to these encodings are also proposed and used to achieve even better performance in different architectures;
6. The use of graph embeddings is analyzed as input for graph classification in conjunction with WNN and non-WNN, with non-WNN performing better than WNN models;
7. The proposed representations are merged with graph embeddings, but the performance of both WNN and non-WNN classifiers does not surpass baselines. However, results for a baseline KernelCanvas alternative are promising, being dominant in almost all datasets.

1.3 Related Works

Concerning shallow machine learning strategies for graph classification, some approaches must be highlighted. NETSimile [35, 36] is a graph representation extraction framework that focuses on local profiles of nodes that are aggregated and then used for downstream tasks. The authors also present an eigenvalue decomposition as a graph feature vector (EIP), in a similar way they later were used in [37] for the SF graph embedding. The EIP baseline is also related to the works of the [38] and other spectral-based graph embeddings, such as the NetLSD [39] and FGSD [40].

Another related work is [41], where authors create a vector representation using a variety of global graph descriptors. More recently, [42] presents a method using vector representations based on graph kernels, where each input is a stack of different kernel values. In [43], several graph metrics are analyzed to distinguish between real and synthetic networks. The work of [44] can be considered a reiteration of [35], reducing the scope to only degree as the metric of interest. Degree-based vector representations were also proposed as features for graph comparison in [45].

Both works of [35] and [44] are very similar to works developed in this thesis, in the way they use signature vectors as aggregation of node-level descriptors for a graph. Nonetheless, the evaluation done in this work adopt different classifiers as baselines and the representation is original and solely based on the WiSARD classifier. We also incorporate every attribute of the datasets and the decision to use them or not is a hyperparameter search. Moreover, we include different node descriptors, such as the onion layer number.

The KernelCanvas approach proposed here has some similarities with GNN graph-level pooling strategies [46], since node-level representations are clustered to compose a final binary vector. In fact, it can be seen as a node clustering pooling or a (multi)set aggregation that does not have learnable parameters [21]. The Set2Set [47] approach that uses LSTM to learn sets is used in several works in a graph learning context [48, 49]. A pioneer work is [50], where authors build on the DeepSets [51] model, another set neural network model, to propose a universal readout function for graph neural networks. Authors in [52] propose a readout with a multiset encoding. A DeepSets approach is also used in [53], in which authors also introduce an MLP readout approach that yields better results than traditional pooling schemes.

The only reference that joined both worlds of graph learning and weightless neural networks prior to this work is [34]. In [34] authors try to classify graphs in the process mining context. The representation and specific context make their attempt not applicable to this work and the datasets here evaluated.

1.4 Thesis Outline

The remainder of this text is organized as follows. In Chapter 2, a background is given on the basics of WNN learning, particularly on the WiSARD model and its derivatives. Chapter 3 continues on related background, formalizing graph and graph learning concepts. Chapter 4 studies the binary representation problem, a necessary step prior to advances in the study of other input representations. Chapter 5 presents initial architectures for the graph classification problem. Chapter 6 explores a promising representation and architecture for graph classification, extending both architectures presented in Chapter 5. Chapter 7 concludes the work by wrapping up the conclusions and showing further research steps.

Chapter 2

Weightless Neural Networks

In this chapter, a quick introduction to artificial neural networks and other machine learning concepts is given, followed by the main background information on weightless neural networks. The focus is on the WiSARD classifier and its several derivatives. These derivatives include simple extensions to more recent models, such as the Differentiable Weightless Neural Network (DWN) [54]. Finally, common binarization algorithms for WNN are introduced.

2.1 Neural Networks and Machine Learning Concepts

2.1.1 Neural Networks

Neural networks are computational models inspired by the biological activity of neurons [55]. As mentioned in [56], biological neurons operate as natural signal processing units, which receive multiple synaptic inputs and produce an output. The pioneer work of mathematical modeling of biological neurons is the McCulloch and Pitts (MCP) neuron model [57], in which synapses are modeled as fixed-weight edges. In this abstraction, the stimuli that a neuron receives are numerical values multiplied by the edge weights, and the output of the neuron is the result of an activation function applied to the sum of these values.

The work of McCulloch and Pitts is considered the cornerstone of one of the most adopted modern artificial neural networks, the Multilayer Perceptron (MLP) or feed-forward neural networks (FFNN) [55]. These are themselves the basis for current deep neural network models. Heavily influenced by the MCP neuron, the Perceptron [58] was planned as a hardware image recognition device, but was first implemented as software by Frank Rosenblatt at IBM [55]. Authors in [55] define

the Perceptron as a linear binary classification model given by

$$y(x) = f(w^\top \phi(x)), \quad (2.1)$$

where x is the input vector, y is the output, w is the weight vector (the learnable parameters of the model), and $\phi(\cdot)$ is a nonlinear transformation applied to the original feature vector. Also, the $f(\cdot)$ function is an activation function given by

$$f(s) = \begin{cases} 1, & \text{if } s \geq 0 \\ -1, & \text{if } s < 0. \end{cases} \quad (2.2)$$

According to [55], the Multilayer Perceptron is a model that comprises several layers of functional transformations [55] that resemble the Perceptron over an input vector $x \in \mathbb{R}^D$. A basic MLP is composed of three layers: an input layer, a hidden layer, and an output layer. The input layer is represented by a node for each x_i in the input vector. This input layer is then transformed into M output nodes, which compose the hidden layer. The outputs of a hidden layer are then combined to form the final K outputs of the network. The outputs of a hidden layer of an MLP are given by

$$h_j = \psi(a_j^{(hidden)}), \quad (2.3)$$

where $j = 1, \dots, M$, $\psi(\cdot)$ is a nonlinear activation function and $a_j^{(hidden)}$ are the activations of the hidden layer, defined as

$$a_j^{(hidden)} = \sum_{i=1}^D w_{ji}^{(0)} x_i + w_{j0}^{(0)}, \quad (2.4)$$

where $w^{(0)}$ is a $M \times D$ matrix of learnable weights of the hidden layer. The results of the output layer are given in a similar way by

$$o_k = \psi(a_k^{(output)}), \quad (2.5)$$

where $k = 1, \dots, K$ and $a_k^{(output)}$ are the activations of the output layer, given by

$$a_k^{(output)} = \sum_{j=1}^M w_{kj} h_j + w_{k0}, \quad (2.6)$$

where $w^{(1)}$ is a $K \times M$ matrix of learnable weights of the output layer. Several hidden layers can also be stacked using the output of the previous layer as input. As an example, Figure 2.1 presents an MLP with an input layer with 2 nodes, two hidden layers of 4 nodes, and an output layer of 2 nodes, where signals flow from

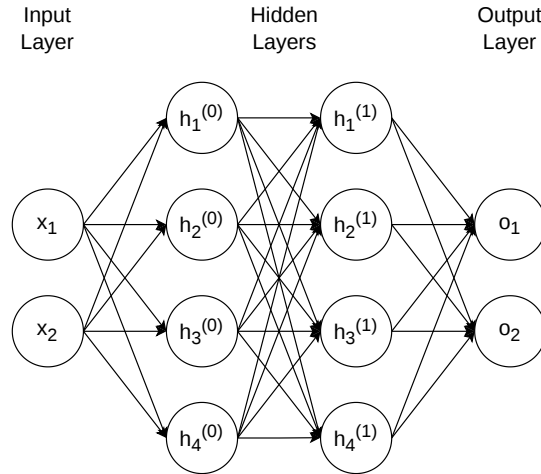


Figure 2.1: Example of an MLP with an input layer with 2 nodes, two hidden layers of 4 nodes and an output layer of 2 nodes.

the input through the output, passing through multiple hidden layers. Finally, more details of how training and learning happen in the MLP can be found in [55].

2.1.2 Machine Learning Concepts

Classification and Learning

The goal of a classifier is to solve a given classification problem. Let Σ be a finite set (or alphabet) of two or more discrete classes, also called labels. Let \mathbf{X} be an $M \times D$ matrix consisting of M data points. Let X_i be the i -th row of $\mathbf{X} \in \mathbb{R}^D$. Let also $y(X_i)$ be a labeling function that maps X_i to a label $y \in \Sigma$. The task of classification in machine learning is to correctly predict these associated labels, mimicking how y works and generalizing it to new examples, whose class is unknown but $\in \Sigma$.

If the cardinality of Σ is exactly two, the classifier has to learn and distinguish between two classes, and the problem is considered a binary classification problem. If the cardinality is greater than two, the problem is called a multiclass classification problem [55].

Generally, the use of a classifier is divided into two phases: training (or learning) and inference. The most common learning process of a classifier is supervised learning [55]. In supervised learning, a training dataset is used, and the classifier receives input examples together with its target labels to learn from. There are other types of learning, such as unsupervised learning, where training examples do not have target labels and learning mostly operates with the goal of finding similar data points and giving labels to them, also called a clustering problem [59].

***K*-Means Algorithm**

An well-known clustering method is the *K*-Means algorithm, proposed by Lloyd [60], and originally used for pulse-code modulation, a method to convert analog signals to a digital format. Its main objective is to create a good partition of the data into *K* clusters, where each cluster is represented by the average element of the partition, also called centroid. In other way, each element is associated with and represented by its closest centroid.

The *K*-Means algorithm is defined in Algorithm 1, with the idea of greedily minimize the distance from each point of the dataset to its closest centroid. It takes 3 arguments as input: \mathbf{T} —a matrix containing all training data points $\in \mathbb{R}^D$, τ —the maximum number of iterations, and k —the number of clusters, and outputs both the centroid matrix containing all centroids and a cluster assignment vector. It assumes that both *argmin* and *dist* functions are defined. The *argmin* function receives an array as an argument and returns the index of the minimum value in an array. The *dist* function receives the centroid matrix and a point as arguments, and returns the distance between the point and every centroid in the centroid matrix. This distance defaults to the Euclidean distance between two vectors, defined by

$$dist_{\text{Euclidean}} = \sqrt{\sum_{i=1}^D (x_i - y_i)^2}, \quad (2.7)$$

where $X = [x_1, \dots, x_D]$, and $Y = [y_1, \dots, y_D]$ are two points represented as arrays $\in \mathbb{R}^D$.

Two other functions compose Algorithm 1: *initCentroids* and *calculateCentroids*. The *initCentroids* function uses k and \mathbf{D} , a matrix of all training points, to initialize a centroid matrix with k centroids. Usually, this initialization is performed by randomly sampling centroids $\in \mathbb{R}$, but many other types of initialization are defined in the literature and may provide faster convergence or better solutions [61]. The *calculateCentroids* function receives k , the current cluster assignment array, and the data points, and returns an updated centroid matrix. The centroid matrix is updated by recalculating each centroid C_k of cluster k as the average point of all points assigned to cluster k . The algorithm stops if no cluster is updated during the iteration or if τ iterations are reached without convergence.

Lloyd’s algorithm is known to be superpolynomial in theory, but of practical efficiency [62]. Moreover, the reader should refer to [55] for a complete introduction to machine learning basic concepts.

Algorithm 1 K-Means Algorithm - Partitions the space into k subspaces assigning each point to cluster

Input:

T ▷ A matrix of all training points
 τ ▷ Maximum iterations
 k ▷ The number of clusters

Output:

$centroids$ ▷ The final centroid matrix
 $clusterAssignment$ ▷ The final cluster assignment vector

```

1:  $t \leftarrow 0$ 
2:  $centroids \leftarrow initCentroids(k, T)$  ▷ Random initialization of centroids
3: for all  $point \in T$  do
4:    $clusterAssignment[point] \leftarrow 0$ 
5: end for
6: for  $t \leftarrow 1$  to  $\tau$  do
7:   for all  $point \in D$  do
8:      $newClusterAssignment[point] \leftarrow argmin(dist(centroids, point))$ 
9:   end for
10:  if  $newClusterAssignment == clusterAssignment$  then
11:    Stop
12:  end if
13:   $centroids \leftarrow calculateCentroids(clusterAssignment, k, T)$ 
14: end for

```

2.2 WiSARD - an n-Tuple Classifier

The learning process in an MLP occurs by adjusting the weights of the edges of the network, hence its knowledge resides in the edges of the network while the neuron has no state or memory. A weightless neural network contrasts with these networks because it memorizes its input instead of having adjustable and learning weights. The learning task literally stores the knowledge in the neuron nodes [63].

The WiSARD [28] is a weightless neural network derived from the n -tuple classifier of Bledsoe and Browning [64]. Originally, WiSARD was a hardware pattern recognition system, that functioned as a supervised multiclass classifier. Its basic structure is composed of discriminators and each discriminator is organized to discriminate and identify one class of the problem. Discriminators are itself groups of RAM nodes, which are the neurons of the network and store all knowledge. Several discriminators are trained independently to deal with all classes of the problem.

WiSARD works only with binary vectors as input. Hence, some binarization process should be performed if the data are not originally represented in a bit vector format. As a supervised classifier, WiSARD functioning is divided into a training and an inference phase, and the learning process is conducted using a training set of labeled samples. During the training phase, each sample of the training set is

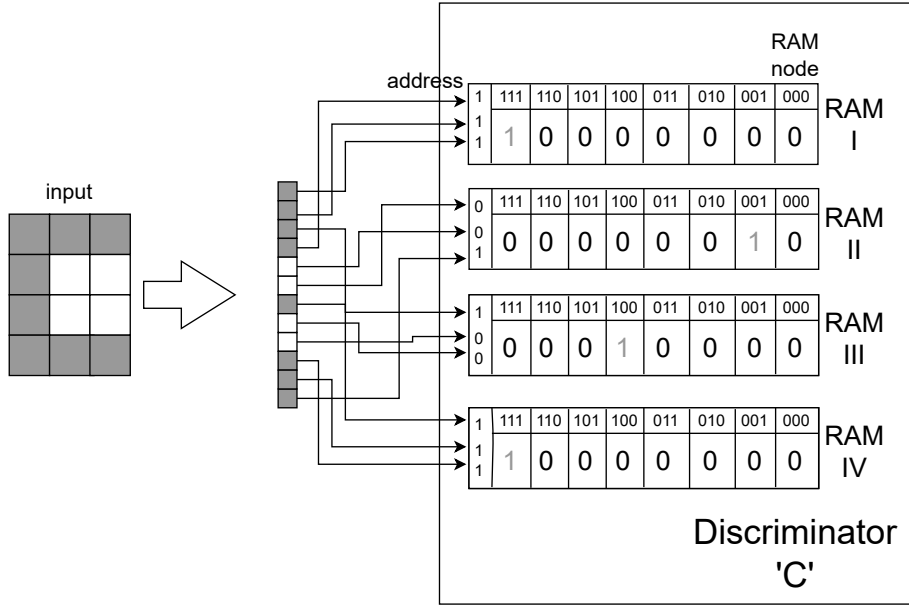


Figure 2.2: Training a WiSARD classifier - A 12-bit input vector representing an example of class 'C' during training phase. This vector is randomly mapped into N tuples of n bits ($N = 4, n = 3$), which serve as RAM addresses that get written.

mapped into N tuples of a fixed size n . Usually, this mapping is random and the same for all examples presented to the classifier. Each of these N tuples works as an n -bit address of a RAM node of 2^n locations. The RAM locations addressed by the input are then written with a single bit value: one or true. These co-occurring n -bits of the n -tuples constitute the patterns memorized by the network.

Figure 2.2 shows the training phase of a WiSARD classifier, where a discriminator for letter C with four RAM neurons is being trained. On the left, the letter C is drawn for better illustration, but a contiguous array is used as input. In the figure, a 12-bit input vector is randomly mapped into $N = 4$ tuples of $n = 3$ bits. These n -tuples indicate the addresses that are written (marked with a 1) on each RAM node.

In order to predict (or classify) an example whose class is unknown, the bit vector is mapped once again into n -tuples using the same mapping used during training phase and presented to each discriminator of the trained classifier. Each n -tuple acts again as one address of the RAM node of a discriminator. A RAM will be activated if its n -bit address presented as a tuple was written during the training phase. Each discriminator may have several RAMs activated, and this number represents the score of a given discriminator (or class) for the example to be classified. The output class is the class of the discriminator that has the highest score. Ties can happen and are randomly broken, whether a confidence over the score is defined. The classification of an example is shown in Figure 2.3, where an example whose label is unknown is presented to all discriminators of a trained network. Each discriminator

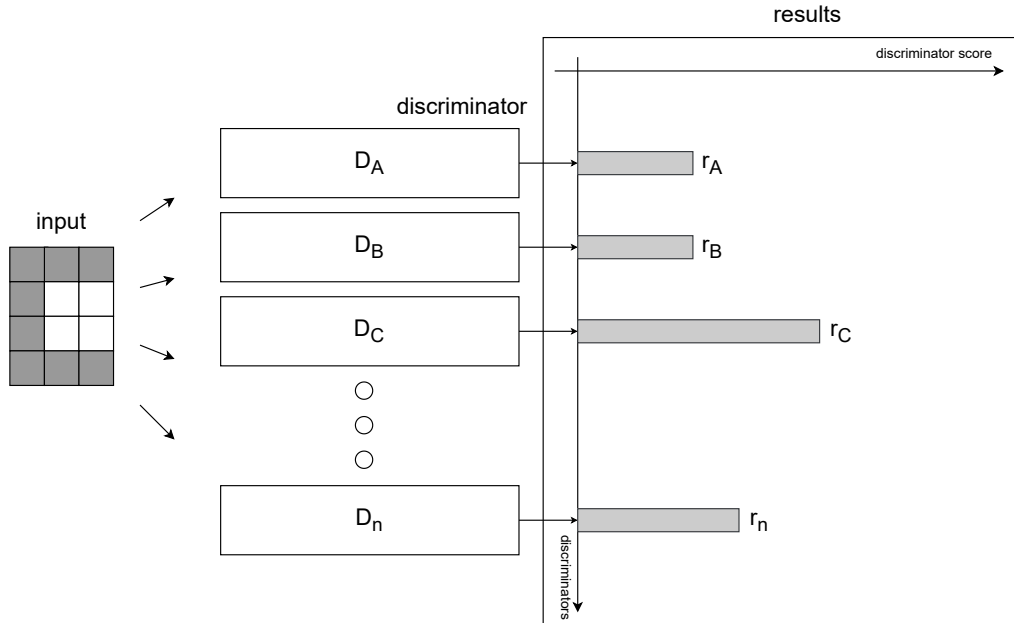


Figure 2.3: Classification using a WiSARD classifier: each input vector is randomly mapped and presented to all trained discriminators, outputting a classification score. The discriminator with the greatest score is chosen as the output class.

outputs its score r_{letter} , and the discriminator trained to identify the letter C has the highest score, r_C , with letter C chosen as the output class.

2.3 WiSARD Extensions

Further literature has expanded the original WiSARD model and its limitations into more sophisticated problems. The next sections try to cover some of these extensions as they are necessary to the complete understanding of this work, but other interesting extensions such as the RegressionWisard [65] are out of the scope of this work. Nonetheless, the hardware natural suitability of WiSARD-based models is still verified even in the most recent extensions [29, 33, 66–68].

2.3.1 Counter & Bleaching Strategies

The original WiSARD architecture suffers from a main concern: overfitting by saturation. Overfitting comes from the literature as a common problem of various machine learning classifiers. As defined by authors in [59], it occurs when a model cannot account for the presence of noise in the training data and after the training phase is incapable of performing decently. In many models, some form of regularization is done to mitigate it, introducing constraints to the learning hypothesis. In the WiSARD case, as each address is either written or not, if noise is present or if many training examples are presented, many addresses can be written. This may result in

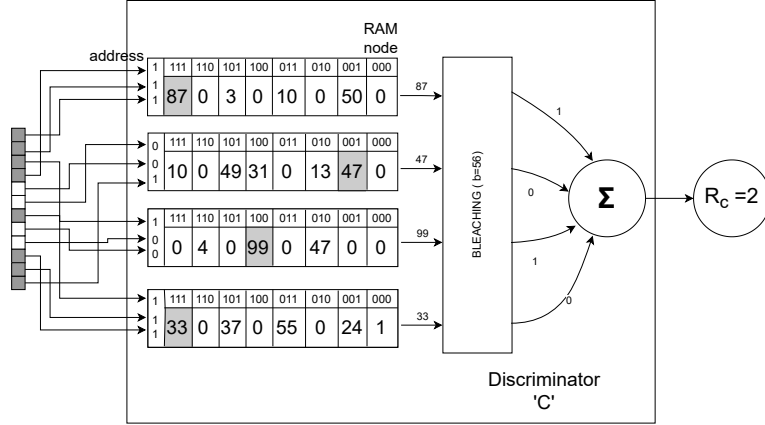


Figure 2.4: Classification using Bleaching: each address is now a counter. Depending on bleaching threshold a different number of RAM gets activated. The output score is still the sum of activated RAMs.

a full memory, causing ties for every decision and producing a completely random classifier. One solution for this problem is to enhance the tuple size, but this has the drawback of exponentially increasing the memory usage. Another solution for this problem is a twofold mechanism: the use of counters as proposed in [69] and the Bleaching [70] procedure or one of its variations.

The use of counters is a trivial modification of the original architecture where instead of writing a single bit value to a RAM address, one writes integer values to it. Hence, each address stores now a counter. The Bleaching procedure is a tie-breaking policy and consists of an iterative process of using an activation threshold b during the classification phase. Instead of activating RAMs that have been written or not, only RAMs with b or more counts are activated. So, when a tie occurs, b is sequentially incremented until there is no tie and a certain confidence threshold is met.

A discriminator using counters and Bleaching for inference is presented in Figure 2.4, where an input vector is presented. All RAM nodes have counters instead of simple boolean flags, and instead of activating if the address is written, the values corresponding to the addresses pass through the Bleaching threshold $b = 56$. This implies that only values greater than b activate the RAM. Activated RAMs are then counted to form the final discriminator score.

As a final note, few works have been published focused on the Bleaching procedure. Apart from the original work of [70] and the work of [71], it was only recently that newer versions of this procedure were proposed [33].

2.3.2 ClusWiSARD

Originally designed as a supervised classifier, the WiSARD has received extensions to unsupervised and semi-supervised learning [72, 73]. The ClusWiSARD (CWIS) [72] is based on the clustering paradigm and can act as an unsupervised learner or a supervised learner.

In its supervised mode, it uses target labels, similar to a classical supervised learner. The difference is that instead of using a single discriminator for each class, each class can have a maximum of LD discriminators, a hyperparameter of the model. The training phase grows a set of multiple discriminators for each class, each class starting with an initial discriminator. Each training sample is first processed by the network and can be accepted or discarded.

The acceptance procedure is such that each labeled example e is presented (in a similar fashion to the inference phase) to the set \mathcal{D}_C of all discriminators of e 's class (C). If the score of that example with respect to every existing discriminator $\in \mathcal{D}_C$ is below a minimum score m , a new discriminator is created for label C if $|\mathcal{D}_C| < LD$. Otherwise, the example is simply discarded. The minimum score m is often combined with a hardening acceptance policy per discriminator, such as

$$f = m + \frac{c}{t}, \quad (2.8)$$

where f is the final acceptance threshold, t is the maximum number examples absorbed per discriminator, and c is the current count of examples in a specific discriminator. The general reasoning behind this mode is to learn multiple profiles for each class and make the acceptance of an example easier at beginning, but as more examples are added to a discriminator, a greater score should be required to be accepted as an example of that discriminator.

During classification, each discriminator is treated the same way as in the original WiSARD classifier, even if they represent the same class. An example of the inference phase of the ClusWiSARD is shown in Figure 2.5, where an input is presented to an already trained instance. Such instance has two trained discriminators for class A (D_{A1}, D_{A2}), two discriminators for B (D_{B1}, D_{B2}) and at least three for class C (D_{C1}, D_{C2}, D_{C3}). While D_{C2} presents the highest output score for the input, the output is the original label, in this case the letter C.

The unsupervised mode of ClusWiSARD works similarly to its supervised mode, but without using class label information from the data. Under the hood, all inputs are now associated with one dummy label and a set of discriminators for this dummy class is grown up to LD discriminators, using a scoring and acceptance policy, similar to the supervised mode. The only difference during inference phase is that each discriminator represents a distinct output class.

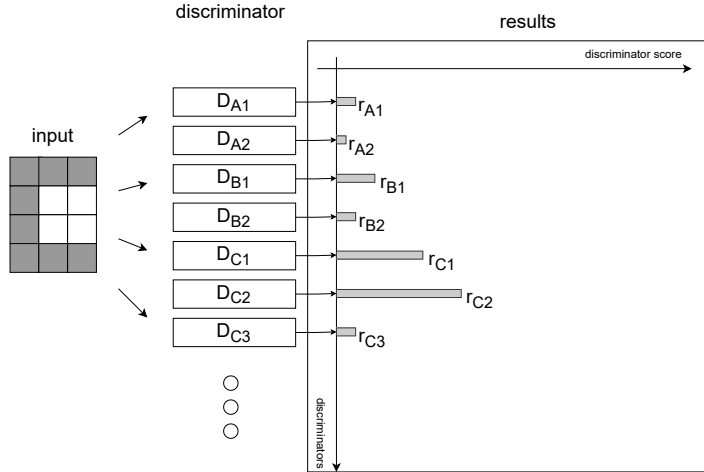


Figure 2.5: Classification using ClusWiSARD in supervised mode. In this trained instance, there are two trained discriminators for class A (D_{A1}, D_{A2}), two discriminators for B (D_{B1}, D_{B2}) and at least three for class C (D_{C1}, D_{C2}, D_{C3}). An input is submitted for inference and D_{C2} presents the highest output score for the input. The output is the original class label, the letter C.

2.3.3 Memory-Efficient WiSARD

The initial architecture has also been studied taking the RAM model as just an abstraction. The key idea behind all studies was to reduce the memory usage of a naive implementation of the WiSARD architecture. One core strategy is based on the assumption of the sparsity of RAMs that actually get written during the training process. Instead of a physical RAM, storing 2^n addresses for a n -bit tuple, some form of dictionary or hash table is used. This solution has been used in several works and was coined in [74] as the DictionaryWisard (or DictWisard). The use of the DictWisard has no drawbacks compared to the original model, since no information is lost.

The work of [74] goes further in this process of memory optimization, introducing Approximate Membership Query (AMQ) structures as a replacement for the original RAMs. Out of these AMQ structures, the use of Bloom filters stands out as the default RAM abstraction with demonstrated lower memory usage compared to dictionaries. A Bloom filter is a probabilistic data structure to represent a set of elements and is composed of an m -bit array and k independent hashes. It provides a query membership mechanism to check if an element belongs or not to it with a false positive rate ϵ . The m -bit array is used to store the elements and the h independent hash functions are used to select h locations of the m -bit array [74] to insert elements.

The BloomWiSARD (BWIS) was also proposed in [74] and it is based on the use of Bloom filters as the neuron abstraction. The hash functions used in the original implementation were developed using a double hashing technique that leverages the

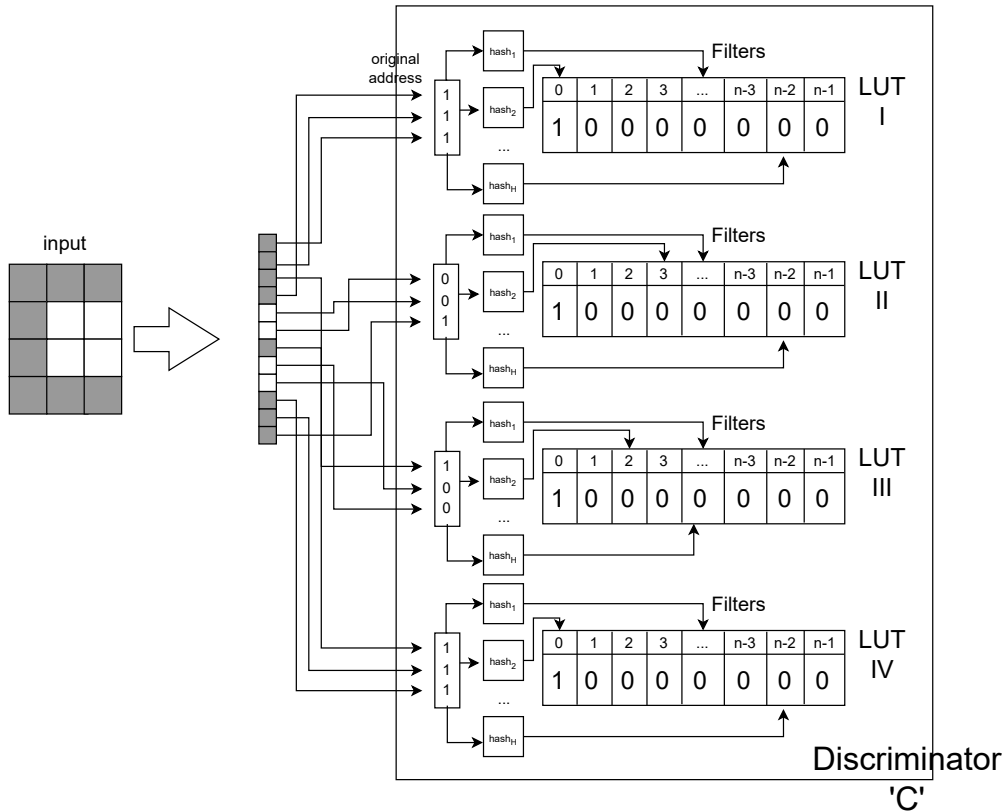


Figure 2.6: BloomWiSARD illustrated - RAMs become lookup tables (LUT) supported by Bloom filters. Instead of a single address, the n -bit tuple becomes the input of the Bloom filter, which uses multiple and independent hash functions. The output of these hash functions act as LUT entries that are written.

family of MurmurHash functions as building blocks.

The training phase with the BloomWiSARD is similar to the WiSARD, but instead of representing a single RAM address, the n -bit tuple becomes the input of an insert operation in the Bloom filter. Hence, the n -tuple passes through k hash functions and the k outputs of these hash functions act as update indexes of the Bloom filter array in that neuron. This procedure is illustrated in Figure 2.6, where an example of the class C is being written.

The inference phase is also very similar to the original WiSARD, but uses the n -tuples as input to the Bloom filter membership query mechanism. If the tuple is within the set according to the Bloom filter, the RAM is activated and outputs '1'. Otherwise, the RAM node is not activated and outputs '0'. As in the original WiSARD, the BloomWiSARD does not account for bleaching mechanisms.

2.3.4 Recent Models

The study of WNN models has gained attention in recent years, given the search for lighter and greener models. This search is a natural answer to the omnipresent

energy-demanding deep neural networks. In this section, some recent models are presented, most of them developed with a focus on edge-inference.

BTHOWeN

The BTHOWeN [33] model is a WNN architecture that expands on the BloomWiSARD by implementing counting Bloom filters. It adopts the H3 family of hash functions and replaces the bit array with an integer array, enabling the storage of counters instead of bits, similar to the adoption of counters in the WiSARD. This implies the possibility of bleaching mechanisms for filter-based WNN, but since the BTHOWeN is an inference-focused model, the bleaching procedure is replaced by a whole-model binarization [29, 33]. This technique applies a fixed and well-suited bleaching to an already trained model, converting it again to a single-bit architecture, but prone to its initial overfitting issues. This also helps to avoid an explosion of memory usage due to the storage of integers instead of bits. Model-binarization can also be seen as a type of bleaching coined here as Static Bleaching, since the bleaching value is learned using validation data, but kept static throughout the classification phase. The BTHOWeN architecture also establishes the Gaussian Thermometer as its binarization procedure and has shown great advances through its FPGA implementation.

LogicWiSARD

LogicWiSARD [29] is a WiSARD implementation developed to reduce the size of the final model. This is done by eliminating lookup tables and memories and using only logic functions on minterms. These minterms are the activated addresses that remain after the model binarization, similar to the one applied in BTHOWeN. LogicWiSARD presents gains of up to 99% in latency and energy per inference when deployed in FPGA circuits and compared to FPGA implementations of other neural networks. Compared to other FPGA implementations of WNN models, LogicWiSARD is capable of delivering equivalent latency with greater accuracy and less energy consumption [29].

ULEEN

ULEEN is an acronym for Ultra-low-Energy Edge Networks [68], another recently published model focused on inference under energy constraints. This model is an evolution of the BTHOWeN model and presents three key innovations: multi-pass training, ensembles, and pruning. The multi-step training incorporated in ULEEN is the direct result of using continuous Bloom filters. These continuous Bloom filters store floating-point values $\in [-1, 1]$, enabling training through optimization methods

that use the gradients of an output loss, such as the Stochastic Gradient Descent [55]. The entries in the Bloom filter are binarized using $sign(x)$. However, since this activation is a non-continuous function, a technique called Straight-Through Estimator (STE) [75] is used for training. The ensemble technique is performed using additive models, trained independently using the same training data. Pruning is adopted mainly to reduce the cost of models, since it avoids hash computation, which is responsible for a great stake of the model’s cost [68].

DWN

The Differentiable Weightless Neural Network (DWN) [54] is a recently proposed model that introduces learnable layers of lookup tables (LUTs). It can be seen as a reimagination of the WiSARD using multiple layers and gradient-based learning, techniques commonly applied to MLPs. A binarized input is introduced to a first layer of LUTs whose outputs are used to compose the next layer of LUTs. These layers can be chained indefinitely until a final layer produces a final output summing the outputs from the previous LUT layer. All LUT outputs are trained using multi-step learning and a well-defined loss. The loss propagation is possible due to an efficient extended finite difference method proposed by the authors. Instead of fixed randomly mapped connections, its connections can also be learned, although authors recommend limiting this to the mapping of the first LUT layer [54].

2.4 Binarization Strategies

Binarization plays a crucial role in the WNN context as it defines the network input. Therefore, how continuous and categorical attributes should be encoded in a bit vector is the subject of many recent works [76–78] that research WiSARD-based systems. Here, some of the most commonly used strategies are covered, especially those used in the development of models presented in the next chapters. Chapter 4 presents an analysis of most of the encodings reviewed in this chapter, together with the proposal and evaluation of a new thermometer called K -Means thermometer.

2.4.1 Thermometers

One of the most adopted encoding algorithms by WNN practitioners is the thermometer encoding [79], whose name is due to its similarity with a mercury thermometer. A mercury thermometer has a maximum value, a minimum value, and some intermediate marks. The thermometer encoding has B bits to represent a continuous value in a discrete way. Its basic behavior can be depicted as the same

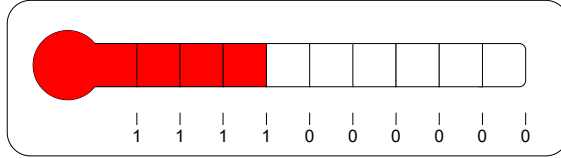


Figure 2.7: The thermometer encoding. Bits resemble a mercury expanding through the thermometer.

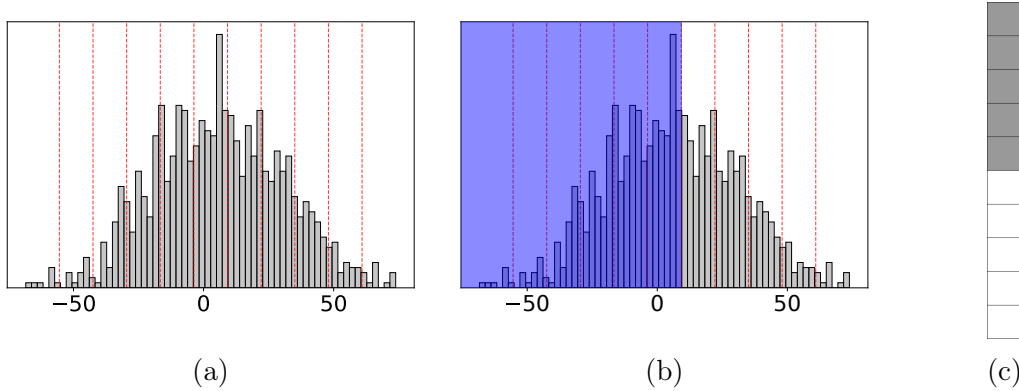


Figure 2.8: A Linear Thermometer example is presented in Subfigure 2.8a. Each dotted red line represents a threshold uniformly splitting the space in 11 intervals using 10 bits. The encoding of value $x = 0$ is represented in Subfigure 2.8b by the blue-shaded area, resulting in the final vector depicted in Subfigure 2.8c.

as a mercury thermometer expanding through the glass. In the mercury thermometer, the more expanded the mercury is, the greater the value. In the thermometer encoding, the greater the value, the more consecutive bits equal to 1 the bit vector has (see the analogy of Figure 2.7).

A thermometer-encoded value x using B bits is equivalent to checking x against T thresholds where the i -th bit of the thermometer is defined by

$$B_i = x \geq T_i. \quad (2.9)$$

Regarding WNN recent literature, several thermometers have been proposed, always following this framework of defining T thresholds for a B -bit vector representation of a each non-binary attribute.

In the Linear Thermometer (LT) [79], these thresholds are equal-width intervals [80], such that they divide the space between the maximum and the minimum values of the training data. This is the same as assuming a uniform distribution of the data, as shown in Figure 2.8. In Subfigure 2.8a, the space is uniformly divided using 10 thresholds. The encoding of value $x = 0$ is represented in Subfigure 2.8b, resulting in the final 10-bit vector depicted in Subfigure 2.8c.

In the Gaussian Thermometer (GT) [33, 81], the assumption is that the data follow a Gaussian distribution with mean and standard deviation obtained from

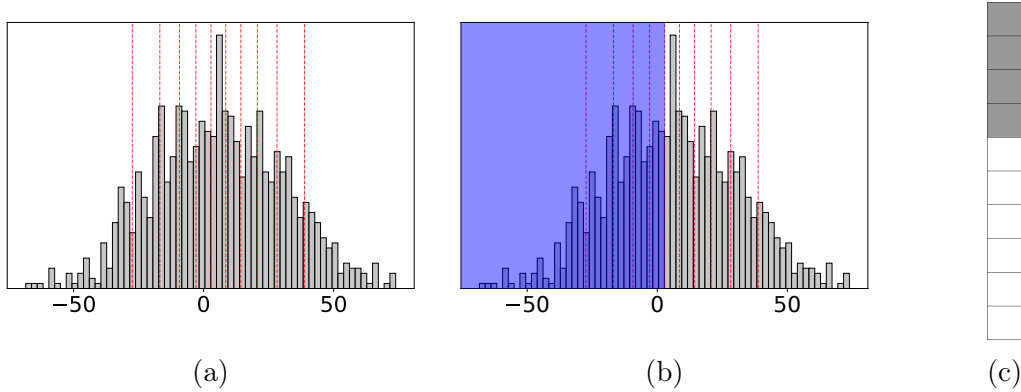


Figure 2.9: A Gaussian Thermometer example is presented in Subfigure 2.9a. Each dotted red line represents a threshold splitting the space in 11 intervals using 10 bits. The encoding of value $x = 0$ is represented in Subfigure 2.9b by the blue-shaded area, resulting in the final vector depicted in Subfigure 2.9c.

the training set. In this case, the thresholds denote quantiles of the Gaussian distribution. The GT is exemplified in Figure 2.9, with Subfigure 2.9a showing the divided space using 10 thresholds. The encoding of value $x = 0$ is represented in Subfigure 2.9b, resulting in the final 10-bit vector depicted in Subfigure 2.9c.

The Distributive Thermometer (DT) [78] drops this Gaussian assumption, making no assumption of the sample distribution. Each threshold is obtained using a quantile of the training set, dividing the space into $B + 1$ equal frequency intervals which are represented by each bit in the final representation. Figure 2.10 explains the Distributive Thermometer using a set of six subfigures. Subfigures 2.10a, 2.10b and 2.10c show the difference between thresholds and the encoding of the value $x = 1$ using 10 bits and the Distributive, Linear, and the Gaussian thermometers, respectively. The blue-shaded area represents bits that get activated and marked with a one in the final representation. In Subfigures 2.10d, 2.10e, and 2.10f, the final encoded vectors are presented for the value $x = 1$ using the Distributive, Linear, and Gaussian thermometers, respectively.

2.4.2 KernelCanvas

The KernelCanvas (KC) [82] was proposed to tackle the time-series problems. It is a dual normalization and binarization method that tries to mimic the act of painting on a canvas. Instead of using a matrix to represent a discrete canvas, KernelCanvas uses \mathcal{K} - a set of kernels - to divide the space similarly to a Voronoi diagram. Each kernel is defined as a randomly sampled point $k \in [-1, 1]^d$, where d is equal to the number of dimensions of the canvas and the original input.

The main KernelCanvas steps are summarized in Algorithm 2, which assumes that five functions are defined: *calculateDimensionalMean*,

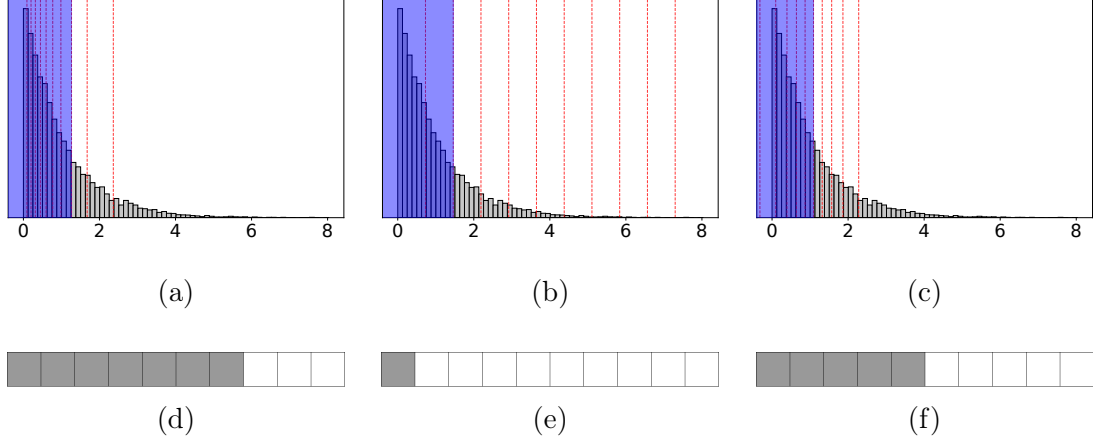


Figure 2.10: The encoding of value $x = 1$ using a Distributive thermometer is presented in Subfigure 2.10a by the blue-shaded area. Each dotted red line represents a quantile/threshold splitting the space in 11 intervals using 10 bits. The Linear and Gaussian thermometers for the same training data and sample are presented in Subfigures 2.10b and 2.10c. The respective final 10-bit vectors are shown in Subfigures 2.10d, 2.10e, and 2.10f.

Algorithm 2 KernelCanvas - Transforms a sequence of points $\in \mathbb{R}^d$ into a binary vector $\in \{0, 1\}^B$

Input:

- sequence ▷ Arbitrary size sequence of points $\in \mathbb{R}^d$
- \mathcal{K} ▷ The kernel set (random points $\in [-1, 1]^d$)
- κ ▷ The number of kernels to activate

Output:

- $input_vector$ ▷ The final binarized input vector

```

1: for  $bit\_idx \leftarrow 1$  to  $B$  do
2:    $input\_vector[bit\_idx] \leftarrow 0$ 
3: end for
4: for  $dim \leftarrow 1$  to  $d$  do
5:    $means[dim] \leftarrow calculateDimensionalMean(dim, sequence)$ 
6:    $stds[dim] \leftarrow calculateDimensionalStd(dim, means, sequence)$ 
7: end for
8: for all  $point \in sequence$  do ▷ Apply dimensional Z-score and tanh
9:   for  $dim \leftarrow 1$  to  $d$  do
10:     $newPoint[dim] \leftarrow tanh(zscore(point, means, stds, dim))$ 
11:   end for
12:    $activatedKernels \leftarrow findknn(newPoint, \mathcal{K}, \kappa)$  ▷ Find newPoint's  $\kappa$  nearest neighbors  $\in \mathcal{K}$ 
13:   for all  $kernel \in activatedKernels$  do ▷ Activate each kernel using  $\beta$  bits
14:     for  $bit \leftarrow 1$  to  $\beta$  do
15:        $bit\_idx \leftarrow (kernel - 1) * \beta + bit$ 
16:        $input\_vector[bit\_idx] \leftarrow 1$ 
17:     end for
18:   end for
19: end for

```

calculateDimensionalStd, *zscore*, *tanh*, and *findknn*. The *calculateDimensionalMean* receives *dim*, a dimension index, and the whole sequence of points and calculates the sample mean with respect to the specified dimension. The *calculateDimensionalStd* operates in a similar way to *calculateDimensionalMean*, receiving also the pre-calculated *means* to output the sample standard deviation (std) with respect to the specified dimension *dim*. The *zscore* function uses both *means* and *stds* to calculate the Z-Score of the *point* with respect to dimension *dim*. The Z-Score is defined as

$$Z_{\text{dim}} = \frac{\text{point}[\text{dim}] - \text{means}[\text{dim}]}{\text{stds}[\text{dim}]}, \quad (2.10)$$

and the *tanh* function is the hyperbolic tangent function, defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad (2.11)$$

where e is the Euler's number. Nonetheless, the *findknn* function receives three arguments: p – a point $\in [-1, 1]^d$, \mathcal{K} – a set of kernels/points $\in [-1, 1]^d$, and κ – an integer, and outputs the κ -nearest neighbors of p within \mathcal{K} .

In a nutshell, an arbitrary size sequence of values $\in \mathbb{R}^d$ is presented as input and each dimension of each point in the sequence is standardized using the Z-Score (with respect to the sequence) and the *tanh* function to fit normalized values in the range of $[-1, 1]^d$. Each point is then mapped to $\kappa = \lambda * |\mathcal{K}|$ kernels that are closer to it on the canvas, where $\lambda \in (0, 1)$ is called the activation rate. All kernels are represented by the same number of bits β , and each activated kernel is marked with an array of β consecutive ones in its index. A non-activated kernel receives an array of β consecutive zeros. The final vector representation is a single 1-D binary vector of size $B = |\mathcal{K}| * \beta$ bits.

Figure 2.11 illustrates a toy example of a KernelCanvas application, divided in three subfigures. Figure 2.11a shows a 2-D plane with several kernels dividing it. These kernels are painted in Figure 2.11b, representing activated kernels in a sequence of points. These activated kernels result in the final vector with 16 bits and 1 bit per kernel, as shown in 2.11c.

2.4.3 KDTree Binarization

The KDTree binarization is another spatial-based binarization procedure. The main idea is to split the \mathbb{R}^K space in l regions using a KDTree [83], being l the number of leaves in the tree and K the number of dimensions of each data point. The algorithm has been proposed initially in [77], with the trajectory classification application in mind. In a context of sparse datasets, the goal is the same as the distributive

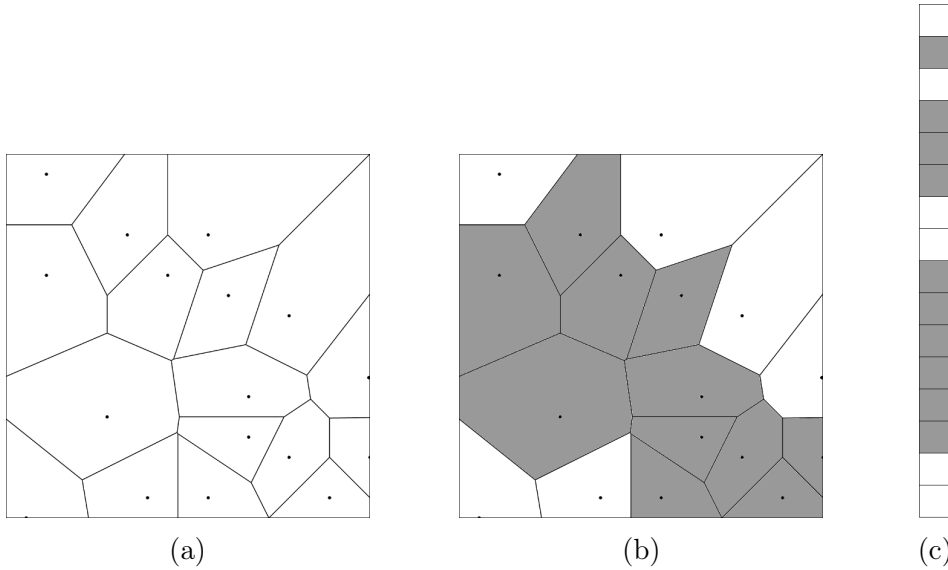


Figure 2.11: Example of KernelCanvas space partitioning. In Subfigure 2.11a, kernels (points) define regions of a 2-D plane, which are painted in Subfigure 2.11b, resulting in the final vector of Subfigure 2.11c

thermometer: fit the training data in a way that bits should account for the same amount of information in the original space.

During the training phase, a KDTree is built using every point of every training data sample. The final vector representation is obtained by activating the tree leaves where the points in a sequence fall in. Figure 2.12 briefly depicts this process in three steps. Subfigure 2.12a shows a plane that was divided using a KDTree built using the training data. A sequence of points is used as input and plotted over the divided plane. In Subfigure 2.12b, the cells or leaves in which the input points fall are painted, resulting in the final vector shown by Subfigure 2.12c.

2.4.4 Other Encodings

There are other several ways of transforming real or continuous values into discrete values, such as the ones presented above. Some other common strategies adopted in the literature are described below.

One-Hot Bit Vectors

One-hot bit vectors are vectors where all bit values are '0' except from one [84]. They are widely used to represent N possible outcomes in a 2^N -bit vector. These outcomes are discrete values, such as words or categories, but can also represent discrete values that divide the sample space. In this work, the one-hot encoding is referred to as the processing of discretizing a continuous value and representing it as a one-hot vector. This may imply assigning a bin to a value within K predefined

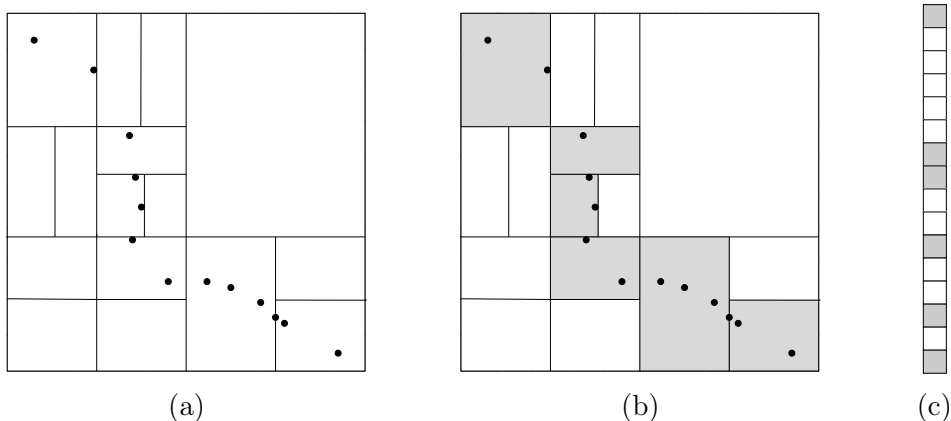


Figure 2.12: Example of KDTree space partitioning. In Figure 2.12a, a KDTree is built based on all training data. Leaves define regions of a 2-D plane. These are painted in Figure 2.12b, resulting in the final vector of Figure 2.12c

bins and then replacing it with the one-hot vector that encompasses the designated bin. Bin generation and assignment can vary from simple schemes such as values falling into regular and uniform-defined intervals to more complex, such as values closer to K -means centroids [85].

Gray Codes

Gray codes are ways of representing a value in a binary format such that in the final representation each sequential value differs by only one digit [86]. Several gray codes have been proposed and the Binary Reflected Gray Code (BRGC) is referenced here as the "main gray code", in the same way as in [87]. For any integer x , its value in BRGC is given by

$$BRGC(x) = x \oplus \lfloor \frac{x}{2} \rfloor, \quad (2.12)$$

where \oplus is the exclusive-or bit-wise boolean operation.

Cerebellar Model Articulation Controller

The Cerebellar Model Articulation Controller (CMAC) [88] is an associative memory neural network that tries to mimic the mammalian cerebellum, a part of the brain. Its operation is based on a large number of overlapping receptive fields and this overlap inspired authors in [89] to propose the CMAC code. The CMAC code is an encoding that aims to reduce the memory usage of thermometer-based encodings. It does that while maintaining some proportionality and monotonicity of its binary-encoded representation to the Hamming distance in their original space.

Given a real-valued input vector $x \in \mathbb{R}^D$, each value x_i is first rescaled to an integer value z_i and encoded separately into a binary vector u_i . All vectors u_i for $i \in (1, D)$ are then further concatenated to form a final binary vector u . A single

attribute binary vector u_i is given by the concatenation of K bit strings, where the k -th string for value x_i is obtained by calculating

$$S_{k,i} = \lfloor \frac{z_i + k - 1}{K} \rfloor, \quad (2.13)$$

where $k=1,\dots,K$, and then converting $S_{k,i}$ to a binary string using Gray codes.

Minchinton Cells

Minchinton Cells (MC) [90] were developed to process grayscale images in the WNN context. Instead of using T thresholds for a single attribute, each k -th element in the final binary vector u is defined by a comparison of the values of the original vector X . The comparison itself is called the Minchinton Cell and is two types are defined in the literature. The Type-0 cell, defined as

$$B_k = X_i > X_j, \quad (2.14)$$

and the Type-1 cell, defined as

$$B_k = (X_i - X_{i+1}) > (X_j - X_{j+1}), \quad (2.15)$$

where X_i, X_j are sampled (and possibly oversampled), but kept fixed for a given k .

Chapter 3

Graphs & Graph Learning

This chapter introduces the background information on graphs and graph learning, such as important definitions and commonly used frameworks. Graph representations for machine learning are also introduced and discussed, such as graph kernels, graph embeddings, and graph neural networks.

3.1 Preliminaries

As defined in the works of [91], a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a pair consisting of \mathcal{V} , the set of vertices or nodes, and \mathcal{E} , its set of edges. An edge $e_{i,j}$ is a connection between two vertices $i, j \in \mathcal{V}$. The most common way to represent a graph is by using an adjacency matrix. An adjacency matrix \mathbf{A} is a $\{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ matrix, where $\mathbf{A}_{i,j}$ assumes a boolean value of whether there is an edge between vertices i and j . The adjacency matrix is often replaced by a weight (or proximity) matrix \mathbf{W} , where $\mathbf{W}_{i,j} \in \mathbb{R}$ represents a quantity about the relationship between i and j .

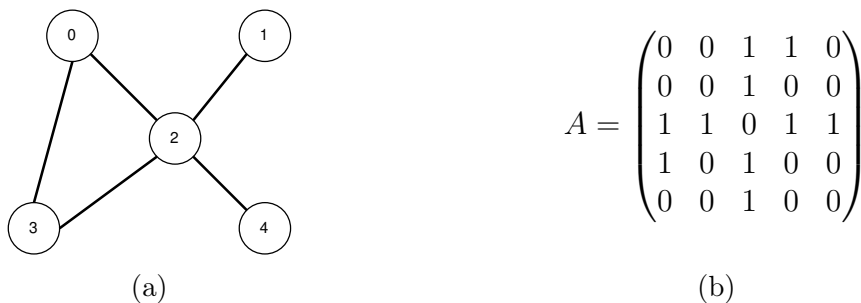


Figure 3.1: Example of a graph with five nodes and its adjacency matrix, a 5×5 matrix. The edge that exists between vertices 2 and 4 is represented with $\mathbf{A}_{2,4} = 1$. Also, since the graph is undirected, $\mathbf{A}_{2,4} = \mathbf{A}_{4,2}$.

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a graph where there is no distinction between edges $e_{i,j}$ and $e_{j,i}$. In this case, the adjacency matrix is symmetric. On the other hand, a directed graph is a graph where there is a distinction between edges $e_{i,j}$ and

$e_{j,i}$. An attributed graph has node and/or edge attributes represented as additional matrices. Such graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{Y})$, where \mathbf{X} is a $|\mathcal{V}| \times a$ matrix representing a attributes per node, and \mathbf{Y} is a $|\mathcal{E}| \times b$ matrix, consisted of b attributes per edge. Figure 3.1 exhibits two different graph representations for a given graph \mathcal{G} . On the left, Subfigure 3.1a displays the graphical visualization of the graph, which has five nodes (represented as numbered circles) and five edges connecting the nodes. On the right, the adjacency matrix of the same graph is shown by Subfigure 3.1b as a 5×5 matrix. The edge that exists between vertices 2 and 4 is represented with $\mathbf{A}_{2,4} = 1$. Also, since the graph is undirected, $\mathbf{A}_{2,4} = \mathbf{A}_{4,2}$.

Traditional machine learning techniques, such as the classifiers used by this work, operate on vectors of (often handcrafted) features. Thus, it is natural that adjacency/weight matrices may emerge as natural candidates. Nonetheless, graph-level learning must account for permutation-invariant representations [16], and the adjacency matrix of a graph is highly dependent on node ordering (or labeling). For example, a given graph \mathcal{G} that has $|\mathcal{V}|$ vertices and a given adjacency matrix \mathbf{A} may have up to $|\mathcal{V}|! - 1$ isomorphic graphs with a different adjacency matrix due to the permutation of its node ordering [92].

Below are presented some context on graph machine learning and a selection of representation methods that are alternatives to the adjacency matrix.

3.2 Machine Learning on Graphs

Graph machine learning can be divided into several task-driven categories [27, 91]. Here, a taxonomy is presented based on the elements of the graph involved, with some examples that help illustrate the problems that each category is trying to solve. Classical machine learning problems such as classification, clustering, and regression have all their counterparts in the context of graph learning.

Node-Level Learning In the heart of node-level problems lies the node representation problem and the node classification. There are uncountable works where the main goal is to propose or learn a representation of a node [27]. Given that learned representation, the next step is to try to predict a property or label that a node should receive. If such property or label is not given as part of the problem, it becomes a clustering/unsupervised problem, similar to a community detection [27].

Edge-Level Learning Sometimes the object of machine learning are the edges of the graph. A task is labeled as edge-level learning if it consists of predicting whether an edge should exist between two nodes or predicting an edge property or label [93].

Graph-Level Learning At last, there are graph-level tasks, where the central learning element is the whole graph [91]. This task is where lies one of the main focus of this work: graph classification. Graph classification is the supervised task of predicting a label in an unlabeled graph. The graph representation learning is also a common task, in a similar fashion to node representation learning. Graph generation is another problem where the main goal is to generate or reconstruct a graph.

3.3 Graph Statistics and Bag-of-Nodes

The bag-of-nodes (BON) strategy is a naive approach to aggregate node and fine-grained measures of a graph to compose a graph-level representation [91]. This aggregation itself can be done using histograms or any other method, such as a simple count, sum or mean. Below are described some measures that are relevant to the development of this work. It is important to note that there are several other node, edge, and graph statistics. Many were considered to be included in this study, but were discarded due to different reasons, such as time and space complexity, similar to the works of [94]. The reader should refer to the works of [95, 96] for a more robust review of node centralities and other graph-level measures.

Node Degree

One of the most basic and primarily used node-level measures. The node degree is a local node statistic that counts the neighbors of a node, vertices that are connected in the graph to a specific node i . Mathematically, in an undirected graph, the node degree $d(i)$ is defined by the cardinality of $\mathcal{N}(i)$ where

$$\mathcal{N}(i) = \{k \in \mathcal{V} \mid (i, k) \in \mathcal{E}, i \neq k\}. \quad (3.1)$$

For directed graphs, two node degrees are observed: the in-degree and the out-degree. The in-degree $d_{in}(k)$ is defined by the cardinality of $\mathcal{N}_{in}(i)$ where

$$\mathcal{N}_{in}(i) = \{k \in \mathcal{V} \mid (k, i) \in \mathcal{E}, i \neq k\}. \quad (3.2)$$

Similarly, the out-degree $d_{out}(k)$ is defined by the cardinality of $\mathcal{N}_{out}(i)$ where

$$\mathcal{N}_{out}(i) = \{k \in \mathcal{V} \mid (i, k) \in \mathcal{E}, i \neq k\}. \quad (3.3)$$

The node degree has a trivial complexity of $\mathcal{O}(|\mathcal{E}|)$, being a measure that naturally scales well.

Node Centralities

Node centralities play a key role in graph analysis [97]. A node centrality defines how important a node is in a graph or a subgraph of the original graph according to some heuristic or goal [91].

PageRank One of the most prominent node centrality is the PageRank centrality [98]. Largely used, the PageRank centrality is also famous for being one of the original building blocks of the Google search engine. With α being a constant $\in (0, 1)$, the page-rank pr_i of a node i is defined as follows:

$$pr_i = \alpha \sum_{k \in \mathcal{N}_{in}(i)} \frac{1}{d_{out}(k)} pr_k + (1 - \alpha). \quad (3.4)$$

The original PageRank article [98] describes it as an algorithm with fast convergence, especially on real networks, which tend to have a very sparse adjacency matrix. Its exact complexity time depends on the algorithm used for its calculation. It is shown in [99] that, for some cases, the time complexity of the power method is linear in $|\mathcal{V}|$.

Clustering Coefficient The clustering coefficient is another relevant node centrality that indicates how clustered nodes tend to be in a graph [100]. For a directed graph, given a vertex i where $|\mathcal{N}_{out}(i)| = k_i$, there are up to $c_{max,i} = k_i \times (k_i - 1)$ edges between all vertices $\in \mathcal{N}_{out}(i)$. Hence, c_v is defined as the fraction of edges that exist over neighbors of i over $c_{max,i}$ - the maximum number of edges that could exist. The undirected variation is analogous, taking into consideration that $c_{max,i} = \frac{k_i \times (k_i - 1)}{2}$. This measure has a direct interpretation in the context of a social network, quantifying how many friends of a given node are also friends between themselves.

Onion Decomposition

The Onion Decomposition is both a local and global-level graph descriptor conceived by gathering more information during the k -core decomposition algorithm of a graph. The k -core decomposition on its own is a core-periphery analysis tool that aims to define a graph's center (or core) of and its periphery (or shell) [101]. A k -core is a maximal subgraph where every node has at least degree k . The k -shell is the set of all nodes that belong to the k -core but not to the $(k+1)$ -core. A layer is defined in [101] as the number of peeling passes needed to reach a given node. Authors in [101] also define the onion spectrum, the fraction of nodes that belong to a specific layer. This spectrum can be plotted similarly to an empirical density function. The onion

decomposition has a time complexity of $\mathcal{O}(|\mathcal{E}|\log|\mathcal{V}|)$ as reported in [101], which relates to the eigenvector centrality, a generalization of the PageRank centrality.

3.4 Graph Kernels

Graph Kernels (GK) are one the most used and successful approaches for graph classification. They stand for methods that use (graph) kernel functions, which are similarity functions defined over a pair of graphs, usually passed to a downstream classifier such as SVM [102]. As defined in [26], let \mathbb{G} be a finite set of graphs, \mathcal{H}_k a Hilbert space and $\phi : \mathbb{G} \rightarrow \mathcal{H}_k$. A kernel \mathcal{K} between any two graphs $g, h \in \mathbb{G}$ is denoted as $\mathcal{K}(g, h) = \phi(g) \cdot \phi(h)$, the dot product of their representation in \mathcal{H}_k . Such space and feature map ϕ exist only and if only \mathcal{K} is positive-semidefinite function.

The BON strategy presented earlier can be seen as one realization of an R -Convolution kernel. The R -convolution is a type of kernel based on the comparison of all pairs of substructures of a graph. Any graph vector decomposition can lead to a new graph kernel [103], and many kernels were developed by new ways of decomposing graphs. For instance, another simple yet effective kernel is the histogram vertex kernel [104], where a simple histogram (graph decomposition) is built for every graph counting their node labels. Nonetheless, any BON strategy will lead to a new kernel.

The use of graph kernels must always be taken with care due to the time complexity involved in their calculations, even if the explicit calculation might be avoided. Besides that, the family of Weisfeiler-Lehman (WL) graph kernels [103], based on the Weisfeiler-Lehman Isomorphism Test, is known to be a good baseline for graph classification problems [26]. Nonetheless, there are various graph kernels defined in the literature and the reader should refer to the works of [26, 105] for a broader overview of graph kernel methods.

3.5 Graph Embeddings & Manifold Learning

Let a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{Y})$ where \mathcal{V} is the set of nodes, \mathcal{E} is the set of edges, $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times n}$ is a node attribute matrix with d attributes per node and $\mathbf{Y} \in \mathbb{R}^{|\mathcal{E}| \times m}$ is an edge attribute matrix with m attributes per node. Graph embeddings are a common framework for representing graphs in a vector format, such as $z : \mathcal{G} \rightarrow \mathbb{R}^d$. Their use is motivated because downstream machine learning and analysis tools usually expect vector-based representations instead of adjacency matrices [27].

As depicted in [106], manifold learning resides in a class of learning where the main assumption is that a high-dimensional data can be sufficiently expressed in

a low-dimensional vector. Hence, methods are developed to reduce matrix dimensionality, creating a graph embedding as result. Manifold learning methods are not intrinsically graph problems, but induced into one by the creation of proximity matrices of data points [107].

According to [27] and [93], several proximity matrices were proposed, yet all method share the principle of matrix factorization with some form of graph structure preservation. The resulted low-dimensional matrix can be seen as a graph representation (embedding), with each row representing a node. The reader interested in these methods should initially refer to [27] and [93], but some relevant methods are presented next.

Local Degree Profile (LDP) [44] is a graph embedding based on a vector of node degree statistics of the neighbors of each vertex in the graph. A final graph representation is built aggregating all nodes feature-wise (i.e., using a histogram).

GeoScattering (SCAT) [108] leverages several moments of the graph scattering operation to compose graph-level features. This scattering operation is itself based on diffusion wavelets.

graph2vec [109] is an application of the doc2vec [110] model to the graph realm. Instead of documents, it uses rooted subgraph features extracted using the WL-kernel.

GL2vec [111] builds up on the graph2vec model using line graphs to incorporate edges features.

FGSD [40] , the Family of Spectral Distances, uses the histogram of spectral features derived from the graph Laplacian as whole graph embeddings.

SF [37] is a graph embedding generated by the k smallest positive eigenvalues of the normalized graph Laplacian.

NetLSD [39] defines the heat trace signature, a collection of heat traces at different time scales. This heat trace is associated with a heat diffusion process on the graph.

Feather (FTH) [112] samples over the characteristic function of a node feature distribution over a neighborhood modeled as an r -scale random walk. Features are pooled over node-level embeddings to create the whole graph embedding.

Diffusion Wavelets (WAVE) [113] uses a similar process to [112] of sampling a characteristic function of the distribution of node features over a neighborhood, but using a wavelet function measuring topological similarities to define the distribution. Features are also pooled to generate the whole graph-level embedding matrix.

There are other ubiquitous graph embeddings that use deep neural network models and have also demonstrated interesting results. Such examples are methods based on context generation through random walks like DeepWalk [114], node2vec [115], and struc2vec [116].

3.6 Graph Neural Networks

Graph Neural Networks are currently one of the most popular strategies regarding graph representation. Since a key feature of deep learning is learning a representation as part of the downstream task learning process, the so-called GNNs define the graph representation as part of the end-to-end learning process, almost as a by-product.

The most common graph neural networks are based on the message-passing framework. The message-passing framework is defined in [16] as a neighborhood aggregation strategy, where the representation of a node is iteratively updated by aggregating its neighbors' representation. Each iteration k captures structural information within the k -hop neighborhood of the node. Authors in [16] define the k -th layer of GNN as

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right), h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right), \quad (3.5)$$

where $h_v^{(k)}$ is the feature vector of node v at layer k and $h_v^{(0)}$ is initialized as X_v , the initial node attribute row, and $\mathcal{N}(v)$ is the already defined neighborhood of node v .

Different GNNs can be represented using this framework and summarized as follows.

GraphSAGE The GraphSAGE [117] model is formulated with AGGREGATE as

$$a_v^{(k)} = \text{MAX}\left(\left\{\text{ReLU}\left(W \cdot h_u^{(k-1)}\right), \forall u \in \mathcal{N}(v)\right\}\right), \quad (3.6)$$

where \mathbf{W} is a learnable matrix, MAX is element-wise max-pooling, \cdot represents matrix multiplication, and ReLU is the Rectified Linear Unit activation function [118]. The COMBINE step is formulated as

$$h_v^{(k)} = W \cdot \left(h_v^{(k-1)} | a_v^{(k)}\right), \quad (3.7)$$

where $|$ represents the concatenation of both vectors.

Graph Convolutional Network The Graph Convolutional Network (GCN) [15] is represented with AGGREGATE and COMBINE operations together, leading to

$$h_v^{(k)} = \text{ReLU}\left(W \cdot \text{MEAN}\left(\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v) \cup v\right\}\right)\right) \quad (3.8)$$

where MEAN is element-wise mean-pooling.

Graph Isomorphism Network The Graph Isomorphism Network (GIN) [16] is also represented in this framework with AGGREGATE and COMBINE operations together and the k -layer is defined as

$$h_v^{(k)} = \text{MLP}^{(k)}\left(\left(1 + \epsilon^{(k)}\right) \cdot h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right), \quad (3.9)$$

where ϵ is a learnable parameter or a fixed scalar, MLP is a multi-layer perceptron block.

Most of the groundwork for graph neural networks was done on generating node embeddings. Generally, node embeddings still need some sort of pooling/aggregation/readout operation to create graph-level representations. A standard readout operation is to use element-wise pooling functions, such as max, sum or mean. Hence, a graph-level representation is simply created by adding or averaging node embeddings within a graph. These simple pooling functions were discussed in [16], where authors argue that the sum pooling has more expressive power for multiset representations, like graphs, but other functions may emphasize different graph characteristics.

Since the initial works of [117], complicated aggregation schemes for GNNs were proposed, such as using LSTM [119] models or composing multiple readouts [120, 121]. Many were initially proposed for neighborhood aggregation, but can be applied for generating graph-level representations, like the SortPooling [122], top-k pooling [123], SAG [49]. There are uncountable works that propose different readout methods, but no pooling scheme is capable of superior performance in all datasets at same time [21]. A more complete overview of these methods can be found in [21, 46, 124]. Nevertheless, the work of [3] also presents a more detailed review of graph-level learning approaches.

Chapter 4

K-Means Thermometer

As discussed in Section 2.4, WNN models rely on binary vector as inputs, with the thermometer encoding standing out as the most adopted approach. This is mostly corroborated by the works of [76], where different encodings were evaluated for an image classification problem and the thermometer encodings' performance was clearly the best. Following this work, different thermometers have been proposed, with the Distributive Thermometer [78] becoming a standard in latest WNN research. Even with its omnipresence, results in the original article and in all the literature on discretization, binarization, and quantization methods show a huge path for further thermometer exploration.

Building upon this fact, a new thermometer is proposed in this chapter, based on the *K*-Means algorithm. An evaluation is also performed using almost all the datasets explored in [78] using the Dictionary WiSARD model (referred as WiSARD or WIS for brevity) and the recently proposed DWN [54]. The analysis is also extended to one-hot counterparts of the thermometers and other shallow classifiers such as the NaiveBayes and RandomForest. This evaluation is very important to establish the thermometer encoding as the prevailing binarization procedure for weightless models and evaluate if thermometers are only relevant in the WNN context.

4.1 The *K*-Means Thermometer

Vector quantization (VQ) has been used in the signal processing area as one way of obtaining meaningful yet compact digital representations [125]. A well-known algorithm used for VQ is also a clustering algorithm, the *K*-Means algorithm. The quantization in this case is performed by using the centroids as prototypes of the signals that are closer to it. Hence, given k prototypes, an encoded value $s = 1, \dots, k$ can be used to store the approximated signal.

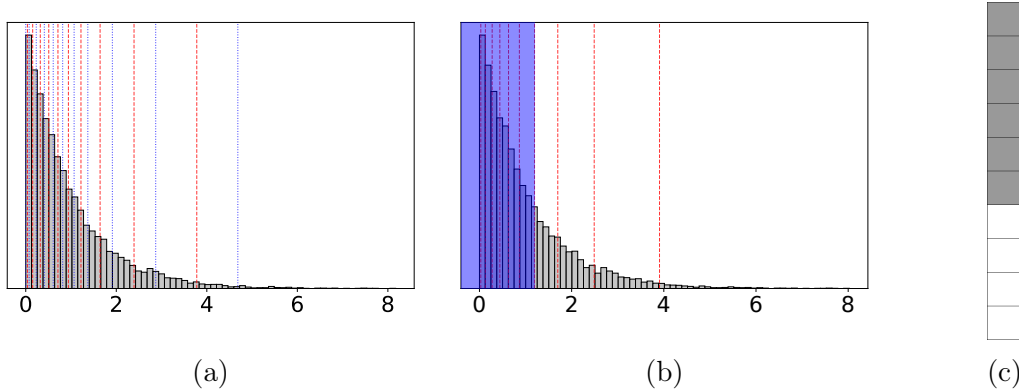


Figure 4.1: The thresholds for a K -Means thermometer are presented in Subfigure 4.1a. Each red dashed line represents a threshold splitting the space in $K + 1$ intervals using $K = 10$ bits. Blue dotted lines illustrate the K -means centroids. The encoding process for the value $x = 1$ is represented in Subfigure 4.1b by the blue-shaded area, which results in the final binary input vector depicted in Subfigure 4.1c.

Finally, a K -Means-based thermometer (KT) is proposed. Instead of using quantiles or a linear approach to binarize values, univariate K -means centroids are used. Each non binary valued attribute distribution can be clustered into K centroids for a vector of size K . These 1-D centroids can be easily sorted in ascending order, with each centroid c having one rank r assigned to it, such that $c_r \leq c_{r+1}$ for $r \in [1, K]$. A naive way of binarizing a value is to first transform it to the rank of its closest centroid. Then, this rank is used as the number of bits marked as 1 in the thermometer.

Another almost similar way of thinking is to use the threshold for bit i as the mean value between c_{i-1} and c_i . This approach is more computationally efficient as it avoids the nearest neighbor calculation, but requires the definition of c_0 . In this work, c_0 is suggested as the minimum observation of the dataset. The KT is demonstrated in Figure 4.1, in a similar way to Figure 2.9. Subfigure 4.1a shows the divided space using K thresholds (red dashed lines) and the K -means centroids (blue dotted lines) obtained by the K -means algorithm with $K = 10$. The encoding of the value $x = 1$ is represented in Subfigure 4.1b, resulting in the final 10-bit vector depicted in Subfigure 4.1c.

As a note, the uni-dimensional problem addressed here is known to be polynomial [126], which is closely related to Jenks Natural Breaks [127] and is frequently referred as the optimal quantization. Nonetheless, there is no evidence in the literature of a thermometer using this quantization as basis for its thresholds.

4.2 Experimental Setup & Results

The experiment analyzed the effects of the use of the proposed K -Means thermometer in several datasets previously used in [78] and [33]. These are relatively small datasets from the UCI datasets ¹. All datasets were evaluated in 80-20 % training-test splits, except for MNIST and FashionMNIST, where the original training-test splits were used. In addition, all models were validated using a ten-fold cross validation using a 90-10% split, with the model with the best validation accuracy being trained in the full training data and evaluated against the test fold. All encoding strategies were subject to the same representation size, i.e. all thermometers had the same search space, being 2,4, 8 or 16 bits for the MNIST and FASHIONMNIST datasets, and 2, 4, 8, 16, 32, 48 or 64 bits for all other datasets. Each pair of classifier and encoding were evaluated with ten different random seeds, except for the DWN ones, which were used five different seeds. The evaluated WiSARD classifier was the DictionaryWiSARD with Bleaching activated derived from the wisardpkg [32]. The DWN architecture was experimented using the authors implementation available on GitHub ², using a fixed two-hidden layer architecture with 2048 and 1024 bits, respectively. The RandomForest and NaiveBayes implementations were taken from the *scikit-learn* [85] python library, with the Bernoulli NaiveBayes implementation being used, except where noted. Full hyperparameter settings are provided in Appendix A.1.

The next sections analyze different aspects of the same experiment. Each section has its dedicated analysis, but all tables (from Table 4.1 to 4.10) share two ranking measures: the average $L1$ -norm between the strategy’s result and the best result for the dataset, and the average rank of the strategy. These values are calculated considering all strategies. A full table with all results is provided in the Appendix B.1.

4.2.1 Thermometers and the WiSARD

In Table 4.1 it is presented the mean accuracy and standard deviation using the WiSARD classifier for all thermometers in all datasets. Many datasets have the Gaussian Thermometer as the best strategy, resulting in the GT being the best ranked thermometer. This result may have occurred because many datasets tend to follow a Gaussian distribution. Concerning the $L1$ -norm, the results show that the Distributive Thermometer is the best alternative, closely followed by the proposed K -Means Thermometer.

¹<https://archive.ics.uci.edu>

²<https://www.github.com/alanbacellar/DWN>

Dataset	Thermometers			
	Distributive	Gaussian	K-Means	Linear
Ecoli	0.819 ± 0.053	0.834 ± 0.023	0.837 ± 0.046	0.834 ± 0.059
EEG Eye State	0.914 ± 0.003	0.767 ± 0.029	0.913 ± 0.005	0.584 ± 0.010
Fashion MNIST	0.826 ± 0.002	0.843 ± 0.001	0.843 ± 0.001	0.839 ± 0.001
Glass	0.776 ± 0.079	0.743 ± 0.051	0.714 ± 0.053	0.700 ± 0.052
Iris	0.937 ± 0.040	0.950 ± 0.036	0.947 ± 0.023	0.957 ± 0.027
Letter	0.942 ± 0.004	0.948 ± 0.003	0.944 ± 0.003	0.942 ± 0.003
Maggic Gamma Telescope	0.856 ± 0.006	0.847 ± 0.005	0.857 ± 0.010	0.834 ± 0.003
MNIST	0.949 ± 0.001	0.962 ± 0.001	0.959 ± 0.001	0.961 ± 0.001
SatImage	0.899 ± 0.006	0.900 ± 0.007	0.899 ± 0.009	0.896 ± 0.008
Shuttle	0.998 ± 0.001	0.998 ± 0.000	0.998 ± 0.000	0.994 ± 0.005
Wine	0.983 ± 0.020	0.969 ± 0.025	0.949 ± 0.030	0.974 ± 0.025
Avg. $L1$ -Norm	0.025	0.037	0.028	0.059
Avg. Rank	13.091	11.909	13.636	16.545

Table 4.1: Mean accuracy and standard deviation for all evaluated thermometers in all datasets (mean over 10 runs with different seeds) for the WiSARD classifier. Best results are highlighted with a black background.

4.2.2 One-Hot Counterparts for the WiSARD

As mentioned in Section 2.4, one-hot binary vectors can be used to represent discrete values that divide the sample space. If thermometers define $B + 1$ intervals using B thresholds and B bits, a similar one-hot encoding can be constructed using the same B bits, but B intervals. Instead of filling all bits up to to i -th bit of the vector, a one-hot counterpart of a thermometer uses the same method used to define thresholds in the thermometer, but define B intervals and only sets to one the bit related to the interval into which the value fall. Hence, a one-hot counterpart does not account for the proportionality between the $L1$ -norm of the original values and the Hamming distance between the encoded values.

Moreover, in addition to the initial evaluation, it was also evaluated whether the thermometers are a better option than using one-hot counterparts (OH) for WNN. This was done evaluating values encoded using one-hot counterparts of the evaluated thermometers in conjunction with the WiSARD model. The one-hot counterparts were developed using the *KBinsDiscretizer* class readily available from the *scikit-learn* python library [85].

Table 4.2 presents the results of this evaluation. In comparison with Table 4.1, the first conclusion is that the thermometer is indeed a better encoding for the WiSARD classifier than the OH counterparts in terms of accuracy. This fact is consistent with previous literature [76]. Among the counterparts, the K -Means variation is a clear winner, with the other two having a similar performance. Moreover, an interesting fact is that this one-hot variation has a better result regarding the average $L1$ -norm than the Gaussian and the Linear thermometers.

Dataset	One-Hot Counterparts		
	Distributive	K-Means	Linear
Ecoli	0.769 ± 0.048	0.799 ± 0.062	0.800 ± 0.045
EEG Eye State	0.751 ± 0.012	0.764 ± 0.008	0.615 ± 0.016
Fashion MNIST	0.812 ± 0.002	0.814 ± 0.003	0.803 ± 0.002
Glass	0.629 ± 0.094	0.674 ± 0.073	0.614 ± 0.070
Iris	0.920 ± 0.028	0.933 ± 0.042	0.913 ± 0.036
Letter	0.767 ± 0.007	0.784 ± 0.019	0.782 ± 0.011
Maggic Gamma Telescope	0.790 ± 0.010	0.800 ± 0.012	0.802 ± 0.009
MNIST	0.931 ± 0.002	0.946 ± 0.002	0.944 ± 0.002
SatImage	0.861 ± 0.008	0.869 ± 0.008	0.865 ± 0.008
Shuttle	0.993 ± 0.003	0.995 ± 0.002	0.996 ± 0.003
Wine	0.923 ± 0.066	0.911 ± 0.041	0.931 ± 0.045
Avg. $L1$ -Norm	0.093	0.031	0.1
Avg. Rank	26.364	14.636	26.364

Table 4.2: Mean accuracy and standard deviation for all evaluated one-hot counterparts in all datasets (mean over 10 runs with different seeds) for the WiSARD classifier. Best results are highlighted with a black background.

4.2.3 Thermometers and the DWN

Continuing on the analysis of thermometers for WNN, it was examined how the different thermometers reflect on the recent DWN architecture. These results are shown in Table 4.3, which shows a similar behavior to Table 4.1, where many datasets have good results with the Gaussian Thermometer. In spite of this fact, the K -Means Thermometer is the best ranked thermometer. Also, with respect to the $L1$ -norm, the Distributive Thermometer is again the winner, but with even closer results compared to the WiSARD evaluation.

Dataset	Thermometers			
	Distributive	Gaussian	K-Means	Linear
Ecoli	0.815 ± 0.035	0.825 ± 0.043	0.834 ± 0.029	0.816 ± 0.049
EEG Eye State	0.888 ± 0.005	0.721 ± 0.030	0.883 ± 0.009	0.544 ± 0.035
Fashion MNIST	0.870 ± 0.003	0.869 ± 0.003	0.874 ± 0.003	0.870 ± 0.003
Glass	0.767 ± 0.079	0.752 ± 0.052	0.740 ± 0.072	0.710 ± 0.074
Iris	0.940 ± 0.034	0.950 ± 0.032	0.943 ± 0.032	0.923 ± 0.074
Letter	0.954 ± 0.003	0.953 ± 0.003	0.953 ± 0.004	0.952 ± 0.003
Maggic Gamma Telescope	0.816 ± 0.010	0.821 ± 0.009	0.818 ± 0.011	0.800 ± 0.016
MNIST	0.980 ± 0.000	0.981 ± 0.001	0.981 ± 0.001	0.981 ± 0.001
SatImage	0.899 ± 0.010	0.899 ± 0.011	0.896 ± 0.005	0.896 ± 0.009
Shuttle	0.998 ± 0.001	0.997 ± 0.001	0.998 ± 0.001	0.996 ± 0.007
Wine	0.963 ± 0.036	0.966 ± 0.030	0.960 ± 0.024	0.963 ± 0.038
Avg. $L1$ -Norm	0.025	0.039	0.026	0.065
Avg. Rank	12.000	11.909	11.455	17.727

Table 4.3: Mean accuracy and standard deviation for all evaluated thermometers in all datasets (mean over 5 runs with different seeds) for the DWN architecture. Best results are highlighted with a black background.

4.2.4 One-hot Counterparts for the DWN

Similar to Section 4.2.2, the effects of one-hot counterparts on the DWN were evaluated. In Table 4.4 the reader can see that the DWN exhibits a similar behavior to the WiSARD classifier, where thermometers play a key role in the final accuracy of the model. Comparing this table with Table 4.3, all thermometers counterparts perform better than their one-hot variations.

Dataset	One-Hot Counterparts		
	Distributive	K-Means	Linear
Ecoli	0.751 ± 0.037	0.760 ± 0.052	0.766 ± 0.037
EEG Eye State	0.776 ± 0.024	0.733 ± 0.018	0.534 ± 0.041
Fashion MNIST	0.859 ± 0.003	0.861 ± 0.004	0.860 ± 0.004
Glass	0.679 ± 0.052	0.648 ± 0.049	0.667 ± 0.082
Iris	0.910 ± 0.042	0.913 ± 0.055	0.903 ± 0.037
Letter	0.908 ± 0.010	0.891 ± 0.006	0.891 ± 0.006
Maggic Gamma Telescope	0.803 ± 0.013	0.787 ± 0.014	0.771 ± 0.012
MNIST	0.977 ± 0.001	0.978 ± 0.001	0.978 ± 0.001
SatImage	0.870 ± 0.013	0.871 ± 0.007	0.868 ± 0.012
Shuttle	0.998 ± 0.001	0.998 ± 0.001	0.993 ± 0.007
Wine	0.909 ± 0.044	0.909 ± 0.040	0.906 ± 0.067
Avg. L1-Norm	0.066	0.074	0.094
Avg. Rank	21.727	21.909	26.000

Table 4.4: Mean accuracy and standard deviation for all evaluated one-hot counterparts in all datasets (mean over 5 runs with different seeds) for the DWN classifier. Best results are highlighted with a black background.

4.2.5 On the Effects of Thermometers in Non-WNN Models

It was also verified whether these results persist when using other classifiers, such as the RandomForest and the NaiveBayes. This analysis is important to assess whether other types of classifiers may benefit from encodings that were originally developed for WNN models. Table 4.5 shows the mean accuracy and standard deviation of all thermometers in all datasets for the RandomForest classifier. It is interesting to see that for this classifier, the *K*-Means thermometer is the best thermometer alternative. Results for the RandomForest classifier using thermometers are generally better than any WNN classifier. This is somewhat expected because it is a very strong ensemble method.

Table 4.6 presents the same metrics for the NaiveBayes classifier, which performs poorly compared to any other classifiers, except for the Wine dataset, where it has the best performance over all strategies when using the Distributive Thermometer. The results also suggest that the proposed *K*-Means Thermometer when combined with the RandomForest classifier is the strongest thermometer strategy overall.

Dataset	Thermometers			
	Distributive	Gaussian	K-Means	Linear
Ecoli	0.834 ± 0.040	0.839 ± 0.033	0.848 ± 0.047	0.843 ± 0.042
EEG Eye State	0.940 ± 0.004	0.851 ± 0.022	0.944 ± 0.005	0.612 ± 0.025
Fashion MNIST	0.872 ± 0.001	0.877 ± 0.001	0.879 ± 0.001	0.877 ± 0.002
Glass	0.767 ± 0.071	0.767 ± 0.047	0.783 ± 0.082	0.774 ± 0.072
Iris	0.937 ± 0.029	0.947 ± 0.017	0.940 ± 0.026	0.957 ± 0.032
Letter	0.960 ± 0.003	0.963 ± 0.002	0.962 ± 0.003	0.967 ± 0.001
Maggic Gamma Telescope	0.879 ± 0.005	0.879 ± 0.005	0.879 ± 0.005	0.869 ± 0.005
MNIST	0.970 ± 0.000	0.973 ± 0.001	0.973 ± 0.001	0.972 ± 0.001
SatImage	0.911 ± 0.007	0.912 ± 0.006	0.912 ± 0.007	0.911 ± 0.007
Shuttle	0.998 ± 0.001	0.999 ± 0.001	0.999 ± 0.000	0.998 ± 0.000
Wine	0.974 ± 0.021	0.983 ± 0.015	0.971 ± 0.023	0.983 ± 0.015
Avg. $L1$ -Norm	0.011	0.016	0.007	0.037
Avg. Rank	8.091	5.727	5.455	7.455

Table 4.5: Mean accuracy and standard deviation for all evaluated thermometers in all datasets (mean over 10 runs with different seeds) for the RF classifier. Best results are highlighted with a black background.

4.2.6 One-Hot Counterparts in Non-WNN Models

In a similar fashion to the analysis for the WNN models, one-hot counterparts were examined in non-WNN classifiers alongside a No-Encoding strategy (NoE). A caveat is that the NoE strategy cannot be directly applied to the NaiveBayes, where a Gaussian Naive Bayes was used instead. Table 4.7 presents results for the RandomForest model. Comparing it with related previous table, it can be also concluded that using a thermometer is better than using a one-hot counterparts for the RandomForest, except for the Linear Thermometer, which performs relatively similar to some one-hot strategies regarding the RandomForest classifier. The clear fact is that the No-Encoding strategy is still superior and the best strategy overall.

Considering the NaiveBayes results in Table 4.8, the one-hot counterparts performed better or similar to the respective thermometers, implying this classifier might not benefit from thermometer encodings. This is quite understandable due to the independence assumption that is the base of this classifier. The thermometer encoding creates a relationship of dependence within the bits of an attribute such that each i -th bit can only occur if the $(i - 1)$ -th has occurred. As a final remark, the NoE Naive Bayes alternative performs the worst among all alternatives using NaiveBayes.

4.2.7 Other Encodings

Another important evaluation is the performance evaluation of other encodings proposed and used in the WNN literature in comparison with the thermometers. The use of the CMAC encoding and the Minchinton Cell Type-0 (MC) encoding was an-

Dataset	Thermometers			
	Distributive	Gaussian	K-Means	Linear
Ecoli	0.806 ± 0.047	0.827 ± 0.047	0.828 ± 0.046	0.840 ± 0.052
EEG Eye State	0.603 ± 0.013	0.577 ± 0.011	0.617 ± 0.013	0.584 ± 0.012
Fashion MNIST	0.671 ± 0.000	0.709 ± 0.000	0.717 ± 0.003	0.709 ± 0.000
Glass	0.605 ± 0.071	0.650 ± 0.078	0.617 ± 0.070	0.581 ± 0.067
Iris	0.937 ± 0.029	0.933 ± 0.035	0.947 ± 0.023	0.943 ± 0.035
Letter	0.658 ± 0.008	0.671 ± 0.007	0.677 ± 0.009	0.691 ± 0.007
Maggic Gamma Telescope	0.785 ± 0.009	0.775 ± 0.005	0.793 ± 0.009	0.786 ± 0.006
MNIST	0.837 ± 0.000	0.846 ± 0.001	0.846 ± 0.000	0.847 ± 0.000
SatImage	0.779 ± 0.007	0.788 ± 0.009	0.792 ± 0.010	0.799 ± 0.013
Shuttle	0.795 ± 0.006	0.838 ± 0.003	0.857 ± 0.010	0.904 ± 0.013
Wine	0.986 ± 0.015	0.977 ± 0.012	0.983 ± 0.015	0.971 ± 0.027
Avg. $L1$ -Norm	0.155	0.143	0.136	0.126
Avg. Rank	28.636	27.545	24.818	25.727

Table 4.6: Mean accuracy and standard deviation for all evaluated thermometers in all datasets (mean over 10 runs with different seeds) for the NaiveBayes classifier. Best results are highlighted with a black background.

Dataset	One-Hot Counterparts			
	NoEncoding	Distributive	K-Means	Linear
Ecoli	0.851 ± 0.042	0.799 ± 0.061	0.764 ± 0.053	0.801 ± 0.040
EEG Eye State	0.935 ± 0.005	0.865 ± 0.005	0.855 ± 0.012	0.619 ± 0.018
Fashion MNIST	0.878 ± 0.001	0.866 ± 0.001	0.868 ± 0.001	0.868 ± 0.001
Glass	0.802 ± 0.045	0.745 ± 0.084	0.717 ± 0.062	0.667 ± 0.059
Iris	0.943 ± 0.032	0.937 ± 0.033	0.957 ± 0.032	0.940 ± 0.044
Letter	0.964 ± 0.003	0.928 ± 0.003	0.912 ± 0.004	0.911 ± 0.004
Maggic Gamma Telescope	0.883 ± 0.006	0.858 ± 0.005	0.853 ± 0.008	0.846 ± 0.007
MNIST	0.971 ± 0.001	0.966 ± 0.000	0.968 ± 0.001	0.967 ± 0.001
SatImage	0.913 ± 0.006	0.886 ± 0.007	0.892 ± 0.009	0.892 ± 0.008
Shuttle	0.999 ± 0.000	0.998 ± 0.000	0.998 ± 0.001	0.997 ± 0.003
Wine	0.969 ± 0.037	0.980 ± 0.027	0.957 ± 0.031	0.963 ± 0.036
Avg. $L1$ -Norm	0.005	0.08	0.039	0.063
Avg. Rank	5.182	23.636	15.273	18.818

Table 4.7: Mean accuracy and standard deviation for all evaluated one-hot counterparts + the No-Encoding approach in all datasets (mean over 10 runs with different seeds) for the RandomForest classifier.

alyzed, instead of thermometers and one-hot encodings. Table 4.9 shows the mean accuracy and standard deviation of the CMAC and Minchinton Cell encodings on all datasets for both evaluated WNN classifiers. It shows that the CMAC encoding is generally better than the Linear Thermometer, but also generally worse than any other thermometer strategies for the WiSARD classifier. When using the DWN, the CMAC performed worse than the Linear Thermometer. The Minchinton Cell’s strategies are generally worse than any other strategy, but in one dataset (Fashion-MNIST) it has achieved the best result overall.

When dealing with non-WNN classifiers, Table 4.10 shows that the adoption of the CMAC encoding produces better results in terms of accuracy than using Minchinton Cells. The RF+CMAC strategy is the fourth best strategy regarding the $L1$ -Norm average, only losing to the complex thermometers and to the null

Dataset	NoEncoding	One-Hot Counterparts		
		Distributive	K-Means	Linear
Ecoli	0.757 ± 0.058	0.803 ± 0.049	0.822 ± 0.052	0.804 ± 0.049
EEG Eye State	0.451 ± 0.010	0.679 ± 0.011	0.682 ± 0.012	0.585 ± 0.017
Fashion MNIST	0.586 ± 0.000	0.730 ± 0.000	0.736 ± 0.000	0.735 ± 0.000
Glass	0.352 ± 0.128	0.643 ± 0.079	0.636 ± 0.071	0.648 ± 0.058
Iris	0.950 ± 0.032	0.910 ± 0.067	0.930 ± 0.037	0.920 ± 0.028
Letter	0.643 ± 0.007	0.729 ± 0.006	0.726 ± 0.006	0.726 ± 0.007
Maggic Gamma Telescope	0.726 ± 0.009	0.773 ± 0.010	0.776 ± 0.008	0.768 ± 0.009
MNIST	0.556 ± 0.000	0.835 ± 0.000	0.845 ± 0.000	0.843 ± 0.000
SatImage	0.794 ± 0.010	0.821 ± 0.012	0.821 ± 0.012	0.821 ± 0.013
Shuttle	0.685 ± 0.011	0.933 ± 0.002	0.936 ± 0.003	0.922 ± 0.006
Wine	0.963 ± 0.036	0.966 ± 0.023	0.969 ± 0.021	0.960 ± 0.031
Avg. L1-Norm	0.246	0.122	0.117	0.13
Avg. Rank	32.000	27.273	25.091	28.364

Table 4.8: Mean accuracy and standard deviation for all evaluated one-hot counterparts + the No-Encoding approach in all datasets (mean over 10 runs with different seeds) for the NaiveBayes classifier.

Dataset	CMAC		Minchinton	
	WiSARD	DWN	WiSARD	DWN
Ecoli	0.855 ± 0.042	0.807 ± 0.048	0.790 ± 0.051	0.757 ± 0.043
EEG Eye State	0.713 ± 0.035	0.800 ± 0.030	0.674 ± 0.009	0.557 ± 0.049
Fashion MNIST	0.837 ± 0.003	0.870 ± 0.003	0.840 ± 0.001	0.879 ± 0.002
Glass	0.721 ± 0.076	0.717 ± 0.070	0.593 ± 0.084	0.574 ± 0.079
Iris	0.953 ± 0.039	0.910 ± 0.035	0.627 ± 0.058	0.660 ± 0.086
Letter	0.948 ± 0.004	0.951 ± 0.004	0.913 ± 0.004	0.922 ± 0.005
Maggic Gamma Telescope	0.840 ± 0.004	0.812 ± 0.017	0.710 ± 0.009	0.643 ± 0.029
MNIST	0.959 ± 0.002	0.981 ± 0.001	0.962 ± 0.001	0.978 ± 0.001
SatImage	0.898 ± 0.007	0.890 ± 0.012	0.794 ± 0.010	0.763 ± 0.015
Shuttle	0.998 ± 0.000	0.997 ± 0.004	0.928 ± 0.003	0.909 ± 0.011
Wine	0.960 ± 0.034	0.934 ± 0.056	0.843 ± 0.034	0.794 ± 0.046
Avg. L1-Norm	0.044	0.045	0.136	0.158
Avg. Rank	13.909	16.818	28.909	28.545

Table 4.9: Mean accuracy and standard deviation for all evaluated CMAC and Minchinton Cell’s approach in all datasets (mean over 10 runs with different seeds) for WNN classifiers. Best results are highlighted with a black background

reference. The Minchinton Cell encoding is generally worse than all other strategies presented, but for the MNIST and FashionMNIST datasets it has a comparable performance.

4.2.8 Visualizing the Thresholds

In order to understand the similarities between the encodings, Figure 4.2 shows a diagram similar to the one presented in [78]. This diagram compares the thresholds generated by the evaluated thermometers when the data follows a specific distribution. In this case, three distributions are plotted using histograms: an exponential distribution with $\lambda = 1$, the rate parameter; a Gaussian distribution with mean $\mu = 128$ and standard deviation $\sigma = 40$, and a Bimodal Gaussian composition with means $\mu_1 = 5$ and $\mu_2 = 255$ and standard deviations $\sigma_1 = 10$ and $\sigma_2 = 40$. The thresholds for each thermometer are plotted using dashed vertical lines. Both

Dataset	CMAC		Minchinton	
	RandomForest	NaiveBayes	RandomForest	NaiveBayes
Ecoli	0.845 ± 0.050	0.840 ± 0.062	0.778 ± 0.055	0.743 ± 0.029
EEG Eye State	0.873 ± 0.015	0.666 ± 0.011	0.677 ± 0.007	0.622 ± 0.009
Fashion MNIST	0.875 ± 0.001	0.708 ± 0.000	0.875 ± 0.001	0.720 ± 0.000
Glass	0.771 ± 0.075	0.633 ± 0.062	0.614 ± 0.077	0.579 ± 0.079
Iris	0.953 ± 0.032	0.940 ± 0.026	0.627 ± 0.058	0.623 ± 0.052
Letter	0.968 ± 0.003	0.692 ± 0.008	0.935 ± 0.003	0.591 ± 0.005
Maggic Gamma Telescope	0.874 ± 0.006	0.787 ± 0.005	0.712 ± 0.008	0.646 ± 0.010
MNIST	0.972 ± 0.000	0.848 ± 0.000	0.970 ± 0.001	0.846 ± 0.000
SatImage	0.913 ± 0.007	0.796 ± 0.009	0.809 ± 0.011	0.720 ± 0.017
Shuttle	0.999 ± 0.000	0.917 ± 0.005	0.929 ± 0.002	0.831 ± 0.003
Wine	0.969 ± 0.025	0.960 ± 0.020	0.854 ± 0.048	0.780 ± 0.050
Avg. L1-Norm	0.014	0.137	0.126	0.224
Avg. Rank	5.455	25.727	26.000	35.182

Table 4.10: Mean accuracy and standard deviation for all evaluated CMAC and Minchinton Cell’s approach in all datasets (mean over 10 runs with different seeds) for non-WNN classifiers. Best results are highlighted with a black background

K -Means and the Distributive exhibit similar behavior of fitting the desired distributions but with clear differences between the final thresholds.

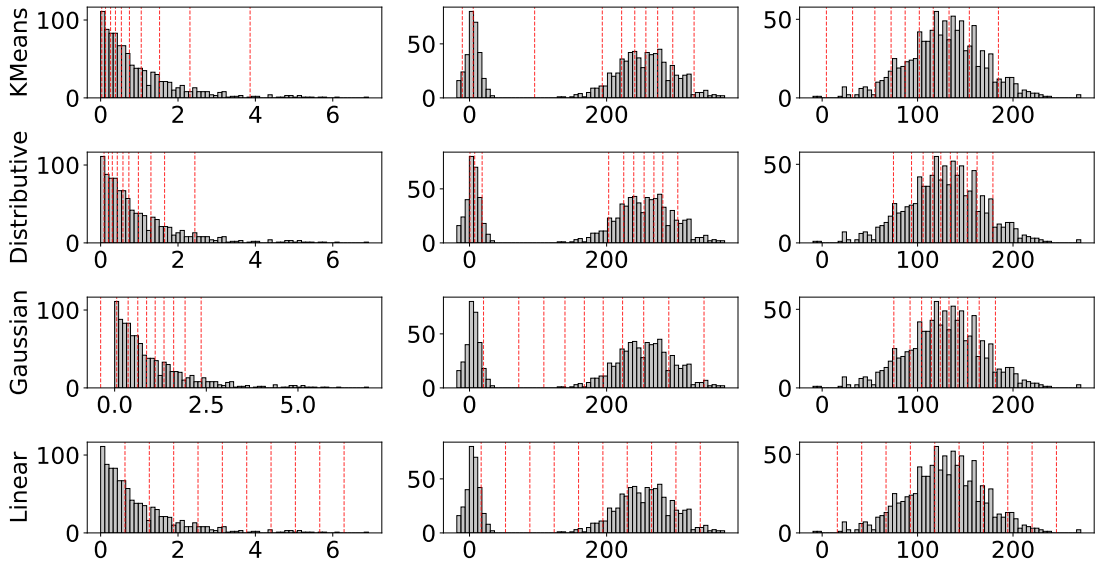


Figure 4.2: Histogram plot of distributions (Exponential $\sim \lambda = 1$, Bimodal $\sim \mu_1 = 5, \mu_2 = 255, \sigma_1 = 10, \sigma_2 = 40$, and Unimodal Gaussian $\sim \mu = 128, \sigma = 40$) and their division by each thermometer strategy. The dashed vertical lines are the thermometer thresholds.

4.3 Conclusion

This chapter has debated the concept of binarization for WNN models, introducing a new thermometer encoding based on the vector quantization theory and the famous K -Means algorithm. It has also evaluated this new encoding and other

thermometers on different datasets. Thermometers were also compared with their one-hot counterparts and other encodings previously adopted in the literature for both WNN and non-WNN models. Yet, the results presented in this chapter confirm that the Distributive Thermometer is the prevailing thermometer strategy. Despite this fact, the proposed thermometer is a better solution in several cases than the omnipresent Distributive Thermometer. While some encodings do not present the best results overall, they are still capable of reaching even better performance than the complex thermometers. The findings also indicate a cost of using binarized inputs compared to the No-Encoding strategy that has to be minimized. While the literature indicates that this cost can be minimized by using bigger thermometers, this fact still has to be verified.

Chapter 5

Weightless Primitives to Graph Classification

After the previous analysis of the binarization problem, this chapter dives into the task of weightless graph classification. The goal of the graph classification task is to assign a categorical label to an unlabeled graph which is given as input. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{Y})$, a model learns a function $h \approx l(\mathcal{G})$, where $l : \mathcal{G} \rightarrow \Sigma$ and Σ is a closed set of possible labels for a graph. In other words, graph classification is a machine learning classification problem where the inputs are graphs and the labels are graph-level labels or properties.

First, an initial attempt leverages existing tools to address the problem of graph classification: established graph embeddings and Distributive thermometer binarization. After this brief evaluation, a second architecture is described, evaluated, and discussed. This approach puts aside regular thermometer-based approaches in favor of a whole-distribution binarization. Although simple, both architectures presented here show very promising results, while preserving all the benefits of using weightless neural networks.

5.1 Leveraging Graph Embeddings for Weightless Graph Classification

As described in Section 3.5, graph embeddings are a common tool for solving graph classification problems, especially when dealing with shallow learning, where unsupervised graph embeddings are very important [128]. Here, a selection of widely spread unsupervised graph embeddings is evaluated in the context of weightless graph classification. A non-weightless baseline is compared against a proposed vanilla weightless architecture.

5.1.1 A Vanilla Architecture Using Graph Embeddings

The proposed architecture can be thought of as a three-function composition: $f = C \circ PB \circ F$, with a feature extraction step - F , a pre-processing and binarization unit - PB , and a classifier - C , on the edge of it. The graph feature extraction is performed by generating graph-level embeddings and these graph-level embeddings form the basis for a final binary vector. The final binary vectors are conceived using the Distributive Thermometer, which constitutes the binarization step, and are then fed to a final classifier. This architecture is summarized by Figure 5.1.

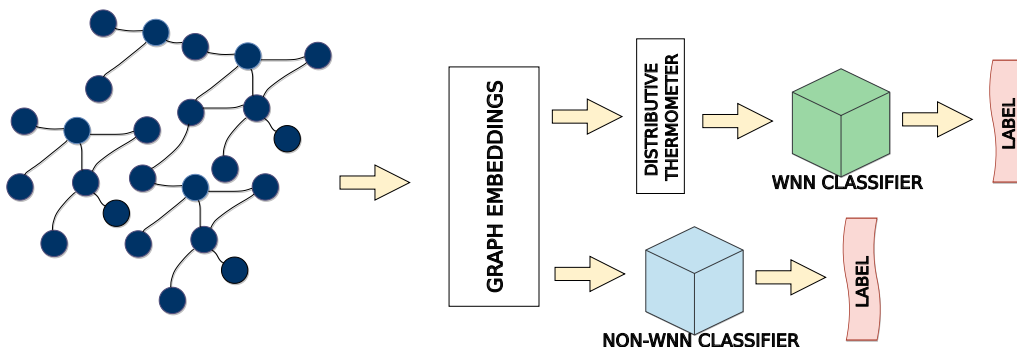


Figure 5.1: Global big picture of the graph embeddings classification framework - graph embeddings are obtained from the graph. Thus, these embeddings are binarized or not depending on the final classifier and used as input for training or inference.

5.1.2 Experimental Setup & Results

In the following sections, the experimental setup such as datasets and hyperparameter settings are established, and results are presented and discussed.

Datasets

The task of graph classification has brought to life innumerable benchmarks and datasets. The datasets used in this initial evaluation were the IMDB-M and COLLAB, composed of unattributed and undirected graphs. These datasets are part of the TUDatasets [129], a collection of datasets that have been used in a variety of recent works regarding graph classification and are considered to be small-size datasets. Tables 5.1 and 5.2 summarize a selection of datasets from this collection that were used not only in this section, but throughout the work, representing three types of graphs: social networks (SocNet), bioinformatics (BioInf), and small molecules (SmlMol).

dataset	type	# Graphs	#Classes	Avg. Nodes	Avg. Edges
COLLAB [130]	SocNet	5000	3	74.49	2457.78
ENZYMES [11, 131]	BioInf	600	6	32.63	62.14
IMDB-M [130]	SocNet	1500	3	13.00	65.94
MUTAG [8, 132]	SmlMol	188	2	17.93	19.79
NCI1 [10, 103]	SmlMol	4110	2	29.87	32.30
PROTEINS [11, 133, 134]	BioInf	1113	2	39.06	72.82

Table 5.1: Statistics of Selected TUDatasets

dataset	Node Labels	Edge Labels	Node Attr.	Edge Attr.
COLLAB	-	-	-	-
ENZYMES	Y	-	Y(18)	-
IMDB-M	-	-	-	-
MUTAG	Y	Y	-	-
NCI1	Y	-	-	-
PROTEINS	Y	-	Y(1)	-

Table 5.2: Attributes of Selected TUDatasets

Hyperparameter Evaluation

All graph-level embeddings described in Section 3.5 were evaluated thanks to their availability provided from the KarateClub library [128]. No hyperparameter search was done for these graph embeddings and they were used with their default (and possibly best) parameters. Different WNN were evaluated: (Dictionary) WiSARD, ClusWiSARD, BTHOWeN, and ULEEN. Moreover, a RandomForest baseline is included, which does not make use of any binarization. Also, all classifiers have their hyperparameters such as *addressSize*, for WiSARD, *addressSize*, *threshold*, *discriminatorsLimit*, and *minScore* for the ClusWiSARD, *addressSize*, *unitHashes*, and *unitEntries* for the BTHOWeN, and *n_estimators*, *criterion*, and *max_features* for the RandomForest. The ULEEN was trained using a single configuration for all classifiers in the ensemble. This configuration has the same parameters as BTHOWeN plus others such as *numberClassifiers* and *numEpochs*. The parameter selection methodology was exactly the same as in [135] and a grid search was used to find the best validation parameters and further use in a test phase. For WNN classifiers, three possibilities of thermometer sizes (2,5,10) were tested. Table A.2 in Appendix A.2 gives a full description of the hyperparameter set for this experiment.

Brief Results & Discussion

Table 5.3 shows the results for the experiment and in both datasets the Random-Forest baseline had the best overall performance. In general, all graph embeddings performed better with the RandomForest classifier than with the weightless models. Concerning the WNN classifiers, the WiSARD model has the best result, regardless of the dataset. As for the graph embeddings, LDP, SCAT, WAVE, and FTH have topped the charts with no clear winner overall. A clear fact is that WNN models tend to give better results when combined WAVE and FTH than when combined with other embeddings. Also, ULEEN models were not able to surpass the WiSARD, even being complex ensembles. One possible reason for this fact is the reduced number of epochs put as a constraint, but this needs to be further studied.

IMDB-M					
	RandomForest	WiSARD	ClusWISARD	BTHOWeN	ULEEN
LDP	66.1 ± 3.6	59.8 ± 3.3	63.8 ± 2.5	62.0 ± 3.3	62.4 ± 2.6
SCAT	65.7 ± 2.8	61.8 ± 2.9	64.5 ± 2.5	63.6 ± 3.4	62.3 ± 3.8
WAVE	65.5 ± 2.1	62.3 ± 1.6	64.4 ± 2.8	64.5 ± 2.4	63.7 ± 2.7
FTH	65.2 ± 2.7	64.8 ± 2.7	64.8 ± 2.2	64.3 ± 2.2	62.8 ± 3.5
FGSD	64.9 ± 2.2	57.9 ± 3.7	62.8 ± 1.5	61.5 ± 2.6	60.3 ± 1.7
SF	64.9 ± 3.1	54.8 ± 3.0	61.4 ± 3.1	61.9 ± 2.0	60.9 ± 2.4
NetLSD	64.4 ± 3.0	61.3 ± 1.6	64.5 ± 2.3	63.2 ± 2.2	62.7 ± 2.6
gl2vec	56.7 ± 3.0	OOO	55.0 ± 2.8	55.7 ± 2.9	56.6 ± 2.1
graph2vec	56.4 ± 1.7	55.5 ± 2.9	56.1 ± 2.3	55.6 ± 1.4	55.9 ± 1.1
COLLAB					
	RandomForest	WiSARD	ClusWISARD	BTHOWeN	ULEEN
WAVE	86.5 ± 1.3	86.0 ± 1.2	85.7 ± 1.4	OOO	OOO
LDP	86.0 ± 1.5	80.0 ± 2.3	80.8 ± 1.4	77.3 ± 2.7	73.4 ± 4.5
FTH	85.5 ± 1.2	84.3 ± 1.2	84.8 ± 1.0	76.6 ± 11.5	OOO
SCAT	84.8 ± 0.9	83.0 ± 1.1	83.0 ± 1.5	76.2 ± 8.5	75.9 ± 2.8
FGSD	83.3 ± 0.9	75.7 ± 1.5	77.5 ± 1.1	76.1 ± 1.2	67.4 ± 6.5
SF	82.5 ± 1.1	78.0 ± 1.1	78.9 ± 1.5	77.1 ± 3.2	72.7 ± 2.4
NetLSD	80.1 ± 0.9	78.4 ± 1.7	79.1 ± 0.9	75.7 ± 7.4	69.8 ± 7.3
graph2vec	68.4 ± 3.3	61.1 ± 6.9	64.0 ± 5.5	66.8 ± 4.7	58.2 ± 5.7

Table 5.3: Accuracy and standard deviation for all evaluated classifiers and graph embeddings on the IMDB-M and COLLAB dataset. Best results are highlighted with a black background, second best with a gray background.

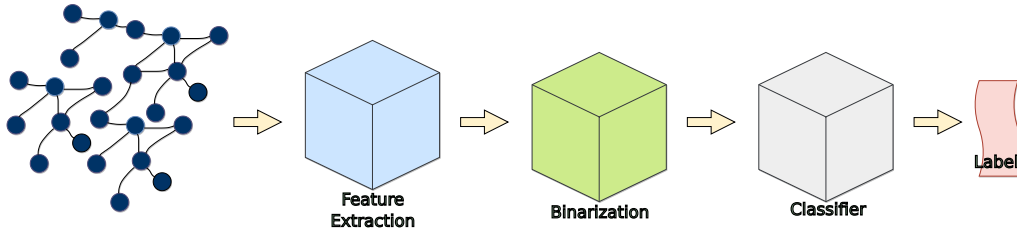


Figure 5.2: Global big picture of a shallow classification framework - a weightless architecture composed of three steps: a feature extraction step - F , a pre-processing and binarization unit - PB , and a classifier - C , at the edge of it.

5.2 A Bag of Nodes Primer on Weightless Graph Classification

Another architecture can be thought using the same structure of three consecutive blocks: a graph measure extraction step (F), a binarization step (PB) and a final classifier (C). The feature extraction step is mandatory due to the necessity of presenting the graph structure as a vectorized learning piece to a final classifier, but graph embeddings may not be necessary. The idea here is to build a full graph representation as an aggregation of node and edge attributes. The binarization step is a requirement when dealing with WNN, but it can also be applied and evaluated using other classifiers.

This architecture specificity lies in the definition of the F and PB steps. With respect to the F step, the adoption of a single measure of the graph, such as the node degree, is proposed to create a representation of each graph. A collection of 2-D points for each node is then built. For each node, the first coordinate is the value of the measure for the current node and the second coordinate is the fraction of values in the graph that is greater than the current value. This is also known as the empirical complimentary cumulative distribution (ECCDF) of the measure. An example of such plot can be seen in Figure 5.3, where the PageRank node centrality is plotted. The X -axis represents an observed value and the Y -axis the fraction of values that is lower than the observed value.

The PB binarization step resides in the application of one of off-the-shelf binarization strategies. The only requirement is the ability of deal with a distribution plot, which the thermometer does not. Some of the candidates are described in Section 2.4 - KDTree and KernelCanvas, plus two proposed in the following. Finally, the input vector is the binary vector produced by the binarization step. This architecture is summarized by Figure 5.2, where the input graph passes through the three colored boxes representing the steps described above and an output label is produced.

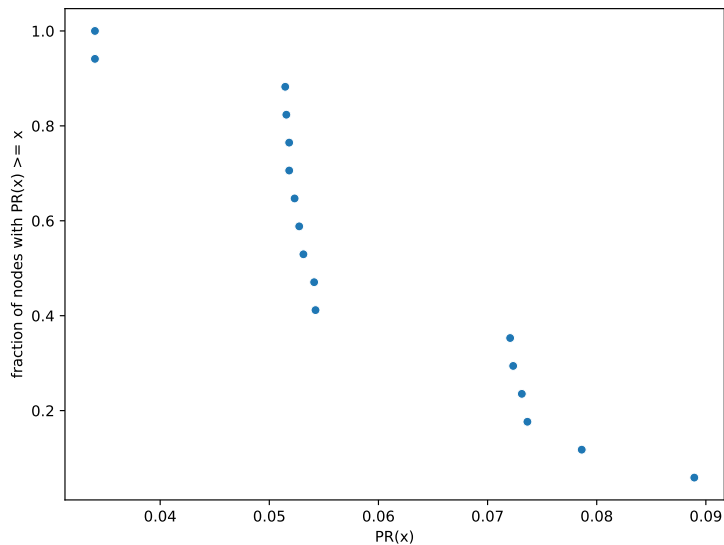


Figure 5.3: Example of an ECCDF plot of the PageRank node centrality. The X-axis represents an observed value and the Y-axis the fraction of values that is lower than the observed value.

5.2.1 On Whole Distribution Binarization

In this section, two binarization strategies developed during this work are formalized: Plot Binarization and Histogram. Different from thermometers, which binarize scalar values, their main goal is to be able to capture a whole distribution (such as an entire node degree distribution of graph) as a single input vector.

Plot Binarization

The Plot binarization is a simple binarization process that mimics the scatter plot of a vector of 2-D points into a canvas, similar to the use of a visualization tool like Matplotlib¹. The plane is divided into a regular grid with a resolution of *width* columns and *height* rows. The *x*-axis and *y*-axis limits (plot area) are normalized to fit the specified resolution. In other words, the pixel at position (0,0) is uses the minimum value for both dimensions and, similarly, the pixel at position (width, height) uses the maximum value for both dimensions. Since the procedure is based on a plot, the axis' scale can be adjusted (between linear or log). A visualization of the Plot binarization is presented in Figure 5.4, which is split in three subfigures. Subfigure 5.4a shows the plot of a sequence of 2-D points representing the ECCDF of the node degree of a graph. In Subfigure 5.4b, the plane is divided and the points of the sequence activate cells on the regular grid. The final output vector is composed

¹<https://matplotlib.org/>

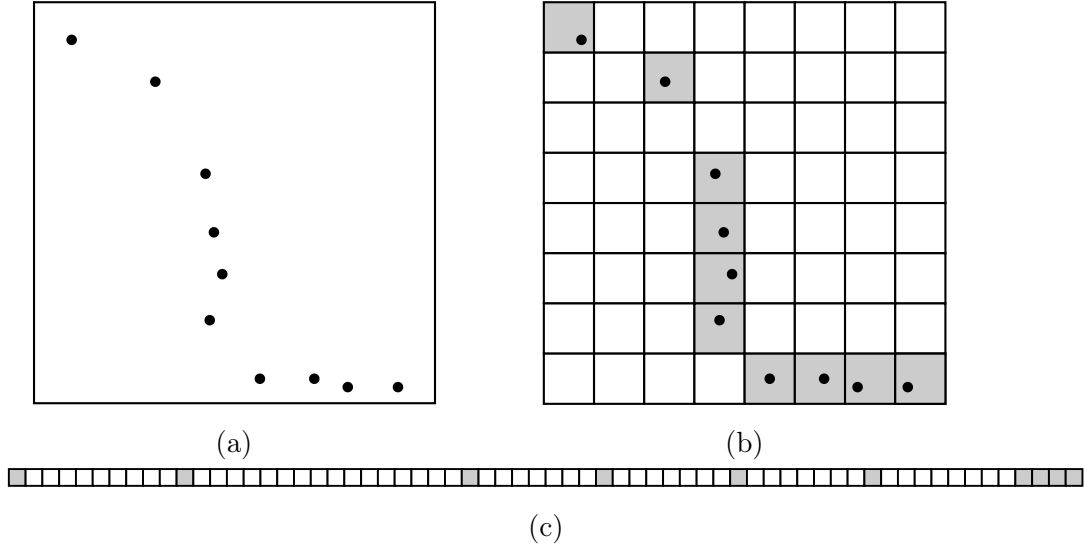


Figure 5.4: Example of Plot Binarization. In Figure 5.4a, points are plotted in a 2-D canvas. A grid is adjusted using this plot as in 5.4b, resulting in the final vector of Figure 5.4c.

of the grid in a single array format, with the activated cells marked with ones and non-activated cells with zeros.

Histogram Binarization

Another binarization process developed in this work is called Histogram binarization. The procedure is defined as follows. A histogram is calculated for each input (a set of node values), using a fixed number of bins in which values fall, defined by the hyperparameter n_bins . For each bin, a linear thermometer is built using $bits_per_bin$, where the y -axis value of the histogram bin is normalized using the number of samples as maximum value, and further encoded. The reader can think of it as a combination of multiple linear thermometers, where both minimum and maximum values are shared and known. Figure 5.5 shows the Histogram binarization process. In the Subfigure 5.5a a histogram with five equal-width bins is drawn. This histogram is transformed into a discrete version using five linear thermometers of five bits each, as depicted in 5.5b. Subfigure 5.5c illustrates a contiguous array that is used as input for the network.

5.2.2 Experimental Setup & Results

The architecture described in the previous section was evaluated as a viable graph classification framework. As in Section 5.1.2, the following sections present the experimental setup, the datasets, and the hyperparameter settings. Also, results are presented and discussed.

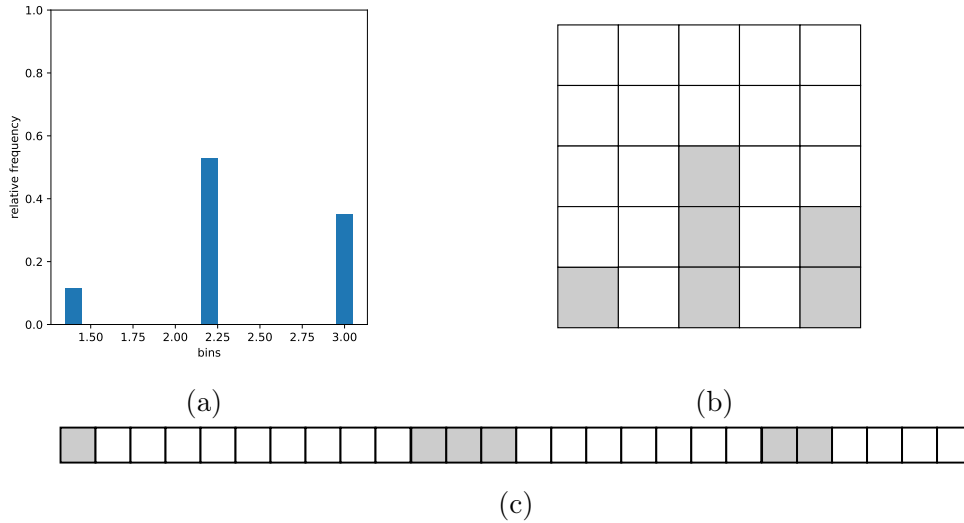


Figure 5.5: Example of Histogram binarization. In Figure 5.5a, a plotted histogram with five equal-width bins (relative to one graph). This histogram is binarized using the same 5 bins with 10 bits each in Figure 5.5b, resulting in the final vector of Figure 5.5c.

dataset	type	# Graphs	#Classes	Avg. Nodes	Avg. Edges
RGG	Synthetic	20480	5	139.68	813.43

Table 5.4: Dataset Statistics

Dataset

As a sanity check, the analysis of this framework was first conducted in a synthetic dataset named RGG, inspired by [94], built specifically for this work. This dataset was created using unattributed and undirected graphs generated by five random graph generators available in the NetworkX [136] Python library: Erdős-Renyi (GNP), Random power law tree, Connected Watts-Strogatz small-world, Holme-Kim and Barabási-Albert. Each generator serves as the class label for the dataset, as described in Table 5.4. Following this own dataset, other five datasets (COLLAB, ENZYMES, IMDB-M, NCI1) were drawn from the TUDatasets collection, using their cleaned versions as distributed in the PyTorch Geometric library [137].

Hyperparameter Evaluation

The performance of every combination of the C , PB and F choices listed on Table 5.5 was assessed, resulting in a total of 48 models (including both WiSARD-based classifiers and RandomForest baselines). The onion decomposition was used on undirected versions of original graphs, in the form of the Onion Spectrum. This eliminates the need of a ECCDF, since the spectrum is already a form of distribution. All measures

were calculated using the NetworkX [136] python library with default parameters as of version 3.3. Additionally, the performance of a complete independent baseline was evaluated, using the WL-Graph Kernel [103], as implemented in [138], and an SVM classifier (WL-SVM) using the scikit-learn library [85]. Each possible combination was tested as an hypothesis in a ten-fold cross validation procedure on a Xeon E5-2630 v3 CPU with 32GB RAM and 8 cores.

Feature Extraction (F)	Binarization(PB)	Classifiers (C)
node degree (IN/OUT)	Plot (PB)	Dictionary WiSARD (WIS)
page rank centrality (PR)	Histogram (HG)	ClusWiSARD (CWIS)
onion decomposition (OD)	KernelCanvas (KC)	RandomForest (RF)
	KDTree (KDT)	

Table 5.5: All hypothesis sets for feature extraction (F), pre-processing and binarization (PB) and classifier (C) steps.

With respect to hyperparameter search, the methodology was the same as in Section 5.1.2, a nested cross-validation scheme where training and test data are separated in 10 folds, then 10% of the training fold is used as validation. Each binarization alternative has also its set of possible parameters. There is also a size constraint on binary vectors to 2048 bits, except for the RGG dataset test where only 1024 bits were used. Table A.3 in Appendix A.2 gives a full description of the hyperparameter set for this experiment.

Brief Results & Discussions

As an initial analysis, results are limited to the main F_1 -score performance results, presented for the five TUDatasets and the RGG dataset in Figure 5.6. A first conclusion is related to the F_1 -score of 0.99 on the RGG dataset, which assess the feasibility of the proposed model. It was capable of successfully completing the task, reaching a comparable performance to the SVM baseline approach. This is an expected result, due to the difficulty of the task. Authors in [139, 140] have already demonstrated that random generated graphs are supposed to be easily distinguished. The model performance on some datasets, such as MUTAG, surpasses the baseline model, reaching a top 0.83 F_1 -score, but, from a statistical point-of-view, no combination can be declared a clear winner.

Concerning the proposed measures, both page rank and onion decomposition ranked better in some scenarios, once again without a global prevalence. As for binarization strategies, there was a slight dominance of the KDTree and Plot approaches, yet other strategies were also successful in some experiments. In general, WiSARD classifiers had a comparable performance to RandomForest alternatives.

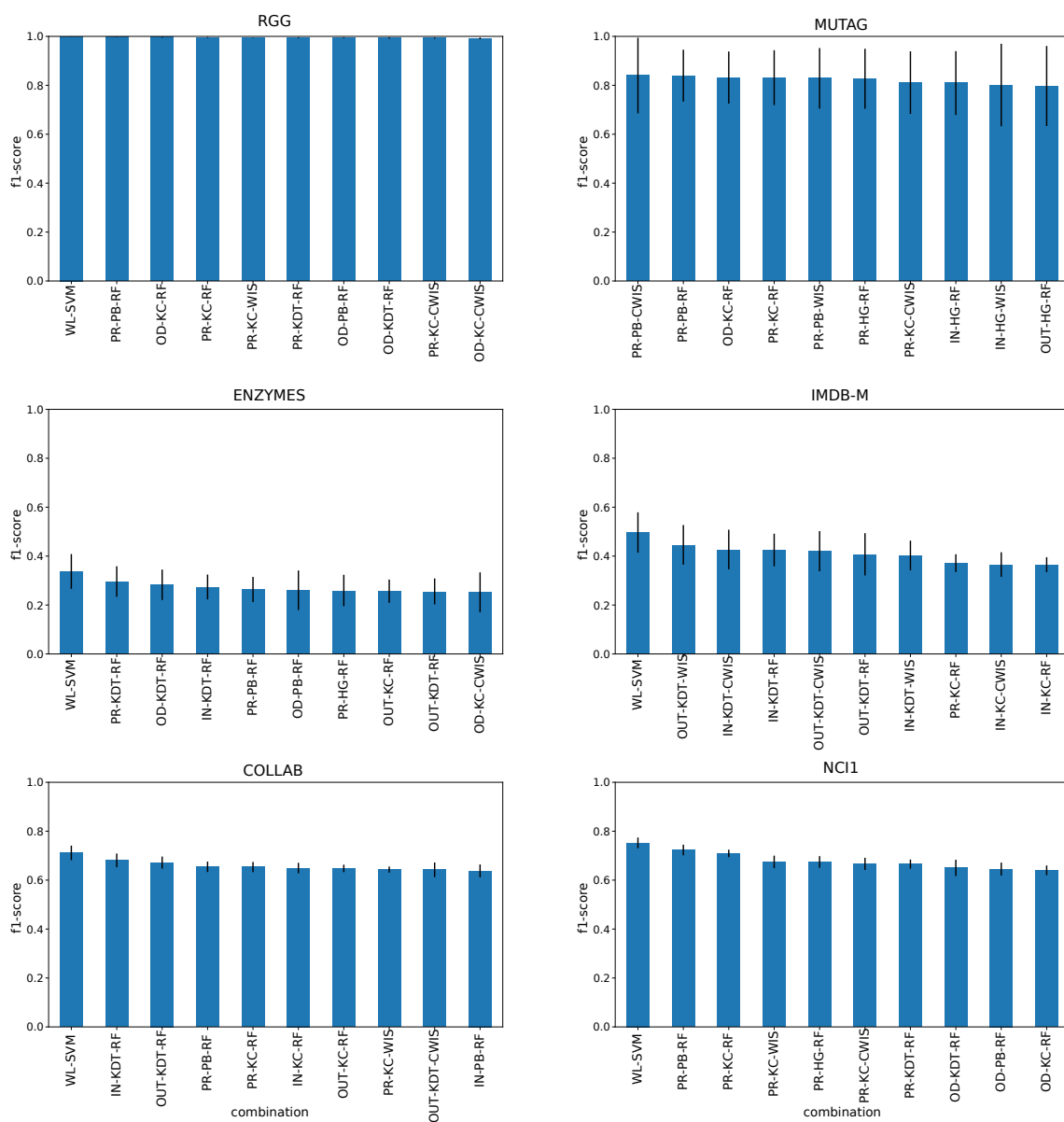


Figure 5.6: Top-ten F1-score results - Initial experiment: five datasets from TU-DATASETS and RGG, a new synthetic dataset.

5.3 Conclusions

This section has proposed two vanilla architectures for weightless graph classification, one based on unsupervised graph embeddings and another based on the distribution of node measures. The former strategy makes use of graph embeddings methods and the Distributive Thermometer as a default platform. The latter strategy relies on different whole-distribution binarization mechanisms, and two different approaches for this kind of binarization were proposed and formalized.

Although simple, both architectures have shown its viability for the graph classification problem. Moreover, the evaluations done in this chapter represent a necessary step towards future research and act as a baseline for weightless graph classification. Nonetheless, both architectures show growth opportunities and need to be further stressed, especially with respect to the incorporation of node and edge attributes.

Chapter 6

KernelCanvas++ for Graph Classification

The architectures for weightless graph classification proposed in the previous chapter were relatively successful in the task, but lack full usage of graph attributes and structure. This chapter introduces a new framework for graph classification that addresses this issue and goes beyond, investigating further improvements. This new framework is based on the development of extensions of the KernelCanvas method. Nonetheless, these extensions follow principles of both vector quantization and distributive approaches previously discussed. These proposals are evaluated in different scenarios, establishing a very competitive framework for graph classification using WNN. The use of this framework in conjunction with graph embeddings is also evaluated.

6.1 Extending the Node Primer

First, another architecture is proposed as an extension to the one presented in the previous sections. It is very similar to the three-phased framework, but F and PB steps differ. In order to further expand on the works of Section 5.2, the scope of possible pre-processing/aggregation strategies is reduced down to two: a KernelCanvas-based attribute aggregation and a histogram-based attribute aggregation. The KernelCanvas approach is used for every continuous attribute, and the histogram is used for categorical values. This proposed architecture is evaluated against different baselines and classifiers.

6.1.1 An Updated Kernel Canvas and Histogram Procedure

The KernelCanvas attribute aggregation was chosen due to a better trade-off between parameter search space and performance in previous experiments (see Sec-

tion 5.2.2). In Section 5.2.2, the empirical cumulative distribution function was plotted for every attribute, except for the onion decomposition, where the onion spectrum was plotted. Since every continuous attribute (being edge or node attribute) has its distribution over the nodes (or edges), a joint distribution of the attributes is plotted. Moreover, the KernelCanvas is adapted using the principles that guide the distributive thermometer: a better (uniform) distribution of bit space. The idea is to capture these joint-distribution using the KernelCanvas, maximizing attribute usage.

Instead of kernels $\in [-1, 1]$, kernels $\in [0, 1]$ are used. This is mainly to drop the current use of *tanh* after a Gaussian normalization. Hence, the value normalization (between 0 and 1) is done applying the empirical cumulative density function, as supported by the Probability Integral Transformation [141]. This can be interpreted as marginal distributive thermometers.

As an additional step, instead of the random kernel generation, the use of the centroids of a K -means run on the multi-dimensional normalized training data is proposed. The reader may call this conjunction of approaches as the KernelCanvas++ (KCcpp, KC++), and its overview is presented in Algorithm 3.

The goal of the algorithm is to transform a sequence of points $\in \mathbb{R}^d$ into a binary vector $\in \{0, 1\}^B$. It takes six arguments as input: *sequence* – a vector of points $\in \mathbb{R}^d$, *trainingData* – a matrix of all training points $\in \mathbb{R}^d$, \mathcal{K} – the set of kernel points $\in [0, 1]^d$, β – the number of bits for each kernel, B – the size of the binary output vector. Two other functions are expected to be defined: *ecdf* and *findknn*. The *ecdf*(x, X) function receives a value x and an array of points X , and returns the value of the empirical cumulative density function of X evaluated at x . The *findknn* function receives three arguments: p – a point $\in [0, 1]^d$, \mathcal{K} – a set of kernels/points $\in [0, 1]^d$, and κ – an integer, and outputs the κ -nearest neighbors of p within \mathcal{K} .

Compared to the original KernelCanvas algorithm (see Algorithm 2), the proposed KCcpp algorithm (detailed in Algorithm 3) only differs from lines 5 to 7, where each dimension of the point is converted to its dimensional ECDF value. After that, the activation is the same, where the κ -nearest kernels $\in \mathcal{K}$ to the converted point are searched, representing the activated kernels, and, from lines 9 to 14, the corresponding index for each activated kernel receives β consecutive bits are marked with one.

Categorical Values The histogram-based approach for categorical values can be defined as a diphasic algorithm. First, for each categorical attribute a in a graph \mathcal{G}_k , all P_a possible values are mapped into P_a bins, and thus, a histogram is created. This histogram is then further binarized using a classic linear thermometer, where the maximum value is the node or edge count (depending on the attribute type),

Algorithm 3 KCpp Algorithm - Transforms a sequence of points $\in \mathbb{R}^d$ into a binary vector $\in \{0, 1\}^B$

Input:

sequence ▷ Arbitrary size sequence of points $\in \mathbb{R}^d$
trainingData ▷ A matrix of all training points
 \mathcal{K} ▷ The kernel set
 β ▷ The number of bits used for each kernel
 κ ▷ The number of kernels to activate
 B ▷ The size of the final output vector

Output:

input_vector ▷ The final binarized input vector

- 1: **for** *bit_idx* $\leftarrow 1$ to B **do**
- 2: *input_vector*[*bit_idx*] $\leftarrow 0$
- 3: **end for**
- 4: **for all** *point* \in *sequence* **do**
- 5: **for** *dim* $\leftarrow 1$ to d **do** ▷ Transform point to its ECDF value using training data w.r.t. dimension
- 6: *newPoint*[*dim*] \leftarrow *ecdf*(*point*[*dim*], *trainingData*[*dim*])
- 7: **end for**
- 8: *activatedKernels* \leftarrow *findknn*(*newPoint*, \mathcal{K} , κ) ▷ Find newPoint's κ nearest neighbors $\in \mathcal{K}$
- 9: **for all** *kernel* \in *activatedKernels* **do** ▷ Activate β bits per kernel
- 10: **for** *bit* $\leftarrow 1$ to β **do**
- 11: *bit_idx* \leftarrow (*kernel* - 1) * β + *bit*
- 12: *input_vector*[*bit_idx*] $\leftarrow 1$
- 13: **end for**
- 14: **end for**
- 15: **end for**

the minimum is 0, and the number of bits per bin is set as a parameter.

As an alternative, a distributive histogram is proposed. In an analogous manner, for each categorical attribute a in a graph \mathcal{G}_k , the set $\mathcal{P}_a = \{p_{a,1}, \dots, p_{a,|\mathcal{P}_a|}\}$ of all possible categorical values of attribute a is mapped into a vector $T_{a,k} = [t_{k,1}, \dots, t_{k,|\mathcal{P}_a|}]$ of size $|\mathcal{P}_a|$, such that $t_{k,i}$ is the percentage of nodes where $a = p_{a,i}$ in \mathcal{G}_k . Second, this percentage $t_{k,i}$ is binarized using a distributive thermometer where quantiles are obtained from respective t_i values across all graphs in the training set.

6.1.2 The Final Representation

The final representation of a graph stacks structural components (s) with original non-structural graph attributes (ns), following a simple proportion:

$$B = \alpha B_s \cup (1 - \alpha) B_{ns}, \quad (6.1)$$

where B_s and B_{ns} are binary vectors of all structural and all non-structural attributes stacked, respectively. These vectors are recursively defined as the stacking of binarization B_i of each attribute i , with β_i defining the amount of information of the attribute i in the final representation. The following equation summarizes the definition of B_s :

$$B_s = \bigcup_i B_{i,s}, \quad (6.2)$$

with B_{ns} being similarly defined.

In terms of bit amount designations, all z_{ns} non-structural continuous attributes are treated as a single attribute (leading to only one z_{ns} -dimensional KernelCanvas). The same occurs for all structural continuous attributes. Figure 6.1 shows an example of a final bit vector, where non-structural and structural attributes split the vector in two. The non-structural portion of the vector is subdivided in a KernelCanvas block for continuous attributes and histograms for the remaining categorical attributes. The amount of bits designated for each block is a hyperparameter of the model and searched during model training. In addition, Figure 6.2 exemplifies the process for two different graphs using only two structural continuous attributes and painting solely the closest kernel (instead of κ kernels). Nodes are presented as an arbitrary sequence of points, painting their closest kernel’s area. Existing node and edge attributes are encoded in similar fashion and stacked to compose the final input vector. Moreover, equation 6.1 is the summary of a model that can be fed with all node and edge attributes originally held by the graphs, processing any node or edge attribute.

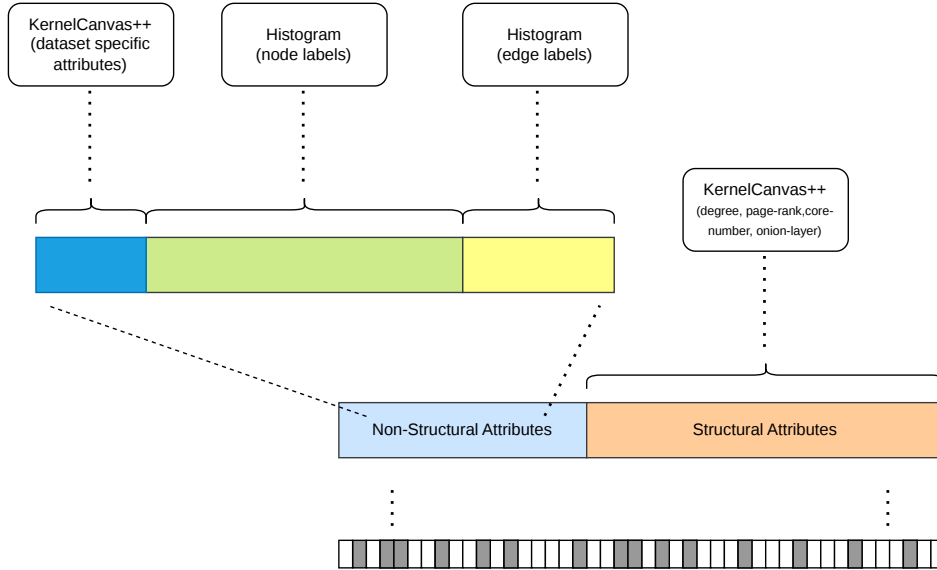


Figure 6.1: An example of a stacked bit vector. Non-structural and structural attributes split the vector in two. The non-structural portion of the vector is subdivided in a KernelCanvas for continuous attributes and histograms for the remaining categorical attributes. The amount of bits designated for each block is a hyperparameter of the model and searched during model training.

6.1.3 Experimental Setup & Results

Datasets & Evaluation

The datasets used to evaluate this architecture were the ones in Table 5.2. Every non-structural attribute in the original dataset is used in the final input vector. Concerning structural information, the node degree, the page-rank centrality, core-number, and onion-layer are used.

The proposed representation was evaluated considering one baseline and four main possibilities. A baseline (BASE) uses no ECDF normalization with random kernels and the vanilla histogram binarization. A first variation (PROP-1) uses the ECDF normalization with the vanilla histogram binarization. Secondly, PROP-2 uses ECDF normalization with the histogram binarization proposed in this section. Moreover, PROP-3 uses ECDF with K -means kernels. PROP-4 combines both PROP-2 and PROP-3.

In this experiment, the Dictionary WiSARD (WIS) and some of its evolutions, namely ClusWiSARD (CWIS) and BloomWiSARD (BWIS) were compared with a RandomForest baseline. A committee/ensemble of WiSARD classifiers was also evaluated, referred as RandomWiSARD (RWIS). The WNN results were also compared to some staple GNN models - GIN [16] and GCN [15], and some other recent GNN models - WLHN [142], a recent model leveraging hyperbolic neural networks

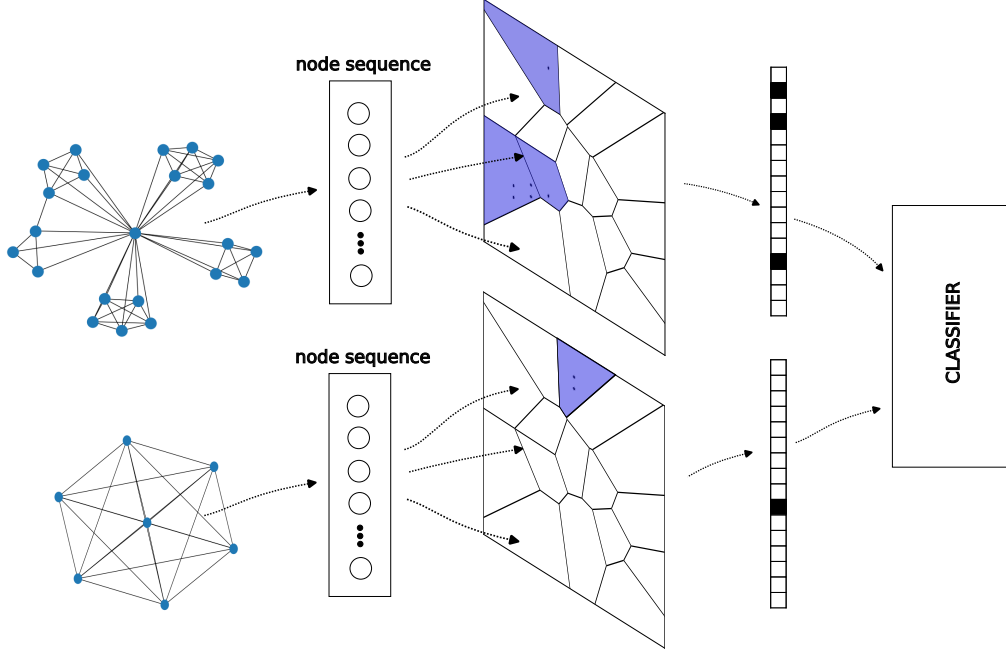


Figure 6.2: The KernelCanvas procedure applied to two different graphs using only two structural continuous attributes. Nodes are presented as an arbitrary sequence of points, painting their closest kernel’s area. Existing node and edge attributes are encoded in similar fashion and stacked to compose the final input vector.

and PMLP [143], a graph neural network that is trained as a multilayer perceptron but adopts a message-passing architecture during inference. The node inputs were the same as the WNN, except for edge-level attributes, since the evaluated GNNs do not use them. A simple mean readout is adopted as the readout layer of these node-based message passing networks.

Although F and PB steps were fixed, several hyperparameters still had to be stressed. One vital hyperparameter is the size of the bit vector representation B . Another important hyperparameter is α , the rate of structural information versus non-structural information, and β , the inner attribute proportion, which is limited to a minimum amount of information - min_info - another hyperparameter. The B and α parameters serve as proxies to define other several parameters. In this experiment, once α , β_s , and β_{ns} are defined, every β_i was also calculated. This was done following a greedy procedure that an attribute i is first uniformly sampled, then its β_i is uniformly sampled from the remaining bits of β_s or β_{ns} , depending on the type of the attribute. For the KernelCanvas, the *activationRate* was kept fixed at 0.07 and the evaluated parameters were *n_kernels* and *bits_per_kernel*. These were defined in the experiment for each attribute i as a random factorization of β_i . For each histogram, a *thermometer_size* was given. This parameter was fixed as β_i divided by the number of distinct possible values of attribute i .

All classifiers have also their hyperparameters such as *addressSize*, for WiSARD, *addressSize*, *threshold*, *discriminatorsLimit*, and *minScore* for the ClusWiSARD, *addressSize*, *unitHashes*, *unitEntries* for the BloomWiSARD, and *n_estimators*, *criterion*, and *max_features* for the RandomForest, and *addressSize*, *max_samples*, *max_features*, *n_estimators* for RandomWiSARD. Appendix A.3 gives a full description of the hyperparameter set used in this experiment.

All experiments were executed using a ten-fold nested cross-validation, with the same hyperparameter selection methodology as in [135]. A random or grid search was used to search for best validation parameters, fixing in a maximum of $P = 1000$ possibilities within a day of experiments for the WNN and RF models, and $P = 100$ for the GNN models. This time constraint was enforced due computing resources availability but to ensure that models are able to deliver good results within a small training window. Also, the search was parallelized as much as possible, using an Intel Xeon E5-2670v3 with 48 cores and 64GB of RAM. All work was developed building necessary and still unpublished extensions to the wisardpkg [30] library. The WLHN model was evaluated using authors’ original code¹ and the other GNN models were implemented using the PyTorch Geometric library [137].

Overall Results

This section starts by introducing the main result on the TUDatasets with no distinction between the variations (BASE, PROP-1, PROP-2, PROP-3, and PROP-4). Table 6.1 shows the overall best results alongside results for SVM, collected from [105] and the evaluated GNN models. The GNN data for the COLLAB dataset were collected for completion from [142] as the experiments could not finish due to shortage of resources. Both works of [105] and [142] have an experimental setup comparable to ours, based on [135]. Results show that the proposed shallow architecture leads to very competitive accuracy in comparison to deep models. The WiSARD-based classifiers achieve best overall results in the IMDB-M, ENZYMES, and PROTEINS datasets. Nonetheless, the RandomForest approach tops other two dataset charts, indicating that shallow architectures may beat deep architectures in almost every dataset. In an overall analysis, RandomForest and Dictionary WiSARD present the best results. Some works [140, 144] debate if the structure is relevant to some of the datasets being evaluated. Although being irrelevant to some, the representation presented here is powerful in both cases. Other datasets and classifiers could be evaluated and there is still room for hyperparameter optimization, since no optimization scheme was adopted besides the random/grid search.

¹<https://github.com/MichailChatzianastasis/WLHN>

	COLLAB	ENZYMES	IMDB-M	MUTAG	NCI1	PROTEINS
SVM	84.5 ± 2.0	67.7 ± 6.5	51.7 ± 4.1	88.3 ± 7.1	86.3 ± 1.6	71.1 ± 4.4
GCN [15]	71.3 ± 2.0	79.6 ± 2.4	65.2 ± 3.0	87.9 ± 6.1	67.6 ± 2.3	74.3 ± 3.5
GIN [16]	75.6 ± 2.3	82.9 ± 1.4	62.6 ± 4.1	84.0 ± 9.4	70.4 ± 2.2	74.2 ± 2.0
WLHN [142]	76.2 ± 2.3	86.1 ± 3.5	65.8 ± 2.6	86.6 ± 10.5	73.2 ± 2.2	74.8 ± 3.0
PMLP [143]	56.4 ± 4.5	83.1 ± 3.6	60.5 ± 2.9	83.5 ± 7.4	56.0 ± 2.9	61.7 ± 4.4
RF	85.7 ± 1.1	90.3 ± 2.9	66.1 ± 4.3	91.0 ± 7.6	74.2 ± 1.5	76.3 ± 4.3
WIS	85.3 ± 0.9	90.2 ± 2.4	66.4 ± 2.9	89.3 ± 7.0	73.5 ± 1.7	76.2 ± 4.8
CWIS	81.7 ± 1.7	89.4 ± 2.9	65.5 ± 3.6	89.8 ± 9.9	73.1 ± 1.7	74.5 ± 3.3
RWIS	80.8 ± 1.9	90.0 ± 2.1	66.2 ± 2.5	88.7 ± 11.0	72.5 ± 1.9	76.9 ± 4.2
BWIS	78.5 ± 1.4	90.5 ± 2.4	66.4 ± 3.2	87.8 ± 6.6	73.0 ± 1.7	76.2 ± 2.8

Table 6.1: Performance summary on TUDatasets (mean accuracy over 10-fold evaluations). Best results are highlighted with a black background, second best with a gray background. Accuracy of SVM and GNN models collected from [105] and [142].

Representation Size Analysis

This section presents an analysis of the performance of the WNN and RF models for different input sizes in every dataset, ranging from 2048 to 16384 bits. Overall, there is no silver bullet when deciding the number of bits of the input vector. Every dataset appears to be a singular problem, as Tables 6.2 to 6.7 show. Apart from the COLLAB dataset, where Dictionary WiSARD and RandomForest appear to be clear winners, performance for all classifiers seems to be similar, within a standard deviation margin. Considering most datasets, there is evidence that greater representations may work better, since best results are closer to the upper limit of the grid search considering the input size. However, increasing input size is also a double-edged sword, particularly when dealing with larger datasets, such as the COLLAB. Larger representation leads to both higher computing time and resource consumption, which may incur execution errors of timeout and/or lack of memory (out of resources - OOR).

Feature Importance Analysis

One key aspect of proposed pipeline is the search for the best representation as part of the hyperparameter search. In Figure 6.3 is presented a summary of the best trade-offs between structural and non-structural features that were found for each dataset in terms of a percentage of the size of the input vector with respect to the WNN and RF models. For the ENZYMES dataset, it can be seen a great majority of non-structural attributes prevailing. The scenario is similar but not so extreme for the MUTAG and the NCI1 datasets, where there is some predominance of non-structural attributes. It is relevant to notice that in the latter there is only one non-structural attribute (node labels). Yet, they seem to play a key role when

Input size	RF	WIS	CWIS	BWIS	RWIS
2048	84.9 ± 1.1	84.8 ± 1.4	80.5 ± 1.4	75.2 ± 7.3	80.1 ± 1.6
4096	85.7 ± 1.1	84.7 ± 1.2	80.9 ± 2.2	73.9 ± 10.1	80.8 ± 1.9
6144	85.1 ± 1.4	85.1 ± 1.4	80.6 ± 1.9	78.5 ± 1.4	80.6 ± 1.9
8192	85.4 ± 1.0	85.3 ± 0.9	81.1 ± 1.4	OOR	OOR
10240	85.5 ± 0.9	85.3 ± 1.1	81.3 ± 2.3	OOR	OOR
12288	85.6 ± 1.1	85.0 ± 0.9	81.3 ± 1.9	OOR	OOR
14336	85.3 ± 0.8	85.1 ± 1.5	81.2 ± 2.0	OOR	OOR
16384	85.5 ± 1.3	85.3 ± 1.3	81.7 ± 1.7	OOR	OOR

Table 6.2: Performance on the COLLAB dataset using different vector input sizes (mean accuracy over 10-fold evaluations). Best results are highlighted with a black background, second best with a gray background. OOR means that results did not complete after 24 hours of experiment.

Input size	RF	WIS	CWIS	BWIS	RWIS
2048	89.6 ± 3.0	88.6 ± 2.1	87.7 ± 2.1	87.5 ± 2.3	88.8 ± 2.8
4096	89.7 ± 2.3	89.2 ± 2.7	88.1 ± 3.1	89.8 ± 3.0	87.1 ± 3.6
6144	90.3 ± 2.9	89.7 ± 2.3	89.1 ± 3.2	89.2 ± 2.7	88.9 ± 1.9
8192	90.2 ± 2.4	89.2 ± 2.6	89.1 ± 2.6	89.1 ± 2.1	88.8 ± 2.3
10240	89.7 ± 1.5	90.1 ± 2.8	88.6 ± 2.8	89.3 ± 3.3	89.8 ± 2.4
12288	89.7 ± 2.7	89.0 ± 1.8	89.4 ± 2.9	89.6 ± 2.5	89.7 ± 1.9
14336	89.2 ± 2.5	89.8 ± 2.2	89.3 ± 2.8	89.4 ± 2.4	88.6 ± 3.2
16384	89.9 ± 2.3	90.2 ± 2.4	88.9 ± 2.7	90.5 ± 2.4	90.0 ± 2.1

Table 6.3: Performance on the ENZYMES dataset using different vector input sizes (mean accuracy over 10-fold evaluations). Best results are highlighted with a black background, second best with a gray background.

Input size	RF	WIS	CWIS	BWIS	RWIS
2048	65.2 ± 3.5	65.0 ± 2.6	62.8 ± 2.6	65.3 ± 2.4	65.2 ± 2.9
4096	64.8 ± 3.4	65.1 ± 2.6	64.2 ± 3.4	65.4 ± 3.2	64.1 ± 3.6
6144	65.8 ± 3.0	65.6 ± 4.1	64.4 ± 3.7	65.3 ± 2.5	65.1 ± 2.8
8192	65.8 ± 3.6	65.9 ± 3.8	64.1 ± 4.0	65.8 ± 2.2	65.3 ± 1.9
10240	65.6 ± 3.6	65.5 ± 3.9	64.6 ± 2.8	65.8 ± 3.2	65.8 ± 3.3
12288	66.0 ± 3.7	66.4 ± 2.9	65.2 ± 4.3	66.4 ± 3.2	65.6 ± 3.5
14336	65.9 ± 3.9	65.4 ± 3.1	65.5 ± 3.6	65.6 ± 2.7	64.2 ± 4.0
16384	66.1 ± 4.3	65.8 ± 3.9	65.0 ± 3.4	65.6 ± 3.0	66.2 ± 2.5

Table 6.4: Performance on the IMDB-M dataset using different vector input sizes (mean accuracy over 10-fold evaluations). Best results are highlighted with a black background, second best with a gray background.

Input size	RF	WIS	CWIS	BWIS	RWIS
2048	86.8 ± 7.2	87.7 ± 7.4	85.5 ± 9.8	85.6 ± 9.0	85.1 ± 8.3
4096	87.3 ± 10.7	86.1 ± 7.0	87.7 ± 8.2	85.0 ± 6.7	85.1 ± 8.5
6144	87.9 ± 11.1	86.1 ± 11.1	86.1 ± 8.3	85.1 ± 6.5	84.1 ± 9.3
8192	89.4 ± 7.4	88.7 ± 8.0	88.2 ± 9.3	85.6 ± 7.9	86.2 ± 7.5
10240	91.0 ± 7.6	87.7 ± 6.8	88.2 ± 8.1	86.2 ± 7.1	82.0 ± 9.9
12288	89.4 ± 8.2	86.6 ± 8.2	87.1 ± 8.4	87.8 ± 6.6	81.9 ± 7.9
14336	87.3 ± 8.7	89.3 ± 7.0	86.6 ± 7.0	87.2 ± 6.8	89.4 ± 5.6
16384	88.3 ± 6.4	88.7 ± 8.5	89.8 ± 9.9	86.1 ± 8.1	87.8 ± 5.0

Table 6.5: Performance on the MUTAG dataset using different vector input sizes (mean accuracy over 10-fold evaluations). Best results are highlighted with a black background, second best with a gray background.

Input size	RF	WIS	CWIS	BWIS	RWIS
2048	74.7 ± 1.9	70.3 ± 2.1	69.3 ± 2.2	69.2 ± 2.4	70.8 ± 2.1
4096	74.8 ± 1.4	72.0 ± 2.3	70.3 ± 1.7	69.9 ± 2.4	70.2 ± 1.4
6144	74.2 ± 2.7	73.0 ± 2.6	70.2 ± 2.9	71.8 ± 1.4	71.5 ± 2.0
8192	75.2 ± 2.5	72.9 ± 2.6	70.9 ± 2.0	71.9 ± 1.5	71.2 ± 2.6
10240	74.6 ± 3.1	72.5 ± 3.0	72.3 ± 2.3	71.9 ± 1.6	71.4 ± 2.1
12288	75.1 ± 2.4	73.9 ± 2.2	72.5 ± 1.2	72.3 ± 2.2	71.5 ± 2.2
14336	75.8 ± 2.3	73.6 ± 2.1	73.1 ± 1.7	72.1 ± 3.1	70.6 ± 1.3
16384	75.8 ± 2.9	72.1 ± 2.9	72.1 ± 2.0	73.0 ± 1.7	72.5 ± 1.9

Table 6.6: Performance on the NCI1 dataset using different vector input sizes (mean accuracy over 10-fold evaluations). Best results are highlighted with a black background, second best with a gray background.

Input size	RF	WIS	CWIS	BWIS	RWIS
2048	74.8 ± 3.5	73.1 ± 3.7	71.0 ± 2.3	73.2 ± 4.5	74.9 ± 4.9
4096	73.5 ± 5.4	73.8 ± 4.7	72.8 ± 3.5	73.6 ± 3.9	75.5 ± 3.5
6144	76.3 ± 4.3	73.9 ± 5.0	72.2 ± 4.4	73.8 ± 3.6	76.2 ± 5.0
8192	73.8 ± 4.9	74.1 ± 4.7	73.2 ± 3.9	74.8 ± 4.1	74.8 ± 4.9
10240	73.2 ± 4.6	75.6 ± 5.8	74.5 ± 3.3	74.0 ± 3.6	76.9 ± 4.2
12288	73.7 ± 4.7	73.7 ± 5.3	72.6 ± 4.1	74.9 ± 3.9	75.9 ± 3.6
14336	70.8 ± 5.3	71.4 ± 5.5	72.5 ± 4.2	73.5 ± 4.6	75.9 ± 4.6
16384	72.9 ± 5.9	76.2 ± 4.8	73.0 ± 3.7	76.2 ± 2.8	76.3 ± 5.1

Table 6.7: Performance on the PROTEINS dataset using different vector input sizes (mean accuracy over 10-fold evaluations). Best results are highlighted with a black background, second best with a gray background.

differentiating the classes. An opposite behavior happens in the PROTEINS dataset

where structural information appears to be more important than non-structural information. IMDB-M and COLLAB datasets leverage only structural attributes. As a concluding remark, there is no clear preset of attributes that works better overall. Every dataset is a unique problem of its own. The most important is to ensure that a final model can absorb these differences.

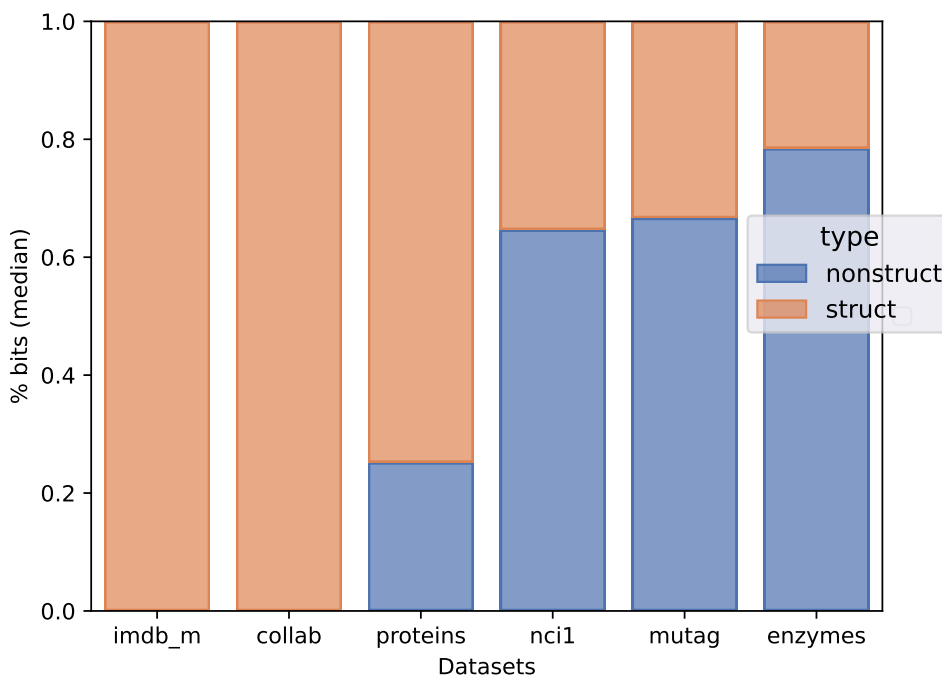


Figure 6.3: Trade-offs between structural and non-structural attributes in the best result for all datasets (median of all folds). Datasets are ordered by structural attributes' usage in final classifier

Resource Usage Analysis

In this section, the resource consumption is evaluated using different metrics. All energy and memory consumption data in Tables 6.8-6.10 were collected from PBS² trace reports for the whole process, including hyperparameter search, training, model selection, and testing, except for the final model size, where the size of the serialized classifier is reported. Results are shown for WNN, RandomForest and the selected GNN baselines. This evaluation was carried out fixing the number of hyperparameter combinations(P) to 100 within a day to establish a fair comparison. In Table 6.8, the results show that the Dictionary WiSARD is generally less energy demanding than the baseline RandomForest classifier, except for cases where it follows RF by a small margin. That fact does not hold though for its variations, where BloomWisard

²<https://altair.com/pbs-professional/>

can be up to twenty times more energy demanding (in IMDB-M dataset), being as energy demanding as some GNN models. Moreover, the GNN models were the worst concerning their energy demands. Table 6.9 presents average final model sizes considering only the best classifiers. Taking into consideration the recent advances of BloomWiSARD model, this is where it shines, providing the most compact WNN model. On the hand, the RWIS approach is the greatest, since it can be seen as an ensemble of models of relative big size. The evaluated GNN were not as deep as other deep network models, resulting in very compact final models, with few kilobytes in size, topping the charts. Concluding, Table 6.10 reveals the memory consumption of all models, with the Dictionary WiSARD being generally the less demanding classifier. On the other hand, GNN models are the most demanding, imposing roughly twice the memory requirements of a WNN model. It is important to note that these memory consumption values are partially due to high data parallelism leveraged for hyperparameter search, not being strictly this high if a single-threaded model was trained.

	COLLAB	ENZYMES	IMDB-M	MUTAG	NCI1	PROTEINS
RF	1342.3 ± 857.3	41.9 ± 15.1	45.3 ± 24.3	4.3 ± 1.7	224.1 ± 101.5	52.8 ± 14.5
WIS	1311.6 ± 823.3	42.0 ± 15.9	17.7 ± 7.5	3.3 ± 1.7	215.6 ± 103.0	53.1 ± 15.6
CWIS	1365.2 ± 858.6	46.1 ± 17.0	38.4 ± 19.4	4.2 ± 2.1	231.5 ± 107.3	59.0 ± 17.8
RWIS	1483.9 ± 996.8	107.5 ± 41.7	128.5 ± 70.2	14.0 ± 6.5	520.2 ± 236.3	130.5 ± 58.1
BWIS	1688.7 ± 1030.6	227.3 ± 97.5	207.1 ± 115.4	18.4 ± 9.7	574.9 ± 263.5	181.3 ± 75.7
GCN	OOR	168.8 ± 3.7	323.7 ± 9.8	36.4 ± 2.2	831.1 ± 44.7	344.5 ± 9.4
GIN	OOR	142.3 ± 4.3	257.3 ± 15.2	35.2 ± 2.2	800.0 ± 46.8	277.4 ± 2.4
WLHN	OOR	671.4 ± 17.7	749.6 ± 39.6	127.2 ± 5.8	3901.9 ± 217.2	1377.9 ± 26.6
PMLP	3005.6 ± 800.2	93.7 ± 5.8	179.9 ± 12.6	27.1 ± 0.9	542.1 ± 19.8	185.6 ± 7.0

Table 6.8: Average energy consumption (in Wh) on TUDatasets. Best results are highlighted with a black background, second best with a gray background.

	COLLAB	ENZYMES	IMDB-M	MUTAG	NCI1	PROTEINS
RF	20.2 ± 3.6	6.0 ± 1.1	2.8 ± 1.0	0.6 ± 0.1	21.6 ± 4.8	5.0 ± 1.2
WIS	19.9 ± 11.5	5.6 ± 5.3	1.6 ± 1.0	0.8 ± 1.8	3.7 ± 1.7	3.3 ± 2.3
CWIS	19.5 ± 11.6	5.4 ± 2.9	2.8 ± 1.6	1.1 ± 0.6	4.7 ± 2.5	4.2 ± 2.8
RWIS	220.9 ± 119.8	113.4 ± 70.0	39.7 ± 34.0	8.4 ± 6.6	78.3 ± 44.0	51.7 ± 42.2
BWIS	2.5 ± 11.1	0.7 ± 0.4	0.3 ± 0.2	0.2 ± 0.1	0.1 ± 0.1	0.3 ± 0.1
GCN	OOR	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
GIN	OOR	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
WLHN	OOR	0.2 ± 0.0	0.1 ± 0.0	0.1 ± 0.0	0.2 ± 0.0	0.1 ± 0.0
PMLP	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 6.9: Average final model size (in MB) on TUDatasets. Best results are highlighted with a black background, second best with a gray background. BWIS results for the COLLAB must be taken with care since only smaller models completed the experiment.

	COLLAB	ENZYMES	IMDB-M	MUTAG	NCI1	PROTEINS
RF	39.1 ± 3.9	11.8 ± 0.2	11.8 ± 0.6	10.7 ± 0.1	14.1 ± 0.6	12.0 ± 0.3
WIS	39.7 ± 4.1	11.5 ± 0.3	11.2 ± 0.4	10.6 ± 0.1	13.2 ± 0.8	11.6 ± 0.4
CWIS	40.4 ± 3.6	11.8 ± 0.4	11.4 ± 0.5	10.8 ± 0.2	13.2 ± 0.6	11.8 ± 0.4
RWIS	49.5 ± 6.3	31.5 ± 13.0	20.7 ± 5.5	14.7 ± 2.0	20.0 ± 3.9	20.9 ± 6.1
BWIS	41.7 ± 1.8	17.7 ± 5.4	13.4 ± 2.0	12.2 ± 0.9	13.9 ± 1.2	12.9 ± 1.0
GCN	OOO	24.4 ± 0.0	24.4 ± 0.0	23.9 ± 0.1	25.7 ± 0.1	24.7 ± 0.0
GIN	OOO	24.4 ± 0.0	24.4 ± 0.1	23.9 ± 0.1	25.7 ± 0.1	24.6 ± 0.0
WLHN	OOO	24.4 ± 0.0	24.2 ± 0.0	23.6 ± 0.0	25.7 ± 0.0	24.8 ± 0.0
PMLP	74.6 ± 0.0	24.3 ± 0.0	24.4 ± 0.1	23.9 ± 0.1	25.7 ± 0.1	24.6 ± 0.0

Table 6.10: Average virtual memory usage on TUDatasets (in GB). Best results are highlighted with a black background, second best with a gray background.

Following this resource usage analysis, the time spent on every key phase of the study is detailed. Table 6.11 shows the time spent on hyperparameter search, training the full model and the inference phase for all evaluated datasets. It is important to point out that the current implementations are not production-ready, and time comparisons can be misleading. The results in Table 6.11 indicate that the Dictionary WiSARD models spent less time searching for hyperparameters and training. Overall, the GNN are faster for inference, followed by the RandomForest baseline. These results are partially explained by a bottleneck caused by the preprocessing of the inputs through the pipeline, and the bleaching technique, which can still be improved with approaches like the one in [145].

	COLLAB			ENZYMES			IMDB-M		
	Search	Training	Inference	Search	Training	Inference	Search	Training	Inference
RF	2368.1 ± 1354.4	285.2 ± 202.8	20.8 ± 11.9	66.6 ± 25.2	9.8 ± 3.7	0.8 ± 0.3	73.5 ± 37.4	8.2 ± 5.2	0.2 ± 0.1
WIS	2296.3 ± 1312.9	279.4 ± 205.1	21.5 ± 12.5	66.6 ± 26.5	9.7 ± 3.9	0.9 ± 0.3	27.5 ± 11.6	3.3 ± 1.4	0.5 ± 0.3
CWIS	2413.0 ± 1364.2	276.2 ± 207.3	21.3 ± 12.7	73.6 ± 28.2	10.1 ± 4.1	1.0 ± 0.3	58.4 ± 27.3	4.7 ± 2.2	2.6 ± 1.7
RWIS	2704.7 ± 1320.8	331.1 ± 204.3	38.6 ± 15.9	150.7 ± 56.4	19.1 ± 8.8	7.7 ± 5.8	175.7 ± 89.8	13.0 ± 7.6	8.5 ± 5.7
BWIS	3324.2 ± 1569.8	394.2 ± 197.1	20.4 ± 12.4	360.7 ± 154.0	62.1 ± 37.4	1.2 ± 0.4	317.8 ± 171.4	50.9 ± 31.5	0.6 ± 0.3
GCN	OOO	OOO	OOO	202.2 ± 2.1	18.8 ± 1.1	0.2 ± 0.0	407.9 ± 5.3	37.9 ± 4.6	0.6 ± 0.1
GIN	OOO	OOO	OOO	173.8 ± 2.4	17.8 ± 2.2	0.2 ± 0.0	346.9 ± 7.5	36.4 ± 3.3	0.5 ± 0.1
WLHN	OOO	OOO	OOO	834.1 ± 7.1	95.2 ± 4.7	0.3 ± 0.0	1003.9 ± 27.7	100.5 ± 11.0	0.6 ± 0.0
PMLP	5286.1 ± 35.1	596.2 ± 19.0	54.2 ± 0.6	125.2 ± 0.9	12.0 ± 0.4	0.2 ± 0.0	247.8 ± 5.8	23.8 ± 2.8	0.5 ± 0.0
	MUTAG			NCI1			PROTEINS		
	Search	Training	Inference	Search	Training	Inference	Search	Training	Inference
RF	6.6 ± 2.6	0.8 ± 0.4	0.1 ± 0.0	382.0 ± 176.5	51.3 ± 25.9	3.6 ± 1.4	86.6 ± 25.2	11.6 ± 3.6	0.9 ± 0.3
WIS	4.9 ± 2.7	0.5 ± 0.4	0.1 ± 0.0	372.0 ± 175.1	48.7 ± 25.5	3.8 ± 1.4	86.3 ± 25.5	11.2 ± 3.6	1.0 ± 0.3
CWIS	6.2 ± 3.2	0.7 ± 0.4	0.1 ± 0.1	403.6 ± 182.9	49.8 ± 25.7	4.0 ± 1.4	96.0 ± 28.3	11.9 ± 4.0	1.2 ± 0.4
RWIS	22.1 ± 9.5	1.4 ± 1.0	0.4 ± 0.3	769.1 ± 341.2	104.2 ± 53.5	21.1 ± 12.7	189.5 ± 75.0	21.9 ± 9.8	6.3 ± 4.3
BWIS	29.0 ± 15.5	3.9 ± 2.8	0.1 ± 0.1	1034.8 ± 466.2	109.6 ± 58.0	4.1 ± 1.5	288.9 ± 117.9	47.5 ± 23.1	1.2 ± 0.3
GCN	47.0 ± 0.7	4.6 ± 0.5	0.0 ± 0.0	1075.0 ± 20.4	110.3 ± 12.1	1.0 ± 0.0	420.9 ± 12.1	31.1 ± 3.7	0.5 ± 0.0
GIN	44.8 ± 0.9	4.3 ± 0.7	0.1 ± 0.0	988.2 ± 24.2	104.4 ± 15.1	1.0 ± 0.1	336.9 ± 4.5	33.0 ± 1.6	0.5 ± 0.0
WLHN	168.6 ± 2.2	13.9 ± 2.7	0.0 ± 0.0	5201.5 ± 60.2	609.7 ± 33.2	1.3 ± 0.1	1788.7 ± 16.6	176.6 ± 8.0	0.6 ± 0.0
PMLP	35.6 ± 0.5	3.1 ± 0.3	0.0 ± 0.0	739.8 ± 10.8	68.3 ± 5.4	1.0 ± 0.1	235.0 ± 5.7	18.7 ± 2.5	0.5 ± 0.0

Table 6.11: Average time (in seconds per fold) spent on hyperparameter search, training and inference on TUDatasets. Best results are highlighted with a black background, second best with a gray background.

KernelCanvas++ Accuracy Analysis

Another aspect of this section are the proposed additions to the KernelCanvas fundamentals. Figures from 6.4 to 6.9 present the results in terms of accuracy according to the strategy implemented in KernelCanvas and the binarization steps, fixing the WiSARD base classifier as the classifier. Starting with the COLLAB dataset, Figure 6.4 shows only three alternatives (BASE, PROP-1, PROP-4). This happens because there is no histogram-based attribute in this dataset (the same applies to Figure 6.6). Regarding the results, the chart shows that PROP-1 is clearly the best alternative in terms of accuracy, followed by PROP-4 and then the baseline. The same difference from the baseline is seen in Figure 6.6, where PROP-1 and PROP-4 have similar performance. Regarding the ENZYMES dataset, Figure 6.5 exhibits all five KernelCanvas' hypotheses, with PROP-1 once again topping the charts. PROP-3 seems to be the worst in performance, with others performing indistinguishably in the middle. Figure 6.7 provides accuracies for the MUTAG dataset, where all hypotheses seem to achieve almost the same results. Results for the NCI1 dataset are presented in Figure 6.8. This figure illustrates PROP-1 and BASE performing similarly at the top, with the other three approaches being in a lower cut. Finally, 6.9 presents results for the PROTEINS dataset, where the only distinct result is the worse accuracy assessed by BASE. The global image brought by the results in this section is that PROP-1 is the to-go strategy, as the other proposals did not perform as expected.

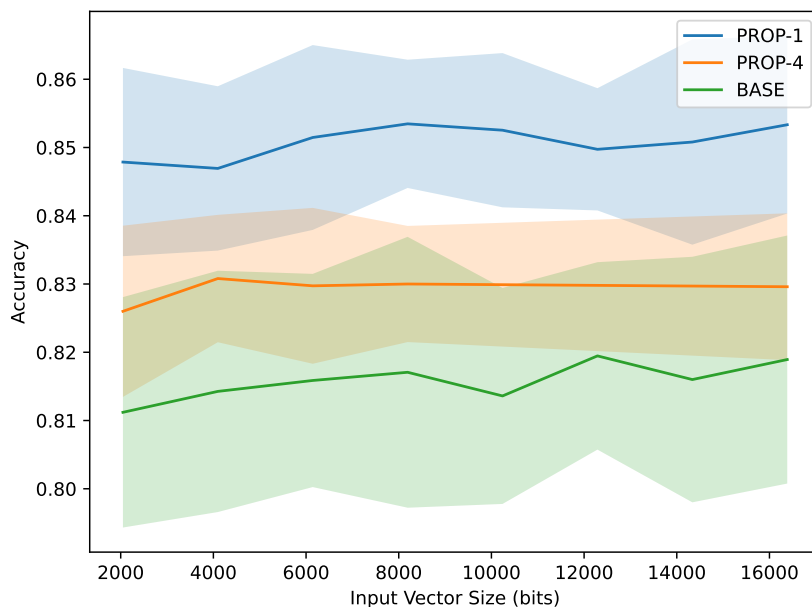


Figure 6.4: Accuracy of different KernelCanvas strategies on the COLLAB dataset

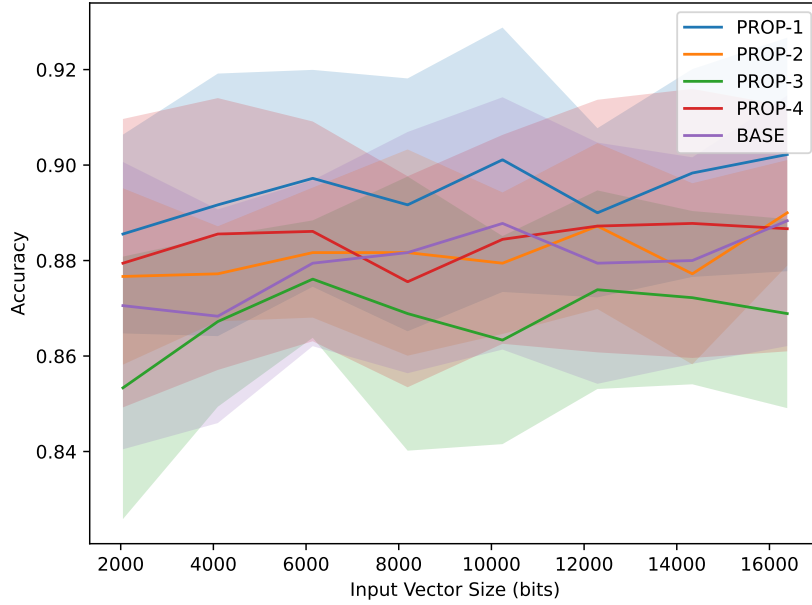


Figure 6.5: Accuracy of different KernelCanvas strategies on the ENZYMES dataset

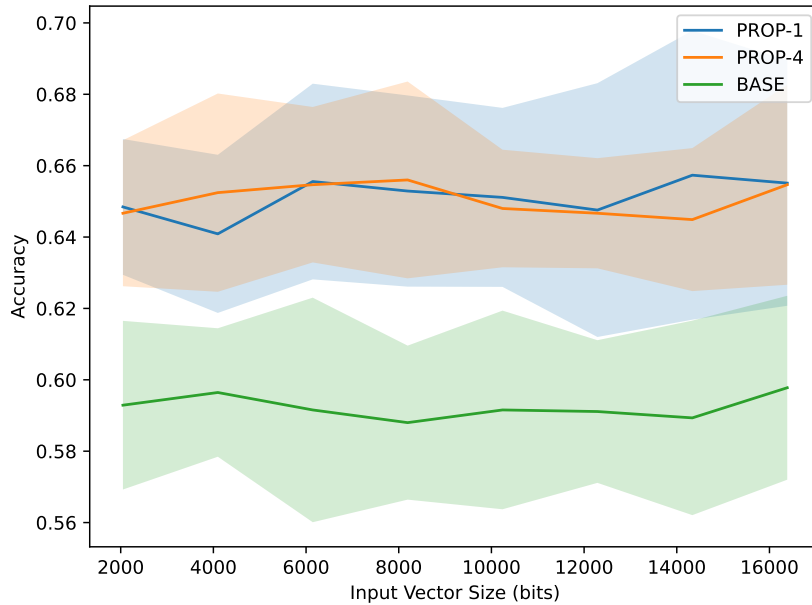


Figure 6.6: Accuracy of different KernelCanvas strategies on the IMDB-M dataset

6.2 KCVQ and DKC++

In this section, two other variations to the KernelCanvas++ are proposed, one exploring its input space and the other its activation step.

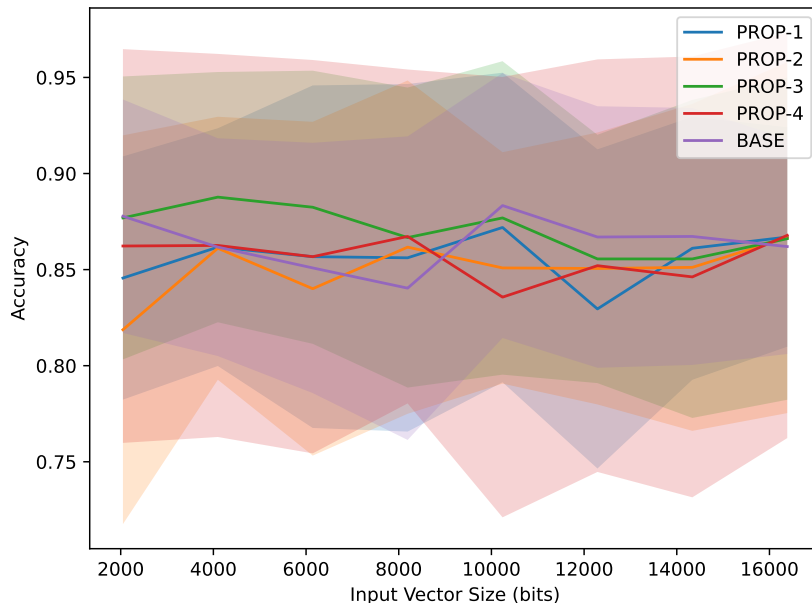


Figure 6.7: Accuracy of different KernelCanvas strategies on the MUTAG dataset

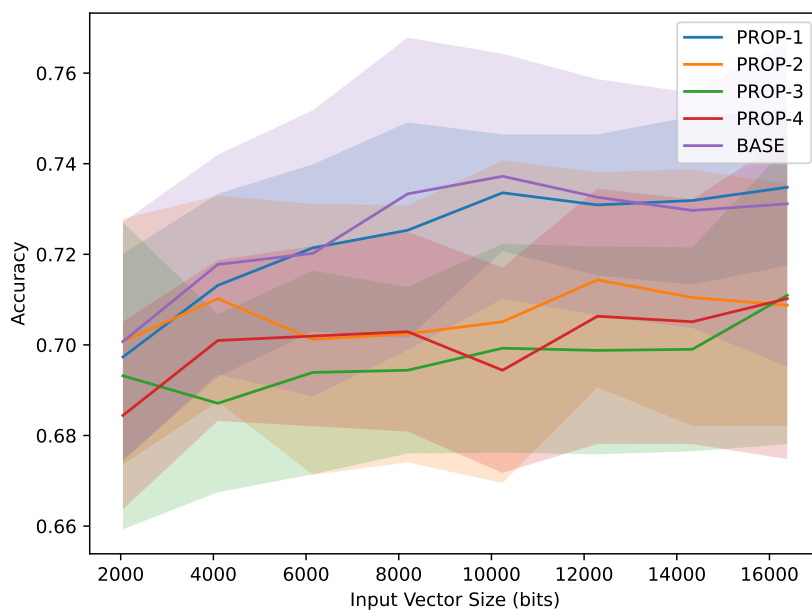


Figure 6.8: Accuracy of different KernelCanvas strategies on the NCI1 dataset

6.2.1 KernelCanvas-VQ

The KernelCanvas++ restricted the input space to $[0, 1]^d$ for a d -dimensional input where all kernels lie. One reason for this constraint is that it establishes a viable bounding box for kernels. This requirement is not strictly necessary if kernels are not generated to divide the space in a random fashion. For example, kernels that are generated via K -means algorithm are well defined in \mathbb{R}^d . Hence, this idea leads to a variation of the KernelCanvas that is close to the vector quantization algorithm,

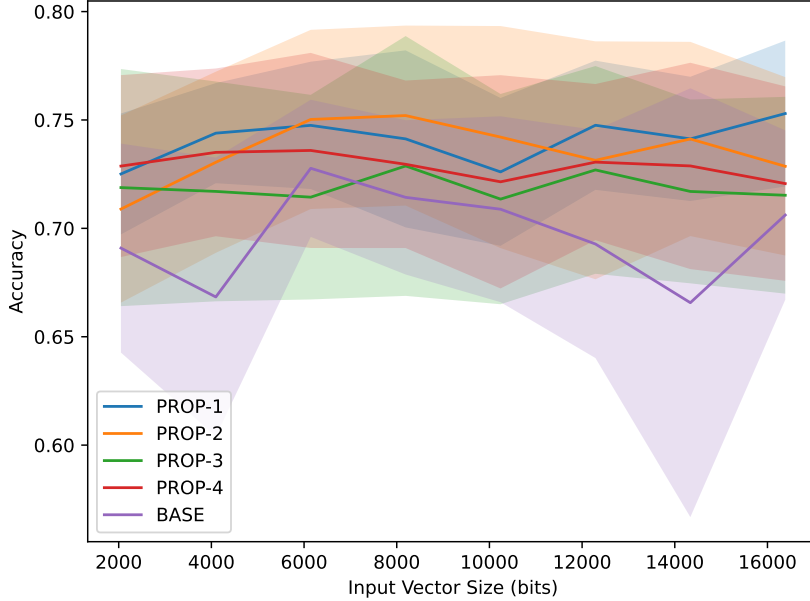


Figure 6.9: Accuracy of different KernelCanvas strategies on the PROTEINS dataset

coined as KernelCanvas-VQ (KCVQ). The only normalization applied is a process called whitening, a simple procedure that divides each dimension by its standard deviation and can lead to better K -means results [146]. The difference from Algorithm 3 is that lines 5 to 7 are replaced with this whitening process, as presented in Algorithm 4.

6.2.2 Distributive KernelCanvas++

Another proposed extension to the KernelCanvas++ algorithm resides in its activation procedure. Recall that each point of a sequence is mapped to $\kappa = \lambda * |\mathcal{K}|$ kernels that are closer to it on the canvas. Since β bits are activated per kernel, this results in two collateral effects: depending on the activation rate λ , many kernels might activate in the same way despite being far from the closest kernel, and, secondly, there is no distinction between a kernel that gets activated by several points and a kernel that is activated once.

With these two opportunities in mind, the Distributive KernelCanvas++ (DKC++, DKC_{pp}) is then proposed. The Distributive KernelCanvas++ transforms each slot of β bits once reserved to a kernel into a distributive thermometer of β bits. Instead of having a binary activation with a stream of β ones or zeros, each kernel is now activated taking into consideration its activation score. Given a sequence S of p points, each kernel k can be activated up to p times, each one with

Algorithm 4 KCVQ - Transforms a sequence of points $\in \mathbb{R}^d$ into a binary vector $\in \{0, 1\}^B$

Input:

sequence ▷ Arbitrary size sequence of points $\in \mathbb{R}^d$
std ▷ Pre-calculated standard deviations for all dim
 \mathcal{K} ▷ The kernel set (generated via K -means)
 κ ▷ The number of kernels to activate

Output:

input_vector ▷ The final binarized input vector

```

1: for bit_idx  $\leftarrow$  1 to  $B$  do
2:   input_vector[bit_idx]  $\leftarrow$  0
3: end for
4: for all point  $\in$  sequence do
5:   for dim  $\leftarrow$  1 to  $d$  do ▷ Whitening
6:     newPoint[dim]  $\leftarrow$   $\frac{\text{point}[\text{dim}]}{\text{std}[\text{dim}]}$ 
7:   end for
8:   activatedKernels  $\leftarrow$  findknn(newPoint,  $\mathcal{K}$ ,  $\kappa$ ) ▷ Find newPoint's  $\kappa$  nearest
   neighbors  $\in \mathcal{K}$ 
9:   for all kernel  $\in$  activatedKernels do ▷ Activate  $\beta$  bits per activated
kernel
10:    for bit  $\leftarrow$  1 to  $\beta$  do
11:      bit_idx  $\leftarrow$  (kernel - 1) *  $\beta$  + bit
12:      input_vector[bit_idx]  $\leftarrow$  1
13:    end for
14:   end for
15: end for

```

an activation score of $AS_{k,p}$. Hence, the final score for a given k in S is given by

$$AS_k = \sum_{p \in S} AS_{k,p}, \quad (6.3)$$

the sum of all partial activation scores for a kernel and point of the sequence $AS_{k,p}$. Afterwards, the thermometer encoding is applied per sequence, considering all kernel activation scores for that sequence. For example, the least activated kernel will receive a stream of β zeros as before, and the most activated kernel will receive its β bits marked with ones.

Let \mathcal{A}_S be the set of activated kernels for a sequence S , two different activation scores $AS_{k,p}$ are then defined as follows. First, the Naive Activation Score (NAS) is defined as

$$NAS_{k,p} = \begin{cases} 1, & \text{if } k \in \mathcal{A}_S, \\ 0, & \text{otherwise,} \end{cases} \quad (6.4)$$

, which solves the problem of accounting how many times a kernel was activated. A consecutive definition is the Exponential Activation Score, defined as

$$EAS_{k,p} = \begin{cases} \frac{e^{-d(p,k)}}{e^{-d^*(p)}}, & \text{if } k \in \mathcal{A}_S, \\ 0, & \text{otherwise,} \end{cases} \quad (6.5)$$

where $d(p,k)$ is the distance between the point p and the kernel k , $d^*(p)$ is the distance between the point and its closest kernel, and e is the Euler’s number. This formula achieves a maximum activation of one at minimum distances and a small activation near the farthest activated kernel.

6.2.3 First Experimental Setup & Results

Datasets & Evaluation

The methodology used to evaluate these proposed architectures was similar to Section 6.1.3. The same datasets and classifiers were used, except for the BloomWiSARD, which was replaced by the BTHOWeN classifier without the thermometer, since the input was already a binary vector. The same representations were used and the same hyperparameters were stressed, but only the PROP-1 and PROP-4 variations from Section 6.1.3 were evaluated, with and without DKC_{pp}, and with the addition of the KCVQ variation. Also, a different baseline was evaluated, BASE-MEAN-DT. This baseline is the same strategy used in Section 5.1, where the mean of node features is used in conjunction with the Distributive Thermometer. The features used for this baseline are the same features used for the KernelCanvas: degree, page-rank, core-number and layer-number.

All experiments were executed using the same ten-fold nested cross-validation. A random search was performed for best validation parameters, fixing in a maximum of $P = 100$ possibilities within a day of experiments for all models.

Overall Results

First, the overall results are presented. These adopted both classifier and the KernelCanvas strategy as hypothesis. Results are marked with suffix KC for KernelCanvas++ (PROP-1,PROP-4 and KCVQ) and DKC for the Distributive KernelCanvas++.

Starting with Table 6.12, the accuracy is shown for each classifier. Once again, the RandomForest baseline is dominant, with three of the best results, followed by the ClusWiSARD, with two. Comparing the use of the KCpp against the DKCpp, the DKCpp presented better or equal results to the KCpp in 66% of the possible scenarios (classifier + dataset). With respect to the best result possible, there is no clear winner, with both performing similarly.

	COLLAB	ENZYMES	IMDB-M	MUTAG	NCI1	PROTEINS
RandomForest-KC	85.8 ± 0.8	90.9 ± 2.3	65.8 ± 1.6	88.9 ± 5.7	75.8 ± 1.6	77.6 ± 4.6
RandomForest-DKC	85.8 ± 0.9	90.2 ± 1.9	66.0 ± 1.2	88.9 ± 5.2	75.8 ± 1.6	76.9 ± 3.8
WIS-KC	85.5 ± 1.1	90.8 ± 1.9	65.9 ± 2.5	88.3 ± 4.8	73.6 ± 2.4	76.4 ± 4.1
WIS+DKC	85.7 ± 0.7	91.2 ± 1.6	65.9 ± 2.5	88.8 ± 5.2	74.0 ± 2.1	76.0 ± 4.2
ClusWiSARD-KC	85.1 ± 1.4	91.2 ± 1.5	65.8 ± 1.6	89.4 ± 5.0	73.7 ± 2.4	75.7 ± 3.7
ClusWiSARD-DKC	85.5 ± 0.7	91.1 ± 1.7	66.2 ± 2.1	88.8 ± 3.9	73.9 ± 2.6	75.7 ± 4.1
RWIS-KC	84.6 ± 1.2	90.5 ± 1.6	66.2 ± 2.4	87.3 ± 3.6	72.8 ± 2.8	76.8 ± 4.4
RWIS-DKC	84.7 ± 1.0	90.1 ± 2.5	66.3 ± 2.6	87.8 ± 4.3	72.5 ± 2.4	76.6 ± 4.1
BWN-KC	82.0 ± 1.6	80.8 ± 2.5	63.6 ± 1.8	80.1 ± 17.5	67.6 ± 1.8	76.1 ± 6.0
BWN-DKC	81.6 ± 1.4	81.8 ± 2.3	63.6 ± 2.2	81.5 ± 18.7	67.7 ± 3.0	75.0 ± 5.9

Table 6.12: Mean accuracy and standard deviation per classifier in the TUDatasets - a comparison between KernelCanvas++ and the Distributive KernelCanvas++. Best results are highlighted with a black background, second best with a gray background.

Breaking Down by Alternatives

In this section, results are presented per strategy, with the Dictionary WiSARD kept as the reference classifier. The goal is to observe how the different strategies behave, similar to the analysis in Section 6.1.3.

Table 6.13 presents the accuracy results for the evaluated KernelCanvas strategies (PROP-1,PROP-4, KCVQ) and its distributive counterparts (Naive Distributive Activation - NDA and Exponential Distributive Activation - EDA, alongside the BASE-MEAN+DT baseline in a ten-fold cross validation execution. In this table, it is possible to conclude that KCVQ is a dominant strategy, despite the fact that

it did not have clear gains when combined with the DKCcpp strategy. Likewise, the PROP-1 achieved again the best result in the COLLAB dataset without using any distributive activation. No variation was capable of surpassing the baseline in the NCI1 dataset, once again pointing into directions that should be noted, especially with respect to this dataset. Nonetheless, both PROP-1 and PROP-4 were able to show accuracy gains when combined with DKCcpp.

Once again, memory and energy consumption data were collected using PBS traces. Energy consumption results are reported in Table 6.14. Clearly, the BASE-MEAN+DT baseline is a dominant strategy in this aspect, since it is the simplest strategy among all others. When analyzing the proposed alternatives, PROP-4 appears as the least energy-demanding strategy. It is worth pointing that the DKCcpp alternatives demand more energy when compared to the binary activation strategies.

The memory usage is shown in Table 6.15. All strategies behave similarly in this aspect, regardless of whether there is distributive activation or not. The exception is the baseline, which spends more energy than the proposed alternatives on average.

	COLLAB	ENZYMES	IMDB-M	MUTAG	NCI1	PROTEINS
PROP-1	85.1 ± 0.3	89.5 ± 0.7	64.8 ± 0.3	86.2 ± 1.4	71.8 ± 1.1	74.8 ± 1.2
PROP-1+NDA	85.1 ± 0.5	89.7 ± 0.6	65.0 ± 0.4	86.1 ± 0.8	71.6 ± 1.0	74.3 ± 1.1
PROP-1+EDA	85.1 ± 0.3	89.8 ± 0.7	64.9 ± 0.5	86.2 ± 1.0	71.3 ± 1.4	74.4 ± 1.0
PROP-4	83.7 ± 0.3	89.3 ± 0.8	65.1 ± 0.3	86.0 ± 1.3	71.6 ± 0.8	74.3 ± 1.1
PROP-4+NDA	84.3 ± 0.7	89.7 ± 0.7	65.0 ± 0.4	86.0 ± 1.0	72.1 ± 1.5	74.2 ± 1.2
PROP-4+EDA	84.4 ± 0.6	89.7 ± 0.7	65.2 ± 0.5	86.5 ± 0.9	72.6 ± 0.9	73.8 ± 0.9
KCVQ	83.0 ± 0.3	90.2 ± 0.6	65.3 ± 0.4	87.0 ± 0.9	71.5 ± 1.0	74.6 ± 0.8
KCVQ+NDA	84.1 ± 0.4	89.9 ± 0.8	65.2 ± 0.4	86.8 ± 1.0	71.9 ± 1.5	74.5 ± 0.6
KCVQ+EDA	84.1 ± 0.5	89.7 ± 0.4	65.2 ± 0.3	86.7 ± 1.0	72.1 ± 1.1	74.8 ± 0.7
BASE-MEAN+DT	83.8 ± 0.4	86.4 ± 0.8	65.1 ± 0.3	86.0 ± 0.9	72.9 ± 0.4	73.3 ± 0.6

Table 6.13: Average accuracy and standard deviation results in the TUDatasets - comparison between KCcpp and DKCcpp strategies. Best results are highlighted with a black background, second best with a gray background.

	COLLAB	ENZYMES	IMDB-M	MUTAG	NCI1	PROTEINS
PROP-1	187.6 ± 162.1	22.0 ± 8.4	14.6 ± 9.8	3.9 ± 1.6	76.3 ± 84.4	28.9 ± 17.1
PROP-1+NDA	343.0 ± 108.5	22.0 ± 9.7	12.8 ± 5.9	4.5 ± 2.7	43.8 ± 18.4	30.1 ± 14.7
PROP-1+EDA	295.0 ± 43.7	24.7 ± 16.6	13.9 ± 6.6	4.0 ± 2.0	45.6 ± 20.7	26.6 ± 14.5
PROP-4	211.4 ± 197.4	10.3 ± 8.0	6.2 ± 8.0	1.8 ± 0.4	35.9 ± 18.7	11.0 ± 7.3
PROP-4+NDA	268.6 ± 37.2	14.0 ± 4.9	11.6 ± 5.1	1.8 ± 0.6	32.9 ± 12.1	14.1 ± 5.4
PROP-4+EDA	293.9 ± 42.3	14.7 ± 5.0	12.5 ± 6.1	1.8 ± 0.6	35.2 ± 14.0	15.1 ± 5.9
KCVQ	230.1 ± 104.2	24.6 ± 13.2	11.1 ± 9.7	3.9 ± 2.2	38.1 ± 22.2	31.6 ± 16.9
KCVQ+NDA	256.4 ± 32.0	20.8 ± 8.2	12.0 ± 5.3	4.0 ± 1.8	42.9 ± 18.2	28.7 ± 16.7
KCVQ+EDA	267.2 ± 49.8	20.4 ± 8.3	12.7 ± 6.2	4.0 ± 1.8	45.0 ± 19.9	26.9 ± 13.2
BASE-MEAN+DT	252.2 ± 17.9	7.3 ± 1.4	8.1 ± 3.3	1.3 ± 0.9	23.1 ± 5.7	9.3 ± 4.4

Table 6.14: Average energy consumption (in Wh) on the TUDatasets - comparison between KCcpp and DKCcpp strategies. Best results are highlighted with a black background, second best with a gray background.

	COLLAB	ENZYMES	IMDB-M	MUTAG	NCII	PROTEINS
PROP-1	31.5 ± 1.8	9.3 ± 0.2	9.0 ± 0.2	8.6 ± 0.0	10.6 ± 0.3	9.5 ± 0.3
PROP-1+NDA	30.9 ± 1.7	9.3 ± 0.2	9.0 ± 0.2	8.5 ± 0.0	10.6 ± 0.3	9.5 ± 0.4
PROP-1+EDA	31.8 ± 1.8	9.3 ± 0.2	9.0 ± 0.2	8.5 ± 0.0	10.6 ± 0.3	9.5 ± 0.3
PROP-4	31.5 ± 1.6	9.6 ± 0.4	9.0 ± 0.2	8.7 ± 0.1	10.9 ± 0.2	9.6 ± 0.3
PROP-4+NDA	31.7 ± 1.5	9.6 ± 0.4	9.0 ± 0.2	8.7 ± 0.1	10.9 ± 0.4	9.7 ± 0.3
PROP-4+EDA	31.6 ± 1.6	9.6 ± 0.4	9.0 ± 0.2	8.7 ± 0.1	10.9 ± 0.3	9.6 ± 0.3
KCVQ	31.0 ± 1.4	11.5 ± 0.3	11.1 ± 0.3	10.6 ± 0.1	13.0 ± 0.6	11.5 ± 0.2
KCVQ+NDA	31.4 ± 1.7	9.4 ± 0.3	9.0 ± 0.2	8.6 ± 0.0	10.7 ± 0.3	9.4 ± 0.2
KCVQ+EDA	31.3 ± 1.6	9.3 ± 0.3	9.0 ± 0.2	8.6 ± 0.0	10.7 ± 0.3	9.4 ± 0.2
BASE-MEAN+DT	30.2 ± 0.7	11.6 ± 0.3	11.7 ± 0.5	10.6 ± 0.1	13.6 ± 1.0	11.8 ± 0.5

Table 6.15: Average virtual memory usage on TUDatasets (in GB) - comparison between KCpp and DKCpp strategies. Best results are highlighted with a black background, second best with a gray background.

Table 6.16 exhibits averages for all evaluated strategies of the time spent in search, training and inference. It is possible to conclude from this table that the baseline BASE-MEAN+DT is clearly faster than all proposed alternatives. Among the proposed alternatives, PROP-4 appears as the fastest. Also, there is no clear increase in the time spent when using the DKCpp during search, training or inference.

	COLLAB			ENZYMES			IMDB-M		
	Search	Training	Inference	Search	Training	Inference	Search	Training	Inference
PROP-1	576.3 ± 81.2	20.9 ± 10.2	2.4 ± 1.2	29.5 ± 14.5	4.3 ± 2.1	0.5 ± 0.3	18.7 ± 9.4	2.0 ± 1.1	0.4 ± 0.2
PROP-1+NDA	635.1 ± 88.7	16.7 ± 8.6	1.9 ± 1.0	30.1 ± 14.7	4.5 ± 2.2	0.5 ± 0.3	19.4 ± 9.6	2.0 ± 1.0	0.4 ± 0.2
PROP-1+EDA	607.8 ± 70.8	16.1 ± 5.5	1.9 ± 0.7	31.4 ± 15.6	4.8 ± 2.3	0.6 ± 0.3	20.6 ± 10.2	2.4 ± 1.3	0.4 ± 0.2
PROP-4	581.1 ± 61.9	14.7 ± 5.4	1.5 ± 0.7	20.9 ± 7.6	2.8 ± 0.9	0.3 ± 0.1	17.0 ± 8.2	1.9 ± 1.0	0.3 ± 0.2
PROP-4+NDA	568.9 ± 71.6	11.7 ± 3.5	1.2 ± 0.4	21.2 ± 7.8	2.6 ± 0.9	0.3 ± 0.1	17.8 ± 8.5	2.0 ± 1.1	0.3 ± 0.2
PROP-4+EDA	611.0 ± 75.6	12.8 ± 4.3	1.3 ± 0.5	22.4 ± 8.4	2.9 ± 1.1	0.3 ± 0.1	18.8 ± 9.1	2.2 ± 1.2	0.4 ± 0.2
KCVQ	556.6 ± 71.4	15.0 ± 8.0	1.6 ± 0.9	35.0 ± 15.8	5.2 ± 2.2	0.5 ± 0.2	21.2 ± 10.1	2.4 ± 1.1	0.3 ± 0.2
KCVQ+NDA	564.1 ± 61.4	9.7 ± 3.8	1.0 ± 0.5	30.3 ± 13.1	4.0 ± 1.5	0.4 ± 0.2	18.3 ± 8.9	2.3 ± 1.2	0.4 ± 0.2
KCVQ+EDA	586.0 ± 73.2	10.8 ± 4.5	1.2 ± 0.5	31.5 ± 13.8	4.1 ± 1.6	0.5 ± 0.2	19.2 ± 9.4	2.4 ± 1.3	0.4 ± 0.2
BASE-MEAN+DT	565.7 ± 47.1	3.2 ± 1.3	0.5 ± 0.2	10.6 ± 2.4	0.7 ± 0.2	0.1 ± 0.0	12.2 ± 4.6	1.2 ± 0.7	0.3 ± 0.2
	MUTAG			NCII			PROTEINS		
	Search	Training	Inference	Search	Training	Inference	Search	Training	Inference
PROP-1	5.8 ± 2.7	0.7 ± 0.4	0.1 ± 0.0	101.6 ± 116.7	8.2 ± 9.3	1.0 ± 1.1	39.2 ± 20.0	4.7 ± 2.6	0.5 ± 0.3
PROP-1+NDA	5.8 ± 2.8	0.7 ± 0.4	0.1 ± 0.0	70.1 ± 31.6	5.7 ± 2.6	0.7 ± 0.3	40.2 ± 20.6	4.8 ± 2.7	0.6 ± 0.3
PROP-1+EDA	5.9 ± 2.8	0.7 ± 0.4	0.1 ± 0.0	74.8 ± 34.6	6.2 ± 3.1	0.7 ± 0.4	42.1 ± 21.5	4.8 ± 2.8	0.6 ± 0.3
PROP-4	2.5 ± 0.9	0.2 ± 0.1	0.0 ± 0.0	54.2 ± 21.8	3.2 ± 1.7	0.4 ± 0.2	20.7 ± 8.1	1.9 ± 1.1	0.2 ± 0.1
PROP-4+NDA	2.6 ± 0.9	0.2 ± 0.1	0.0 ± 0.0	57.1 ± 22.3	3.5 ± 1.3	0.4 ± 0.2	22.3 ± 8.2	2.0 ± 1.0	0.2 ± 0.1
PROP-4+EDA	2.7 ± 1.0	0.2 ± 0.1	0.0 ± 0.0	61.5 ± 24.8	3.5 ± 1.4	0.4 ± 0.2	24.4 ± 9.7	2.2 ± 1.2	0.3 ± 0.1
KCVQ	5.9 ± 2.7	0.8 ± 0.4	0.1 ± 0.0	81.2 ± 37.2	6.6 ± 2.9	0.6 ± 0.3	41.5 ± 20.5	4.7 ± 2.5	0.5 ± 0.3
KCVQ+NDA	5.8 ± 2.7	0.7 ± 0.4	0.1 ± 0.0	72.2 ± 31.8	5.4 ± 2.2	0.6 ± 0.3	38.6 ± 19.1	4.4 ± 2.4	0.5 ± 0.3
KCVQ+EDA	5.9 ± 2.8	0.7 ± 0.4	0.1 ± 0.0	77.2 ± 35.1	5.7 ± 2.5	0.6 ± 0.3	41.1 ± 20.5	4.9 ± 2.6	0.6 ± 0.3
BASE-MEAN+DT	2.1 ± 0.7	0.1 ± 0.1	0.0 ± 0.0	38.3 ± 9.4	2.5 ± 1.2	0.3 ± 0.1	14.7 ± 2.3	0.9 ± 0.5	0.1 ± 0.1

Table 6.16: Average time (seconds) spent on Search, Training and Inference on TUDatasets (in seconds) - comparison between KCpp and DKCpp strategies. Best results are highlighted with a black background, second best with a gray background.

6.2.4 Larger Scale Experiment

Several other datasets were proposed in the graph literature. In [9] were proposed the OGB datasets, which have a clear goal of structuring benchmarks related to graph

learning tasks. These datasets have predefined metrics to be evaluated, predefined training and test splits, and present problems at larger scales (some of them having terabytes of size). While they are categorized over small, medium and large scales, even their small scale datasets are larger than the TUDatasets used in previous experiments.

Datasets & Evaluation

In this work, due to size constraints, only one OGB dataset was chosen for experiments, called OGB-molhiv. This is the smallest dataset in the group and it is associated with a binary classification problem. The task is to predict whether the graph (molecule) inhibits HIV virus replication or not. More details are presented in Tables 6.17 and 6.18.

type	# Graphs	Nodes per graph(avg)	Edges per graph (avg)	Task	Evaluation Metric
BioInf	41,127	25.5	27.5	Binary Classification	ROC-AUC

Table 6.17: OGB-molhiv dataset statistics

Several trials were conducted varying the input size between in the range [8192, 40960] in steps of 4096. The number of random possibilities was fixed in $P = 100$, within a day of experiment. Since it is not feasible to load several copies of this dataset in the memory at the same time, a batch-oriented learning framework was used. In other words, only a batch b of 256 examples of the training data was used at a time, restricting the memory utilization, thus allowing more parallelization, up to 8 jobs. Some modifications were then introduced: the calculation of CDFs via data sketches [147] and the use of a K -means mini batch variation available in [85]. Also, the analysis was constrained to the classifiers that allowed for this kind of learning: WiSARD, ClusWiSARD and BloomWiSARD.

Overall Results

Authors in [9] have also pre-defined how the evaluation on the OGB-molhiv dataset is done. Hence, the results presented here follow their evaluation metric (ROC-AUC). For each experiment and each classifier, the best ROC-AUC is reported in Table 6.19.

Node Labels	Edge Labels	Node Attr.	Edge Attr.
-	-	Y(9)	Y(3)

Table 6.18: OGB-molhiv dataset attributes

	Validation			Test		
	WiSARD	ClusWiSARD	BloomWiSARD	WiSARD	ClusWiSARD	BloomWiSARD
PROP-1	0.6020	0.6037	0.5375	0.5968	0.5933	0.5045
PROP-1+NDA	0.6143	0.5915	0.5370	0.5745	0.5815	0.5035
PROP-1+EDA	0.5906	0.5773	0.5307	0.5815	0.5748	0.5000
PROP-4	0.6145	0.6215	0.5784	0.6092	0.6164	0.5939
PROP-4+NDA	0.6229	0.6250	0.5760	0.6086	0.6190	0.5940
PROP-4+EDA	0.6541	0.6058	0.5755	0.6143	0.6060	0.5938
KCVQ	0.6122	0.6093	0.5086	0.5567	0.5663	0.5077
KCVQ+NDA	0.6201	0.6685	0.5187	0.5567	0.5723	0.5248
KCVQ+EDA	0.6262	0.6146	0.5233	0.5562	0.5568	0.5096

Table 6.19: ROC-AUC for both validation and test sets for the *OGB-molhiv* datasets for each classifier. Best results are highlighted with a black background, second best with a gray background.

From this table, it is possible to conclude that the best overall classifier was the ClusWISARD, with best overall result being the one using the Naive Distributive Activation (DKC++). It is also worth noting that the validation score was not a good proxy for the test score, since several approaches with good validation scores were not able to repeat the same success on the test data. Among the KC++ propositions, the PROP-4 stands out as a better strategy, but with respect to the activation policy, the results vary per classifier. Nevertheless, all values obtained in this experiment were still far away from the values in the OGB online rank ³, where the first place has a ROC-AUC of 0.8475 ± 0.0003 and the last candidate has a ROC-AUC of 0.7549 ± 0.0163 .

There might be several reasons for this results, such as

- Models sizes are not comparable: even the last place in the online rank has more than 500k optimized parameters;
- Many dataset attributes are categorical and the histogram strategy is failing to capture them as the KernelCanvas does on other datasets;
- Extreme low percentage of positive data, leading to biased models.

6.2.5 Sampled Input Size and Activation Rate

In this section, another experiment was drawn to analyze the effects of two aspects not analyzed before. In previous experiments, input sizes were searched but fixed as a hypothesis. Given that the input size was fixed, other parameters were searched for each input size. In this experiment, the input size was searched, but sampled. Also, it was analyzed the effects of the *activationRate* in the proposed KC++ and DKC++, which was kept fixed in prior experiments.

³https://ogb.stanford.edu/docs/leader_graphprop/#ogbg-molhiv

Datasets & Evaluation

The input size sampling was done with two hypothesis: sample a small budget - SSB, where the input size was sampled from the interval [2048, 16384] in steps of 2048, and sample a big budget - SBB, where the input size was sampled from the interval [8192, 40960] in steps of 4096. After that, all parameters were calculated in the same way as before. Also, for the *activationRate*, it was searched within the values 0.0, 0.01, 0.03, 0.07, 0.1, while it was kept fixed at 0.07 in previous experiments.

The datasets were the same of the TUDatasets from previous section with the addition of the OGB-molhiv. The TUDatasets were evaluated with 16 workers and across the WiSARD, ClusWiSARD, BloomWiSARD, BTHOWEN, and RandomForest classifiers. The OGB-molhiv was evaluated using a total of 8 workers, being only evaluated in the same set of classifiers as before: WiSARD, ClusWiSARD and BloomWiSARD. All other parameters were searched in the same way as the previous experiments.

Accuracy Results in TUDatasets

Tables 6.20 to 6.25 show the mean accuracy for a ten-fold cross validation execution per classifier and strategy. Each table focus on the results for a specific dataset. Overall, the same behavior from previous experiments is observed again, where each dataset is perceived as an unique problem by itself. In a comparison between WiSARD flavors and the RandomForest classifier, there is a tie, with three best results for each.

In Table 6.20 is possible to see that in the COLLAB dataset, the PROP-1 is the leading strategy, as before. Generally, in this dataset, the DKC_{pp} alternatives are marginally better than pure KC_{pp} alternatives. Concerning the ENZYMES dataset, the results in Table 6.21 show a divided scenario, with no clear predominance of strategies. A similar scenario happens in Table 6.23, and Table 6.24, which refers respectively to the MUTAG and NCI1 dataset. For the IMDB-M dataset, with results presented in 6.22, there is a clear accuracy superiority for the strategies using the PROP-4. This is mostly due to the *K*-means generated centroids, because there are only structural attributes in this dataset. Like in the COLLAB dataset, strategies with DKC_{pp} are generally better than pure KC_{pp} strategies. With respect to the PROTEINS dataset, data in Table 6.25 show a predominance divided by two strategies, PROP-1 and KCVQ, but again, DKC_{pp} performed generally better than pure KC_{pp} alternatives.

In comparison with the results presented in all previous experiments using the TUDatasets, the best overall results are generally lower in this experiment. This is mainly due to fact that the search space is greater, while the number of combina-

Strategy	Classifier					
	RandomForest	WiSARD	ClusWiSARD	BloomWiSARD	BTHOWeN	RandomWiSARD
PROP-1-SSB	85.52 ± 1.02	84.93 ± 1.02	84.35 ± 0.89	84.89 ± 1.07	81.92 ± 1.56	83.92 ± 1.35
PROP-1-SBB	85.69 ± 1.09	85.08 ± 1.00	84.95 ± 1.35	84.69 ± 1.01	79.05 ± 8.58	OOB
PROP-1+NDA-SSB	85.47 ± 1.25	85.45 ± 0.95	84.23 ± 1.16	85.05 ± 1.02	67.49 ± 13.25	83.96 ± 1.42
PROP-1+NDA-SBB	85.23 ± 1.26	85.16 ± 1.02	85.11 ± 1.03	84.93 ± 0.87	76.52 ± 11.39	84.01 ± 1.29
PROP-1+EDA-SSB	85.79 ± 0.98	85.11 ± 0.73	84.68 ± 1.30	84.53 ± 1.46	70.96 ± 13.77	83.64 ± 1.26
PROP-1+EDA-SBB	85.23 ± 1.23	85.39 ± 0.90	84.67 ± 1.17	84.67 ± 1.55	78.44 ± 8.48	OOB
PROP-4-SSB	85.13 ± 1.14	84.08 ± 1.58	83.92 ± 1.11	83.93 ± 1.25	78.84 ± 8.44	OOB
PROP-4-SBB	85.52 ± 0.94	84.72 ± 1.15	83.80 ± 1.23	83.56 ± 1.72	74.59 ± 11.74	OOB
PROP-4+NDA-SSB	85.08 ± 1.05	84.87 ± 0.66	84.37 ± 1.11	84.12 ± 1.40	72.65 ± 12.26	82.97 ± 1.33
PROP-4+NDA-SBB	85.37 ± 1.18	84.57 ± 1.04	83.79 ± 1.47	84.47 ± 1.16	70.91 ± 13.59	OOB
PROP-4+EDA-SSB	85.20 ± 1.17	84.40 ± 1.01	84.12 ± 1.71	83.96 ± 1.78	74.69 ± 10.83	OOB
PROP-4+EDA-SBB	85.37 ± 0.96	84.44 ± 1.42	84.13 ± 0.89	84.12 ± 1.37	76.19 ± 9.84	OOB
KCVQ-SSB	84.27 ± 1.10	83.72 ± 1.94	83.17 ± 1.28	82.05 ± 1.04	79.93 ± 2.97	82.27 ± 1.37
KCVQ-SBB	84.76 ± 1.04	83.51 ± 1.30	82.92 ± 1.11	82.36 ± 1.72	78.80 ± 8.41	OOB
KCVQ+NDA-SSB	84.69 ± 1.25	83.53 ± 1.36	83.96 ± 1.02	83.45 ± 1.01	80.59 ± 1.67	82.29 ± 1.50
KCVQ+NDA-SBB	85.11 ± 1.51	83.71 ± 1.28	83.81 ± 0.76	83.57 ± 1.35	81.12 ± 1.49	OOB
KCVQ+EDA-SSB	84.64 ± 2.22	83.81 ± 0.92	83.75 ± 1.16	82.97 ± 1.15	80.56 ± 1.42	82.29 ± 1.36
KCVQ+EDA-SBB	85.37 ± 1.52	84.49 ± 1.05	83.49 ± 1.71	83.53 ± 1.56	81.12 ± 1.49	OOB

Table 6.20: Average accuracy and standard deviation per strategy in the COLLAB dataset. Best results are highlighted with a black background, second best with a gray background.

tions is kept fixed in $P = 100$. This fact still has to be verified, but results here are generally similar than the results observed in previous experiments. This concludes that the input size can be treated as an hyperparameter. The *activationRate* parameter was not that relevant and varying it implied in no increase in the accuracy. The number of combinations could be increased, but due to time constraints, it was treated as out of scope.

Results for the OGB-molhiv

Regarding the OGB-molhiv dataset, ROC-AUC results are presented, as it is the designed evaluation metric. The best ROC-AUC is reported for each classifier and strategy in Table 6.26.

Results from this table show the ClusWiSARD as the best classifier for this dataset. Once again, the best validation score did not imply the best test score, indicating possibly overfitted models. Among the KCpp alternatives, the PROP-4 is again the leading strategy, but there is no clear DKCcpp alternative that prevailed. Comparing with the previous results on this dataset, there is a marginal increase in the best overall result with the ClusWiSARD, but a huge decrease with the other classifiers. The most plausible explanation to this fact is the same as before, as a limited number of possibilities in a greater search space leads the search to alternatives that might not be as successful as before.

Strategy	Classifier					
	RandomForest	WiSARD	ClusWiSARD	BloomWiSARD	BTHoWeN	RandomWiSARD
PROP-1-SSB	89.78 ± 1.70	89.83 ± 2.22	90.06 ± 2.33	89.50 ± 2.48	78.50 ± 1.80	89.50 ± 2.56
PROP-1-SBB	89.50 ± 1.69	90.72 ± 1.91	89.78 ± 2.07	90.44 ± 1.96	72.72 ± 1.58	89.17 ± 2.05
PROP-1+NDA-SSB	90.06 ± 2.07	89.61 ± 1.78	89.00 ± 2.69	87.94 ± 2.87	78.28 ± 2.81	89.83 ± 2.40
PROP-1+NDA-SBB	89.44 ± 1.76	90.39 ± 2.03	89.50 ± 2.49	89.72 ± 2.24	72.89 ± 2.11	89.22 ± 2.07
PROP-1+EDA-SSB	89.61 ± 2.05	89.28 ± 2.36	90.17 ± 1.72	88.50 ± 2.49	77.72 ± 3.41	89.89 ± 2.23
PROP-1+EDA-SBB	89.94 ± 1.62	90.22 ± 1.78	88.28 ± 3.04	90.22 ± 1.55	72.22 ± 0.00	88.78 ± 2.64
PROP-4-SSB	90.00 ± 1.70	89.11 ± 2.00	88.17 ± 3.13	89.17 ± 2.26	78.06 ± 2.00	90.89 ± 2.51
PROP-4-SBB	89.78 ± 2.54	90.67 ± 2.69	89.06 ± 2.54	89.94 ± 2.15	81.94 ± 3.15	OOOR
PROP-4+NDA-SSB	89.67 ± 1.84	88.11 ± 2.07	89.06 ± 2.81	90.17 ± 1.96	79.56 ± 3.48	OOOR
PROP-4+NDA-SBB	90.00 ± 2.27	90.00 ± 2.87	90.33 ± 2.00	89.78 ± 2.37	82.44 ± 2.35	OOOR
PROP-4+EDA-SSB	89.61 ± 1.96	89.06 ± 1.87	88.11 ± 3.16	88.78 ± 2.19	79.56 ± 3.38	OOOR
PROP-4+EDA-SBB	89.33 ± 1.97	89.83 ± 2.14	89.89 ± 1.30	89.89 ± 2.78	82.33 ± 2.83	OOOR
KCVQ-SSB	89.67 ± 2.16	88.33 ± 1.59	89.28 ± 1.96	88.39 ± 2.17	78.72 ± 2.98	OOOR
KCVQ-SBB	89.89 ± 1.45	90.78 ± 2.03	89.83 ± 2.16	90.11 ± 1.36	81.00 ± 4.01	OOOR
KCVQ+NDA-SSB	90.06 ± 1.75	88.72 ± 1.31	89.11 ± 2.11	88.33 ± 2.21	78.17 ± 3.57	OOOR
KCVQ+NDA-SBB	89.39 ± 1.93	90.00 ± 2.22	90.28 ± 1.12	89.83 ± 2.21	82.00 ± 2.61	OOOR
KCVQ+EDA-SSB	89.17 ± 2.49	88.89 ± 1.91	88.61 ± 3.13	88.61 ± 2.29	78.94 ± 3.48	OOOR
KCVQ+EDA-SBB	89.28 ± 2.43	89.78 ± 1.78	89.83 ± 1.96	89.28 ± 2.70	82.67 ± 2.98	OOOR

Table 6.21: Average accuracy per strategy in the ENZYMES dataset. Best results are highlighted with a black background, second best with a gray background.

Strategy	Classifier					
	RandomForest	WiSARD	ClusWiSARD	BloomWiSARD	BTHoWeN	RandomWiSARD
PROP-1-SSB	65.11 ± 1.65	65.02 ± 2.36	64.53 ± 2.76	64.67 ± 2.26	62.40 ± 3.24	64.80 ± 1.66
PROP-1-SBB	64.93 ± 1.87	65.11 ± 1.85	64.84 ± 3.07	64.80 ± 2.11	62.18 ± 2.01	64.67 ± 1.98
PROP-1+NDA-SSB	65.07 ± 1.83	64.80 ± 2.71	65.07 ± 3.07	64.40 ± 2.03	61.51 ± 3.51	65.16 ± 2.79
PROP-1+NDA-SBB	65.38 ± 2.28	65.42 ± 2.24	65.33 ± 3.44	64.93 ± 3.00	62.80 ± 2.77	64.89 ± 2.45
PROP-1+EDA-SSB	64.53 ± 1.57	64.89 ± 2.45	64.76 ± 2.87	64.93 ± 2.17	61.64 ± 2.43	65.24 ± 3.47
PROP-1+EDA-SBB	64.76 ± 2.21	65.24 ± 2.57	65.29 ± 2.19	64.67 ± 2.27	63.64 ± 3.41	64.93 ± 2.25
PROP-4-SSB	65.78 ± 2.66	65.38 ± 3.65	64.31 ± 2.51	64.44 ± 2.69	62.36 ± 2.17	65.07 ± 3.03
PROP-4-SBB	65.07 ± 1.76	65.64 ± 2.58	65.11 ± 2.20	64.98 ± 2.67	62.13 ± 3.34	65.11 ± 2.06
PROP-4+NDA-SSB	65.29 ± 1.72	65.24 ± 2.63	65.29 ± 1.96	64.71 ± 2.74	63.38 ± 2.36	64.98 ± 2.26
PROP-4+NDA-SBB	65.24 ± 2.78	65.64 ± 3.24	64.98 ± 2.22	64.80 ± 2.65	61.87 ± 2.70	65.56 ± 2.59
PROP-4+EDA-SSB	65.02 ± 2.85	65.69 ± 3.26	64.76 ± 1.66	65.82 ± 2.82	63.02 ± 2.96	66.09 ± 2.55
PROP-4+EDA-SBB	65.38 ± 2.83	65.51 ± 2.50	65.73 ± 3.16	65.87 ± 3.45	62.04 ± 2.94	64.13 ± 2.07
KCVQ-SSB	65.42 ± 2.23	64.62 ± 2.11	64.53 ± 2.53	64.58 ± 2.74	62.53 ± 3.07	65.64 ± 3.19
KCVQ-SBB	64.98 ± 1.96	65.38 ± 1.91	65.29 ± 2.47	64.67 ± 2.64	62.36 ± 3.16	64.49 ± 2.58
KCVQ+NDA-SSB	65.16 ± 2.23	65.11 ± 2.10	64.53 ± 2.89	64.04 ± 2.26	61.73 ± 3.08	65.56 ± 3.03
KCVQ+NDA-SBB	64.89 ± 1.77	65.51 ± 2.52	65.07 ± 1.82	65.11 ± 2.89	62.00 ± 3.40	64.53 ± 2.20
KCVQ+EDA-SSB	65.11 ± 1.85	64.31 ± 2.34	64.71 ± 2.58	64.27 ± 2.86	62.76 ± 1.99	65.69 ± 3.13
KCVQ+EDA-SBB	64.22 ± 1.47	65.60 ± 2.29	64.71 ± 1.99	64.84 ± 2.84	61.87 ± 3.39	64.53 ± 2.33

Table 6.22: Average accuracy per strategy in the IMDB-M dataset. Best results are highlighted with a black background, second best with a gray background.

Strategy	Classifier					
	RandomForest	WiSARD	ClusWiSARD	BloomWiSARD	BTHOWeN	RandomWiSARD
PROP-1-SSB	83.57 ± 6.66	84.59 ± 7.60	81.37 ± 8.34	82.49 ± 6.96	73.27 ± 21.28	80.88 ± 5.62
PROP-1-SBB	83.04 ± 5.84	86.17 ± 7.53	84.59 ± 5.19	84.04 ± 9.97	51.08 ± 22.84	79.33 ± 10.26
PROP-1+NDA-SSB	84.09 ± 6.46	81.43 ± 7.45	82.98 ± 7.39	82.49 ± 6.96	79.85 ± 19.45	80.82 ± 6.96
PROP-1+NDA-SBB	85.15 ± 6.85	85.61 ± 9.38	86.17 ± 6.19	83.04 ± 8.79	61.37 ± 22.82	77.75 ± 9.73
PROP-1+EDA-SSB	84.59 ± 6.27	81.43 ± 7.02	85.61 ± 7.51	83.01 ± 7.30	68.01 ± 24.53	80.82 ± 7.39
PROP-1+EDA-SBB	84.62 ± 5.69	82.95 ± 6.55	84.53 ± 8.50	81.40 ± 10.30	71.32 ± 20.13	81.96 ± 11.64
PROP-4-SSB	84.09 ± 5.49	83.63 ± 11.16	83.51 ± 6.30	83.57 ± 5.67	45.09 ± 19.00	81.37 ± 8.43
PROP-4-SBB	86.23 ± 6.59	81.37 ± 6.71	82.98 ± 6.01	84.09 ± 6.97	51.93 ± 24.73	78.71 ± 8.96
PROP-4+NDA-SSB	85.61 ± 7.56	86.73 ± 6.15	82.46 ± 7.06	85.15 ± 6.91	62.75 ± 24.68	81.93 ± 6.61
PROP-4+NDA-SBB	84.09 ± 6.46	82.51 ± 7.35	84.04 ± 8.95	85.67 ± 6.98	48.25 ± 24.24	79.82 ± 8.83
PROP-4+EDA-SSB	86.20 ± 5.01	85.67 ± 6.98	85.64 ± 7.03	85.70 ± 9.24	48.25 ± 24.24	79.80 ± 7.74
PROP-4+EDA-SBB	84.09 ± 7.76	86.17 ± 5.67	85.12 ± 7.75	84.62 ± 5.69	44.27 ± 19.06	80.82 ± 8.38
KCVQ-SSB	83.48 ± 7.30	84.59 ± 7.18	79.33 ± 7.06	84.59 ± 7.18	65.44 ± 23.68	80.82 ± 9.74
KCVQ-SBB	87.25 ± 5.68	85.12 ± 9.53	83.48 ± 6.40	85.64 ± 9.62	57.43 ± 21.63	76.64 ± 11.41
KCVQ+NDA-SSB	86.67 ± 6.35	85.12 ± 7.75	84.09 ± 9.54	85.12 ± 7.75	55.96 ± 24.90	80.82 ± 9.74
KCVQ+NDA-SBB	83.04 ± 7.71	86.17 ± 9.35	82.95 ± 5.53	85.12 ± 10.16	57.43 ± 21.63	76.64 ± 11.41
KCVQ+EDA-SSB	82.98 ± 7.39	85.12 ± 7.75	82.49 ± 8.55	84.59 ± 7.99	55.44 ± 24.16	81.40 ± 9.68
KCVQ+EDA-SBB	83.07 ± 7.67	85.67 ± 10.50	82.98 ± 7.39	85.67 ± 10.20	52.16 ± 20.77	76.64 ± 11.41

Table 6.23: Average accuracy per strategy in the MUTAG dataset. Best results are highlighted with a black background, second best with a gray background.

Strategy	Classifier					
	RandomForest	WiSARD	ClusWiSARD	BloomWiSARD	BTHOWeN	RandomWiSARD
PROP-1-SSB	73.89 ± 2.11	70.85 ± 2.32	71.53 ± 1.19	70.44 ± 2.04	65.01 ± 6.03	71.78 ± 1.51
PROP-1-SBB	73.26 ± 2.41	72.73 ± 1.94	71.41 ± 3.50	72.07 ± 2.04	64.82 ± 5.73	72.65 ± 1.54
PROP-1+NDA-SSB	74.53 ± 1.95	72.53 ± 2.56	71.31 ± 1.99	71.73 ± 1.71	66.89 ± 3.36	70.61 ± 2.79
PROP-1+NDA-SBB	73.92 ± 2.15	73.07 ± 2.27	72.53 ± 1.77	71.19 ± 2.26	64.94 ± 4.14	70.97 ± 1.63
PROP-1+EDA-SSB	73.24 ± 2.68	71.68 ± 2.66	71.22 ± 1.74	71.39 ± 2.86	62.34 ± 5.42	71.36 ± 2.06
PROP-1+EDA-SBB	74.53 ± 2.23	72.77 ± 2.17	71.70 ± 1.52	72.43 ± 2.16	66.03 ± 3.28	71.73 ± 2.18
PROP-4-SSB	73.21 ± 1.90	70.63 ± 2.83	71.46 ± 2.02	71.19 ± 1.68	67.08 ± 2.95	71.00 ± 2.60
PROP-4-SBB	73.97 ± 2.56	72.55 ± 2.68	72.92 ± 2.40	72.97 ± 1.46	66.42 ± 2.71	72.19 ± 1.40
PROP-4+NDA-SSB	72.99 ± 2.01	71.39 ± 2.33	73.28 ± 2.42	71.36 ± 2.25	65.60 ± 3.08	71.87 ± 2.20
PROP-4+NDA-SBB	74.06 ± 2.67	73.80 ± 2.64	71.58 ± 3.06	73.14 ± 2.02	65.82 ± 6.03	71.05 ± 1.86
PROP-4+EDA-SSB	73.67 ± 1.57	72.94 ± 2.58	72.70 ± 2.13	70.90 ± 2.61	65.89 ± 3.19	72.31 ± 2.23
PROP-4+EDA-SBB	74.06 ± 2.43	73.09 ± 3.10	72.34 ± 1.76	72.99 ± 2.20	67.03 ± 3.31	71.70 ± 2.22
KCVQ-SSB	74.77 ± 3.03	72.21 ± 2.34	71.90 ± 2.48	71.29 ± 2.52	63.55 ± 3.84	72.17 ± 2.52
KCVQ-SBB	73.65 ± 3.13	72.36 ± 2.50	72.75 ± 2.97	73.28 ± 1.96	64.84 ± 2.30	71.90 ± 1.75
KCVQ+NDA-SSB	74.50 ± 2.70	72.09 ± 2.64	71.56 ± 3.00	70.66 ± 2.43	60.95 ± 5.15	72.17 ± 2.51
KCVQ+NDA-SBB	73.82 ± 3.47	72.80 ± 1.78	73.60 ± 2.06	73.33 ± 1.96	66.08 ± 2.23	71.92 ± 2.08
KCVQ+EDA-SSB	74.50 ± 2.70	72.21 ± 2.68	71.31 ± 1.86	71.05 ± 2.02	61.75 ± 5.67	72.02 ± 2.50
KCVQ+EDA-SBB	74.28 ± 3.30	72.80 ± 2.18	73.02 ± 2.49	73.11 ± 2.12	66.06 ± 2.40	72.17 ± 2.37

Table 6.24: Average accuracy per strategy in the NCI1 dataset. Best results are highlighted with a black background, second best with a gray background.

Strategy	Classifier					
	RandomForest	WiSARD	ClusWiSARD	BloomWiSARD	BTHOWeN	RandomWiSARD
PROP-1-SSB	75.21 ± 4.41	74.22 ± 3.68	74.94 ± 5.03	73.68 ± 3.50	64.88 ± 17.25	75.30 ± 4.36
PROP-1-SBB	76.01 ± 3.66	74.31 ± 4.61	74.31 ± 5.56	74.40 ± 4.29	74.04 ± 5.76	75.57 ± 3.85
PROP-1+NDA-SSB	75.65 ± 2.92	74.76 ± 3.71	72.33 ± 3.31	73.50 ± 3.21	62.89 ± 16.04	75.93 ± 4.68
PROP-1+NDA-SBB	75.65 ± 4.21	75.20 ± 3.75	73.32 ± 4.16	72.60 ± 4.59	74.58 ± 4.18	76.02 ± 3.97
PROP-1+EDA-SSB	75.47 ± 3.35	74.94 ± 5.45	73.14 ± 5.80	74.30 ± 3.67	66.92 ± 14.62	74.58 ± 3.65
PROP-1+EDA-SBB	74.12 ± 4.45	75.93 ± 4.89	75.48 ± 4.42	74.85 ± 3.73	70.47 ± 11.25	75.12 ± 4.02
PROP-4-SSB	75.48 ± 3.76	75.12 ± 4.66	72.51 ± 4.27	74.49 ± 4.04	67.64 ± 14.56	74.57 ± 3.95
PROP-4-SBB	73.76 ± 3.82	73.76 ± 3.47	73.59 ± 5.80	72.77 ± 4.44	69.71 ± 11.11	74.39 ± 4.26
PROP-4+NDA-SSB	74.85 ± 4.05	75.02 ± 3.50	73.32 ± 4.19	73.32 ± 4.70	64.52 ± 16.92	75.12 ± 4.89
PROP-4+NDA-SBB	74.67 ± 3.15	74.04 ± 4.27	74.12 ± 3.02	73.85 ± 3.99	70.88 ± 11.70	74.94 ± 4.23
PROP-4+EDA-SSB	74.58 ± 3.84	74.39 ± 3.11	72.95 ± 5.11	74.84 ± 5.00	62.99 ± 15.95	75.39 ± 4.16
PROP-4+EDA-SBB	72.96 ± 3.93	73.68 ± 4.48	72.15 ± 4.00	73.41 ± 4.54	63.53 ± 16.90	75.83 ± 4.42
KCVQ-SSB	74.21 ± 4.35	75.11 ± 3.51	74.57 ± 3.08	75.29 ± 3.19	67.41 ± 15.04	74.22 ± 4.08
KCVQ-SBB	74.40 ± 4.60	74.66 ± 3.59	72.78 ± 4.28	74.84 ± 3.33	69.72 ± 11.17	75.84 ± 4.54
KCVQ+NDA-SSB	74.48 ± 5.48	75.57 ± 4.51	74.57 ± 2.65	75.57 ± 4.51	68.81 ± 15.46	76.11 ± 4.45
KCVQ+NDA-SBB	74.13 ± 5.28	74.13 ± 4.33	73.59 ± 2.96	74.58 ± 3.81	62.99 ± 15.77	75.65 ± 4.42
KCVQ+EDA-SSB	75.30 ± 5.36	76.55 ± 2.89	74.48 ± 2.82	76.02 ± 3.68	70.80 ± 11.80	75.21 ± 4.95
KCVQ+EDA-SBB	74.22 ± 5.59	74.58 ± 3.56	75.20 ± 4.55	74.76 ± 3.62	62.96 ± 15.72	76.28 ± 4.42

Table 6.25: Average accuracy per strategy in the PROTEINS dataset. Best results are highlighted with a black background, second best with a gray background.

6.3 Leveraging Embeddings with KC++ and DKC++

Following the approach discussed in Section 5.1, two extensions to the initial pipelines are proposed and evaluated. They both try to maximize the use of node attributes, with the experimental adoption of the KernelCanvas as a graph-level readout function. This performance is also compared with the performance of using strategies previously defined.

6.3.1 Architecture for Attributed Graphs and KC Investigation

In [113], authors propose a generic approach for attributed graphs, although the evaluated datasets do not present any. Manually created features using nodes' log-degree and clustering coefficient are used instead. With the use of node attributes, graph embeddings can assume a prohibitive size. Authors in [112] experiment with a node-level attributed version that uses the SVD [148] dimensionality reduction technique. The caveat is that in this work there is no clear distinction of the results using the structural manually created features and no graph-level attributed version is evaluated or proposed. In the context of WNN models, the use of thermometers also increases the input size by a factor of B , where B is the number of bits per attribute, being mandatory to reduce the input dimensionality.

Let the final embedding matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$, where \mathbf{X}_i is the embedding for graph \mathcal{G}_i after the pooling function. The application of SVD to these embeddings

	Validation			Test		
	WiSARD	ClusWiSARD	BloomWiSARD	WiSARD	ClusWiSARD	BloomWiSARD
PROP-1-SSB	0.5670	0.5759	0.5246	0.5748	0.5705	0.5000
PROP-1-SBB	0.6136	0.6138	0.5123	0.5375	0.5599	0.5000
PROP-1+NDA-SSB	0.6019	0.5811	0.5185	0.5516	0.5021	0.5000
PROP-1+NDA-SBB	0.6033	0.6045	0.5306	0.5155	0.5613	0.5000
PROP-1+EDA-SSB	0.5829	0.5834	0.5185	0.5196	0.5164	0.4999
PROP-1+EDA-SBB	0.5912	0.5861	0.5246	0.5149	0.5568	0.5000
PROP-4-SSB	0.5474	0.6011	0.5200	0.5935	0.6229	0.5292
PROP-4-SBB	0.5746	0.5786	0.5484	0.5606	0.5007	0.5000
PROP-4+NDA-SSB	0.5750	0.5557	0.5192	0.5995	0.5984	0.5243
PROP-4+NDA-SBB	0.5904	0.5757	0.5327	0.5922	0.5132	0.5175
PROP-4+EDA-SSB	0.5897	0.5670	0.5181	0.5436	0.6053	0.5403
PROP-4+EDA-SBB	0.5785	0.5680	0.5719	0.5682	0.5253	0.5388
KCVQ-SSB	0.5324	0.5424	0.5206	0.5979	0.5099	0.4941
KCVQ-SBB	0.5558	0.5825	0.5060	0.5353	0.5691	0.5037
KCVQ+NDA-SSB	0.5340	0.5438	0.5091	0.5917	0.5087	0.4929
KCVQ+NDA-SBB	0.5609	0.5825	0.5060	0.5052	0.5691	0.5037
KCVQ+EDA-SSB	0.5289	0.5439	0.5183	0.4983	0.5174	0.4928
KCVQ+EDA-SBB	0.5514	0.5825	0.5059	0.5344	0.5691	0.5033

Table 6.26: ROC-AUC for both validation and test sets for the *OGB-molhiv* dataset for each classifier, strategy and sample strategy. Best results are highlighted with a black background, second best with a gray background.

is proposed, resulting in a matrix $\mathbf{X}^* \in \mathbb{R}^{N \times k}$, where k is number of dominant components of the singular value decomposition. The remainder of the architecture is kept the same, using the Distributive Thermometer and a final classifier. This variation will be referenced using the prefix POOL-SVD.

Following the approach developed in the previous section, another variation is proposed, referenced by the prefix SVD-KC and defined as follows. After all L node embeddings have been calculated, SVD is performed and the resulting matrix $\mathbf{X}^\dagger \in \mathbb{R}^{L \times k}$ is fed to KernelCanvas++, configuring the binarization step. The resulting binary vector is then fed to the final classifier in a similar fashion to the other alternatives.

6.3.2 Experimental Setup & Results

Following the initial experiment in Section 5.1, both WAVE and FTH have presented the best results considering the average only on RandomForest, WiSARD and ClusWiSARD. Hence, these two embeddings were elected as candidates for the evaluation of the attributed alternatives. They also make use of a mean pooled average of node embeddings to form their graph embeddings, which enables the comparison with a different readout such as the proposed SVD+KC alternative. The datasets in this evaluation were the same as in Section 6.1, also using the same manually created features for the IMDB-M and COLLAB datasets. The same methodology was used for hyperparameter search and a subset of WNN classifiers was used along the RandomForest baseline: WiSARD and ClusWiSARD.

Two different baseline strategies were also evaluated : BASE-MEAN-DT and BASE-KC. The BASE-MEAN-DT strategy is defined by using mean pooled node attributes (log-degree and clustering coefficient) as structural attributes and mean pooled node and edge attributes when applicable. These means are quantized using a Distributive Thermometer and fed to a classifier. A binary histogram is used for categorical attributes, also similar to Section 6.1. The final representation also follows Section 6.1, concatenating all quantized means and histograms into a single binary vector used as input. The BASE-KC is the same as the PROP-1 from 6.1, but using only log-degree and clustering coefficient as structural features.

	COLLAB			ENZYMES			IMDB-M		
	RandomForest	WiSARD	ClusWiSARD	RandomForest	WiSARD	ClusWiSARD	RandomForest	WiSARD	ClusWiSARD
A-POOL-SVD-FTH	85.5 ± 1.1	77.7 ± 0.8	OOO	83.8 ± 3.0	80.3 ± 1.8	80.2 ± 1.9	66.1 ± 2.1	59.3 ± 2.9	60.6 ± 3.2
A-SVD-KC-FTH	81.5 ± 0.9	77.9 ± 0.6	OOO	76.4 ± 1.9	75.6 ± 1.9	75.0 ± 1.8	64.0 ± 1.8	63.8 ± 2.9	63.1 ± 1.9
A-POOL-SVD-WAVE	85.9 ± 1.2	OOO	OOO	83.6 ± 2.0	81.3 ± 2.9	81.2 ± 3.1	66.2 ± 2.4	65.3 ± 1.8	64.6 ± 2.3
A-SVD-KC-WAVE	77.5 ± 1.0	76.8 ± 0.9	OOO	75.2 ± 1.9	75.7 ± 2.7	75.2 ± 1.8	63.3 ± 1.9	63.6 ± 2.4	64.0 ± 1.4
SVD-KC-FTH	81.1 ± 1.2	77.6 ± 0.7	OOO	N/A	N/A	N/A	63.2 ± 2.1	64.0 ± 2.1	63.2 ± 2.8
SVD-KC-WAVE	77.3 ± 1.1	76.5 ± 0.8	OOO	N/A	N/A	N/A	62.8 ± 2.9	63.8 ± 2.2	62.4 ± 2.5
BASE-KC	87.5 ± 1.3	86.0 ± 1.3	85.5 ± 1.7	89.9 ± 2.1	90.3 ± 2.7	89.7 ± 2.2	66.5 ± 1.8	66.4 ± 2.0	65.7 ± 2.9
BASE-MEAN-DT	80.5 ± 0.8	81.0 ± 1.3	81.0 ± 1.5	87.2 ± 2.2	86.1 ± 1.7	86.1 ± 2.5	64.8 ± 1.8	64.8 ± 3.3	64.4 ± 1.3
	MUTAG			NCI1			PROTEINS		
	RandomForest	WiSARD	ClusWiSARD	RandomForest	WiSARD	ClusWiSARD	RandomForest	WiSARD	ClusWiSARD
A-POOL-SVD-FTH	84.0 ± 7.5	70.2 ± 13.7	76.7 ± 11.0	78.6 ± 2.8	73.7 ± 2.0	74.4 ± 2.0	71.4 ± 3.7	56.4 ± 5.2	59.4 ± 3.5
A-SVD-KC-FTH	70.3 ± 9.3	74.0 ± 6.2	73.9 ± 7.5	65.6 ± 1.8	64.6 ± 1.6	63.6 ± 2.8	71.4 ± 5.3	71.0 ± 5.4	72.1 ± 3.1
A-POOL-SVD-WAVE	84.1 ± 5.5	86.2 ± 7.1	83.0 ± 6.4	78.2 ± 1.8	74.8 ± 2.1	75.3 ± 1.9	71.1 ± 2.6	60.1 ± 3.1	60.3 ± 2.7
A-SVD-KC-WAVE	69.3 ± 9.9	64.4 ± 8.3	71.8 ± 6.8	64.5 ± 2.6	64.2 ± 1.6	62.7 ± 1.9	71.0 ± 4.4	71.5 ± 5.4	71.6 ± 2.4
BASE-KC	85.1 ± 6.0	77.1 ± 6.8	78.1 ± 11.6	66.0 ± 1.9	66.4 ± 2.4	65.4 ± 1.8	76.6 ± 4.9	74.6 ± 4.2	73.2 ± 4.5
BASE-MEAN-DT	86.7 ± 5.8	84.6 ± 6.3	84.0 ± 5.6	70.1 ± 2.7	68.3 ± 2.2	67.2 ± 3.6	73.5 ± 5.5	74.1 ± 5.4	72.3 ± 4.4

Table 6.27: Average accuracy in 10-fold cross validation on a selection of TU-Datasets. Best results are highlighted with a black background, second best with a gray background.

Table 6.27 depicts the same scenario already mentioned in other experiments: every dataset is a different problem and there is no strategy that can beat all other strategies in all datasets. The results show that two proposed strategies can beat the baselines only in the NCI1 dataset with both A-POOL-SVD-FTH and A-POOL-SVD-WAVE. When comparing Table 5.3 with Table 6.27, it is possible to conclude that replacing the mean pool readout with SVD+KC does not yield better results. This is also reflected in the attributed variations, where the A-POOL-SVD variations generally perform better than their counterparts A-SVD-KC. The only clear exception in the PROTEINS dataset. This may be explained by the use of SVD on a matrix of graph embeddings instead of a matrix of node embeddings, but needs to be further investigated.

Another interesting point to take into account is that the strategy proposed in Section 6.1 is indeed a strong representation. It surpasses all graph embeddings in both COLLAB and IMDB-M datasets when using the RandomForest and generally has a better performance when the WiSARD is used.

6.4 Conclusions

Both architectures from previous chapter demonstrated to be successful attempts to graph classification. However, the whole distribution strategy lacks the use of all attributes in the graph, providing opportunities for further exploration. Following this exploration opportunity, this chapter introduced a new framework for graph classification based on a extensions for the KernelCanvas method, the KernelCanvas++. This framework was thoroughly analyzed, and results confirm that WNNs were able to surpass established graph learning models not only in resource usage but also in accuracy.

Two extensions to the KernelCanvas++ are also proposed and evaluated: Distributive KernelCanvas++ and KernelCanvas-VQ. These extensions resulted in improvements over the initial KernelCanvas++ proposals in some scenarios. The extensions were also evaluated in a large-scale dataset, where results were not as promising as in the TUDatasets. As the literature presents much higher scores for this large-scale dataset, this point needs to be further addressed.

Also, continuing on the work of Section 5.1, it was proposed the use of the KernelCanvas++ framework in conjunction with graph embeddings. While this conjunction was unable to surpass the baselines without graph embeddings, the results reinforce the strength of the KCpp architecture as a representation for graph classification.

Chapter 7

Conclusion & Future Works

In this chapter, conclusions are presented alongside a collection of ideas for future work and next steps.

7.1 Conclusions

Graph learning is a very important topic nowadays. Graphs are everywhere and almost everything can be represented as a graph. Several tasks have been defined in the literature as graph learning tasks, and in this work was focused on a branch of entire graph level learning, graph classification. As any graph related problem, graph classification has applications for every field of study, such as biology, chemistry, or math. Although many architectures have been proposed for graph classification, recent works mainly leverage deep architectures as their backbones. These can lead to successful results, but are huge resource demanding, which WNNs are not.

This work has evaluated different encoding methods from the literature prior to any experiment with graph classification. Moreover, it has defined the dominance of the Distributive Thermometer as the to-go binarization strategy for WNN. Both DT approaches studied by this work present 0.25 as their $L1$ -norm value, the best values for any WNN strategy.

A new encoding method, the K -Means Thermometer, was also introduced, resulting in a competitive alternative in different scenarios. When combined with RandomForest, the KT encoding topped 3 out of 11 datasets in terms of accuracy, beating its DT counterpart in both rank and $L1$ -norm metrics. The KT+WNN classifiers were also just marginally behind their DT counterparts with values of 0.26 and 0.28 with respect to the average $L1$ -norm metric. The dominance of thermometers over one-hot representations was also concluded, since all WNN models performed on average better using thermometers than using one-hot counterparts. However, all thermometers were surpassed by a No-Encoding strategy with RandomForest, demonstrating that there is still a performance gap to be filled.

Secondly, initial naive architectures for graph classification using weightless neural networks were proposed. Although based on naive bag of nodes features, initial experimental results in real and synthetic datasets pointed that it had comparable performance with other shallow learning and yet more complex alternatives, such as graph kernels. Based on ideas of current state-of-the-art representation, the Distributive Thermometer, this initial architecture was extended, obtaining even better results. The building blocks of this architecture were the complete redesign of the KernelCanvas, introducing the KernelCanvas++. When compared to GNN models, this confirmed the idea that the less can be more, given that the vanilla WiSARD has presented better results than most of the staple GNN models, with an average increase of 4% to 21% in terms of accuracy. The extended model is a fully capable model albeit strictly less memory and power demanding, resulting in a model that consumes at least 3x less energy than a lighter GNN, and up to 20x less energy when compared to other heavier candidates. Two major extensions to the KernelCanvas++ were proposed and tested: the KCVQ and the Distributive KernelCanvas, which can be used in conjunction. These extensions were capable of achieving better results in some experiments without sacrificing resource consumption or time demand. This architecture was also evaluated on a large scale dataset, the OGB-molhiv, demonstrating its scalability. As a remark, it could not carry on with the same performance in comparison with heavy-scale models, where performance is roughly 0.3 points behind the best candidate in the online ranking regarding the ROC-AUC score.

Hyperparameter optimization is still a bottleneck, but it is a much smaller problem in a WNN context than in models that depend on trillions of parameters and GPUs. This is heavily supported by the results of this work, where WNN models spent up to 18x less than in GNN models in hyperparameter search and training. The application of the DKCcpp framework to graph embeddings was also experimented, but the results were not satisfactory as with the bag-of-nodes approach, resulting in worse performance in terms of accuracy. However, these results reinforced the strength of the baseline KernelCanvas approach, which was dominant in all but the MUTAG and NCI datasets.

During the development of this research, several improvements have been made to the *wisardpkg*[30] library. These improvements are not claimed as direct contributions of this work, but will certainly empower the users of this library with more tools and several improvements.

7.1.1 Paper Contributions

The ideas and works developed in this thesis led to two accepted publications and one publication that is currently under review.

Accepted Publications

- BARBOSA, R. B.; CARVALHO, D. ; LIMA, P. M. V. ; FRANCA, F. M. G. . A bag of nodes primer on weightless graph classification. In: European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2021, Bruges. Proceedings of ESANN 2021. Brussels: i6doc.com, 2021. p. 659-665.
- BEZERRA BARBOSA, RAUL; LEONEL CADETTE DUTRA, DIEGO ; MACHADO VIEIRA LIMA, PRISCILA ; CARVALHO, DIEGO ; MAIA GALVÃO FRANÇA, FELIPE . Updating KernelCanvas for weightless graph classification. NEUROCOMPUTING, v. 646, p. 130458, 2025.

Submitted Publications

- BARBOSA, R. B.; DUTRA, L. C. DIEGO ; LIMA, M. V. PRISCILA ; CARVALHO, DIEGO ; FRANÇA, M. G. FELIPE . K-Means Thermometers for Weightless Input Embeddings. ESANN 2026.

7.2 Future Works

Considering the analysis in Chapter 4, there is a huge search space in the codebook and discretization literature that can be stressed. Other representations such as entropy or tree-based [149] encodings can be evaluated under the same framework. Other recent architectures such as ULEEN [68] should be stressed to consolidate the findings. Moreover, other representations can be developed exploring the potentials of both CMAC and effective thermometers such as the K-Means or the Distributive Thermometers. Also, a full resource usage analysis for the evaluated thermometers is necessary, including the proposed *K*-Means thermometer.

Several experiments can still be conducted following the ones in Chapters 5 and 6. A crucial one is the evaluation of the *K*-Means thermometer for graph classification. This evaluation was put out of scope due to time and resource constraints in the development of this thesis. The degree, PageRank and Onion Decomposition measures used as backbones for the KCpp and DKCcpp strategies are just a tiny subset of the possible measures that can be used for graph problems. Depending on time constraints, many other useful node centralities such as the betweenness

and harmonic centrality can boost the classification performance. As future work, the KCpp-based representations should be tested with other classifiers to confirm the results presented here and conclude if they can benefit from using them. Other datasets may also be tested to improve the comparison between both representations, such as the datasets presented in [128] or in [144].

Better results might be achieved through better parameter optimization. This can be handled through the use of dedicated libraries and Bayesian Methods. More complex models, with greater input sizes, could be trained without time restriction, particularly for a better conclusion in large-scale scenarios. Regarding the graph embeddings use, different techniques for dimensionality reduction should be evaluated, as SVD poses as a clear bottleneck of the proposed approach. Other graph embeddings can be enhanced with the replacement of mean and histogram-based approaches by KCpp-based ones. Recent WNN approaches benefit from multi-shot learning in an accuracy vs computing time trade-off. These can be studied to establish where this relationship stands in the context of graph learning.

Different applications can benefit from the developed framework and should be evaluated, such as malware detection and mental disease diagnosis. The main graph architecture was supported by the proposal of the new KernelCanvas++ and Distributive KernelCanvas++ strategies, but these are key innovations for the WNN scenario that can lead to better results in many areas. These approaches can also be leveraged to other multi-set problems such as trajectory classification [77].

One possible drawback that these approaches present is the lack of direct use of the adjacency matrix, similar to message-passing networks. Hence, a mechanism that mirrors the message-passing framework of GNN could be developed, added to the process of representation building and further evaluated for weightless graph classification.

References

- [1] BEZERRA BARBOSA, R., LEONEL CADETTE DUTRA, D., MACHADO VIEIRA LIMA, P., et al. “Updating KernelCanvas for weightless graph classification”, *Neurocomputing*, v. 646, pp. 130458, Sep. 2025. ISSN: 09252312. doi: 10.1016/j.neucom.2025.130458. Available at: <<https://linkinghub.elsevier.com/retrieve/pii/S0925231225011300>>.
- [2] LI, P., WANG, H., BÖHM, C. “Scalable graph classification via random walk fingerprints”. In: *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 10912–10915, Montreal, Canada, Sep. 2025. International Joint Conferences on Artificial Intelligence Organization. ISBN: 978-1-956792-06-5. doi: 10.24963/ijcai.2025/1215. Available at: <<https://www.ijcai.org/proceedings/2025/1215>>.
- [3] YANG, Z., ZHANG, G., WU, J., et al. “State of the art and potentialities of graph-level learning”. May 2023. arXiv:2301.05860 [cs].
- [4] BUTEREZ, D., JANET, J. P., OGLIC, D., et al. “An end-to-end attention-based approach for learning on graphs”, *Nature Communications*, v. 16, n. 1, pp. 5244, Jun. 2025. ISSN: 2041-1723. doi: 10.1038/s41467-025-60252-z. Available at: <<https://www.nature.com/articles/s41467-025-60252-z>>.
- [5] CHAMI, I., ABU-EL-HAIJA, S., PEROZZI, B., et al. “Machine learning on graphs: a model and comprehensive taxonomy”, *Journal of Machine Learning Research*, v. 23, n. 89, pp. 1–64, 2022. Available at: <<http://jmlr.org/papers/v23/20-852.html>>.
- [6] WU, Z., PAN, S., CHEN, F., et al. “A comprehensive survey on graph neural networks”, *IEEE Trans. Neural Netw. Learning Syst.*, v. 32, n. 1, pp. 4–24, Jan. 2021. ISSN: 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2020.2978386.
- [7] BRONSTEIN, M. M., BRUNA, J., LECUN, Y., et al. “Geometric deep learning: Going beyond euclidean data”, *IEEE Signal Process. Mag.*, v. 34, n. 4,

pp. 18–42, Jul. 2017. ISSN: 1053-5888, 1558-0792. doi: 10.1109/MSP.2017.2693418.

- [8] DEBNATH, A. K., LOPEZ DE COMPADRE, R. L., DEBNATH, G., et al. “Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity”, *J. Med. Chem.*, v. 34, n. 2, pp. 786–797, Feb. 1991. ISSN: 0022-2623, 1520-4804. doi: 10.1021/jm00106a046.
- [9] HU, W., FEY, M., ZITNIK, M., et al. “Open graph benchmark: datasets for machine learning on graphs”. Feb. 2021. arXiv:2005.00687 [cs, stat].
- [10] WALE, N., WATSON, I. A., KARYPIS, G. “Comparison of descriptor spaces for chemical compound retrieval and classification”, *Knowl Inf Syst*, v. 14, n. 3, pp. 347–375, Mar. 2008. ISSN: 0219-1377, 0219-3116. doi: 10.1007/s10115-007-0103-5.
- [11] BORGWARDT, K. M., ONG, C. S., SCHONAUER, S., et al. “Protein function prediction via graph kernels”, *Bioinformatics*, v. 21, n. Suppl 1, pp. i47–i56, Jun. 2005. ISSN: 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/bti1007.
- [12] HELMA, C., KING, R. D., KRAMER, S., et al. “The predictive toxicology challenge 2000–2001”, *Bioinformatics*, v. 17, n. 1, pp. 107–108, Jan. 2001. ISSN: 1367-4811, 1367-4803. doi: 10.1093/bioinformatics/17.1.107. Available at: <<https://academic.oup.com/bioinformatics/article/17/1/107/217049>>.
- [13] FREITAS, S., DONG, Y., NEIL, J., et al. “A large-scale database for graph representation learning”. Nov. 2021. Available at: <<http://arxiv.org/abs/2011.07682>>.
- [14] LIU, L., WEN, G., CAO, P., et al. “BrainTGL: A dynamic graph representation learning model for brain network analysis”, *Computers in Biology and Medicine*, v. 153, pp. 106521, Feb. 2023. ISSN: 00104825. doi: 10.1016/j.combiomed.2022.106521. Available at: <<https://linkinghub.elsevier.com/retrieve/pii/S001048252201229X>>.
- [15] KIPF, T. N., WELLING, M. “Semi-supervised classification with graph convolutional networks”. 2016. arXiv:1609.02907 [cs.LG].
- [16] XU, K., HU, W., LESKOVEC, J., et al. “How powerful are graph neural networks?” Feb. 2019. arXiv:1810.00826 [cs, stat].

- [17] VELIČKOVIĆ, P., CUCURULL, G., CASANOVA, A., et al. “Graph attention networks”. Feb. 2018. Available at: <<http://arxiv.org/abs/1710.10903>>. arXiv:1710.10903 [cs, stat].
- [18] GILMER, J., SCHOENHOLZ, S. S., RILEY, P. F., et al. “Neural message passing for quantum chemistry”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML’17*, pp. 1263–1272. JMLR.org, 2017.
- [19] SCARSELLI, F., GORI, M., TSOI, A. C., et al. “The graph neural network model”, *IEEE Transactions on Neural Networks*, v. 20, n. 1, pp. 61–80, 2009. doi: 10.1109/TNN.2008.2005605.
- [20] MICHELI, A. “Neural network for graphs: A contextual constructive approach”, *IEEE Trans. Neural Netw.*, v. 20, n. 3, pp. 498–511, Mar. 2009. ISSN: 1045-9227, 1941-0093. doi: 10.1109/TNN.2008.2010350.
- [21] LIU, C., ZHAN, Y., WU, J., et al. “Graph pooling for graph neural networks: progress, challenges, and opportunities”. Jun. 2023. Available at: <<http://arxiv.org/abs/2204.07321>>. arXiv:2204.07321 [cs].
- [22] MORRIS, C., RITZERT, M., FEY, M., et al. “Weisfeiler and Leman go neural: higher-order graph neural networks”, *AAAI*, v. 33, n. 01, pp. 4602–4609, Jul. 2019. ISSN: 2374-3468, 2159-5399. doi: 10.1609/aaai.v33i01.33014602.
- [23] STRUBELL, E., GANESH, A., MCCALLUM, A. “Energy and policy considerations for deep learning in NLP”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1355.
- [24] THOMPSON, N. C., GREENEWALD, K., LEE, K., et al. “The computational limits of deep learning”. Jul. 2022. arXiv:2007.05558 [cs, stat].
- [25] SCHWARTZ, R., DODGE, J., SMITH, N. A., et al. “Green AI”, *Commun. ACM*, v. 63, n. 12, pp. 54–63, Nov. 2020. ISSN: 0001-0782, 1557-7317. doi: 10.1145/3381831.
- [26] KRIEGE, N. M., JOHANSSON, F. D., MORRIS, C. “A survey on graph kernels”, *Appl Netw Sci*, v. 5, n. 1, pp. 6, Dec. 2020. ISSN: 2364-8228. doi: 10.1007/s41109-019-0195-3.

- [27] CAI, H., ZHENG, V. W., CHANG, K. C.-C. “A comprehensive survey of graph embedding: problems, techniques, and applications”, *IEEE Trans. Knowl. Data Eng.*, v. 30, n. 9, pp. 1616–1637, Sep. 2018. ISSN: 1041-4347, 1558-2191, 2326-3865. doi: 10.1109/TKDE.2018.2807452.
- [28] ALEKSANDER, I., DE GREGORIO, M., FRANÇA, F. M. G., et al. “A brief introduction to weightless neural systems”. In: *17th European Symposium on Artificial Neural Networks, ESANN 2009, Bruges, Belgium, April 22-24, 2009*, 2009.
- [29] MIRANDA, I. D., ARORA, A., SUSSKIND, Z., et al. “LogicWiSARD: memoryless synthesis of weightless neural networks”. In: *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 19–26, Gothenburg, Sweden, Jul. 2022. IEEE. ISBN: 978-1-6654-8308-7. doi: 10.1109/ASAP54787.2022.00014.
- [30] FILHO, A. S. L., GUARISA, G. P., FILHO, L. A. D. L., et al. “Wisardpkg – a library for WiSARD-based models”. May 2020. arXiv:2005.00887 [cs].
- [31] FRANÇA, F. M. G., DE GREGORIO, M., LIMA, P. M. V., et al. “Advances in weightless neural systems”. In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pp. 497–504, 2014.
- [32] FILHO, A. L., GUARISA, G. P., FILHO, L. L., et al. “Interpretation of model agnostic classifiers via local mental images”. In: *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2020, Bruges, Belgium, October 2-4, 2020*, pp. 37–42, 2020.
- [33] SUSSKIND, Z., ARORA, A., MIRANDA, I. D. S., et al. “Weightless neural networks for efficient edge inference”. In: *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pp. 279–290, Chicago Illinois, Oct. 2022. ACM. ISBN: 978-1-4503-9868-8. doi: 10.1145/3559009.3569680.
- [34] BARBASTEFANO, R. G., DE ARAUJO CARVALHO, D. M., LIPPI, M. C. “Process mining classification with a weightless neural network”. In: Thomé, A. M. T., Barbastefano, R. G., Scavarda, L. F., et al. (Eds.), *Industrial Engineering and Operations Management*, pp. 349–356, Cham, 2020. Springer International Publishing. ISBN: 978-3-030-56920-4. doi: 10.1007/978-3-030-56920-4_28.

- [35] BERLINGERIO, M., KOUTRA, D., ELIASSI-RAD, T., et al. “NetSimile: A scalable approach to size-independent network similarity”. Sep. 2012. arXiv:1209.2684 [physics, stat].
- [36] BERLINGERIO, M., KOUTRA, D., ELIASSI-RAD, T., et al. “Network similarity via multiple social theories”. In: *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 1439–1440, Niagara Ontario Canada, Aug. 2013. ACM. ISBN: 978-1-4503-2240-9. doi: 10.1145/2492517.2492582.
- [37] DE LARA, N., PINEAU, E. “A simple baseline algorithm for graph classification”. In: *Relational Representation Learning Workshops (NIPS 2018)*, Nov. 2018.
- [38] TSITSULIN, A., MUNKHOEVA, M., PEROZZI, B. “Just SLaQ when you approximate: accurate spectral distances for web-scale graphs”, *Proceedings of The Web Conference 2020*, pp. 2697–2703, Apr. 2020. doi: 10.1145/3366423.3380026. arXiv: 2003.01282.
- [39] TSITSULIN, A., MOTTIN, D., KARRAS, P., et al. “NetLSD: hearing the shape of a graph”, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2347–2356, Jul. 2018. doi: 10.1145/3219819.3219991. arXiv: 1805.10712.
- [40] VERMA, S., ZHANG, Z.-L. “Hunt for the unique, stable, sparse and fast feature learning on graphs”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, pp. 87–97. Curran Associates Inc., 2017. ISBN: 978-1-5108-6096-4.
- [41] LI, G., SEMERCI, M., YENER, B., et al. “Effective graph classification based on topological and label attributes”, *Statistical Analy Data Mining*, v. 5, n. 4, pp. 265–283, Aug. 2012. ISSN: 19321864. doi: 10.1002/sam.11153.
- [42] BOHNSACK, K. S., KADEN, M., VOIGT, J., et al. “Efficient classification learning of biochemical structured data by means of relevance weighting for sensoric response features”. In: *ESANN 2022 proceedings*, pp. 445–450, Bruges (Belgium) and online event, 2022. Ciaco - i6doc.com. ISBN: 978-2-87587-084-1. doi: 10.14428/esann/2022.ES2022-36.
- [43] NAGY, M., MOLONTAY, R. “Network classification based structural analysis of real networks and their model-generated counterparts”. Apr. 2022. arXiv: 1810.08498 [physics].

- [44] CAI, C., WANG, Y. “A simple yet effective baseline for non-attributed graph classification”. May 2022. arXiv:1811.03508 [cs, stat].
- [45] ALIAKBARY, S., HABIBI, J., MOVAGHAR, A. “Quantification and comparison of degree distributions in complex networks”. In: *7th International Symposium on Telecommunications (IST’2014)*, pp. 464–469. IEEE, Sep. 2014. ISBN: 978-1-4799-5359-2 978-1-4799-5358-5. doi: 10.1109/ISTEL.2014.7000748.
- [46] LI, Z.-P., WANG, S.-G., ZHANG, Q.-H., et al. “Graph pooling for graph-level representation learning: a survey”, *Artif Intell Rev*, v. 58, n. 2, pp. 45, Dec. 2024. ISSN: 1573-7462. doi: 10.1007/s10462-024-10949-2. Available at: <<https://link.springer.com/10.1007/s10462-024-10949-2>>.
- [47] VINYALS, O., BENGIO, S., KUDLUR, M. “Order matters: sequence to sequence for sets”. Feb. 2016. Available at: <<http://arxiv.org/abs/1511.06391>>. arXiv:1511.06391 [stat].
- [48] KNYAZEVA, B., TAYLOR, G. W., AMER, M. R. “Understanding attention and generalization in graph neural networks”. Oct. 2019. Available at: <<http://arxiv.org/abs/1905.02850>>. arXiv:1905.02850 [cs].
- [49] LEE, J., LEE, I., KANG, J. “Self-attention graph pooling”. Jun. 2019. Available at: <<http://arxiv.org/abs/1904.08082>>. arXiv:1904.08082 [cs].
- [50] NAVARIN, N., TRAN, D. V., SPERDUTI, A. “Universal readout for graph convolutional neural networks”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, Budapest, Hungary, Jul. 2019. IEEE. ISBN: 978-1-7281-1985-4. doi: 10.1109/IJCNN.2019.8852103.
- [51] ZAHEER, M., KOTTUR, S., RAVANBAKHSI, S., et al. “Deep sets”. Apr. 2018. Available at: <<http://arxiv.org/abs/1703.06114>>. arXiv:1703.06114 [cs].
- [52] BAEK, J., KANG, M., HWANG, S. J. “Accurate Learning of Graph Representations with Graph Multiset Pooling”. Jun. 2021. Available at: <<http://arxiv.org/abs/2102.11533>>. arXiv:2102.11533 [cs].
- [53] BUTERZ, D., JANET, J. P., KIDDLE, S. J., et al. “Graph neural networks with adaptive readouts”. Nov. 2022. Available at: <<http://arxiv.org/abs/2211.04952>>. arXiv:2211.04952 [cs].
- [54] BACELLAR, A. T. L., SUSSKIND, Z., BRETERNITZ JR, M., et al. “Differentiable weightless neural networks”. In: Salakhutdinov, R., Kolter, Z.,

Heller, K., et al. (Eds.), *Proceedings of the 41st International Conference on Machine Learning*, v. 235, *Proceedings of Machine Learning Research*, pp. 2277–2295. PMLR, Jul. 2024.

- [55] BISHOP, C. M. *Pattern recognition and machine learning*. Information science and statistics. New York, Springer, 2006. ISBN: 9780387310732.
- [56] CARDOSO, D. D. O. *Rejection-oriented learning without complete class information*. Ph.D. Thesis, Federal University of Rio de Janeiro, Mar. 2017. Available at: <<http://pantheon.ufrj.br/handle/11422/10186>>.
- [57] MCCULLOCH, W. S., PITTS, W. “A logical calculus of the ideas immanent in nervous activity”, *The bulletin of mathematical biophysics*, v. 5, pp. 115–133, 1943.
- [58] ROSENBLATT, F. “Principles of neurodynamics: Perceptrons and the theory of brain mechanisms”. 1962. Available at: <<https://books.google.com.br/books?id=7FhRAAAAMAAJ>>.
- [59] ABU-MOSTAFA, Y. S., MAGDON-ISMAIL, M., LIN, H.-T. *Learning from data: a short course*. S.l., AMLbook.com, 2012. ISBN: 978-1-60049-006-4.
- [60] LLOYD, S. “Least squares quantization in PCM”, *IEEE transactions on information theory*, v. 28, n. 2, pp. 129–137, 1982.
- [61] ARTHUR, D., VASSILVITSKII, S. “k-means++: the advantages of careful seeding”. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pp. 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. ISBN: 9780898716245. event-place: New Orleans, Louisiana.
- [62] ARTHUR, D., VASSILVITSKII, S. “How slow is the k -means method?” In: *Proceedings of the twenty-second annual symposium on Computational geometry*, pp. 144–153, Sedona Arizona USA, Jun. 2006. ACM. ISBN: 978-1-59593-340-9. doi: 10.1145/1137856.1137880. Available at: <<https://dl.acm.org/doi/10.1145/1137856.1137880>>.
- [63] ARAÚJO, L. S. D. “On optimization of hardware-assisted security”, 2019.
- [64] BLEDSOE, W. W., BROWNING, I. “Pattern recognition and reading by machine”. In: *Proceedings of Eastern Joint Computer Conference*, pp. 225–232, 1959.

- [65] LUSQUINO FILHO, L. A., OLIVEIRA, L. F., FILHO, A. L., et al. “Extending the weightless WiSARD classifier for regression”, *Neurocomputing*, v. 416, pp. 280–291, Nov. 2020. ISSN: 09252312. doi: 10.1016/j.neucom.2019.12.134.
- [66] SUSSKIND, Z., ARORA, A., BACELLAR, A. T. L., et al. “An FPGA-based weightless neural network for edge network intrusion detection”. In: *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 232–232, Monterey CA USA, Feb. 2023. ACM. ISBN: 978-1-4503-9417-8. doi: 10.1145/3543622.3573140.
- [67] MIRANDA, I. D. S., ARORA, A., SUSSKIND, Z., et al. “COIN: combinational intelligent networks”. In: *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 27–28, Porto, Portugal, Jul. 2023. IEEE. ISBN: 979-8-3503-4685-5. doi: 10.1109/ASAP57973.2023.00016.
- [68] SUSSKIND, Z., ARORA, A., MIRANDA, I. D. S., et al. “ULEEN: A novel architecture for ultra-low-energy edge neural networks”, *ACM Trans. Archit. Code Optim.*, v. 20, n. 4, pp. 61:1–61:24, Dec. 2023. ISSN: 1544-3566. doi: 10.1145/3629522.
- [69] BLEDSOE, W. W., BISSON, C. L. “Improved Memory Matrices for the n-Tuple Pattern Recognition Method”, *IRE Transactions on Electronic Computers*, v. EC-11, n. 3, pp. 414–415, Jun. 1962. ISSN: 0367-9950. doi: 10.1109/IRETELC.1962.5407930. Available at: <<http://ieeexplore.ieee.org/document/5407930/>>.
- [70] GRIECO, B. P., LIMA, P. M., DE GREGORIO, M., et al. “Producing pattern examples from “mental” images”, *Neurocomputing*, v. 73, n. 7-9, pp. 1057–1064, Mar. 2010. ISSN: 09252312. doi: 10.1016/j.neucom.2009.11.015.
- [71] CARVALHO, D. S., CARNEIRO, H. C. C., FRANÇA, F. M. G., et al. “B-bleaching: Agile overtraining avoidance in the WiSARD weightless neural classifier”. In: *21st european symposium on artificial neural networks, ESANN 2013, bruges, belgium, april 24-26, 2013*, 2013. Available at: <<https://www.esann.org/sites/default/files/proceedings/legacy/es2013-22.pdf>>. tex.bibsource: dblp computer science bibliography, <https://dblp.org> tex.timestamp: Tue, 02 Aug 2022 16:46:01 +0200.
- [72] CARDOSO, D. O., CARVALHO, D. S., ALVES, D. S., et al. “Financial credit analysis via a clustering weightless neural classifier”, *Neu-*

rocomputing, v. 183, pp. 70–78, Mar. 2016. ISSN: 09252312. doi: 10.1016/j.neucom.2015.06.105.

- [73] WICKERT, I., FRANÇA, F. M. G. “AUTOWISARD: unsupervised modes for the wisard”. In: *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, v. 2084, Springer Berlin Heidelberg, pp. 435–441, Berlin, Heidelberg, 2001. ISBN: 978-3-540-42235-8 978-3-540-45720-6. doi: 10.1007/3-540-45720-8_51.
- [74] SANTIAGO, L., VERONA, L., RANGEL, F., et al. “Weightless neural networks as memory segmented bloom filters”, *Neurocomputing*, v. 416, pp. 292–304, Nov. 2020. ISSN: 09252312. doi: 10.1016/j.neucom.2020.01.115.
- [75] BENGIO, Y., LÉONARD, N., COURVILLE, A. “Estimating or propagating gradients through stochastic neurons for conditional computation”. Aug. 2013. Available at: <<http://arxiv.org/abs/1308.3432>>. arXiv:1308.3432 [cs].
- [76] KAPPAUN, A., CAMARGO, K., RANGEL, F., et al. “Evaluating binary encoding techniques for WiSARD”. In: *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pp. 103–108, Recife, Brazil, Oct. 2016. IEEE. ISBN: 978-1-5090-3566-3. doi: 10.1109/BRACIS.2016.029.
- [77] BARBOSA, R., CARDOSO, D. O., CARVALHO, D., et al. “Weightless neuro-symbolic GPS trajectory classification”, *Neurocomputing*, v. 298, pp. 100–108, Jul. 2018. ISSN: 09252312. doi: 10.1016/j.neucom.2017.11.075.
- [78] BACELLAR, A., SUSSKIND, Z., A. Q. VILLON, L., et al. “Distributive thermometer: a new unary encoding for weightless neural networks”. In: *ESANN 2022 proceedings*, pp. 31–36, Bruges (Belgium) and online event, 2022. Ciaco - i6doc.com. ISBN: 978-2-87587-084-1. doi: 10.14428/esann/2022.ES2022-94. Available at: <<https://www.esann.org/sites/default/files/proceedings/2022/ES2022-94.pdf>>.
- [79] CARNEIRO, H. C. C., FRANÇA, F. M. G., LIMA, P. M. V. “Multilingual part-of-speech tagging with weightless neural networks”, *Neural Networks*, v. 66, pp. 11–21, 2015. ISSN: 0893-6080. doi: 10.1016/j.neunet.2015.02.012.
- [80] GARCIA, S., LUENGO, J., SÁEZ, J. A., et al. “A survey of discretization techniques: taxonomy and empirical analysis in supervised learning”, *IEEE*

Trans. Knowl. Data Eng., v. 25, n. 4, pp. 734–750, Apr. 2013. ISSN: 1041-4347. doi: 10.1109/TKDE.2012.35.

- [81] XAVIER, P., GREGORIO, M. D., FRANÇA, F. M. G., et al. “Detection of elementary particles with the WiSARD n-tuple classifier”. In: *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2020, Bruges, Belgium, October 2-4, 2020*, pp. 643–648, 2020.
- [82] SOUZA, D. F. P., FRANCA, F. M. G., LIMA, P. M. V. “Spatio-temporal pattern classification with KernelCanvas and WiSARD”. In: *2014 Brazilian Conference on Intelligent Systems*, pp. 228–233, Sao Paulo, Brazil, Oct. 2014. IEEE. doi: 10.1109/BRACIS.2014.49.
- [83] BENTLEY, J. L. “Multidimensional binary search trees used for associative searching”, *Commun. ACM*, v. 18, n. 9, pp. 509–517, Sep. 1975. ISSN: 0001-0782, 1557-7317. doi: 10.1145/361002.361007. Available at: <<https://dl.acm.org/doi/10.1145/361002.361007>>.
- [84] HARRIS, D. M. *Digital design and computer architecture*. 2nd ed ed. Waltham, MA, Morgan Kaufmann, 2013. ISBN: 978-0-12-394424-5 978-0-12-397816-5.
- [85] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., et al. “Scikit-learn: Machine learning in Python”, *Journal of Machine Learning Research*, v. 12, pp. 2825–2830, 2011.
- [86] GILBERT, E. N. “Gray codes and paths on the n-cube”, *Bell System Technical Journal*, v. 37, n. 3, pp. 815–826, May 1958. ISSN: 00058580. doi: 10.1002/j.1538-7305.1958.tb03887.x. Available at: <<https://ieeexplore.ieee.org/document/6773372>>.
- [87] WHEALTON, S. “The reflected binary gray code transform”. In: Sarhangi, R., Séquin, C. (Eds.), *Bridges: Mathematical Connections in Art, Music, and Science*, pp. 149–156, Southwestern College, Winfield, Kansas, 2004. Bridges Conference. ISBN: 0-9665201-5-7. Available at: <<http://archive.bridgesmathart.org/2004/bridges2004-149.html>>. ISSN: 1099-6702.
- [88] ALBUS, J. S. “A new approach to manipulator control: the cerebellar model articulation controller (CMAC)”, *Journal of Dynamic Systems, Measurement, and Control*, v. 97, n. 3, pp. 220–227, Sep. 1975. ISSN: 0022-0434, 1528-9028. doi: 10.1115/1.3426922. Available at: <<https://>>

[//asmedigitalcollection.asme.org/dynamicsystems/article/97/3/220/400927/A-New-Approach-to-Manipulator-Control-The](https://asmedigitalcollection.asme.org/dynamicsystems/article/97/3/220/400927/A-New-Approach-to-Manipulator-Control-The)>.

- [89] KOLCZ, A., ALLINSON, N. “Application of the CMAC input encoding scheme in the N-tuple approximation network”, *IEEE Proceedings-Computers and Digital Techniques*, v. 141, n. 3, pp. 177–183, 1994. Publisher: IET.
- [90] BISHOP, J., MINCHINTON, P., MITCHELL, R. “Real time invariant grey level image processing using digital neural networks”. In: *Proc. IMechE Conf., EuroDirect ‘91-Computers in the Engineering Industry*, pp. 187–189, 1991.
- [91] HAMILTON, W. L. *Graph representation learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Cham, Springer International Publishing, 2020. ISBN: 978-3-031-00460-5 978-3-031-01588-5.
- [92] MURPHY, R., SRINIVASAN, B., RAO, V., et al. “Relational pooling for graph representations”. In: Chaudhuri, K., Salakhutdinov, R. (Eds.), *Proceedings of the 36th International Conference on Machine Learning*, v. 97, *Proceedings of Machine Learning Research*, pp. 4663–4673. PMLR, Jun. 2019. Available at: <<https://proceedings.mlr.press/v97/murphy19a.html>>.
- [93] GOYAL, P., FERRARA, E. “Graph embedding techniques, applications, and performance: A survey”, *Knowledge-Based Systems*, v. 151, pp. 78–94, Jul. 2018. ISSN: 09507051. doi: 10.1016/j.knosys.2018.03.022.
- [94] AVELAR, P., LEMOS, H., PRATES, M., et al. “Multitask learning on graph neural networks: Learning multiple graph centrality measures with a unified network”. In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Workshop and Special Sessions*, v. 11731, pp. 701–715, Cham, 2019. Springer International Publishing. ISBN: 978-3-030-30492-8 978-3-030-30493-5. doi: 10.1007/978-3-030-30493-5_63. Available at: <http://link.springer.com/10.1007/978-3-030-30493-5_63>.
- [95] WAN, Z., MAHAJAN, Y., KANG, B. W., et al. “A survey on centrality metrics and their network resilience analysis”, *IEEE Access*, v. 9, pp. 104773–104819, 2021. ISSN: 2169-3536. doi: 10.1109/ACCESS.2021.3094196.
- [96] KAMISIŃSKI, A., CHOŁDA, P., JAJSZCZYK, A. “Assessing the structural complexity of computer and communication networks”, *ACM Comput. Surv.*, v. 47, n. 4, pp. 1–36, Jul. 2015. ISSN: 0360-0300, 1557-7341. doi: 10.1145/2755621.

- [97] BOLDI, P., VIGNA, S. “Axioms for centrality”, *Internet Mathematics*, v. 10, pp. 222 – 262, 2013.
- [98] PAGE, L., BRIN, S., MOTWANI, R., et al. “The PageRank citation ranking: Bringing order to the web”. In: *The Web Conference*, 1999.
- [99] VARGAS, B. *Exploring pagerank algorithms: power iteration & Monte Carlo methods*. Dissertation, California State University, San Marcos, 2020.
- [100] WATTS, D. J., STROGATZ, S. H. “Collective dynamics of ‘small-world’ networks”, *Nature*, v. 393, n. 6684, pp. 440–442, Jun. 1998. ISSN: 0028-0836, 1476-4687. doi: 10.1038/30918. Available at: <<https://www.nature.com/articles/30918>>.
- [101] HÉBERT-DUFRESNE, L., GROCHOW, J. A., ALLARD, A. “Multi-scale structure and topological anomaly detection via a new network statistic: The onion decomposition”, *Sci Rep*, v. 6, n. 1, pp. 31708, Aug. 2016. ISSN: 2045-2322. doi: 10.1038/srep31708.
- [102] HEARST, M., DUMAIS, S., OSUNA, E., et al. “Support vector machines”, *IEEE Intell. Syst. Their Appl.*, v. 13, n. 4, pp. 18–28, Jul. 1998. ISSN: 1094-7167. doi: 10.1109/5254.708428.
- [103] SHERVASHIDZE, N., SCHWEITZER, P., VAN LEEUWEN, E. J., et al. “Weisfeiler-Lehman graph kernels”, *J. Mach. Learn. Res.*, v. 12, n. null, pp. 2539–2561, Nov. 2011. ISSN: 1532-4435.
- [104] SUGIYAMA, M., BORGWARDT, K. “Halting in random walk kernels”. In: Cortes, C., Lawrence, N., Lee, D., et al. (Eds.), *Advances in Neural Information Processing Systems*, v. 28. Curran Associates, Inc., 2015.
- [105] NIKOLENTZOS, G., SIGLIDIS, G., VAZIRGIANNIS, M. “Graph kernels: A survey”, *jair*, v. 72, pp. 943–1027, Nov. 2021. ISSN: 1076-9757. doi: 10.1613/jair.1.13225. Available at: <<http://jair.org/index.php/jair/article/view/13225>>.
- [106] IZENMAN, A. J. “Introduction to manifold learning”, *WIREs Computational Stats*, v. 4, n. 5, pp. 439–446, Sep. 2012. ISSN: 1939-5108, 1939-0068. doi: 10.1002/wics.1222.
- [107] VON LUXBURG, U. “A tutorial on spectral clustering”, *Stat Comput*, v. 17, n. 4, pp. 395–416, Dec. 2007. ISSN: 0960-3174, 1573-1375. doi: 10.1007/s11222-007-9033-z.

- [108] GAO, F., WOLF, G., HIRN, M. “Geometric scattering for graph data analysis”. In: Chaudhuri, K., Salakhutdinov, R. (Eds.), *Proceedings of the 36th International Conference on Machine Learning*, v. 97, *Proceedings of Machine Learning Research*, pp. 2122–2131. PMLR, Jun. 2019. Available at: <<https://proceedings.mlr.press/v97/gao19e.html>>.
- [109] NARAYANAN, A., CHANDRAMOHAN, M., VENKATESAN, R., et al. “Graph2vec: learning distributed representations of graphs”. Jul. 2017. Available at: <<http://arxiv.org/abs/1707.05005>>. Issue: arXiv:1707.05005 arXiv: 1707.05005 [cs].
- [110] LE, Q., MIKOLOV, T. “Distributed representations of sentences and documents”. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML’14, pp. II–1188–II–1196, Beijing, China, 2014. JMLR.org.
- [111] CHEN, H., KOGA, H. “GL2vec: graph embedding enriched by line graphs with edge features”. In: Gedeon, T., Wong, K. W., Lee, M. (Eds.), *Neural Information Processing*, v. 11955, Springer International Publishing, pp. 3–14, Cham, 2019. ISBN: 978-3-030-36717-6 978-3-030-36718-3. doi: 10.1007/978-3-030-36718-3_1. Available at: <http://link.springer.com/10.1007/978-3-030-36718-3_1>. Series Title: Lecture Notes in Computer Science.
- [112] ROZEMBERCZKI, B., SARKAR, R. “Characteristic functions on graphs: birds of a feather, from statistical descriptors to parametric models”. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 1325–1334, Virtual Event Ireland, Oct. 2020. ACM. ISBN: 978-1-4503-6859-9. doi: 10.1145/3340531.3411866. Available at: <<https://dl.acm.org/doi/10.1145/3340531.3411866>>.
- [113] WANG, L., HUANG, C., MA, W., et al. “Graph embedding via diffusion-wavelets-based node feature distribution characterization”. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3478–3482, Oct. 2021. doi: 10.1145/3459637.3482115.
- [114] PEROZZI, B., AL-RFOU, R., SKIENA, S. “DeepWalk: online learning of social representations”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, New York New York USA, Aug. 2014. ACM. ISBN: 978-1-4503-2956-9. doi: 10.1145/2623330.2623732.

- [115] GROVER, A., LESKOVEC, J. “Node2vec: Scalable feature learning for networks”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864, San Francisco California USA, Aug. 2016. ACM. ISBN: 978-1-4503-4232-2. doi: 10.1145/2939672.2939754.
- [116] RIBEIRO, L. F., SAVERESE, P. H., FIGUEIREDO, D. R. “Struc2vec: Learning node representations from structural identity”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 385–394, Halifax NS Canada, Aug. 2017. ACM. ISBN: 978-1-4503-4887-4. doi: 10.1145/3097983.3098061.
- [117] HAMILTON, W. L., YING, R., LESKOVEC, J. “Inductive representation learning on large graphs”. Sep. 2018. Available at: <<http://arxiv.org/abs/1706.02216>>. arXiv:1706.02216 [cs, stat].
- [118] NAIR, V., HINTON, G. E. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML’10*, pp. 807–814, Madison, WI, USA, 2010. Omnipress. ISBN: 978-1-60558-907-7. event-place: Haifa, Israel.
- [119] HOCHREITER, S., SCHMIDHUBER, J. “Long short-term memory”, *Neural Computation*, v. 9, n. 8, pp. 1735–1780, Nov. 1997. ISSN: 0899-7667, 1530-888X. doi: 10.1162/neco.1997.9.8.1735. Available at: <<https://direct.mit.edu/neco/article/9/8/1735-1780/6109>>.
- [120] BINKOWSKI, J., SAWCZYN, A., JANIĄK, D., et al. “Graph-level representations using ensemble-based readout functions”. Apr. 2023. Available at: <<http://arxiv.org/abs/2303.02023>>. arXiv:2303.02023 [cs].
- [121] CORSO, G., CAVALLERI, L., BEAINI, D., et al. “Principal neighbourhood aggregation for graph nets”. Dec. 2020. Available at: <<http://arxiv.org/abs/2004.05718>>. arXiv:2004.05718 [cs].
- [122] ZHANG, M., CUI, Z., NEUMANN, M., et al. “An end-to-end deep learning architecture for graph classification”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI’18/IAAI’18/EAAI’18*, New Orleans, Louisiana, USA, 2018. AAAI Press. ISBN: 978-1-57735-800-8.

- [123] GAO, H., JI, S. “Graph U-Nets”. May 2019. Available at: <<http://arxiv.org/abs/1905.05178>>. arXiv:1905.05178 [cs].
- [124] WANG, P., LUO, J., SHEN, Y., et al. “A comprehensive graph pooling benchmark: effectiveness, robustness and generalizability”. Oct. 2024. Available at: <<http://arxiv.org/abs/2406.09031>>. arXiv:2406.09031 [cs].
- [125] GRAY, R., NEUHOFF, D. “Quantization”, *IEEE Trans. Inform. Theory*, v. 44, n. 6, pp. 2325–2383, Oct. 1998. ISSN: 00189448. doi: 10.1109/18.720541. Available at: <<http://ieeexplore.ieee.org/document/720541/>>.
- [126] WANG, H., SONG, M. “Ckmeans.1d.dp: optimal k-means clustering in one dimension by dynamic programming.” *R J*, v. 3, n. 2, pp. 29–33, Dec. 2011. ISSN: 2073-4859. Place: United States.
- [127] JENKS, G., GEOGRAPHY, U. O. K. D. O. “Optimal data classification for choropleth maps”. 1977. Available at: <<https://books.google.com.br/books?id=HvAENQAACAAJ>>.
- [128] ROZEMBERCZKI, B., KISS, O., SARKAR, R. “Karate club: an API oriented open-source python framework for unsupervised learning on graphs”. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, pp. 3125–3132. ACM, 2020.
- [129] MORRIS, C., KRIEGE, N. M., BAUSE, F., et al. “TUDataset: A collection of benchmark datasets for learning with graphs”. In: *ICML 2020 workshop on graph representation learning and beyond (GRL+ 2020)*, 2020.
- [130] YANARDAG, P., VISHWANATHAN, S. “Deep graph kernels”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374, Sydney NSW Australia, Aug. 2015. ACM. ISBN: 978-1-4503-3664-2. doi: 10.1145/2783258.2783417.
- [131] SCHOMBURG, I. “BRENDA, the enzyme database: updates and major new developments”, *Nucleic Acids Research*, v. 32, n. 90001, pp. 431D–433, Jan. 2004. ISSN: 1362-4962. doi: 10.1093/nar/gkh081.
- [132] KRIEGE, N., MUTZEL, P. “Subgraph matching kernels for attributed graphs”. Jun. 2012. arXiv:1206.6483 [cs, stat].
- [133] KIM, S., CHEN, J., CHENG, T., et al. “PubChem 2023 update”, *Nucleic Acids Research*, v. 51, n. D1, pp. D1373–D1380, Jan. 2023. ISSN: 0305-1048, 1362-4962. doi: 10.1093/nar/gkac956.

- [134] DOBSON, P. D., DOIG, A. J. “Distinguishing enzyme structures from non-enzymes without alignments”, *Journal of Molecular Biology*, v. 330, n. 4, pp. 771–783, Jul. 2003. ISSN: 00222836. doi: 10.1016/S0022-2836(03)00628-4.
- [135] ERRICA, F., PODDA, M., BACCIU, D., et al. “A fair comparison of graph neural networks for graph classification”. 2022. arXiv:1912.09893 [cs.LG].
- [136] HAGBERG, A. A., SCHULT, D. A., SWART, P. J. “Exploring network structure, dynamics, and function using NetworkX”. In: *Proceedings of the 7th python in science conference*, pp. 11 – 15, Pasadena, CA USA, 2008.
- [137] FEY, M., LENSSEN, J. E. “Fast graph representation learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
- [138] SIGLIDIS, G., NIKOLENTZOS, G., LIMNIOS, S., et al. “GraKeL: A graph kernel library in python”, *Journal of Machine Learning Research*, v. 21, n. 54, pp. 1–5, 2020.
- [139] ROSSI, R. A., AHMED, N. K. “Complex networks are structurally distinguishable by domain”, *Soc. Netw. Anal. Min.*, v. 9, n. 1, pp. 51, Sep. 2019. ISSN: 1869-5469. doi: 10.1007/s13278-019-0593-7. Available at: <<https://doi.org/10.1007/s13278-019-0593-7>>.
- [140] WELKE, P., SCHULZ, T. H. “On the necessity of graph kernel baselines”. In: *Graph Embedding and Mining Workshop at ECML PKDD*, 2019.
- [141] DAVID, F. N., JOHNSON, N. L. “The probability integral transformation when parameters are estimated from the sample”, *Biometrika*, v. 35, n. 1/2, pp. 182–190, 1948. ISSN: 0006-3444. doi: 10.2307/2332638.
- [142] NIKOLENTZOS, G., CHATZIANASTASIS, M., VAZIRGIANNIS, M. “Weisfeiler and Leman go hyperbolic: Learning distance preserving node representations”. In: Ruiz, F., Dy, J., van de Meent, J.-W. (Eds.), *Proceedings of the 26th international conference on artificial intelligence and statistics*, v. 206, *Proceedings of machine learning research*, pp. 1037–1054. PMLR, Apr. 2023.
- [143] YANG, C., WU, Q., WANG, J., et al. “Graph neural networks are inherently good generalizers: insights by bridging gnns and mlps”. In: *The Eleventh International Conference on Learning Representations*, 2023. Available at: <<https://openreview.net/forum?id=dqnNW2omZL6>>.

- [144] LIU, R., CANTÜRK, S., WENKEL, F., et al. “Taxonomy of benchmarks in graph representation learning”. In: Rieck, B., Pascanu, R. (Eds.), *Proceedings of the first learning on graphs conference*, v. 198, *Proceedings of machine learning research*, pp. 6:1–6:25. PMLR, Dec. 2022.
- [145] CODA, G., DE GREGORIO, M., SORGENTE, A., et al. “Improving the DRASiW performance by exploiting its own "Mental Images"”. In: *ESANN 2023 proceedings*, pp. 363–368, Bruges (Belgium) and online, 2023. Ciaco - i6doc.com. ISBN: 978-2-87587-088-9. doi: 10.14428/esann/2023.ES2023-25. Available at: <<https://www.esann.org/sites/default/files/proceedings/2023/ES2023-25.pdf>>.
- [146] VIRTANEN, P., GOMMERS, R., OLIPHANT, T. E., et al. “SciPy 1.0: fundamental algorithms for scientific computing in python”, *Nature Methods*, v. 17, pp. 261–272, 2020. doi: 10.1038/s41592-019-0686-2.
- [147] KARNIN, Z., LANG, K., LIBERTY, E. “Optimal quantile approximation in streams”. Apr. 2016. Available at: <<http://arxiv.org/abs/1603.05346>>. arXiv:1603.05346 [cs].
- [148] HALKO, N., MARTINSSON, P. G., TROPP, J. A. “Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions”, *SIAM Rev.*, v. 53, n. 2, pp. 217–288, Jan. 2011. ISSN: 0036-1445, 1095-7200. doi: 10.1137/090771806. Available at: <<http://epubs.siam.org/doi/10.1137/090771806>>.
- [149] LINNEBERG, C., JORGENSEN, T. M. “Discretization methods for encoding of continuous input variables for boolean neural networks”. In: *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, v. 2, pp. 1219–1224. IEEE, 1999.

Appendix A

Hyperparameter Settings

A.1 Hyperparameter Settings for the K -Means Thermometer Experiments

Table A.1 presents all hyperparameters searched in this work for the evaluated classifiers and other (hyper) parameters used for the experiments executed in Section 4. Parameters for RandomForest and NaiveBayes are the default parameters in the scikit-learn python library (version 1.1.1), except where noted.

A.2 Hyperparameter Settings for the Initial Graph Classification Architectures

Table A.3 presents all hyperparameters searched in this work for the evaluated classifiers and other (hyper) parameters used for the experiments executed in Section 5.1. ClusWiSARD parameters are the union of all WiSARD parameters plus the ones presented here. ULEEN parameters are the union of all BTHOWeN parameters plus the ones presented here. Parameters for RandomForest are the default parameters in the scikit-learn python library (version 1.1.1), except where noted.

In addition, Table A.3 presents all hyperparameters searched in this work for the evaluated classifiers and other (hyper) parameters used for the experiments executed in Section 5.2. This table adheres to the same principles mentioned above for the ClusWiSARD and RandomForest classifiers.

Table A.1: Hyperparameters search in the experiments executed in Section 4

Parameter	Description	Search Space ¹
<i>addressSize</i>	Number of bits used for an address in RAM (tupleSize)	{2, 16, 32, 48, 64}
<i>classificationMethod</i>	How ties are broken	Bleaching
<i>mappingGenerator</i>	How the inputs are broken into tuples	RandomMapping
<i>completeAddressing</i>	If address should be completed in case there is a remainder in the division	true
<i>balanced</i>	inputVectorSize/tupleSize If the number of training examples for a class should be considered when classifying	false
<i>ignoreZero</i>	If zero-only address should be discarded when classifying	false
<i>monoMapping</i>	If the mapping is shared between all classes	true
<i>base</i>	Base of input vector	2 (binary vector)
<i>n_estimators</i>	Number of trees	{10, 100, 200}
<i>criterion</i>	Function to evaluate the quality of the split	{"gini", "entropy"}
<i>max_features</i>	Maximum features used when searching for the best split	{"sqrt", "log2"}
<i>lut_size</i>	Size of the LUTs	6
<i>num_epochs</i>	Number of training epochs	10
<i>batch_size</i>	Batch size	32
<i>tau</i>	GroupSum's tau parameter	1/0.3
<i>learning_rate</i>	Optimizer learning rate	1e-2
<i>optimizer</i>	Optimizer	Adam

^aA set denotes a grid of evaluated hyperparameter values and a constant value represents a fixed hyperparameter value

Parameter	Description	Search Space ¹
<i>addressSize</i>	Number of bits used for an address in RAM (tupleSize)	{2, 16, 32, 48, 64}
<i>classificationMethod</i>	How ties are broken	Bleaching
<i>mappingGenerator</i>	How the inputs are broken into tuples	RandomMapping
<i>completeAddressing</i>	If address should be completed in case there is a remainder in the division	true
<i>balanced</i>	inputVectorSize/tupleSize If the number of training examples for a class should be considered when classifying	false
<i>ignoreZero</i>	If zero-only address should be discarded when classifying	false
<i>monoMapping</i>	If the mapping is shared between all classes	true
<i>base</i>	Base of input vector	2 (binary vector)
<i>minScore</i>	Minimum score to accept a training example	{0.5, 0.7, 0.9}
<i>discriminatorLimit</i>	Maximum number of discriminators that can be created for a single class	{1, 5, 10}
<i>unitHashes</i>	Number of hash functions	{1,2,3,4,5,6}
<i>unitEntries</i>	Number of entries per bloom filter	{128, 256, 512, 1024, 2048, 4096, 8192}
<i>addressSize</i>	Number of bits used for an address in RAM (tupleSize)	{2, 4, 8, 16, 32, 48, 64}
<i>numberClassifiers</i>	Number of classifiers in the ensemble	{1,3,5,10}
<i>numEpochs</i>	Number of training epochs	10
<i>dropoutRate</i>	Dropout rate	{0, 0.1, 0.25}
<i>learningRate</i>	Optimizer learning rate	1e-3
<i>batch_size</i>	Batch size	32
<i>optimizer</i>	Optimizer	Adam
<i>n_estimators</i>	Number of trees	{10, 100, 200}
<i>criterion</i>	Function to evaluate the quality of the split	{"gini", "entropy"}
<i>max_features</i>	Maximum features used when searching for the best split	{"sqrt", "log2"}

Table A.2: Hyperparameters search in the experiments executed in Section 5.1

^aA set denotes a grid of evaluated hyperparameter values and a constant value represents a fixed hyperparameter value

Parameter	Description	Search Space ¹
WISARD parameters	<i>addressSize</i>	{2, 16, 32, 48, 64}
	<i>classificationMethod</i>	Bleaching
	<i>mappingGenerator</i>	RandomMapping
	<i>completeAddressing</i>	true
	<i>balanced</i>	false
	<i>ignoreZero</i>	false
ClusWISARD parameters	<i>monoMapping</i>	true
	<i>base</i>	2 (binary vector)
ClusWISARD parameters	<i>minScore</i>	{0.5, 0.7, 0.9}
	<i>discriminatorsLimit</i>	{1, 5, 10}
RandomForest parameters	<i>n_estimators</i>	{10, 100, 200}
	<i>criterion</i>	{"gini", "entropy"}
PlotBinarizer parameters	<i>max_features</i>	{"sqrt", "log2"}
	<i>width</i>	{8, 16, 24, 32, 48, 64}
	<i>height</i>	{8, 16, 24, 32, 48, 64}
	<i>xAxisLog</i>	{False, True}
	<i>yAxisLog</i>	{False, True}
	<i>numberLeaves</i>	{256, 512, 1024, 1536, 2048}
	<i>height</i>	{8, 12, 16, 20}
	<i>xAxisLog</i>	{False, True}
	<i>yAxisLog</i>	{False, True}
	<i>cutStrategy</i>	{"median", "mean", "half"}
KDTreeBinarizer parameters	<i>priorityPolicy</i>	{"size", "density", "std"}
	<i>numberKernels</i>	{256, 512, 1024, 1536, 2048}
KernelCanvas parameters	<i>bitsPerKernel</i>	{1, 2, 4, 8}
	<i>numberBins</i>	{256, 512, 1024, 1536, 2048}
HistogramBinarizer parameters	<i>bitsPerBin</i>	{1, 2, 4, 8}
	<i>bitBudget</i>	{1024, 2048}

Table A.3: Hyperparameters search in the experiments executed in Section 5.2

^aA set denotes a grid of evaluated hyperparameter values and a constant value represents a fixed hyperparameter value

A.3 Hyperparameter Settings for the KCpp Experiments

Table A.4 presents all hyperparameters searched in this work for the evaluated classifiers and other (hyper) parameters used for the experiments executed in Section 6.1.3. ClusWiSARD parameters are the union of all WiSARD parameters plus the ones presented here. Parameters for RandomForest are the default parameters in the scikit-learn python library (version 1.1.1), except where noted. Parameters for GNN models are the default in the Pytorch Geometric Library (version 2.6.1), except where noted.

Parameter	Description	Search Space ¹
WisARD parameters	<p><i>addressSize</i> Number of bits used for an address in RAM (tupleSize)</p> <p><i>classificationMethod</i> How ties are broken</p> <p><i>mappingGenerator</i> How the inputs are broken into tuples</p> <p><i>completeAddressing</i> If address should be completed in case there is a remainder in the division inputVectorSize/tupleSize</p> <p><i>balanced</i> If the number of training examples for a class should be considered when classifying</p> <p><i>ignoreZero</i> If zero-only address should be discarded when classifying</p> <p><i>monoMapping</i> If the mapping is shared between all classes</p> <p><i>base</i> Base of input vector</p>	<p>{2, 16, 32, 48, 64}</p> <p>Bleaching</p> <p>RandomMapping</p> <p>true</p> <p>false</p> <p>false</p> <p>true</p> <p>2 (binary vector)</p>
ClusWisARD parameters	<p><i>minScore</i> Minimum score to accept a training example</p> <p><i>threshold</i> Damping factor for training example acceptance</p> <p><i>discriminatorsLimit</i> Maximum number of discriminators that can be created for a single class</p>	<p>{0.5, 0.7, 0.9}</p> <p>{0.1,0.2,0.5,1}</p> <p>{1, 5, 10}</p>
RandomWisARD parameters	<p><i>addressSize</i> Number of bits used for an address in RAM (tupleSize)</p> <p><i>n_estimators</i> Number of estimators used in the bagging</p> <p><i>max_samples</i> Fraction of samples used to train each base estimator</p> <p><i>max_features</i> Fraction of features used in each base estimator</p>	<p>{2, 4, 8, 16, 32}</p> <p>{10, 100, 200}</p> <p>{0.1,0.5,1}</p> <p>{"log2", "sqrt", 0.1,0.5,1}</p>
BloomWisARD - BTHOWeN parameters	<p><i>unitHashes</i> Number of hash functions</p> <p><i>unitEntries</i> Number of entries per bloom filter</p> <p><i>addressSize</i> Number of bits used for an address in RAM (tupleSize)</p>	<p>{1,2,3,4,5,6}</p> <p>{128, 256, 512, 1024, 2048, 4096, 8192}</p> <p>{2, 4, 8, 16, 32,48,64}</p>
RandomForest parameters	<p><i>n_estimators</i> Number of trees</p> <p><i>criterion</i> Function the evaluate the quality of the split</p> <p><i>max_features</i> Maximum features used when searching for the best split</p>	<p>{10, 100, 200}</p> <p>{"gini", "entropy"}</p> <p>{"sqrt", "log2"}</p>
GNN parameters	<p><i>layers</i> Number of hidden layers</p> <p><i>batch_size</i> Number of graphs per iteration</p> <p><i>optimizer</i> Optimizer used during training</p> <p><i>learning_rate</i> Learning rate used for optimization</p> <p><i>hidden_units</i> Hidden units per hidden layer</p> <p><i>num_epochs</i> Number of epochs</p> <p><i>dropout</i> Dropout rate</p>	<p>{2, 3, 4}</p> <p>64</p> <p>Adam</p> <p>{1e-1,1e-3, 1e-6}</p> <p>{32,64}</p> <p>{100, 200}</p> <p>{0,0.25,0.5}</p>
Other parameters	<p><i>varianceThreshold</i> A variance threshold</p> <p><i>P</i> Maximum of combinations evaluated in a day</p> <p><i>min_info</i> Minimum β_i - amount of bits for an attribute</p> <p>α Rate of structural information</p>	<p>{None, 0}</p> <p>1000</p> <p>{64}</p> <p>{0.1,0.25,0.4,0.5,0.6,0.75,0.9}</p>

Table A.4: Hyperparameters search in the experiments executed in Section 6.1.3

^aA set denotes a grid of evaluated hyperparameter values and a constant value represents a fixed hyperparameter value

Appendix B

Full Experiment Results

B.1 Full Experiment Results - *K*-Means Thermometer

Table B.1 presents the full results for the experiments executed in Section 4.

strategy	Ecoli	EEG Eye State	Fashion MNIST	Glass	Iris	Letter	Telescope	MNIST	SatImage	Shuttle	Wine	L1-Norm	Rank
NoE+RandomForest	0.851 ± 0.042	0.935 ± 0.005	0.878 ± 0.001	0.802 ± 0.045	0.943 ± 0.032	0.964 ± 0.003	0.883 ± 0.006	0.971 ± 0.001	0.913 ± 0.006	0.999 ± 0.000	0.969 ± 0.037	0.005	5.182
KT+RandomForest	0.848 ± 0.047	0.944 ± 0.005	0.879 ± 0.001	0.783 ± 0.082	0.940 ± 0.029	0.962 ± 0.003	0.879 ± 0.005	0.973 ± 0.001	0.912 ± 0.007	0.999 ± 0.000	0.971 ± 0.023	0.007	5.455
DT+RandomForest	0.834 ± 0.040	0.873 ± 0.015	0.872 ± 0.001	0.767 ± 0.071	0.937 ± 0.029	0.960 ± 0.003	0.879 ± 0.005	0.970 ± 0.001	0.911 ± 0.007	0.998 ± 0.001	0.974 ± 0.021	0.011	8.091
CMAC+RandomForest	0.845 ± 0.050	0.875 ± 0.001	0.877 ± 0.001	0.771 ± 0.075	0.953 ± 0.032	0.968 ± 0.003	0.874 ± 0.006	0.972 ± 0.001	0.913 ± 0.007	0.999 ± 0.000	0.969 ± 0.025	0.014	5.455
GT+RandomForest	0.839 ± 0.033	0.851 ± 0.022	0.875 ± 0.001	0.767 ± 0.047	0.947 ± 0.017	0.963 ± 0.002	0.879 ± 0.005	0.973 ± 0.001	0.912 ± 0.006	0.999 ± 0.000	0.983 ± 0.015	0.016	5.727
DT+DWN	0.815 ± 0.035	0.888 ± 0.005	0.870 ± 0.003	0.767 ± 0.079	0.940 ± 0.034	0.954 ± 0.003	0.816 ± 0.010	0.980 ± 0.000	0.899 ± 0.010	0.998 ± 0.001	0.963 ± 0.036	0.025	12.000
DT+WISARD	0.819 ± 0.053	0.914 ± 0.003	0.826 ± 0.002	0.776 ± 0.079	0.937 ± 0.040	0.942 ± 0.004	0.856 ± 0.006	0.949 ± 0.001	0.899 ± 0.006	0.998 ± 0.001	0.983 ± 0.020	0.025	13.091
KT+DWN	0.834 ± 0.029	0.883 ± 0.009	0.874 ± 0.003	0.740 ± 0.072	0.943 ± 0.032	0.953 ± 0.004	0.818 ± 0.011	0.981 ± 0.001	0.896 ± 0.005	0.998 ± 0.001	0.960 ± 0.024	0.026	11.455
KT+WISARD	0.837 ± 0.046	0.913 ± 0.005	0.843 ± 0.001	0.714 ± 0.053	0.947 ± 0.023	0.944 ± 0.003	0.857 ± 0.010	0.959 ± 0.001	0.899 ± 0.009	0.998 ± 0.000	0.949 ± 0.030	0.028	13.636
OH-KT+WISARD	0.799 ± 0.062	0.764 ± 0.008	0.814 ± 0.003	0.774 ± 0.073	0.933 ± 0.042	0.784 ± 0.019	0.800 ± 0.012	0.946 ± 0.002	0.869 ± 0.008	0.995 ± 0.002	0.911 ± 0.041	0.031	14.636
LT+RandomForest	0.843 ± 0.042	0.612 ± 0.025	0.877 ± 0.002	0.674 ± 0.072	0.957 ± 0.032	0.967 ± 0.001	0.869 ± 0.005	0.972 ± 0.001	0.911 ± 0.007	0.998 ± 0.000	0.983 ± 0.015	0.037	7.455
GT+WISARD	0.834 ± 0.023	0.767 ± 0.029	0.843 ± 0.001	0.743 ± 0.051	0.950 ± 0.036	0.948 ± 0.003	0.847 ± 0.005	0.962 ± 0.001	0.900 ± 0.007	0.998 ± 0.000	0.969 ± 0.025	0.037	11.909
OH-KT+RandomForest	0.764 ± 0.053	0.855 ± 0.012	0.868 ± 0.001	0.717 ± 0.062	0.957 ± 0.032	0.912 ± 0.004	0.853 ± 0.008	0.968 ± 0.001	0.892 ± 0.009	0.998 ± 0.001	0.957 ± 0.031	0.039	15.273
GT+DWN	0.825 ± 0.043	0.721 ± 0.030	0.869 ± 0.003	0.752 ± 0.052	0.950 ± 0.032	0.953 ± 0.003	0.821 ± 0.009	0.981 ± 0.001	0.899 ± 0.011	0.997 ± 0.001	0.966 ± 0.030	0.039	11.909
CMAC+WISARD	0.855 ± 0.042	0.713 ± 0.035	0.837 ± 0.003	0.721 ± 0.076	0.953 ± 0.039	0.948 ± 0.004	0.840 ± 0.004	0.959 ± 0.002	0.898 ± 0.007	0.998 ± 0.000	0.960 ± 0.034	0.044	13.909
CMAC+DWN	0.807 ± 0.048	0.800 ± 0.030	0.870 ± 0.003	0.717 ± 0.070	0.910 ± 0.035	0.951 ± 0.004	0.812 ± 0.017	0.981 ± 0.001	0.890 ± 0.012	0.997 ± 0.004	0.934 ± 0.056	0.045	16.818
LT+WISARD	0.834 ± 0.059	0.584 ± 0.010	0.839 ± 0.001	0.700 ± 0.052	0.957 ± 0.027	0.942 ± 0.003	0.834 ± 0.003	0.961 ± 0.001	0.896 ± 0.008	0.994 ± 0.005	0.974 ± 0.025	0.059	16.545
OH-LT+RandomForest	0.801 ± 0.040	0.619 ± 0.018	0.868 ± 0.001	0.667 ± 0.059	0.940 ± 0.044	0.911 ± 0.004	0.846 ± 0.007	0.967 ± 0.001	0.892 ± 0.008	0.997 ± 0.003	0.963 ± 0.036	0.063	18.818
LT+DWN	0.816 ± 0.049	0.544 ± 0.035	0.870 ± 0.003	0.710 ± 0.074	0.923 ± 0.074	0.952 ± 0.003	0.800 ± 0.016	0.981 ± 0.001	0.896 ± 0.009	0.996 ± 0.007	0.963 ± 0.038	0.065	17.727
OH-DT+DWN	0.751 ± 0.037	0.776 ± 0.024	0.859 ± 0.003	0.679 ± 0.052	0.910 ± 0.042	0.908 ± 0.010	0.803 ± 0.013	0.977 ± 0.001	0.870 ± 0.013	0.998 ± 0.001	0.909 ± 0.044	0.066	21.727
OH-KT+DWN	0.760 ± 0.052	0.733 ± 0.018	0.861 ± 0.004	0.648 ± 0.049	0.913 ± 0.055	0.891 ± 0.006	0.787 ± 0.014	0.978 ± 0.001	0.871 ± 0.007	0.998 ± 0.001	0.900 ± 0.040	0.074	21.909
OH-DT+RandomForest	0.799 ± 0.061	0.865 ± 0.005	0.866 ± 0.001	0.745 ± 0.084	0.937 ± 0.033	0.928 ± 0.003	0.858 ± 0.005	0.966 ± 0.000	0.886 ± 0.007	0.998 ± 0.000	0.980 ± 0.027	0.080	23.636
OH-DT+WISARD	0.769 ± 0.048	0.751 ± 0.012	0.812 ± 0.002	0.629 ± 0.094	0.920 ± 0.028	0.767 ± 0.007	0.790 ± 0.010	0.931 ± 0.002	0.861 ± 0.008	0.993 ± 0.003	0.923 ± 0.066	0.093	26.364
OH-LT+DWN	0.766 ± 0.037	0.534 ± 0.041	0.860 ± 0.004	0.667 ± 0.082	0.903 ± 0.037	0.891 ± 0.006	0.771 ± 0.012	0.978 ± 0.001	0.868 ± 0.012	0.993 ± 0.007	0.906 ± 0.067	0.094	26.000
OH-LT+WISARD	0.800 ± 0.045	0.615 ± 0.016	0.803 ± 0.002	0.614 ± 0.070	0.913 ± 0.036	0.782 ± 0.011	0.802 ± 0.009	0.944 ± 0.002	0.865 ± 0.008	0.996 ± 0.003	0.931 ± 0.045	0.100	26.364
OH-KT+NaiveBayes	0.822 ± 0.052	0.682 ± 0.012	0.736 ± 0.000	0.636 ± 0.071	0.930 ± 0.037	0.726 ± 0.006	0.776 ± 0.008	0.845 ± 0.000	0.821 ± 0.012	0.936 ± 0.003	0.969 ± 0.021	0.117	25.091
OH-DT+NaiveBayes	0.803 ± 0.049	0.679 ± 0.011	0.730 ± 0.000	0.643 ± 0.079	0.910 ± 0.067	0.729 ± 0.006	0.773 ± 0.010	0.835 ± 0.000	0.821 ± 0.012	0.933 ± 0.002	0.966 ± 0.023	0.122	27.273
MC+RandomForest	0.778 ± 0.055	0.677 ± 0.007	0.875 ± 0.001	0.614 ± 0.077	0.627 ± 0.058	0.935 ± 0.003	0.712 ± 0.008	0.970 ± 0.001	0.809 ± 0.011	0.929 ± 0.002	0.854 ± 0.048	0.126	26.000
LT+NaiveBayes	0.840 ± 0.052	0.584 ± 0.012	0.709 ± 0.000	0.581 ± 0.067	0.943 ± 0.035	0.691 ± 0.007	0.786 ± 0.006	0.847 ± 0.000	0.799 ± 0.013	0.904 ± 0.013	0.971 ± 0.027	0.126	25.727
OH-LT+NaiveBayes	0.804 ± 0.049	0.585 ± 0.017	0.735 ± 0.000	0.648 ± 0.058	0.920 ± 0.028	0.726 ± 0.007	0.768 ± 0.009	0.846 ± 0.000	0.821 ± 0.013	0.922 ± 0.006	0.960 ± 0.031	0.130	28.364
KT+NaiveBayes	0.828 ± 0.046	0.617 ± 0.013	0.717 ± 0.003	0.617 ± 0.070	0.947 ± 0.023	0.677 ± 0.009	0.793 ± 0.009	0.846 ± 0.000	0.792 ± 0.010	0.857 ± 0.010	0.983 ± 0.015	0.136	24.818
MC+WISARD	0.790 ± 0.051	0.674 ± 0.009	0.840 ± 0.001	0.593 ± 0.084	0.627 ± 0.058	0.913 ± 0.004	0.710 ± 0.009	0.962 ± 0.001	0.794 ± 0.010	0.928 ± 0.003	0.843 ± 0.034	0.136	28.909
CMAC+NaiveBayes	0.840 ± 0.062	0.666 ± 0.011	0.708 ± 0.000	0.633 ± 0.062	0.940 ± 0.026	0.692 ± 0.008	0.787 ± 0.005	0.848 ± 0.000	0.796 ± 0.009	0.917 ± 0.005	0.960 ± 0.020	0.137	25.727
GT+NaiveBayes	0.827 ± 0.047	0.577 ± 0.011	0.709 ± 0.000	0.650 ± 0.078	0.933 ± 0.035	0.671 ± 0.007	0.775 ± 0.005	0.846 ± 0.001	0.788 ± 0.009	0.838 ± 0.003	0.977 ± 0.012	0.143	27.545
DT+NaiveBayes	0.806 ± 0.047	0.603 ± 0.013	0.671 ± 0.000	0.605 ± 0.071	0.937 ± 0.029	0.658 ± 0.008	0.785 ± 0.009	0.837 ± 0.000	0.779 ± 0.007	0.795 ± 0.006	0.986 ± 0.015	0.155	28.636
MC+DWN	0.757 ± 0.043	0.557 ± 0.049	0.879 ± 0.002	0.574 ± 0.079	0.660 ± 0.086	0.922 ± 0.005	0.643 ± 0.029	0.978 ± 0.001	0.763 ± 0.015	0.909 ± 0.011	0.794 ± 0.046	0.158	28.545
MC+NaiveBayes	0.743 ± 0.029	0.622 ± 0.009	0.720 ± 0.000	0.579 ± 0.079	0.623 ± 0.052	0.591 ± 0.005	0.646 ± 0.010	0.846 ± 0.000	0.720 ± 0.017	0.831 ± 0.003	0.780 ± 0.050	0.224	35.182
NoE+NaiveBayes	0.451 ± 0.010	0.451 ± 0.010	0.586 ± 0.000	0.352 ± 0.128	0.950 ± 0.032	0.643 ± 0.007	0.726 ± 0.009	0.556 ± 0.000	0.794 ± 0.010	0.685 ± 0.011	0.963 ± 0.036	0.246	32.000

Table B.1: Full accuracy results for the experiments executed in Section 4