The University of Winnipeg

Master Thesis

N-Dimensional Polynomial Neural Networks and their Applications

by

Habib Ben Abdallah

A thesis submitted to the Faculty of Graduate Studies of
The University of Winnipeg
in partial fulfillment of the requirements of the degree of

Master of Science

Department of Applied Computer Science
University of Winnipeg
Winnipeg, Manitoba, Canada

N-Dimensional Polynomial Neural Networks and their Applications

by

Habib Ben Abdallah

Supervisory Committee

Dr. Christopher J. Henry, Co-supervisor
Department of Applied Computer Science (The University of Winnipeg)

Dr. Sheela Ramanna, Co-supervisor
Department of Applied Computer Science (The University of Winnipeg)

Dr. Blair Jamieson, Member
Department of Physics (The University of Winnipeg)

Dr. Ian Stavness, External Member
Department of Computer Science (University of Saskatchewan)

Abstract

In addition to being extremely non-linear, modern machine learning problems require millions if not billions of parameters to solve or at least to get a good approximation of the solution, and neural networks are known to assimilate that complexity by deepening and widening their topology in order to increase the level of non-linearity needed for a better approximation. However, compact topologies are always preferred to deeper ones as they offer the advantage of using less computational units and less parameters. This compactness comes at the price of reduced non-linearity and thus, of limited solution search space. This thesis proposes the N-Dimensional Polynomial Neural Network (NDPNN) model that uses automatic polynomial kernel estimation for N-Dimensional Convolutional Neural Networks (NDCNNs) and introduces a high degree of non-linearity from the first layer which can compensate the need for deep and/or wide topologies. We first theoretically formalized the 1DPNN model which can process 1-dimensional signals and we demonstrated that its inherent non-linearity enables it to yield better results with less computational and spatial complexity than a regular 1DCNN on various classification and regression problems related to audio signals, even though it introduces more computational and spatial complexity on a neuronal level. The experiments were conducted on three publicly available datasets and demonstrate that the proposed 1DPNN model can extract more relevant information from the data than a 1DCNN in less time and with less memory. We subsequently extended the theoretical foundation of the 1DPNN to NDPNN which can process 2D signals such as images and 3D signals such as videos. Also, we theoretically created a general polynomial degree reduction formula that we used to develop a heuristic algorithm, which enables the degree reduction of any pre-trained NDPNN. This algorithm compresses an NDPNN without altering its performance, thus making the model faster and lighter. Following that, we used 2DPNNs and 3DPNNs to tackle the problem of plant species recognition on a publicly available plant species recognition dataset composed of 40,000 images with different sizes consisting of 8 plant species. As a result, we created a novel method, called Variably Overlapping Time—Coherent Sliding Window (VOTCSW), that transforms a dataset composed of images with variable size to a 3D representation with fixed size that is suitable for convolutional neural networks, and we demonstrated that this representation is more informative than resizing the images of the dataset to a given size. We theoretically formalized the use cases of the method as well as its inherent properties and proved that it has an oversampling and a regularization effect on the data. By combining the VOTCSW method with 3DPNNs, we were able to create a model that achieved a state-of-the-art accuracy of 99.9% on the considered dataset, surpassing well-known architectures such as ResNet and Inception. Furthermore, we established that the currently available plant species dataset could not be used for machine learning in its present form, due to a substantial class imbalance between the training set and the test set. Hence, we created a specific preprocessing and a model development framework that enabled us to improve the

accuracy from 49.23% to 99.9%. The contributions of this thesis are the creation of a novel generic model called NDPNN that can extract more information from data than a NDCNN with less computational and spatial complexity, the evaluation of the performance of NDPNNs on audio signals, images and videos, the creation of a general direct polynomial reduction formula, the design of a heuristic algorithm for NDPNN compression that generates faster and lighter models, the formalization of an image transformation method that circumvents image resizing without altering fine-grained information, and the production of a state-of-the-art 3DPNN for plant species recognition.

Keywords: Convolutional neural networks, polynomial approximation, deep learning, audio signal processing, polynomial degree reduction, plant species recognition.

Acknowledgements

I would like to express my deepest and sincerest gratitude to Dr. Christopher J. Henry and Dr. Sheela Ramanna who have supported me tirelessly throughout my Master's studies and who have become a pillar in my academic career by setting up an unmatched example of what a thoughtful, resourceful, attentive and helpful supervisor should be. Without the confidence that they have in me and their unfailing constant encouragements, this thesis would not have come to fruition. I would also like to thank Dr. Talal Halabi and Dr. Sergio Camorlinga for their kindness and availability and for the help that they provided me in turning course projects to academic papers.

I would like to express my most respectful recognition to the supervisory committee, Dr. Blair Jamieson and Dr. Ian Stavness, for considering my thesis and for devoting their time and efforts to read it thoroughly. I wish to thank all the valued members of the Department of Applied Computer Science and the Faculty of Graduate Studies who work daily to offer the best studying conditions to students. Special thanks go to Dylan Jones for never failing to resolve my inquiries efficiently and for being an excellent student coordinator. I also want to thank Connie Arnhold for always being cheerful and for the peaceful and lively atmosphere she brings to the ACS department.

I wish to express my thanks and appreciation to Mitacs, the NSERC, and the Faculty of Graduate Studies of the University of Winnipeg for funding my research and for making it possible.

Finally, I would like to thank my parents for their patience and for raising me to become what I am today, my friends, with whom I can share anything, for always brightening my mood with a good laugh, my parents-in-law for believing in me and always supporting me, and lastly, my beloved wife who completes me and infuses my life with bliss.

To my father.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Artificial Neural Network (ANN) has nowadays become an extremely popular model for machine-learning applications that involve classification or regression [1]. Due to its effectiveness on feature-based problems, it has been extended with many variants such as Convolutional Neural Networks (CNNs) [2, 3] or Recurrent Neural Networks (RNNs) [4, 5] that aim to solve a broader panel of problems involving signal processing [6, 7] and/or time-series [8, 9], for example. However, as data is becoming more available to use and exploit, problems are becoming richer and more complex, and deeper and bigger topologies [10, 11] of neural networks are used to solve them. Moreover, the computational load to train such models is steadily increasing - despite advances in high performance computing systems - and it may take up to several days or weeks just to develop a trained model that generalizes well on a given problem. This aggravation is partially due to the fact that complex problems involve highly non-linear solution spaces, and, to achieve a high level of non-linearity, deeper topologies [12] are required since every network layer introduces a certain level of non-linearity with an activation function.

A well-known machine-learning trick to alleviate such a problem is to resort to a *kernel transformation* [13, 14]. The basic idea is to apply a non-linear function (that has certain properties) to the input features so that the search space becomes slightly non-linear and maybe (not always) more adequate to solve the given problem. However, such a trick highly depends on the choice of the so-called *kernel*, and may not be successful due to the fact that the non-linearity of a problem can not always be expressed through usual functions; e.g. polynomial, exponential, logarithmic or circular functions. Moreover, the search for an adequate kernel involves experimenting with different functions and evaluating their individual performance which is time consuming. One can also use the kernel trick with neural networks [15, 16] but the same problem remains: What is an adequate kernel for the given problem?

1.1 Problem Statement

To solve the problem of automatically designing a kernel for any given problem, Ivakhnenko introduced the concept of polynomial networks [17] where he expanded the definition of a Rosenblatt's perceptron [18] by enabling it to take two inputs and estimate the weights corresponding to their quadratic expansion in order to better approximate an output. This concept was further generalized into polynomial neural networks (PNNs) introduced by Oh *et al.* [19] where each neuron could use weights to estimate the polynomial expansion of any degree of any pair of inputs. By dynamically interconnecting neurons into layers using a construction algorithm, they built a network that could approximate a scalar output using a full polynomial expansion of an input vector. However, the structure of a PNN could not allow for weight sharing or densely connected neurons as each neuron could have access to only 2 components of an input vector. In this thesis, we aim to create the foundation of a new model that extends the PNN model to allow for weight sharing via convolution such as in N-Dimensional CNN (NDCNN) and our objective is to demonstrate that the non-linearity introduced by polynomial approximation may help to reduce the depth (number of layers) or at least the width (number of neurons per layer) of the conventional neural network architectures due to the fact that the kernels estimated are dependent on the problems that are considered, and more specific than the well-known general kernels.

1.2 Proposed Approach

The model that we propose is a novel variant of NDCNN which we call N-Dimensional Polynomial Neural Network (NDPNN) designed to create the adequate kernels for each neuron in an automated way using a polynomial approximation with a given degree for each layer, and thus, better approximate the non-linearity of the solution space by increasing the complexity of the search space. The NDPNN model is fundamentally different from the PNN model in the sense that any neuron can take any number of inputs (not just 2 as in the PNN model), a neuron's input is a finite number of N-dimensional signals (not scalars), a neuron's output is not necessarily a polynomial (it can be a non-linear function applied to a polynomial), the weights of a neuron can be shared between the components of an input via convolution, and the output of an NDPNN is, in general, a finite number of N-dimensional signals (feature maps).

In this thesis, we created a three-step approach to develop the NDPNN model. The first step of our approach is to restrict ourselves to only 1-dimensional signals and, thus, to defining and using 1DPNNs so that we can incrementally build a foundation for 2DPNNs and 3DPNNs. We present a formal definition of the 1DPNN model that is easily extensible for 2-dimensional and 3-dimensional signals and which includes forward and backward propagation, a new weight initialization method, and a detailed theoretical computational complexity analysis. Our proposed

1DPNN is evaluated in terms of the number of parameters, the computational complexity, and the estimation of performance with different activation functions for various audio signal applications involving either classification or regression. Two classification problems were considered, musical note recognition on a subset of the NSynth[20] datset and spoken digit recognition on the Free Spoken Digits dataset [21]. Only one regression problem was considered for which a subset of the MUSDB18 [22] dataset was used, namely audio signal denoising.

The second step of our approach is to extend the 1DPNN to NDPNN by generalizing the formal definition of the 1DPNN to higher dimensions (2 and 3) and to find a way to minimize the degree of each layer of a pre-trained NDPNN with little to no compromise on its performance in order to improve its computational efficiency and to reduce its spatial complexity. To do so, we use the assumption that the samples of the feature map produced by each layer of a pre-trained NDPNN are always bounded in a certain interval. Therefore, we are able to formally develop a mathematical formula to generate a low degree polynomial that optimally approximates a higher degree polynomial on a symmetric interval. Consequently, we develop a heuristic algorithm that makes use of the polynomial degree reduction formula and that makes it possible to determine the smallest degree of each layer of a pre-trained NDPNN that preserves its performance on its test set, thus, enabling it to use less memory and less computational power while maintaining the same performance on its test set.

The third step is to use the newly defined 2DPNNs and 3DPNNs in conjunction with the degree reduction heuristic algorithm to produce state-of-the-art results on a given problem. We consider the problem of plant species recognition on the "Weed seedling images of species common to Manitoba, Canada" (WSISCMC) dataset [23] in order to enable the automation of the data acquisition and the data labeling processes. The dataset is composed of images of various sizes which makes it mandatory to create a representation with fixed size in order to use neural networks. Therefore, we design a novel method based on sliding windows that we call Variably Overlapping Time–Coherent Sliding Window (VOTCSW) and which allows the transformation of images with variable sizes to a 3D representation with fixed size that is suitable for both 3DCNNs and 3DPNNs and that offers a compromise between shrinking and padding/magnifying the images. Following that, we create a model development framework that enables the creation of a highly reliable and accurate NDPNN plant recognition model. Furthermore, we redistribute the samples of the dataset to maximize the learning efficiency of any machine learning model that may use it. We also train various well-know architectures such as ResNetV2 [24], InceptionV3 [25] and Xception [26], and we evaluate the advantage of using the VOTCSW method and the NDPNN degree reduction heuristic with respect to regular 2DCNNs architectures.

1.3 Thesis Contribution

The work presented in this thesis has led to the following three journal papers:

- "1-Dimensional Polynomial Neural Networks for audio signal related problems", Knowledge-Based Systems, Volume 240, 2022, p. 108174 [27].
- "Polynomial degree reduction in the L2-norm on a symmetric interval for the canonical basis", Results in Applied Mathematics, Volume 12, 2021, p. 100185 [28].
- "Plant Species Recognition with Optimized 3D Polynomial Neural Networks and Variably Overlapping Time-Coherent Sliding Window", ArXiv, vol.abs/2203.02611, 2022 [29].

The contribution of this thesis is as follows:

- Creating a novel generic model called NDPNN that can extract more information from data than a NDCNN with less computational and spatial complexity [27, 29].
- Formalizing the definition of the forward and backward propagation of a 1DPNN [27].
- Elaborating a detailed theoretical computational complexity analysis of the 1DPNN with respect to the 1DCNN [27].
- Linearizing the computational complexity of the 1DPNN by implementing the model on GPU [27].
- Creating a theorem for generating a 1DCNN with the same number of parameters of a given 1DPNN [27].
- Evaluating the influence of various activation functions on 1DPNNs [27].
- Comparing the performance of 1DPNNs with that of 1DCNNs on three audio signal related problems with three different comparison strategies [27].
- Creating a mathematical formula for polynomial degree reduction which is more stable and less complex than classical methods and which can be applied on any machine learning model that uses polynomials as kernels or as building blocks [28].
- Using the formula to develop a heuristic algorithm for the degree reduction of pre-trained NDPNNs which creates lighter and faster NDPNNs with little to no compromise to their initial performances [29].
- Extending the formal definition of 1DPNNs to NDPNNs which can be used on 2D and 3D signals such as images and videos [29].
- Formalizing the VOTCSW method which circumvents the need to resize images from an image dataset that has variable sizes by transforming each image to a 3D representation

of fixed size with minimal added synthetic data (padding, magnifying) and minimal loss of data (shrinking) which is suitable for 3DCNNs and 3DPNNs and which improves their inference on the WSISCMC dataset [29].

- Creating an NDPNN model development framework that makes use of the degree reduction heuristic and the VOTCSW method and allows the creation of the best fitting neural network architecture on the WSISCMC dataset [29].
- Resampling the WSISCMC dataset with respect to class distribution and size distribution in order to enhance the performance of any machine learning model trained on it [29].
- Creating a simple 3DPNN architecture that achieves a state-of-the-art 99.9% accuracy on the WSISCMC dataset, and which outperforms highly complex neural network architectures such as ResNet50V2 and InceptionV3, in less time and with substantially less parameters [29].
- Determining the existence of aberrant samples in the WSISCMC dataset which are not suitable for the single-plant species recognition task that this dataset was created for [29].

1.4 Thesis Layout

The layout of this thesis is as follows. Chapter 2 discusses the works related to the introduction of additional non-linearities in neural networks. Chapter 3 formulates the 1DPNN model mathematically and evaluates its performance with regards to the 1DCNN model on various audio signal related problems. Chapter 4 presents the generalization of 1DPNNs to NDPNNs and the formal creation of the polynomial degree reduction formula which is used to design a heuristic algorithm that minimizes the degree of each layer of an NDPNN with a negligible effect on its performance. Chapter 5 presents the use of 2DPNNs and 3DPNNs in the task of plants species recognition and introduces the VOTCSW method. Finally, Chapter 6 addresses the strengths and the weaknesses of the NDPNN model and proposes different ways of extending and improving it.

Chapter 2

Literature review

Many of the works that try to add a degree of non-linearity to the neural network model focus either on creating new layers, designing new activation functions or implementing neuronal enhancements that change the behavior of conventional neurons such as in the work achieved in this thesis. This chapter presents an overview of the most known techniques and methods used to enhance the non-linearity of a neural network in order to improve its feature expressiveness and its search space complexity. Section 2.1 discusses the most important activation functions that are used to introduce non-linearity. Section 2.2 discusses the layers that were specifically designed as non-linear extensions to regular neural network layers. Section 2.3 discusses the ways in which neurons were modified, enhanced and extended to allow for better non-linear approximations.

2.1 Activation Functions

Turian et. al [30] implemented a variant of the tangent hyperbolic called softsign that was proposed by Elliot [31] to reduce the problem of gradient vanishing with a less abrupt saturation than the tangent hyperbolic. They used it in conjunction with a quadratic filter extractor for the purpose of token chunking and they achieved better classification accuracy than conventional models. Although the problem of gradient vanishing was attenuated, using softsign produces higher gradient updates due to its fractional nature which can cause unstable training. Jarrett *et al.* [32] are the first to consider the Rectified Linear Unit (ReLU) as an activation function for neural networks. They were inspired to do so by biological models which use rectifying non-linearities such as the absolute value and they showed that the non-linearity of the ReLU function is one of the most important factors in improving the performance of convolutional neural networks trained on three different datasets for object recognition. ReLU's simplicity allows the networks to be trained faster and to better propagate the gradients by minimizing the vanishing gradient problem. However, ReLU can cause the early death of neurons if the network weights are not initialized in a certain way and can make mean shifts towards zero hard due to its

saturation to zero for negative values. This issue was considered in the work of Clevert *et al.* [33] where they designed the Exponential Linear Unit (ELU) to allow for a certain range of negative values before saturation. They use it to demonstrate that the bias shift correction it introduces not only speeds up learning but also improves the generalization capability of networks with more than 5 layers trained on 4 different datasets. Following that, Ramachandran *et al.* [34] have considered improving the ReLU and ELU by creating a reinforcement learning-based framework that searches for multiple combinations of predetermined functions that perform well on certain datasets. They discovered the swish function which outperforms every known improvement of the ReLU in terms of stability, convergence speed and generalization capability.

2.2 Non-linear Neural Network Layers

The most used non-linear layer in the literature is the max pooling layer which subsamples an input signal by taking the maximum value between samples contained in adjacent observation windows. It was proposed by Riesenhuber *et al.* [35] in an attempt to more accurately model object recognition in the visual cortex. They showed that the max operation enables object recognition models to achieve translation, scale and elastic distortion invariance which improves their generalization capability as well as their stability. However, the max pooling layer performance decreases as the network becomes deeper due to the drastic reduction in the number of available samples that will be convolved together and due to the disjoint nature of the pooling regions. As a result, the Fractional Max Pooling layer [36] was proposed by Graham to address these issues. He allows non-integer subsampling coefficients which helps in creating deeper architectures of convolutional neural networks and he shows that this layer reduces overfitting which makes regularization an option for deep networks. Nevertheless, like the max pooling layer, the fractional max pooling layer only uses the *max* operation to output its values which may not always be appropriate depending on the problem. This is where the work of Lee *et al.* [37] on mixed, gated and tree pooling comes to circumvent this issue. They proposed a way to generate pooling operations specific to the data at hand by adding a learnable parameter that determines the proportion of max pooling and average pooling to perform, and then sum the quantities together which they call "mixed" pooling because it mixes the max and average operations. Although this improved the performance of regular convolutional neural networks on various benchmark datasets compared to the simple use of max pooling, the "mixed" pooling was not a data-responsive method since the proportion coefficient becomes static after the learning process ends. Thus, they proposed the "gated" pooling which learns a gating mask that produces a proportion coefficient via its dot product with the values in the pooled region. This coefficient was then used in the same way as in the "mixed" pooling and produced even better results. The last improvement that they achieved was the "tree" pooling which consists in creating multiple gating masks, organizing them in a binary tree, and mixing each two children

node into a parent node until only one parent node remains which becomes the final learned pooling operation. This allowed them to achieve state-of-the-art results on three benchmark datasets. Other non-linear layers have been developed such as dropout layers [38] and batch normalization layers [39], but they are used as regularizers rather than approximators.

2.3 Non-linear Neuronal Enhancements

Many of the works that try to add a degree of non-linearity to the neural network model focus either on enriching pooling layers [36, 37] by allowing more granularity or by incorporating various additional operations, creating new layers (such as dropout layers [38] and batch normalization layers [39]), or designing new activation functions [32, 30]. However, Campbell *et al.* [40] used Ivakhnenko's polynomial networks along with Hidden Markov Models for speech recognition. They have proposed a novel training algorithm that can experimentally converge to allow a polynomial network to approximate transition probabilities by which they demonstrated the efficiency of such networks and achieved a high accuracy. However, they only used Ivakhnenko's neurons which have a number of limitations (as described in Section 1), and they only obtained convergence for the cases they have tried without providing a formal proof.
Livni *et al.* [41] have considered changing the way Ivakhnenko's neurons behave by making them either compute a linear combination of the input components or a weighted product of the input components. By stacking layers and creating a deep network, the network is able to learn any polynomial of a degree not exceeding the number of layers. Moreover, they provide an algorithm that can construct the network progressively. While achieving relatively good results on various problems with a small topology and with minimal human intervention, they only dealt with the perceptron model which can be inappropriate for the problems they tackled, which are computer vision problems. In fact, the perceptron model does not take into account the spatial proximity of the pixels and only considers an image as a vector with no specific spatial arrangement or relationship between neighboring pixels, which is why a convolutional model is more appropriate for these kind of problems. Similarly, Hughes *et al.* [42] have considered introducing the use of quadratic polynomials in RNN nodes because of their ability to approximate more functions than affine functions can. By using this subtle tweak, they were able to outperform state-of-the-art models on voice activity detection while using less number of parameters, which suggest that the non-linearity is well captured by the quadratic nodes. The study is, however, limited to quadratic polynomials and no further exploration of higher polynomial degrees is performed.
Wang *et al.* [43], on the contrary, considered the 2D convolutional model and changed the way a neuron operates by applying a kernel function on its input which they call kervolution, thus, not changing the number of parameters a network needs to train, but adding a level of non-linearity that can lead to better results than regular CNNs. They used a sigmoidal, a gaussian and a

polynomial kernel and studied their influence on the accuracy measured on various datasets. Furthermore, they used well-known architectures such as ResNet [44] and modified some layers to incorporate the kervolution operator. However, they show that this operator can make the model become unstable when they introduce more complexity than what the problem requires. Although they have achieved better accuracy than state-of-the-art models, they still need to manually choose the kernel for each layer which can be inefficient due to the sheer number of possibilities they can choose from and because it can be really difficult to estimate how much non-linearity a problem needs.

Mairal [45] on the other hand, proposed a way to learn layer-wise kernels in a 2D convolutional neural network by learning a linear subspace in a Reproducing Kernel Hilbert Space [46] at each layer from which feature maps are built using kernel compositions. Although the method introduces new parameters that need to be learned during the backpropagation, its main advantage is that it is able to overcome the main problem of using kernels with machine learning, namely learning and data representation decoupling. The problems that the author tackled are image classification and image super-resolution [47], and the results that were obtained outperform approaches solely based on classical convolutional neural networks. Nevertheless, due to technical limitations, the kernel learning could not be tested on large networks and its main drawback is that a pre-parametrized kernel function should be defined for each layer of the network.

However, Tran *et al.* [48] went even further by proposing a relaxation of the fundamental operations used in the multilayer perceptron model called Generalized Operational Perceptron (GOP) which allows changes to the summation in the linear combination between weights and inputs by any other operation such as median, maximum or product, and they call it a pool operator. Moreover, they propose the concept of nodal operators which basically applies a non-linear function on the product between the weights and the inputs. They show that this configuration surpasses the regular multilayer perceptron model on well-known classification problems, with the same number of parameters, but with a slight increase in the computational complexity. However, as a pool operator, a nodal operator and an activation function have to be chosen for every neuron in every layer. As a result, they have devised an algorithm to construct a network with the operators that are meant to minimize the loss chosen for any given problem. Nevertheless, the main limitation of the model is that such operators need to be created and specified manually as an initial step, and that choosing an operator for every neuron is time-consuming. This compulsory preliminary step has to be performed in order to be able to use the model and to train it.

Kiranyaz *et al.* [49] extended the notion of GOP to 2-dimensional signals such as images and created the Operational Neural Network (ONN) model which generalizes the 2D convolutions to any non-linear operation without adding any new parameter. They prove that ONN models can solve complex problems like image transformation, image denoising and image synthesis

with a minimal topology, where CNN models fail to do so with the same topology and number of parameters. They also propose an algorithm that can create homogeneous layers (all neurons inside the layer have the same pool operator, nodal operator and activation function) and choose the operators that minimize any given loss, in a greedy fashion. However, the aforementioned limitation still applies to the model since it is based on GOPs and the operator choosing algorithm does not take into account the intra-layer dependency of the so-called operators. Therefore, they proposed the concept of generative neurons in [50] to approximate nodal operators by adding learnable parameters into the network. While achieving comparable results to the ONN model, they were unable to overcome the need to manually choose the pool operator and the activation function.

As was discussed, works [41, 43, 45] still need to manually predetermine which kernel to use on each layer, and works [48, 49, 50] need to predetermine, for each neuron, the nodal operator, the pool operator and the activation function to use in order to create the network. The main gain of the proposed 1DPNN model is that there is no longer the need to search for a kernel that produces good results, as the model itself approximates a problem-specific polynomial kernel for each neuron.

Chapter 3

1-Dimensional Polynomial Neural Networks for Audio Signal Related Problems

3.1 Introduction

We propose the 1DPNN model which is an extension of the 1DCNN model that uses polynomial approximation to perform highly non-linear filtering from the first layer. We formulate the forward propagation equation and the gradient estimations which allow the model to be trained using the gradient descent algorithm. In addition, we provide a theoretical computational complexity analysis to estimate how computationally complex a 1DPNN is compared to a 1DCNN. Consequently, we describe the GPU implementation of the model and we empirically show that, although the 1DPNN model's theoretical computational complexity is polynomial, the parallel nature of the implementation enables a linearization of the model's inference and learning process. We also formalize a theorem that generates a 1DCNN that has the same number of parameters as a given 1DPNN. Finally, we consider two classification problems, namely musical note recognition and spoken digit recognition, to evaluate the 1DPNN as a feature extractor and one regression problem, audio signal denoising, to evaluate the 1DPNN as a regressor. We also evaluate the influence of various activation functions on the 1DPNN model, and we devise a model comparison strategy on every problem which allows us to estimate the efficiency of the 1DPNN model compared to the 1DCNN model.

Although the problems that are tackled can be efficiently solved using much more powerful audio signal representations such as time-frequency representations and much more complex models such as 2DCNNs [51] and RNNs, our objective is to illustrate how the 1DPNN can introduce just enough non-linearity to the 1DCNN model to achieve better performance with less spatial complexity and computational complexity thanks to the polynomial kernel estimation. Therefore, comparing our model to the 2DCNN model, for example, can be problematic since

the input of a 2DCNN is a 2-dimensional signal, and the input of a 1DPNN is a 1-dimensional signal which makes the information that both models have access to, completely different. Furthermore, comparing our model to the RNN model would result in the same complication since the internal definition of both models are fundamentally very different, although the information that is accessed is the same. Hence, the model that shares the most common characteristics with our proposed 1DPNN is the 1DCNN since the 1DPNN extends it. As a result, our evaluation methodology is based on comparing the performance, the computational complexity and the spatial complexity of the 1DPNN to that of the 1DCNN under certain conditions, notably, the equality of the number of trainable parameters or the equality of the performance evaluation. Therefore, we chose not to create very deep and wide networks and to restrict our analysis to a fairly manageable number of parameters due to technical limitations.
The outline of this chapter is as follows. Section 3.2 formulates the 1DPNN model mathematically by detailing how the forward propagation and the backward propagation are performed as well as providing a new way to initialize the weights and a detailed theoretical computational analysis of the model while Section 3.3 describes how it was implemented and tackles the computational analysis from an experimental perspective. Section 3.4 describes in detail the experiments that were conducted in order to evaluate the model, and shows its results.

3.2 Theoretical Framework

In this section, the background of the PNN model is presented and the 1DPNN is formally defined with its relevant hyperparameters, its trainable parameters and the way to train the model with the gradient descent algorithm [2]. A theoretical analysis of the computational complexity of the model is also made with respect to the complexity of the regular convolutional model.

3.2.1 Preliminaries

As defined by Oh et al. [19], a PNN $\hat{f}$ takes a vector of N independent variables $(x_1, ..., x_N)$ and estimates a single scalar output $\hat{y}$ as such:

$$\hat{y} = \hat{f}(x_1, ..., x_N) = c_0 + \sum_{k_1} c_{k_1} x_{k_1} + \sum_{k_1,k_2} c_{k_1 k_2} x_{k_1} x_{k_2} + ...,$$

where c_k's are the coefficients of the model and $\hat{y}$ is a polynomial expansion of the input vector $(x_1, ..., x_N)$ to a degree D. A PNN is constructed dynamically by determining a partial description (PD) of each combination of r independent variables in the input vector. A PD of two variables is a polynomial expansion to a given degree (or order) D' where weights are determined using the least square method applied on a given training data. For example, a PD

z_{12} of x_1 and x_2 of order 2 can be expressed as such:

$$z_1 2 = c_0 + c_1 x_1 + c_2 x_2 + c_{11} x_1^2 + c_{22} x_2^2 + c_{12} x_1 x_2.$$

Figure 3.1 (extracted from [19]) shows a PNN with $r = 2$ and $D' = 2$. Every PD is then evaluated on the test set and only a predetermined number of them are chosen to create a new feature vector composed of the chosen PDs. The number of selected PDs can also be determined with a performance cutoff value such that the PDs whose performances are below this value are discarded. Following that, the same process applied on the input vector is repeated on the feature vector consisting of the chosen PDs. The above steps are subsequently repeated until a stopping criterion has been satisfied. After that, the node that produced the best approximation is selected and all the nodes that are independent of it are discarded for inference. Therefore, the degree D of the polynomial expansion of $(x_1, ..., x_N)$ can be determined dynamically via the depth of the network as well as the order of the PD used in the PNN. Although PNNs were used

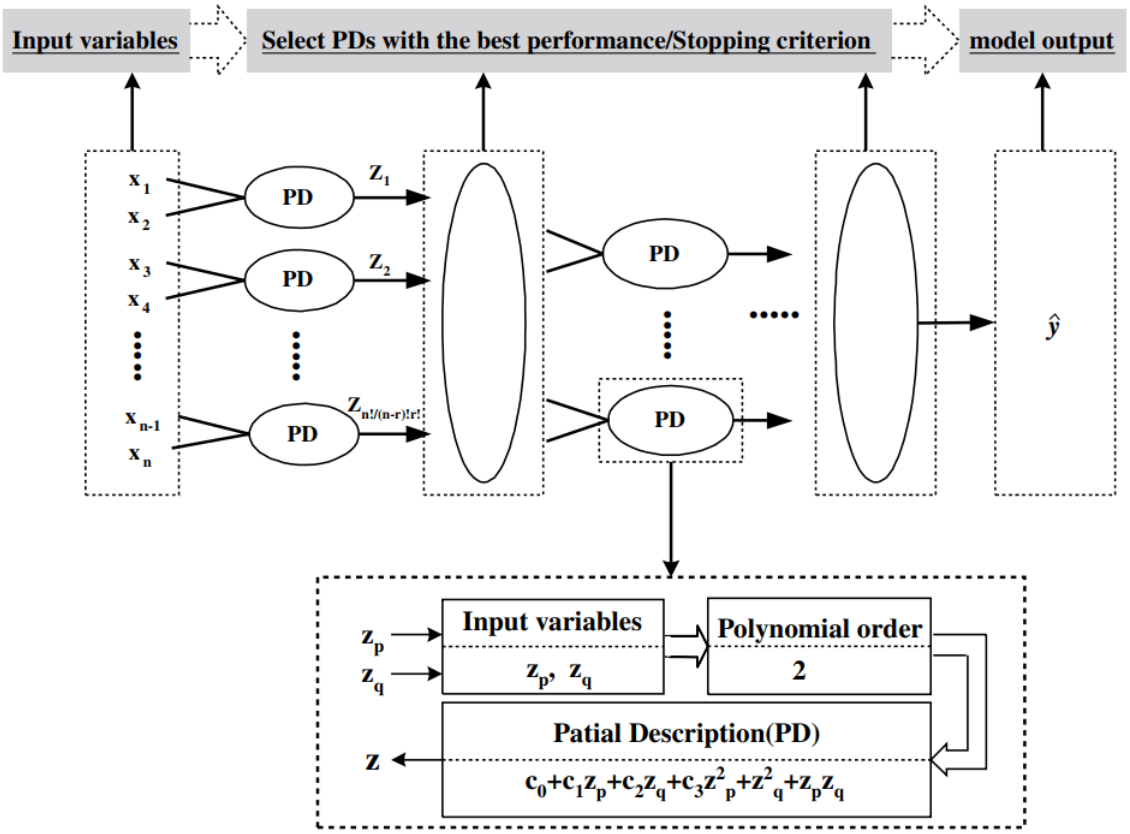


FIGURE 3.1: An overall architecture of the PNN [19].

to exploit non-linear relationships between independent features, they cannot be used efficiently on ordered structures such as signals that have a causal relationship where each of their samples are constructed using a process applied on all of the previous samples. Therefore, we propose

the 1DPNN model to remedy this drawback and to improve the non-linearity of the 1DCNN model.

3.2.2 1DPNN Model Definition

The aim is to create a network whose neurons can perform non-linear filtering using a Maclaurin series decomposition. For 1-Dimensional signals, a regular neuron in 1DCNN performs a convolution between its weight vector and its input vector whereas the neuron that needs to be modeled for 1DPNN should perform convolutions not only with its input vector but also with its exponentiation as shown in Figure 3.2 below. In the following, we designate by L the number

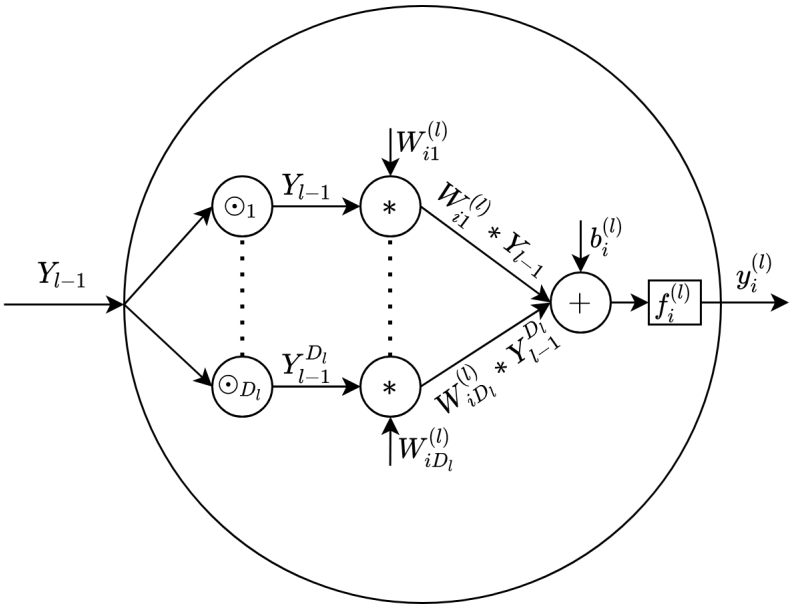


FIGURE 3.2: A graphical representation of a 1DPNN neuron.

of layers of a 1DPNN. $\forall l \in [\![1, L]\!], N_l$ is the number of neurons in layer l, D_l is the degree of the Taylor decomposition of the neurons in layer l and $\forall i \in [\![1, N_l]\!], y_i^{(l)}$ is the output vector of neuron i in layer l considered as having 1 row and M_l columns representing the output samples indexed in $[\![0, M_l - 1]\!]$. We consider the input layer as layer $l = 0$ with N_0 inputs in general (for a single input network, $N_0 = 1$). $\forall l \in [\![0, L]\!]$, we construct the N_l by M_l matrix Y_l such that:

$$Y_l = \begin{bmatrix} y_1^{(l)} \\ \vdots \\ y_{N_l}^{(l)} \end{bmatrix}.$$

$\forall l \in [\![1, L]\!], \forall (i, j, d) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!] \times [\![1, D_l]\!], w_{ijd}^{(l)}$ is the weight vector of neuron i in layer l corresponding to the exponent d and to the output of neuron j in layer $l - 1$ considered as having 1 row and K_l columns indexed in $[\![0, K_l - 1]\!]$. $\forall l \in [\![1, L]\!], \forall (i, d) \in [\![1, N_l]\!] \times [\![1, D_l]\!]$, we

construct the N_{l-1} by K_l matrix $W_{id}^{(l)}$ such that:

$$W_{id}^{(l)} = \begin{bmatrix} w_{i1d}^{(l)} \\ \vdots \\ w_{iN_{l-1}d}^{(l)} \end{bmatrix}.$$

$\forall l \in [\![1, L]\!], \forall i \in [\![1, N_l]\!], b_i^{(l)}$ is the bias of neuron i in layer l and $f_i^{(l)}$ is a differentiable function called the activation function of neuron i in layer l such that $y_i^{(l)} = f_i^{(l)}\left(x_i^{(l)}\right)$ where $x_i^{(l)}$ is the pre-activation output of neuron i in layer l.

Definition 3.2.1. *The output of a neuron in the 1DPNN is defined as such:*

$$\forall l \in [\![1, L]\!], \forall i \in [\![1, N_l]\!],\ y_i^{(l)} = f_i^{(l)}\left(\sum_{d=1}^{D_l} W_{id}^{(l)} * Y_{l-1}^d + b_i^{(l)}\right) = f_i^{(l)}\left(x_i^{(l)}\right), \tag{3.1}$$

where $*$ is the convolution operator, $Y_{l-1}^d = \underbrace{Y_{l-1} \odot \cdots \odot Y_{l-1}}_{d \text{ times}}$, and $\odot$ is the Hadamard product.

Remark 3.2.1. *As stated above, the whole focus of this work is to learn the best polynomial function in each neuron for a given problem, which is entirely defined by the weights $W_{id}^{(l)}$ associated with Y_{l-1} to the power of d.*

Remark 3.2.2. *Since the 1DPNN neuron creates a polynomial function using the weights $W_{id}^{(l)}$, the activation function $f_i^{(l)}$ can seem unnecessary to define, and can be replaced by the identity function. However, in the context of the 1DPNN model, the activation function plays the role of a bounding function, meaning that it can be used to control the range of the values of the created polynomial function.*

3.2.3 1DPNN Model Training

In order to enable the weights of the model to be updated so that it learns, we need to define a loss function that measures whether the output of the network is close or far from the output that is desired since it is a supervised model. We denote by Y the desired output, by $\hat{Y}$ the output that is produced by the network, and by $\epsilon\left(Y, \hat{Y}\right)$ the loss between the desired output and the estimated output. ϵ needs to be differentiable since the estimation of its derivative with respect to different variables is the key of learning the weights due to the fact that gradient descent is used as a numeric optimization technique.

3.2.3.1 Gradient Descent Algorithm

Given a function $\phi : \mathbb{R}^N \times \mathbb{R}^P \to \mathbb{R}^M$, where $(M, N, P) \in \mathbb{N}^{*3}$, the input is a tuple $X = (X_1, ..., X_N) \in \mathbb{R}^N$, and the parameters are $\theta = (\theta_1, ..., \theta_P)$; we consider a desired output from

X given ϕ called Y where $Y \in \mathbb{R}^M$. The objective is to estimate θ so that $\epsilon\left(Y, \phi\left(X, \theta\right)\right)$ is minimum where $\epsilon : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}_+$ is a differentiable loss function. Gradient descent [52] is an algorithm that iteratively estimates new values of θ for T iterations or until a target loss ϵ_t is attained. $\forall t \in [\![0, T]\!], \hat{\theta}^{(t)} = (\hat{\theta}_1^{(t)}, ..., \hat{\theta}_P^{(t)})$ is the estimated value of θ at iteration t. Given an initial value $\hat{\theta}^{(0)}$, and a learning rate $\eta \in (0, 1]$, the gradient descent estimations are

$$\forall t \in [\![0, T-1]\!], \hat{\theta}^{(t+1)} = \hat{\theta}^{(t)} - \eta \nabla_\theta \epsilon \left(\hat{\theta}^{(t)}\right),$$

where $\nabla_\theta \epsilon \left(\hat{\theta}^{(t)}\right)$ is the gradient of ϵ with respect to θ applied on $\hat{\theta}^{(t)}$. In the case of the 1DPNN, the parameters are the weights and the biases, so there is a need to estimate the weight gradients $\dfrac{\partial \epsilon}{\partial w_{ijd}^{(l)}}$ and the bias derivatives $\dfrac{\partial \epsilon}{\partial b_i^{(l)}}$, $\forall l \in [\![1, L]\!], \forall (i, j, d) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!] \times [\![1, D_l]\!]$.

3.2.3.2 Weight Gradient Estimation

In order for the weights to be updated using the Gradient Descent algorithm, there is a need to estimate the contribution of each weight of each neuron in the loss by means of calculating the gradient.

Proposition 3.2.1. *The gradient of the loss with respect to the weights of a 1DPNN neuron can be estimated using the following formula:*

$$\begin{aligned} &\forall l \in [\![1, L]\!], \forall (i, j, d) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!] \times [\![1, D_l]\!], \\ &\frac{\partial \epsilon}{\partial w_{ijd}^{(l)}} = \left(\frac{\partial \epsilon}{\partial y_i^{(l)}} \odot \frac{\partial y_i^{(l)}}{\partial x_i^{(l)}}\right) * \left(y_j^{(l-1)}\right)^d = \frac{\partial \epsilon}{\partial x_i^{(l)}} * \left(y_j^{(l-1)}\right)^d, \end{aligned} \tag{3.2}$$

where

- $\dfrac{\partial y_i^{(l)}}{\partial x_i^{(l)}} = \dfrac{\partial f_i^{(l)}}{\partial x_i^{(l)}} \left(x_i^{(l)}\right)$ *is the derivative of the activation function of neuron i in layer l with respect to $x_i^{(l)}$; and*
- $\dfrac{\partial \epsilon}{\partial y_i^{(l)}}$ *is the gradient of the loss with respect to the output of neuron i in layer l, that we call the output gradient.*

Proof. To estimate the weight gradients, we use the chain-rule such that:

$$\begin{aligned} &\forall l \in [\![1, L]\!], \forall (i, j, k, d) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!] \times [\![0, K_l - 1]\!] \times [\![1, D_l]\!], \\ &\frac{\partial \epsilon}{\partial w_{ijd}^{(l)}}(k) = \sum_{m=0}^{M_l - 1} \left(\frac{\partial \epsilon}{\partial y_i^{(l)}} \odot \frac{\partial y_i^{(l)}}{\partial x_i^{(l)}}\right)(m) . \frac{\partial x_i^{(l)}}{\partial w_{ijd}^{(l)}(k)}(m). \end{aligned} \tag{3.3}$$

From Eq. (3.1), we can write:

$$\forall l \in [\![1, L]\!], \forall (i, m, d) \in [\![1, N_l]\!] \times [\![0, M_l - 1]\!] \times [\![1, D_l]\!],$$
$$x_i^{(l)}(m) = \sum_{j'=1}^{N_{l-1}} \sum_{d'=1}^{D_l} \sum_{k'=0}^{K_l-1} w_{ij'd'}^{(l)}(k') \left(y_{j'}^{(l-1)}\right)^{d'} (m + k'),$$

from which we deduce:

$$\forall l \in [\![1, L]\!], \forall (i, j, m, k, d) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!] \times [\![0, M_l - 1]\!] \times [\![0, K_l - 1]\!] \times [\![1, D_l]\!],$$
$$\frac{\partial x_i^{(l)}}{\partial w_{ijd}^{(l)}(k)}(m) = \left(y_j^{(l-1)}\right)^d (m + k). \tag{3.4}$$

When injecting Eq. (3.4) in Eq. (3.3), we find that:

$$\forall l \in [\![1, L]\!], \forall (i, j, k, d) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!] \times [\![0, K_l - 1]\!] \times [\![1, D_l]\!],$$
$$\frac{\partial \epsilon}{\partial w_{ijd}^{(l)}}(k) = \sum_{m=0}^{M_l-1} \left(\frac{\partial \epsilon}{\partial y_i^{(l)}} \odot \frac{\partial y_i^{(l)}}{\partial x_i^{(l)}} \right) (m). \left(y_j^{(l-1)}\right)^d (m + k),$$

which is equivalent to:

$$\forall l \in [\![1, L]\!], \forall (i, j, d) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!] \times [\![1, D_l]\!],$$
$$\frac{\partial \epsilon}{\partial w_{ijd}^{(l)}} = \left(\frac{\partial \epsilon}{\partial y_i^{(l)}} \odot \frac{\partial y_i^{(l)}}{\partial x_i^{(l)}} \right) * \left(y_j^{(l-1)}\right)^d = \frac{\partial \epsilon}{\partial x_i^{(l)}} * \left(y_j^{(l-1)}\right)^d. \tag{3.5}$$

■

Remark 3.2.3. *The weight gradient estimation of a 1DPNN neuron is equivalent to that of a 1DCNN neuron when $D_l = 1$. This was to be expected as the former is only a mere extension of the latter.*

Remark 3.2.4. *The output gradient can easily be determined for the last layer since $\hat{Y} = Y_L$ which makes the loss directly dependent on Y_L. However, it can not be determined as easily for the other layers since the loss is indirectly dependent on their outputs.*

3.2.3.3 Output Gradient Estimation

As stated in Remark 3.2.4, there is a need to find a way to estimate the gradient of the loss with respect to the inner layers' outputs in order to estimate the weight gradients of their neurons as in Eq. (3.2).

Proposition 3.2.2. *The output gradient of a neuron in an inner layer can be estimated using the following formula:*

$$\forall l \in [\![1, L-1]\!], j \in [\![1, N_l]\!],$$
$$\frac{\partial \epsilon}{\partial y_j^{(l)}} = \sum_{d=1}^{D_l} d \left(\sum_{i=1}^{N_{l+1}} \tilde{w}_{ijd}^{(l+1)} * \frac{\mathring{\partial} \epsilon}{\partial x_i^{(l+1)}} \right) \odot \left(y_j^{(l)} \right)^{d-1}, \tag{3.6}$$

where

- $\forall k \in [\![0, K_{l+1}-1]\!], \tilde{w}_{ijd}^{(l+1)}(k) = w_{ijd}^{(l+1)}(K_{l+1}-1-k)$*; and*
- $\forall m \in [\![0, M_l-1]\!], \frac{\mathring{\partial} \epsilon}{\partial x_i^{(l+1)}}(m) = \begin{cases} \frac{\partial \epsilon}{\partial x_i^{(l+1)}}(m-K_{l+1}) & if\ m \in [\![K_{l+1}, M_{l+1}+K_{l+1}-1[\![\\ 0 & else \end{cases}$.

Proof. Since Y_{l+1} is directly computed from $Y_l, \forall l \in [\![1, L-1]\!]$, we can assume that ultimately, the last layer's output Y_L is totally dependent on Y_{l+1} so that we can write:

$$\forall l \in [\![1, L-1]\!], \epsilon\left(Y, \hat{Y}\right) = \epsilon\left(Y, \psi_{l+1}\left(Y_{l+1}\right)\right), \tag{3.7}$$

where Y is a given desired output and ψ_{l+1} is the application of Eq. (3.1) from layer $l+1$ to layer L. From Eq. (3.7) we can write the differential of ϵ as such:

$$\forall l \in [\![1, L-1]\!], d\epsilon = \sum_{i=1}^{N_{l+1}} \frac{\partial \epsilon}{\partial y_i^{(l+1)}} \odot dy_i^{(l+1)}. \tag{3.8}$$

We can then derive the general expression of the output gradient of any neuron in layer l as such:

$$\forall l \in [\![1, L-1]\!], j \in [\![1, N_l]\!], \frac{\partial \epsilon}{\partial y_j^{(l)}} = \sum_{i=1}^{N_{l+1}} \frac{\partial \epsilon}{\partial y_i^{(l+1)}} \odot \frac{\partial y_i^{(l+1)}}{\partial y_j^{(l)}} = \sum_{i=1}^{N_{l+1}} \frac{\partial \epsilon}{\partial y_i^{(l+1)}} \odot \frac{\partial y_i^{(l+1)}}{\partial x_i^{(l+1)}} \odot \frac{\partial x_i^{(l+1)}}{\partial y_j^{(l)}}.$$

This expression can be qualitatively interpreted as finding the contribution of each sample in $y_j^{(l)}$ in the loss by finding its contribution to every output vector in layer $l+1$. Considering a layer $l \in [\![1, L-1]\!]$, a neuron j in layer l, and a neuron i in layer $l+1$, a sample $m \in [\![0, M_l-1]\!]$ in $y_j^{(l)}$ contributes to $x_i^{(l+1)}$ in the following samples:

$$\forall l \in [\![1, L-1]\!], \forall (i, m, k, d) \in [\![1, N_{l+1}]\!] \times [\![0, M_l-1]\!] \times [\![k_{lm}, k'_{lm}]\!] \times [\![1, D_{l+1}]\!],$$
$$x_i^{(l+1)}(m-k) = \sum_{j'=1}^{N_l} \sum_{d=1}^{D_{l+1}} \sum_{k'=0}^{K_{l+1}-1} w_{ij'd}^{(l+1)}(k') \left(y_{j'}^{(l)} \right)^d (m-k+k'),$$

where $\forall l \in [\![1, L-1]\!], k_{lm} = max\,(0, m - M_{l+1} + 1)$ and $k'_{lm} = min\,(m, K_{l+1} - 1)$. We can then determine the exact contributions of $y_j^{(l)}(m)$ in $x_i^{(l+1)}$ as such:

$$\begin{aligned}&\forall l \in [\![1, L-1]\!], \forall (i, j, m, k, d) \in [\![1, N_{l+1}]\!] \times [\![1, N_l]\!] \times [\![0, M_l - 1]\!] \times [\![k_{lm}, k'_{lm}]\!] \times [\![1, D_{l+1}]\!], \\ &\frac{\partial x_i^{(l+1)}}{\partial y_j^{(l)}(m)}(m-k) = \sum_{d=1}^{D_{l+1}} d.w_{ijd}^{(l+1)}(k) \left(y_j^{(l)}\right)^{d-1}(m).\end{aligned} \tag{3.9}$$

Therefore, we can finally determine the full expression of the output gradient for each sample:

$$\begin{aligned}&\forall l \in [\![1, L-1]\!], (j, m) \in [\![1, N_l]\!] \times [\![0, M_l - 1]\!], \\ &\frac{\partial \epsilon}{\partial y_j^{(l)}}(m) = \sum_{i=1}^{N_{l+1}} \sum_{k=k_{lm}}^{k'_{lm}} \sum_{d=1}^{D_l} d. \left(\frac{\partial \epsilon}{\partial y_i^{(l+1)}} \odot \frac{\partial y_i^{(l+1)}}{\partial x_i^{(l+1)}}\right)(m-k) w_{ijd}^{(l+1)}(k) \left(y_j^{(l)}\right)^{d-1}(m).\end{aligned} \tag{3.10}$$

Eq. (3.10) can be decomposed as the sum along i and d of the product of $\left(y_j^{(l)}\right)^{d-1}$ with the correlation between $\frac{\partial \epsilon}{\partial x_i^{(l+1)}}$ and $w_{ijd}^{(l+1)}$, the correlation being a rotated version of the convolution where the samples of the weights are considered in an inverted order to that of the convolution. Therefore, a correlation can be transformed into a convolution by considering the rotated weights $\tilde{w}_{ijd}^{(l+1)}$ defined as such:

$$\forall k \in [\![0, K_{l+1} - 1]\!], \tilde{w}_{ijd}^{(l+1)}(k) = w_{ijd}^{(l+1)}(K_{l+1} - 1 - k).$$

However, we can only perform a valid convolution when $k_{lm} = 0$ and $k'_{lm} = K_{l+1} - 1$. Thus, to obtain a valid convolution otherwise, we consider the zero-padded version of $\frac{\partial \epsilon}{\partial x_i^{(l+1)}}$ designated by $\frac{\mathring{\partial} \epsilon}{\partial x_i^{(l+1)}}$ and defined as such:

$$\forall m \in [\![0, M_l - 1]\!], \frac{\mathring{\partial} \epsilon}{\partial x_i^{(l+1)}}(m) = \begin{cases} \frac{\partial \epsilon}{\partial x_i^{(l+1)}}(m - K_{l+1}) & if\ m \in [\![K_{l+1}, M_{l+1} + K_{l+1} - 1[\![\\ 0 & else \end{cases}.$$

Finally, we can obtain the desired expression by considering the rotated version of the weights, and the zero-padded version of the gradient of the error with respect to $x_i^{(l+1)}$ in Eq. (3.10). ■

Remark 3.2.5. *The main difference between the output gradient estimation of a 1DPNN inner layer's neuron and that of a 1DCNN inner layer's neuron is that in the former, the gradient depends on the output values of the considered layer and in the latter, it does not. By injecting Eq. (3.6) in Eq. (3.2), we notice that, unlike a 1DCNN neuron, the weight gradient of a 1DPNN*

inner layer's neuron carry the information of its output values as well as its previous layer's output values.

3.2.3.4 Bias Gradient Estimation

The bias is a parameter whose contribution to the loss needs to be estimated in order to properly train the model.

Proposition 3.2.3. *The bias gradient of a 1DPNN neuron can be estimated using the following formula:*

$$\forall l \in [\![1, L]\!], \forall i \in [\![1, N_l]\!], \frac{\partial \epsilon}{b_i^{(l)}} = \sum_{m=0}^{M_l - 1} \left(\frac{\partial \epsilon}{\partial y_i^{(l)}} \odot \frac{\partial y_i^{(l)}}{\partial x_i^{(l)}} \right)(m) = \sum_{m=0}^{M_l - 1} \frac{\partial \epsilon}{\partial x_i^{(l)}}(m).$$

Proof. Using the differential of ϵ, we can determine the gradient of the loss with respect to the bias as such:

$$\forall l \in [\![1, L]\!], \forall i \in [\![1, N_l]\!], \frac{\partial \epsilon}{b_i^{(l)}} = \sum_{m=0}^{M_l - 1} \left(\frac{\partial \epsilon}{\partial y_i^{(l)}} \odot \frac{\partial y_i^{(l)}}{\partial x_i^{(l)}} \odot \frac{\partial x_i^{(l)}}{\partial b_i^{(l)}} \right)(m).$$

And since from Eq. (3.1), we can notice that $\frac{\partial x_i^{(l)}}{\partial b_i^{(l)}}(m) = 1, \forall l \in [\![1, L]\!], \forall (i, m) \in [\![1, N_l]\!] \times [\![0, M_l - 1]\!]$, we obtain the desired expression. ■

Remark 3.2.6. *The bias gradient formula of a 1DPNN neuron is the same as that of a 1DCNN neuron regardless of the degree of the polynomial approximation.*

3.2.3.5 Training Procedure

Given a tuple (X, Y) representing an input and a desired output, we generate $\hat{Y}$ using a defined architecture of the 1DPNN, then calculate $\frac{\partial \epsilon}{\partial Y_L}$ directly from the loss expression, in order to determine the weight gradients and the bias gradients for the output layer. Then using $\frac{\partial \epsilon}{\partial Y_L}$ and Eq. (3.6), calculate the output gradients, the weight gradients and the bias gradients of the previous layer. Repeat the process until reaching the first layer. After computing the gradients, we use gradient descent to update the weights as such:

$$\forall l \in [\![1, L]\!], \forall (i, j, d) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!] \times [\![1, D_l]\!], \left(w_{ijd}^{(l)}\right)^{(t+1)} = \left(w_{ijd}^{(l)}\right)^{(t)} - \eta \frac{\partial \epsilon}{\partial w_{ijd}^{(l)}}, \tag{3.11}$$

where η is the learning rate and t is the epoch. The same goes for the updating the biases:

$$\forall l \in [\![1, L]\!], \forall i \in [\![1, N_l]\!], \left(b_i^{(l)}\right)^{(t+1)} = \left(b_i^{(l)}\right)^{(t)} - \eta \frac{\partial \epsilon}{b_i^{(l)}}.$$

3.2.4 1DPNN Weight Initialization

Since the 1DPNN model uses polynomials to generate non-linear filtering, it is highly likely that, due to the fact that every feature map of every layer is raised to a power higher than 1, the weight updates become exponentially big (gradient explosion) or exponentially small (gradient vanishing), depending on the nature of the activation functions that are used. To remedy that, the weights have to be initialized in a way that the highest power of any feature map is associated with the lowest weight values.

Definition 3.2.2. *Let $\mathcal{R}(\alpha_l), l \in [\![1, L]\!]$ be a probability law with a parameter vector α_l used to initialize the weights of any layer l. The proposed weight initialization is defined as such:*

$$\forall l \in [\![1, L]\!], (i, j, d) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!] \times [\![1, D_l]\!], w_{ijd}^{(l)} \sim \frac{\mathcal{R}(\alpha_l)}{d!}.$$

Remark 3.2.7. *This initialization offers the advantage of allowing the use of any known deep-learning initialization scheme such as the Glorot Normalized Uniform Initialization [53] while adapting it to the weights associated with any degree. However, this does not ensure that there will be no gradient explosion as it only provides an initial insight on how the weights should evolve. Completely avoiding gradient explosion can be achieved by either choosing activation functions bounded between -1 and 1, by performing weight or gradient clipping or by using weight or activity regularization.*

3.2.5 1DPNN Theoretical Computational Complexity Analysis

The use of polynomial approximations in the 1DPNN model introduces a level of complexity that has to be quantified in order to estimate the gain of using it against using the regular convolutional model. Therefore, a thorough analysis is conducted with the aim to determine the complexity of the 1DPNN model with respect to the complexity of the 1DCNN model for both the forward propagation and the learning process (forward propagation + backpropagation).

Definition 3.2.3. *Let γ be a mathematical expression that can be brought to a combination of summations and products. We define $\mathcal{C}$ as a function that takes as input a mathematical expression such as γ and outputs an integer designating the number of operations performed to calculate that expression so that $\mathcal{C}(\gamma) \in \mathbb{N}$.*

Example 1. *Let* $N \in \mathbb{N}^*$ *and* $\gamma = \sum_{i=1}^{N} i^3$, *then* $\mathcal{C}(\gamma) = 2N + N - 1 = 3N - 1$ *because* $N - 1$ *summations are performed, and 2 products are performed* N *times.*

Example 2. $\forall z \in \mathbb{C}, 0 \leq \mathcal{C}(z) \leq 2$ *because a complex number can be written as* $z = a + ib, (a, b) \in \mathbb{R}^2$.

Remark 3.2.8. $\mathcal{C}$ *does not take into account any possible optimization such as the one for the exponentiation in Example 1 which can have a complexity of* $\mathcal{O}(\log_2 m)$ *where* m *is the exponent, nor does it take into account any possible simplification of a mathematical expression such as* $\sum_{i=1}^{N} i^3 = \frac{N^2(N+1)^2}{4}$. *Therefore,* $\mathcal{C}$ *provides an upper bound complexity that is independent of any implementation.*

Since the smallest computational unit in both models is the neuron, the complexity is calculated at its level for every operation performed during the forward propagation and the backpropagation. Moreover, every 1DPNN operation's complexity denoted by $\mathcal{C}_p$ is calculated as a function of the corresponding 1DCNN operation's complexity denoted by $\mathcal{C}_c$ since the aim is to compare both models with each other.

3.2.5.1 Forward Propagation Complexity

The forward propagation complexity is a measure relevant to the estimation of how complex a trained 1DPNN neuron is and can provide an insight on how complex it is to use a trained model compared to using a trained 1DCNN model.

Proposition 3.2.4. *The computational complexity of a 1DPNN neuron's forward propagation with respect to that of a 1DCNN neuron is given by the following formula:*

$$\forall l \in [\![1, L]\!], \forall i \in [\![1, N_l]\!], \mathcal{C}_p\left(y_i^{(l)}\right) = D_l \mathcal{C}_c\left(y_i^{(l)}\right) + (D_l - 1)\left(\frac{1}{2} M_{l-1} N_{l-1} D_l - 2M_l\right). \tag{3.12}$$

Proof. The 1DPNN forward propagation is fully defined by Eq. (3.1), which can also be interpreted as the 1DCNN forward propagation if the degree of the polynomials is 1. Therefore, we can determine the 1DPNN complexity as a function of the 1DCNN complexity by first determining the complexity of $x_i^{(l)}$ before adding the biases as such:

$$\begin{aligned} &\forall l \in [\![1, L]\!], \forall i \in [\![1, N_l]\!], \\ &\mathcal{C}_p\left(x_i^{(l)} - b_i^{(l)}\right) = \sum_{d=1}^{D_l} \left(\mathcal{C}_c\left(x_i^{(l)} - b_i^{(l)}\right) + (d-1) M_{l-1} N_{l-1}\right) = D_l \mathcal{C}_c\left(x_i^{(l)} - b_i^{(l)}\right) \\ &\qquad\qquad + \frac{1}{2} D_l (D_l - 1) M_{l-1} N_{l-1}. \end{aligned} \tag{3.13}$$

Assuming that the activation functions are atomic meaning that their complexity is $\mathcal{O}(1)$, we have:

$$\forall l \in [\![1, L]\!], \forall i \in [\![1, N_l]\!], \begin{cases} \mathcal{C}_c\left(x_i^{(l)}\right) & = \mathcal{C}_c\left(x_i^{(l)} - b_i^{(l)}\right) + M_l \\ \mathcal{C}_c\left(y_i^{(l)}\right) & = \mathcal{C}_c\left(x_i^{(l)}\right) + M_l \\ \mathcal{C}_p\left(x_i^{(l)}\right) & = \mathcal{C}_p\left(x_i^{(l)} - b_i^{(l)}\right) + M_l \\ \mathcal{C}_p\left(y_i^{(l)}\right) & = \mathcal{C}_p\left(x_i^{(l)}\right) + M_l \end{cases}. \tag{3.14}$$

By using the relationships in Eq. (3.14) in Eq. (3.13), we obtain the desired expression. ■

Remark 3.2.9. *Eq. (3.12) shows that the forward propagation's complexity of the 1DPNN does not scale linearly with the degrees of the polynomials, which means that a 1DPNN neuron with degree D_l is more computationally complex than D_l 1DCNN neurons despite having less trainable parameters (same number of weights but only 1 bias). However, this complexity can become linear since Y_{l-1}^d can be calculated only once, stored in memory and used for all neurons in layer l.*

3.2.5.2 Learning Complexity

The learning complexity is a measure relevant to the estimation of how complex it is to train a 1DPNN neuron and can provide an overall insight on how complex a model is to train compared to training a 1DCNN model. Since the learning process of the inner layers' neurons is more complex than the output layer's neurons, which can learn faster by estimating the output gradient directly from the loss function, the learning complexity will only be determined for the inner layers' neurons.

Proposition 3.2.5. *The computational complexity of the learning process of a 1DPNN inner layer's neuron denoted by $\mathcal{L}_p^{(l)}$ is given by the following formula:*

$$\forall l \in [\![1, L-1]\!], \mathcal{L}_p^{(l)} = D_l \mathcal{L}_c^{(l)} + (D_l - 1)\left(\left(M_{l-1}N_{l-1} + \frac{1}{2}M_l\right)D_l - M_l\right), \tag{3.15}$$

where $\mathcal{L}_c^{(l)}$ is the learning complexity of a 1DCNN neuron in a layer l.

Proof. The difference between the two models in the backpropagation phase resides in the weight gradient estimation and the output gradient estimation. In fact, the bias gradient estimation is the same for both models so there is no need to quantify its complexity. Since the weight gradient estimation is dependent on the output gradient estimation, the output gradient estimation will be quantified first. From Eq.(3.6), we can quantify the output gradient estimation complexity

for the 1DPNN model as :

$$\forall l \in [\![1, L-1]\!], j \in [\![1, N_l]\!],$$
$$\mathcal{C}_p\left(\frac{\partial \epsilon}{\partial y_j^{(l)}}\right) = \mathcal{C}_c\left(\frac{\partial \epsilon}{\partial y_j^{(l)}}\right) + \sum_{d=2}^{D_l}\left(\mathcal{C}_c\left(\frac{\partial \epsilon}{\partial y_j^{(l)}}\right) + 2M_l + (d-2)M_l\right) = D_l\mathcal{C}_c\left(\frac{\partial \epsilon}{\partial y_j^{(l)}}\right) + \frac{1}{2}(D_l+2)(D_l-1)M_l.$$

Since the 1DPNN neuron introduces D_l times more weights than the 1DCNN neuron, the complexity of calculating the weight gradients for a pair of neurons (i, j) as in Eq. (3.2) is the sum of their corresponding weight gradients with respect to the degree, as:

$$\forall l \in [\![1, L]\!], \forall (i,j) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!],$$
$$\mathcal{C}_p\left(\frac{\partial \epsilon}{\partial w_{ij}^{(l)}}\right) = \sum_{d=1}^{D_l}\mathcal{C}_p\left(\frac{\partial \epsilon}{\partial w_{ijd}^{(l)}}\right) = \sum_{d=1}^{D_l}\left(\mathcal{C}_c\left(\frac{\partial \epsilon}{\partial w_{ij}^{(l)}}\right) + (d-1)M_{l-1}\right) = D_l\mathcal{C}_c\left(\frac{\partial \epsilon}{\partial w_{ij}^{(l)}}\right) + \frac{1}{2}D_l(D_l-1)M_{l-1}.$$

Since one 1DPNN neuron in a layer l has D_l times more weights than a 1DCNN neuron, its weight update complexity is D_l times that of the 1DCNN. Therefore, the complexity of Eq. (3.11) is:

$$\forall l \in [\![1, L]\!], \forall (i,j) \in [\![1, N_l]\!] \times [\![1, N_{l-1}]\!], \mathcal{C}_p\left(\left(w_{ij}^{(l)}\right)^{(t+1)}\right) = D_l\mathcal{C}_c\left(\left(w_{ij}^{(l)}\right)^{(t+1)}\right).$$

From the above expressions, we can determine the learning complexity of any neuron which will be the sum of the forward propagation complexity and the backward propagation complexity. Therefore, the learning complexity denoted by $\mathcal{L}_{p,i}^{(l)}$ of a 1DPNN inner layer's neuron is:

$$\forall l \in [\![1, L-1]\!], i \in [\![1, N_l]\!], \mathcal{L}_{p,i}^{(l)} = \mathcal{C}_p\left(y_i^{(l)}\right) + \mathcal{C}_p\left(\frac{\partial \epsilon}{\partial y_i^{(l)}}\right) + \sum_{j=1}^{N_{l-1}}\left(\mathcal{C}_p\left(\frac{\partial \epsilon}{\partial w_{ij}^{(l)}}\right) + \mathcal{C}_p\left(\left(w_{ij}^{(l)}\right)^{(t+1)}\right)\right).$$

Since we suppose that the network is fully connected, $\mathcal{L}_{p,i}^{(l)}$ does not depend on i, thus we can replace it by $\mathcal{L}_p^{(l)}$. Therefore, by replacing each complexity in the above equation by their full expressions, we obtain the desired expression where:

$$\forall l \in [\![1, L-1]\!], i \in [\![1, N_l]\!], \mathcal{L}_c^{(l)} = \mathcal{C}_c\left(y_i^{(l)}\right) + \mathcal{C}_c\left(\frac{\partial \epsilon}{\partial y_i^{(l)}}\right) + \sum_{j=1}^{N_{l-1}}\left(\mathcal{C}_c\left(\frac{\partial \epsilon}{\partial w_{ij}^{(l)}}\right) + \mathcal{C}_c\left(\left(w_{ij}^{(l)}\right)^{(t+1)}\right)\right).$$

■

3.3 Implementation

This section describes the application programming interface (API) used to implement the 1DPNN model as well as an evaluation of the implementation in the form of an experimental computational complexity analysis for the forward propagation and the learning process of a 1DPNN neuron.

3.3.1 Keras Tensorflow API Implementation

Tensorflow [54] is an API created by Google that is widely used for graphics processing unit (GPU)-based symbolic computation and especially for neural networks. It allows the implementation of a wide range of models and automatically takes into account the usual derivatives without the need to define them manually. However, Tensorflow is a low-level API that involves a lot of coding and memory management to define a simple model. The 1DPNN model is mainly based on convolutions between inputs and filters, so it can be considered as an extension of the basic 1DCNN with slight changes. Therefore, there is no need to use such a low-level API like Tensorflow or CUDA [55] to define the 1DPNN, so the Keras API was used to implement the model.
Keras [56] is a high level API built on top of Tensorflow that provides an abstraction of the internal operations performed to compute gradients or anything related. Keras makes it possible to build a network as a combination of layers whether they are stacked sequentially or not. It uses the concept of layers which is the key element to perform any operation. This allows the definition of custom layers that can be jointly used with predefined layers, thus, allowing the creation of a heterogeneous network composed of different types of layers. In Keras, there is only the need to define the forward propagation of 1DPNN since convolutions and exponents are considered basic operations, and the API takes care of storing the gradients and using them to calculate the weight updates.

3.3.2 Experimental Computational Complexity Analysis

In order to evaluate the efficiency of the implementation, we compare the forward propagation complexity and the learning process complexity of a 1DPNN neuron with respect to the theoretical upper bound complexities determined in Section 3.2.5, as well as with a 1DCNN neuron's complexity, and a 1DPNN-equivalent neuron by varying the degrees of the polynomials in a given range. This experimental analysis is designed to show that the theoretical complexity is indeed an upper bound and that, as stated in Section 3.2.5, a 1DPNN neuron with a degree D is actually more complex than D 1DCNN neurons, which is the genesis of the idea of a 1DPNN-equivalent neuron.

3.3.2.1 1DPNN-Equivalent Network

Theorem 3.3.1. *Any 1DPNN with $L \geq 1$ layers can be transformed into a 1DCNN with $L+1$ layers that has the same number of parameters as the 1DPNN.*

Proof. Let $L \in \mathbb{N}^*$ be the number of layers of a 1DPNN and $\forall l \in [\![1, L+1]\!]$, let N'_l be the number of neurons in a 1DCNN layer. Given any inner 1DPNN layer, we can create a 1DCNN layer that has the same number of parameters using the equality between the number of parameters of each model's layer as such:

$$\forall l \in [\![1, L-1]\!], N'_l N'_{l-1} K_l + N'_l = N_l N_{l-1} K_l D_l + N_l,$$

where $N'_0{=}N_0$ since the input layer remains the same. We then determine N'_l from that equation as follows:

$$\forall l \in [\![1, L-1]\!], N'_l = \left\lfloor \frac{N_l(N_{l-1}K_l D_l + 1)}{N'_{l-1}K_l + 1} + \frac{1}{2} \right\rfloor, \tag{3.16}$$

where N'_l is rounded by adding $1/2$ and applying the floor function since it should be an integer. If we determine N'_L in the same manner as we determine the number of neurons in the inner layers, we will change the number of neurons in the output layer of the 1DCNN, which is not a desired effect. To remedy that problem, we add another 1DCNN layer with $N'_{L+1} = N_L$ neurons having a filter size of 1 in the 1DCNN, and we adjust the number of neurons in layer L so that the number of parameters in layer L and layer $L+1$ equals the number of parameters in the 1DPNN layer L using the following equality:

$$N'_L N'_{L-1} K_L + N'_L + N_L N'_L + N_L = N_L N_{L-1} K_L D_L + N_L.$$

By extracting N'_L from this expression and by rounding it, we end up with

$$N'_L = \left\lfloor \frac{N_L N_{L-1} K_L D_L}{N'_{L-1}K_L + N_L + 1} + \frac{1}{2} \right\rfloor. \tag{3.17}$$

■

Remark 3.3.1. *We call any 1DCNN recurrently constructed using the aforementioned theorem and Eqs. (3.16) and (3.17) a 1DPNN-equivalent network because it has the same number of parameters as the 1DPNN it was constructed from. However, their respective search spaces and computational complexities are generally not equivalent.*

Remark 3.3.2. *In a 1DPNN having only 1 layer ($L = 1$), $\left\lfloor \frac{N'_L}{N_L} + \frac{1}{2} \right\rfloor$ 1DCNN neurons are considered equivalent to only one 1DPNN neuron, and thus comes the concept of a 1DPNN-equivalent neuron. Comparing a 1DPNN neuron with its equivalent 1DCNN neuron is indeed helpful to determine the gain or loss of using one over the other by providing an insight on how to estimate the balance between searching features in a more complex search space and searching less complex features in less time.*

3.3.2.2 Experimental Setup

Since the implementation is GPU-based, it is very difficult to estimate the actual execution time of a mathematical operation since it can be split among parallel execution kernels and since the memory bandwidth greatly impacts it. Nevertheless, for a given amount of data, we can roughly estimate how fast a mathematical operation was performed by running it multiple times and averaging. However, that estimate also includes accesses to the memory which are the slowest operations that can run on a GPU.
Despite this limitation, a rough estimate is used to determine the execution times of a 1DPNN neuron, a 1DCNN neuron and a 1DPNN-equivalent neuron with different hyperparameters. Various networks with 2 layers serving as a basis for the complexity estimation are created with the hyperparameters defined in Table 3.1 below. Recall that N_l is the number of neurons in layer l, M_l is the number of samples of the signals created from the neurons of layer l, K_l is the kernel size of the neurons in layer l and D_l is the degree of the polynomials that need to be estimated for each neuron in layer l. Since the output layer is a single 1DCNN neuron, its

TABLE 3.1: Hyperparameters for each layer of the networks used for the complexity estimation.

	Layer		
Hyperparameter	$l = 0$	$l = 1$	$l = 2$
M_l	100	76	52
N_l	2	10	1
K_l	-	25	25
D_l	-	$[\![1, 100]\!]$	1

execution time can be estimated separately from the first layer. That time will then be deducted from the overall execution time of the network. Subsequently, for each degree of polynomials in the previous set of hyperparameters, a 1DPNN is created, as well as its equivalent 1DCNN counterpart. 1000 forward propagations are first performed, then 1000 learning cycles (forward propagation and backpropagation) are performed. The execution times for 1 neuron are then estimated by deducting the average execution times of the last layer and then averaging over 1000 and dividing by N_1.
The theoretical complexities determined in Eqs. (3.12) and (3.15) are expressed in terms of

number of operations and need to be expressed in seconds. Therefore, given the execution times of the 1DPNN with $D_1 = 1$, we can estimate how long it would theoretically take to perform the same operations with a different polynomial degree. For instance, we can estimate the time T_1 it takes to perform a forward propagation as a function of the polynomial degree D_1 using Eq. (3.12) as such:

$$\forall D_1 \in [\![1, 100]\!], T_1(D_1) = D_1 T_0 + (D_1 - 1)(c_1 D_1 - c_2) T,$$

where

- T_0 is the forward propagation execution time of a 1DCNN neuron,
- $c_1 = \frac{1}{2} M_0 N_0$,
- $c_2 = 2M_1$, and
- T is an estimate of the time it takes to perform one addition or one multiplication.

T can only be estimated from the fact that T_1 is an increasing function of D_1 which means that

$$\forall D_1 \in [\![1, 100]\!], \frac{\partial T_1}{\partial D_1}(D_1) = T_0 + (2c_1 D_1 - (c_1 + c_2)) T \geq 0.$$

This is equivalent to:

$$\forall D_1 \in [\![1, 100]\!], T \geq \frac{T_0}{c_1 + c_2 - 2c_1 D_1}.$$

Since $c_1 + c_2 - 2c_1 D_1$ is a decreasing function of D_1, the final estimation of T is

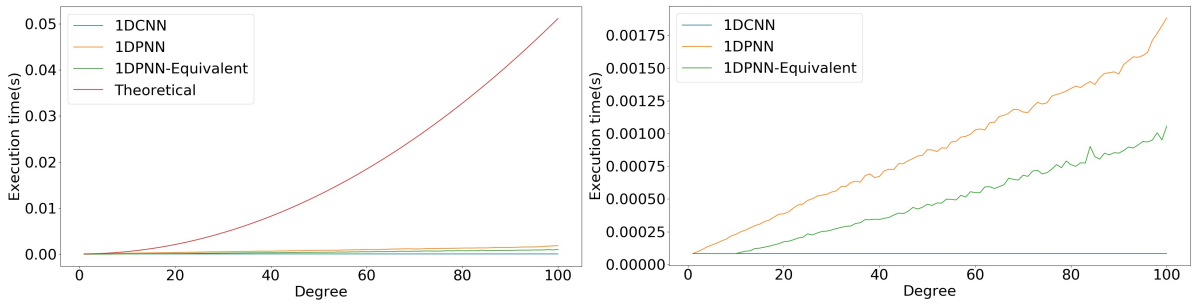
$$T = \frac{T_0}{c_1 + c_2}.$$

The same can be done for the theoretical learning complexity defined in Eq. (3.15) by replacing c_1, c_2 and T_0 accordingly.

3.3.2.3 Results

Figure 3.3 shows the evolution of the execution time of a neuron's forward propagation with respect to the degree of the polynomial for each of the 1DCNN neuron, 1DPNN neuron and 1DPNN-equivalent neuron. Figure 3.3(a) also shows the evolution of the theoretical complexity with respect to the degree. This confirms that Eq. (3.12) is indeed an upper bound complexity and that, with optimization, the forward propagation of a 1DPNN neuron can run in quasi-linear time as shown in Figure 3.3(b). Moreover, with the chosen hyperparameters, the 1DPNN neuron is, on average, 1.94 times slower than the 1DPNN-equivalent neuron, which confirms

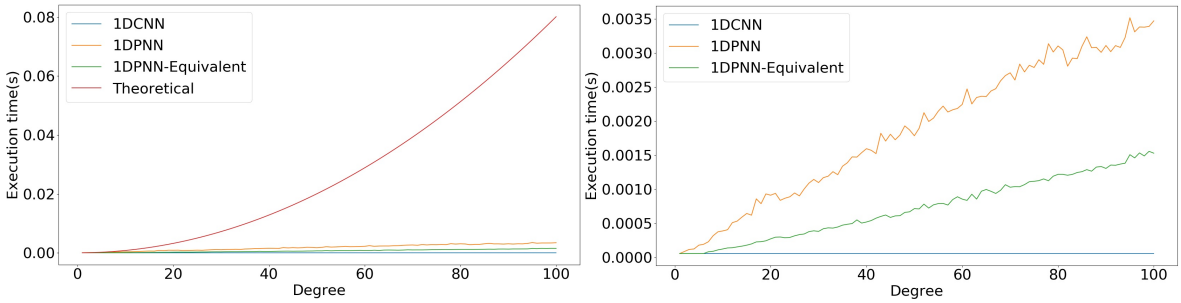
the expectation that a 1DPNN neuron is more complex than a 1DPNN-equivalent neuron. Figure 3.4 shows the evolution of the execution time of neuron's learning process with respect



(a) With theoretical execution time. (b) Without theoretical execution time.

FIGURE 3.3: Forward propagation execution time for 1 neuron of each network.

to the degree of the polynomial for each of the 1DCNN neuron, 1DPNN neuron and 1DPNN-equivalent neuron. Figure 3.4(a) showing the evolution of the theoretical complexity with respect to the degree confirms that Eq. (3.15) is in fact an upper bound, and Figure 3.4(b) shows that the learning process of a 1DPNN neuron can also run in quasi-linear time, as stated in Section 3.2.5. However, the 1DPNN neuron's learning process is, on average, 2.72 times slower than the 1DPNN-equivalent neuron, as it is to be expected from Eq. (3.15).



(a) With theoretical execution time. (b) Without theoretical execution time.

FIGURE 3.4: Learning process execution time for 1 neuron of each network.

3.4 Experiments And Results

Since 1DPNN is basically an extension of 1DCNN, there is a need to compare our proposed model's performance with the 1DCNN model's performance under certain conditions. In fact, the number of parameters in a 1DPNN differs from the number of parameters in a 1DCNN with the same topology, and the computational complexity of a 1DPNN is higher than the computational complexity of a 1DPNN-equivalent 1DCNN. Hence, we devised 3 strategies that to compare the two models, 1DPNN and 1DCNN:

1. **Topology-wise** comparison which consists in comparing the performances of a 1DCNN and a 1DPNN that have the same topology.
2. **Parameter-wise** comparison which consists in comparing the performances of a 1DCNN and a 1DPNN that have the same number of parameters. The 1DCNN is created according to the definition of the 1DPNN-equivalent network detailed in Section 3.3.2.
3. **Performance-wise** comparison which consists in comparing the spatial and the computational complexities of a 1DCNN and a 1DPNN that achieve equal or nearly equal performance based on an evaluation metric. A performance-wise network is constructed by gradually adding neurons to the layers of a 1DPNN-equivalent network until reaching the same performance as the reference 1DPNN.

Figure 3.5 shows an overview of the methodology used to evaluate the 1DPNNs performance. Both models were evaluated on the same problems consisting of 2 classification problems which

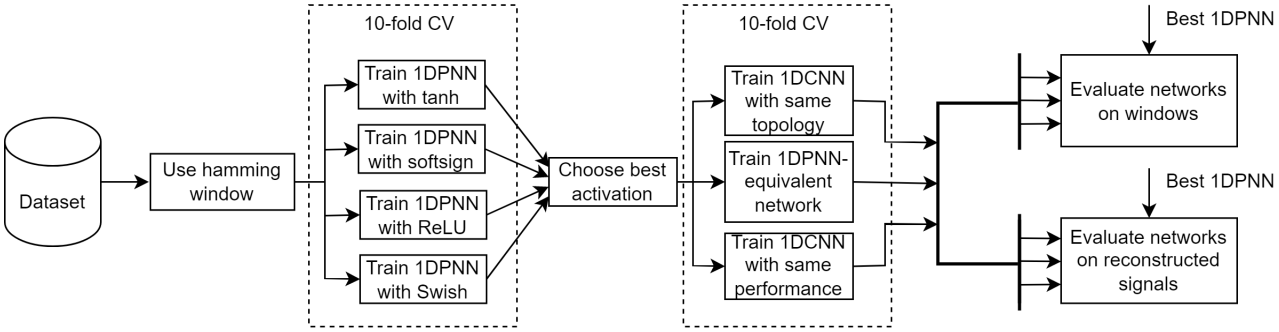


FIGURE 3.5: Block diagram of the experimental methodology.

are musical note recognition on monophonic instrumental audio signals and spoken digit recognition; and 1 regression problem, audio signal denoising at 0db Signal-to-Noise Ratio (SNR) with Additive White Gaussian Noise (AWGN). Moreover, the spatio-temporal complexities of the 1DPNN and the 3 architectures of 1DCNN were evaluated on each problem. The same learning rate (10^{-3}) and the same gradient optimizer (ADAM [57]) were used for all networks on any given problem. Furthermore, we also experimentally studied the influence of the activation function on the performance and convergence of the 1DPNN since the presence of the term $\left(y_j^{(l-1)}\right)^d$ in Eq. (3.2) and the term $\left(y_j^{(l)}\right)^{d-1}$ in Eq. (3.6) strongly suggests that a gradient explosion is very likely to occur during training if the activation function is not bounded. As a result, we chose to evaluate the influence of the following activation functions on the 1DPNN:

1. Tangent hyperbolic which is bounded between -1 and 1.
2. Softsign [31] which is also bounded between -1 and 1 but has a higher saturation limit than the tangent hyperbolic.

3. Rectified Linear Unit (ReLU) [32] which produces a sparse representation of features due to its hard saturation to zero for negative values and its linear output for positive values.
4. Swish [34] which circumvents the hard saturation of the ReLU to allow for more flexibility during learning.

Since the audio signals are bounded between -1 and 1 and ReLU and Swish are highly likely to nullify a negative input, the signals are normalized between 0 and 1 when these activation functions are used.
All problems involve 1 dimensional audio signals sampled at a given sampling rate and with a specific number of samples. However, the datasets used for the experiments contain either signals that have a high number of samples, or signals that have different numbers of samples inside the same dataset. Due to technical limitations, creating a network that takes a high number of samples as input or different number of samples per signal is time-consuming and irrelevant because the main aim of the experiments is to compare both models with each other, and not to produce state-of-the-art results on the considered problems. Therefore, the sliding window technique described below has been adopted for both problems as a preprocessing step. In the following experiments, $PNNLayer(x, y, z, f)$ refers to a PNN layer having x neurons, a mask size y, a polynomial degree z, and an activation function f. $Conv1D(x, y, f)$ refers to a convolutional layer having x neurons, a mask size y, and an activation function f. Finally, $MLP(x, f)$ refers to a multilayer perceptron layer with x neurons, and an activation function f.

3.4.1 Sliding Window Technique

The sliding window technique consists of applying a sliding observation window on a signal to extract a certain number of consecutive samples with a certain overlap between two consecutive windows. Usually, the samples that are contained in an observation window are multiplied by certain weights that constitute a window function. The technique is useful when dealing with signals that have a high number of samples and when studying a locally stationary signal property (such as the frequency of a tone which lasts for a certain duration).

Definition 3.4.1. *Let $x = (x_0, ..., x_{N-1}) \in \mathbb{R}^N$ be a temporal signal. Let $w \in [\![1, N]\!]$ be the size of the window. Let $\alpha \in [0, 1[$ be the overlap ratio between two consecutive windows. Let $h = (h_0, ..., h_{w-1}) \in [0, 1]^w$ be a window function. We define $\mathcal{W}_{w,\alpha}(h, x)$ as the set of all the observed windows for the signal x with respect to the window function h, and we express it as such:*

$$\mathcal{W}_{w,\alpha}(h, x) = \{(h_0 x_i, ..., h_{w-1} x_{i+w-1}) | i = \lfloor n(1-\alpha)w \rfloor, n \in [\![0, \left\lfloor \frac{N-w}{(1-\alpha)w} \right\rfloor]\!]\}.$$

Remark 3.4.1. *The sliding window has the effect of widening the spectrum of the signal due to the Heisenberg-Gabor uncertainty principle [58], thus, distorting it in a certain measure. Therefore, the size of the window and the window function have to be chosen so that a given observation window of the signal can respectively contain enough information to process it accordingly, and as less distortion as possible to preserve the spectral integrity of the original signal. The window function used for all the problems is the Hamming window [59].*

3.4.2 Classification Problems

Since 1DPNN and 1DCNN are based on convolutions, they are basically used as regressors in the form of feature extractors. Their objective is to extract temporal features that will be used to classify the signals. In the case of this work, they are used to create a feature extractor block that will be linked to a classifier which will classify the input signals based on the features extracted as described in Figure 3.6, where x is a temporal signal, $(f_1, ..., f_p)$ are p features extracted using either 1DPNN or 1DCNN, and $(c_1, ...c_q)$ are probabilities describing whether the signal x belongs to a certain class (there are q classes in general). The classifier will be a multilayer perceptron (MLP) in both problems and the metric used to evaluate the performance of the models is the classification accuracy. However, since the sliding window technique is used on all

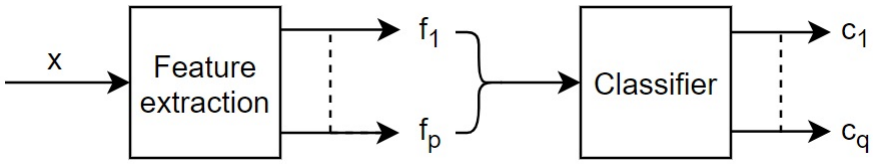


FIGURE 3.6: Block diagram of the classification procedure.

signals, the classifier will only be able to classify one window at a time. Therefore, for a given signal x, a window size w, an overlap ratio α and a window function h as defined in the previous subsection, we obtain $|\mathcal{W}_{w,\alpha}(h, x)|$ different classes for the same signal, where $|\mathcal{W}_{w,\alpha}(h, x)|$ is the cardinal of $\mathcal{W}_{w,\alpha}(h, x)$. Thus, we define the class of the signal as the statistical mode of all the classes, estimated from every window extracted from the signal. For instance, if we have 3 classes and a signal gets decomposed in 10 windows such that 5 windows are classified as class 0, 2 windows as class 1, and 3 windows as class 2, the signal will be classified as class 0 since it is the class that occurs most frequently in the estimated classes. As a result, the performance of each model is evaluated with respect to the per-window accuracy and the per-signal accuracy.

3.4.2.1 Note Recognition on Monophonic Instrumental Audio Signals

The dataset used for this problem is NSynth [20] which is a large scale dataset of annotated musical notes produced by various instruments. It contains more than 300,000 signals sampled at 16kHz, and lasting 4 seconds each. The usual musical range is composed of 88 different notes

which represent our classes. Since the dataset is huge, we only use 12,678 signals for training, and 4096 for testing. We also use a sliding window with a window size $w = 1600$ (100ms) and an overlap of 0.5 which makes the training set composed of 728,828 signals and the test set composed of 235,480 signals. We then use 10-fold cross validation [60] on the training set so that we estimate the average performance for every topology used on the test set. Four different networks were created:

1. A network composed of a 1DPNN feature extractor.
2. A network composed of a 1DCNN feature extractor with the same topology as the previous one.
3. A network composed of a 1DPNN-equivalent feature extractor with the same number of parameters as the 1DPNN.
4. A network composed of a 1DCNN feature extractor that achieves the same accuracy as the 1DPNN.

Table 3.2 shows the hyperparameters of each layer of the 1DPNN, the 1DPNN-equivalent network, and the performance-equivalent network. The 1DPNN is trained on the windows extracted

Table 3.2: Networks' topologies for note recognition.

1DPNN	1DPNN-equivalent network	Performance-equivalent network
PNNLayer(12, 49, 1, tanh)	Conv1D(12, 49, tanh)	Conv1D(18, 49, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
PNNLayer(12, 25, 1, tanh)	Conv1D(12, 25, tanh)	Conv1D(14, 25, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
PNNLayer(12, 13, 2, tanh)	Conv1D(24, 13, tanh)	Conv1D(24, 13, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
PNNLayer(12, 7, 3, tanh)	Conv1D(26, 7, tanh)	Conv1D(26, 7, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
PNNLayer(12, 3, 5, tanh)	Conv1D(12, 3, tanh)	Conv1D(12, 3, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
Flatten	Flatten	Flatten
MLP(96, ReLU)	MLP(96, ReLU)	MLP(96, ReLU)
MLP(88, softmax)	MLP(88, softmax)	MLP(88, softmax)

from the signals with the activation functions presented in Section 3.4 and the 10-fold statistics are reported in Table 3.3 below. The results show that the 1DPNN with the tangent hyperbolic activation function achieves the highest minimum, maximum and average accuracy overall and that the Swish activation function is indeed a better alternative than ReLU due to its flexibility in processing negative values [34]. However, during the experiments, both ReLU and Swish exhibited unstable behavior in the first epochs of training which was expected due to the fact

that very high weight updates occur when the network begins learning. This reinforces the need for a bounded activation function when using the 1DPNN. The networks in Table 3.2 use the

TABLE 3.3: Accuracy per window for each activation function for note recognition over 10 folds.

	Activation function			
Statistic (%)	Tanh	Softsign	ReLU	Swish
Minimum accuracy	**84.73**	84.39	82.34	83.62
Maximum accuracy	**86.03**	85.7	82.91	85.11
Average accuracy	**85.11**	85.02	82.42	84.2

tangent hyperbolic function and are also trained on the windows extracted from the signals, and their minimum, maximum and average accuracy over the 10-fold cross validation are reported in Table 3.4 below in percentages. We can see that the average accuracy per window of the 1DPNN is slightly better than the other networks, except the performance-equivalent one.

TABLE 3.4: Accuracy per window for each network for note recognition over 10 folds.

	Network			
Statistic (%)	1DPNN	1DCNN same topology	1DPNN-equivalent	Performance-equivalent
Minimum accuracy	84.73	83.84	84.6	**84.82**
Maximum accuracy	86.03	84.91	85.17	**86.07**
Average accuracy	85.11	84.47	84.93	**85.2**

Figure 3.7 also shows that 91% of the time, over 200 epochs, the average accuracy of the 1DPNN is higher than the 1DPNN-equivalent network. Moreover, the evolution of the average accuracy is very smooth and the 1DPNN shows a slightly faster convergence in the first 50 epochs.

The networks are evaluated on whether they can classify a window from a signal correctly,

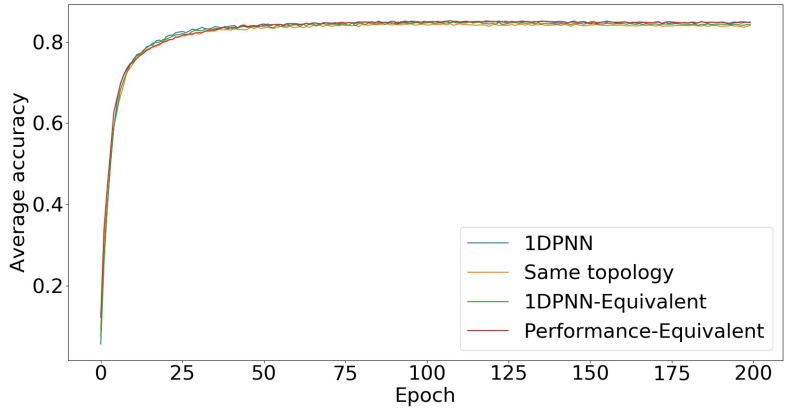


FIGURE 3.7: Average accuracy for each epoch of all the networks trained on the note recognition dataset.

but they should be evaluated on whether they classify a complete signal since the dataset is

originally composed of 4 second signals. In the case of this work, multiple windows are derived from a single signal that belongs to a certain class. Therefore, the windows are also considered as belonging to the same class of the signal that they are derived from. However, the models may give a different classification for each window, so to determine the class of a signal given the classification of its windows, we use the statistical mode of the different classifications. By performing this postprocessing step, we end up with the average accuracy per signal for each network, as shown in Table 3.5. We notice that the per-signal accuracies are better than the per-window accuracies for all networks, and that the 1DPNN is more accurate than the 1DPNN-equivalent network by 0.22% and is only 0.02% better at classifying the signals than the performance-equivalent network which has the best per-window accuracy statistics. Nevertheless, the performance-equivalent network is **1.1** times slower than the 1DPNN and uses **1.05** times more parameters as shown in Table 3.6 in which the number of parameters and the execution time of each network are reported. Therefore, the 1DPNN is able to extract more information from the data in less space and less time for the note recognition problem.

Table 3.5: Accuracy per signal for each network for note recognition over 10 folds.

	Network			
Statistic (%)	1DPNN	1DCNN same topology	1DPNN-equivalent	Performance-equivalent
Average accuracy	**88.81**	88.47	88.59	88.79

Table 3.6: Average spatio-temporal complexities for each network for note recognition.

	Network			
Complexity	1DPNN	1DCNN same topology	1DPNN-equivalent	Performance-equivalent
Spatial	71,344	65,728	71,490	75,116
Temporal(μs)	78	73	76	86

3.4.2.2 Spoken Digit Recognition

The dataset used for this problem is the Free Spoken Digits Dataset [21] which contains 2000 audio signals of different duration sampled at 8kHz consisting of people uttering digits from 0 to 9 which will represent the classes of the signals. In this work, 1800 signals are considered for training and 200 signals are considered for testing the networks. A closer look at the dataset shows that 83% of the signals last less than 500ms and that the voiced sections of the remaining 27% are contained within the first 500ms, leaving the rest of the durations filled with noise and/or silence. Since the dataset has signals of different durations, we use a sliding window with a window size of $w = 4000$ (500ms) and an overlap of 0.9 to ensure that one window contains enough information to classify the signal. This process yields a training set of 8333 signals and a test set of 796 signals. The sliding window only operates on the signals whose durations exceed 500ms while the signals that last less than 500ms are padded with zeros. 10-fold cross validation

is also used to evaluate the four networks. Table 3.7 shows the hyperparameters of each layer

TABLE 3.7: Networks' topologies for spoken digit recognition.

1DPNN	1DPNN-equivalent network	Performance-equivalent network
PNNLayer(8, 81, 1, tanh)	Conv1D(8, 81, tanh)	Conv1D(16, 81, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
PNNLayer(8, 25, 2,tanh)	Conv1D(8, 25, tanh)	Conv1D(16, 25, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
PNNLayer(8, 9, 2, tanh)	Conv1D(8, 9, tanh)	Conv1D(16, 9, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
PNNLayer(8, 9, 3, tanh)	Conv1D(24, 9, tanh)	Conv1D(16, 9, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
PNNLayer(8, 3, 6, tanh)	Conv1D(24, 3, tanh)	Conv1D(8, 3, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
Flatten	Conv1D(8, 1, tanh)	Flatten
MLP(64, ReLU)	Flatten	MLP(64, ReLU)
MLP(48, ReLU)	MLP(64, ReLU)	MLP(48, ReLU)
MLP(10, softmax)	MLP(48, ReLU)	MLP(10, softmax)
-	MLP(10, softmax)	-

of the 1DPNN, 1DPNN-equivalent network and the performance-equivalent network.
The 1DPNN is trained on the windows extracted from the signals with the activation functions presented in Section 3.4 and the 10-fold statistics are reported in Table 3.8 below. The results are highly similar to the ones obtained on the note recognition problem where the tangent hyperbolic activation function outperforms the other activation functions and the ReLU and Swish functions exhibit unstable behavior. The 10-fold accuracy statistics of each network is

TABLE 3.8: Accuracy per window for each activation function for spoken digit recognition over 10 folds.

	Activation function			
Statistic (%)	Tanh	Softsign	ReLU	Swish
Minimum accuracy	**93.59**	93.13	91.06	92.98
Maximum accuracy	**93.97**	93.82	92.51	93.6
Average accuracy	**93.71**	93.56	91.24	93.15

shown in Table 3.9 where the average accuracy of the 1DPNN is almost 1% better than the 1DPNN-equivalent network. Figure 3.8 shows that the average accuracies of all the networks start to stagnate from epoch 50 and that their evolution becomes stochastic in nature. Since the classification is performed window-wise, the average accuracy per signal can be estimated with the same principle used in the note recognition problem. Table 3.10 shows that the average accuracy per signal of the 1DPNN surpasses that of the 1DPNN-equivalent network by almost 3% and that of the performance-equivalent by only 0.06% which is to be expected. However,

Table 3.11 shows that the performance-equivalent network has **1.11** times more parameters than the 1DPNN and runs **1.08** times slower than the 1DPNN. Therefore, the 1DPNN can extract more information in less space and less time for the digit recognition problem.

TABLE 3.9: Accuracy per window for each network for spoken digit recognition over 10 folds.

	Network			
Statistic (%)	1DPNN	1DCNN same topology	1DPNN-equivalent	Performance-equivalent
Minimum accuracy	93.59	90.98	92.64	**93.63**
Maximum accuracy	93.97	91.27	92.91	**94.07**
Average accuracy	93.71	91.2	92.83	**93.96**

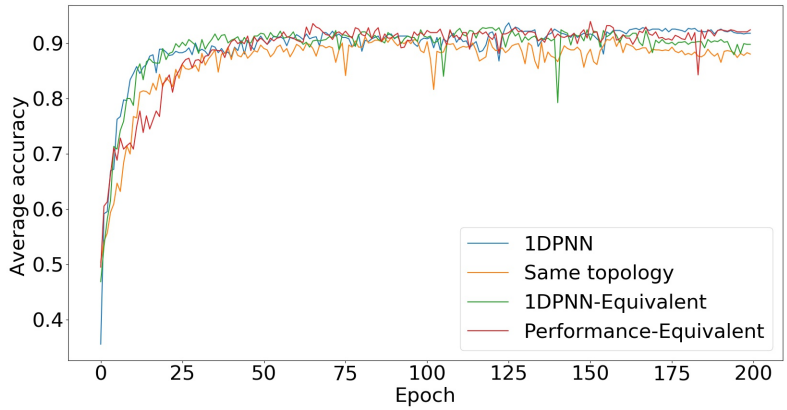


FIGURE 3.8: Average accuracy for each epoch of all the networks trained on the spoken digit recognition dataset.

TABLE 3.10: Accuracy per signal for each network for spoken digit recognition over 10 folds.

	Network			
Statistic (%)	1DPNN	1DCNN same topology	1DPNN-equivalent	Performance-equivalent
Average accuracy	**94.23**	91.31	93.11	94.17

TABLE 3.11: Average spatio-temporal complexities for each network for spoken digit recognition.

	Network			
Complexity	1DPNN	1DCNN same topology	1DPNN-equivalent	Performance-equivalent
Spatial	68,746	67,210	69,274	76,338
Temporal(μs)	97	93	96	105

3.4.3 Regression Problem: Audio Signal Denoising

Both 1DPNN and 1DCNN models take as input a signal and output a signal that usually has a lower temporal dimension due to border effects. However, in this regression problem, we need the output signal to have the same temporal dimension as the input signal. Therefore, we use zero-padding to avoid border effects. 4 different metrics are used to evaluate the performances of the models for this problem:

1. The Signal-to-Noise Ratio (SNR) measured in decibel (dB) defined as such:

   $$SNR = 10log_{10}\left(\frac{\mu_{s^2}}{\mu_{n^2}}\right),$$

   where μ_{s^2} is the mean square of the signal without noise, and μ_{n^2} is the mean square of the noise.

2. The Mean Squared Error (MSE) which is equivalent to μ_{n^2}.

3. The segmental SNR (SNRseg) [61] which is the average of the SNR values calculated on short segments of the signal.

4. The perceptual evaluation of speech quality (PESQ) score [62] which is an objective evaluation of subjective speech quality that takes into account a large range of signal distortions. The score ranges from -0.5 for very bad speech quality, to 4.5 for excellent speech quality.

The dataset used for this problem is the MUSDB18 [22] dataset containing 150 high quality full length stereo musical tracks (10 hours of recordings) with their isolated drums, bass, and vocal stems sampled at 44.1kHz. It is mainly used for source separation, but can also be used for instrument tracking or for noise reduction. The aim of this problem is to restore voice recordings that are drowned in AWGN making their individual SNR around 0dB. However, since the dataset is huge and the experiments are restricted by technical limitations, we take a small subset of the training set and the test set, downsample the voice tracks to 16kHz, and extract windows of 100ms to obtain 40,000 short clips (80% for training, 20% for testing).
All networks are then trained to estimate a clean window from a noisy one. The trained networks are then used as building blocks for an end-to-end denoising system which takes any noisy signal and outputs a cleaner signal of the same length. Figure 3.9 shows how an input signal x is processed into an output signal y where the "Model" block represents a denoising model, which in this case corresponds to any of the trained networks. The "Sliding window" block takes a signal x and outputs n windows where each is fed to the model that processes it into a denoised window. The n denoised windows are then divided by the Hamming window function and reconstructed into a signal y. The end-to-end system is then evaluated for each trained network using the 4 metrics described above on 700 singing clips of 10 seconds that

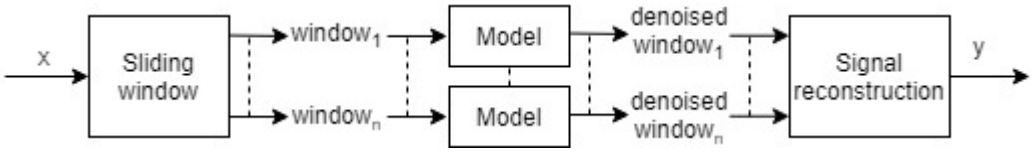


FIGURE 3.9: Block diagram of the end-to-end denoising system.

are drowned in AWGN. However, to estimate the generalization capability of the networks, the singing clips are corrupted into -5dB clips, 0dB clips, 5dB clips, 10dB clips and 15dB clips. Therefore 20 measures are reported per network.

The topologies used to solve the problem are detailed in Table 3.12 below. The 1DPNN is trained on the windows extracted from the signals with the activation functions presented in Section 3.4. However, since ReLU and Swish are unbounded and the output of the 1DPNN needs to have the same range as its input, we use the sigmoid activation function in the last layer when using ReLU and Swish in the inner layers. The 10-fold SNR statistics for each 1DPNN reported in Table 3.13 below show that although the tangent hyperbolic function has the highest minimum accuracy and average accuracy overall, the softsign achieved the highest maximum accuracy. Moreover, it shows that the Swish function is far more effective than the ReLU and that it almost performs as well as the tangent hyperbolic and the softsign. However, contrary to the classification experiments, the ReLU and the Swish activation functions did not exhibit any form of instability. This may be due to the use of the sigmoid function in the last layer which highly restricts the values of the gradients. The statistics of the SNR per

TABLE 3.12: Networks' topologies for audio signal denoising.

1DPNN	1DPNN-equivalent network	Performance-equivalent network
PNNLayer(16, 21, 5, tanh)	Conv1D(77, 21, tanh)	Conv1D(128, 21, tanh)
MaxPooling1D(2)	MaxPooling1D(2)	MaxPooling1D(2)
PNNLayer(16, 7, 5, tanh)	Conv1D(17, 7, tanh)	Conv1D(64, 7, tanh)
UpSampling1D(2)	UpSampling1D(2)	UpSampling1D(2)
PNNLayer(1, 11, 5, tanh)	Conv1D(5, 11, tanh)	Conv1D(32, 11, tanh)
-	Conv1D(1, 1, tanh)	Conv1D(1, 1, tanh)

window gathered using 10-fold cross validation are reported in Table 3.14 where we notice that the 1DPNN's average SNR is $0.12dB$ better than the 1DPNN-equivalent and very close to the performance-equivalent 1DCNN, which is to be expected. However, Figure 3.10 representing the evolution of the 10-fold average SNR per window over the epochs shows that the 1DPNN has a highly faster convergence than the other networks. Table 3.15 shows the performance evaluation of the end-to-end system built on top of the best of each network based on the 4 performance evaluation metrics described above and for different noise levels ranging from -5dB to 15dB. The corrupted signals are also evaluated with respect to the original ones in the "No

TABLE 3.13: SNR per window for each activation function for audio signal denoising over 10 folds.

	Activation function			
Statistic (%)	Tanh	Softsign	ReLU	Swish
Minimum accuracy	**9.25**	9.18	8.3	9.11
Maximum accuracy	9.34	**9.38**	8.71	9.27
Average accuracy	**9.28**	9.25	8.44	9.2

TABLE 3.14: SNR statistics per window for each network for audio signal denoising over 10 folds.

	Network			
Statistic (dB)	1DPNN	1DCNN same topology	1DPNN-equivalent	Performance-equivalent
Minimum SNR	**9.25**	8.63	9.11	9.2
Maximum SNR	9.34	8.94	9.23	**9.35**
Average SNR	9.28	8.88	9.16	**9.3**

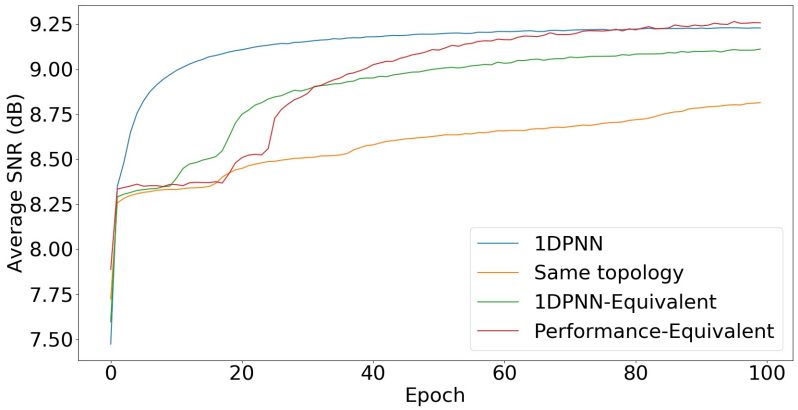


FIGURE 3.10: Average SNR in dB per window for each epoch of all the networks trained on audio signal denoising.

network" column. The overall results show that, for all the networks, the SNRseg is always much higher than the SNR which may be due to the fact that the networks perform better on voiced segments of the signal than on non-voiced segments (silence segments) of the signal. However, the SNRseg of the corrupted signals is always very inferior to the SNR which can be explained by the fact that the corruption of speech quality and intelligibility is far more severe than the corruption of the overall signal when using AWGN. The results show that for all noise levels except -5dB, the 1DPNN provides better results than all the other networks and shows a decent level of noise suppression notably for 0dB noisy signals with a 9.44dB segmental SNR representing a 24dB improvement compared to the corrupted signals and a 2.255 PESQ score which means that the restored speech is fairly intelligible.

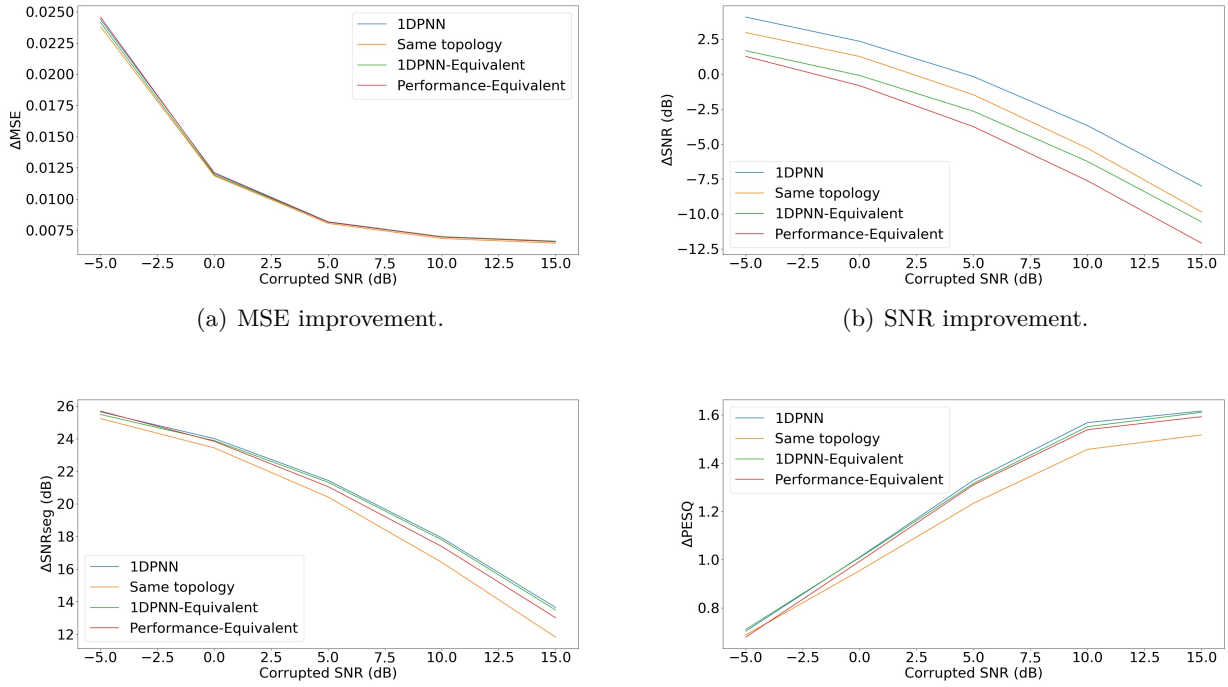
Figure 3.11 shows how the end-to-end system built on top of every best trained network improves

TABLE 3.15: Performance evaluation of the end-to-end system with respect to the best trained networks, 5 different levels of noise and 4 performance evaluation metrics. The "No network" column shows the corrupted signals evaluated with respect to the original ones.

		Network				
SNR	Metric	No network	1DPNN	1DCNN same topology	1DPNN-equivalent	Performance-equivalent
-5dB	MSE	0.0283	0.0039	0.0045	0.0041	**0.0037**
	SNR (dB)	-4.99	**1.82**	0.61	1.12	1.37
	SNRseg (dB)	-19.56	6.07	5.67	5.92	**6.13**
	PESQ score	1.15	1.858	1.843	**1.866**	1.834
0dB	MSE	0.0136	**0.0015**	0.0017	0.0016	0.0015
	SNR (dB)	5.23E-5	**2.34**	1.25	1.33	1.47
	SNRseg (dB)	-14.56	**9.44**	8.87	9.31	9.27
	PESQ score	1.24	**2.255**	2.19	2.253	2.23
5dB	MSE	0.0089	**0.0007**	0.0009	0.0008	0.0008
	SNR (dB)	4.99	**4.81**	3.51	2.33	1.23
	SNRseg (dB)	-9.56	**11.87**	10.85	11.76	11.49
	PESQ score	1.42	**2.74**	2.65	2.73	2.73
10dB	MSE	0.0074	**0.0004**	0.0006	0.0005	0.0005
	SNR (dB)	9.99	**6.31**	4.69	3.75	2.36
	SNRseg (dB)	-4.56	**13.34**	11.84	13.22	12.81
	PESQ score	1.69	**3.26**	3.14	3.24	3.23
15dB	MSE	0.007	**0.00039**	0.0005	0.0004	0.0004
	SNR (dB)	14.99	**6.99**	5.14	4.43	2.91
	SNRseg (dB)	0.43	**14.07**	12.25	13.91	13.44
	PESQ score	2.06	**3.68**	3.58	3.67	3.66

the 4 performance metrics with respect to the corrupted signal for different levels of noise. The improvement of a metric is calculated by determining the difference between the metric estimated from a given network's output and the metric estimated from the corrupted signals (the opposite is calculated for the MSE) such that an improvement in a metric is positive for a given network when that network enhances the corrupted signal with respect to that same metric. The main observation that can be drawn is that the PESQ score is the only metric that the networks improve better for higher SNR values whereas the improvements of the other metrics decrease as the SNR of the corrupted signals increase (as the noise becomes less severe). Furthermore, the only metric that all the networks fail to improve is the SNR for corrupted signals with an SNR of 5dB or higher despite an improvement in all other metrics. This may be due to the fact that the networks are better at enhancing the quality of the speech locally which explains the PESQ and SNRseg improvements, and that this local enhancement compensates the degradation that occurs in the non-voiced segments which leads to an improvement in the signal reconstruction and thus, the MSE. A closer look at the results also shows that the 1DPNN-equivalent network actually performs better than the performance-equivalent network on all noise levels greater or equal to 5dB. This may be due to the fact that the performance-equivalent network has more parameters which enables it to focus more on severe noise than on medium/low noise. Overall, the 1DPNN shows a better ability to generalize on different noise

levels. Moreover, the performance-equivalent network has **7.18** times more parameters than the 1DPNN and runs **1.19** times slower than the 1DPNN as shown in Table 3.16 below. This means that the 1DPNN is more efficient than the 1DCNN in audio signal denoising and can encapsulate more relevant information in less space and less time.



(a) MSE improvement.

(b) SNR improvement.

(c) SNRseg improvement.

(d) PESQ improvement.

FIGURE 3.11: Performance metric improvements of the end-to-end system built on top of every best trained network with respect to the corrupted signals for 5 different levels of noise (values between two noise levels are linearly interpolated).

TABLE 3.16: Average spatio-temporal complexities for each network for audio signal denoising.

	Network			
Complexity	1DPNN	1DCNN same topology	1DPNN-equivalent	Performance-equivalent
Spatial	11,553	2,337	11,820	83,009
Temporal(μs)	104	76	81	124

3.5 Chapter Summary

In this chapter, we have formally introduced a novel 1-Dimensional Polynomial Neural Network (1DPNN) model that induces a high degree of non-linearity starting from the initial layer in

an effort to produce compact topologies in the context of audio signal applications. Our experiments demonstrate that it has the potential to produce more accurate feature extraction and better regression than the conventional 1DCNN with less spatial and computational complexities. Furthermore, our model also shows faster convergence in the audio signal denoising problem and shows a significant gap in the performance compared to the 1DCNN which needs to use more space and time to produce the same results. We have also showed that the 1DPNN model converges with no instability when we use activation functions bounded between -1 and 1 (like *tanh*) or when all the degrees are set to 1 because it becomes equivalent to a 1DCNN.

Chapter 4

Polynomial Degree Reduction and NDPNN Generalization

4.1 Introduction

Polynomials expressed in the canonical basis are widely used in different scientific fields such as machine learning, statistics and computer science. They can be used in different regression and classification models in machine learning [63] [64], they can be used as the main building block for some statistical models [65], and they can also be used to compute hyperbolic and circular functions [66]. Although modern technology is able to rapidly exploit and manipulate polynomials, it is undeniable that saving computational and spatial resources is paramount for hardware and software optimization. Furthermore, in many cases, high degree polynomials are used on inputs (data) that are bounded in a certain interval. For instance, audio signal values are, in most of the cases, comprised between -1 and 1, and grayscale images contain values that are comprised between 0 and 255 or 0 and 1. Moreover, some machine learning applications require polynomial-based models such as the 1DPNN model that we introduced in Chapter 3 or the polynomial activation neural network model [67] that use high degree polynomials in every neuron. Therefore, reducing the degree of a polynomial to obtain nearly the same results on a particular interval with increased spatial and computational efficiency can be vital for some new machine learning models to thrive.

Polynomial reduction has been extensively explored for various norms to improve computer aided design (CAD) [68] [69] [70], but the polynomials that are reduced are mostly expressed using Bernstein-Bézier coefficients because they are suitable to model graphical curves [71]. However, the polynomials that are used in many machine learning models are usually expressed using coefficients in the canonical basis, or canonical coefficients. In fact, they can either be used as kernels [13][14][15][16][64][72] or as core building blocks [63][67][73][74] which is the case of the 1DPNN. We therefore develop a direct formula to produce the canonical coefficients of a polynomial of low degree that approximates a polynomial of high degree which is expressed in

the canonical basis. The reduced polynomial minimizes the $\mathcal{L}^2$-norm on any symmetric interval of the form $[-l, l]$ where $l \in \mathbb{R}_+^*$. We also demonstrate the theoretical proof that using the formula is more computationally efficient than using a classical approach consisting of matrix multiplications and we empirically show that this formula is more stable than the classical approach. Following that, we generalize the 1DPNN to NDPNN and we use the polynomial degree reduction formula to design an NDPNN layer-wise degree reduction heuristic algorithm that enables the compression of a pre-trained NDPNN with little to no compromise to its performance on the dataset it was trained on.
The outline of this chapter is as follows. In Section 4.2, we define the problem and we provide a method to determine its solution. In Section 4.3, we define an orthonormal basis that is used as an intermediary for computing the canonical coefficients. In Section 4.4, we detail the steps that lead to determining the canonical coefficients of the reduced polynomial. In Section 4.5, we analyze the computational complexities of the use of the direct formula and the use of the classical approach. We also provide two examples of how to use the results developed in this chapter. Finally, Section 4.6 generalizes the 1DPNN to NDPNN and details the NDPNN layer-wise degree reduction heuristic algorithm.

4.2 Problem Statement

In this section, we define the problem and we provide a general way to determine the solution. Let $\mathbb{R}[X]$ be the polynomial algebra in one indeterminate X over $\mathbb{R}$. Let $P, Q \in \mathbb{R}[X]$ such that $deg(Q) < deg(P)$. Let $N = deg(P)$ and $M = deg(Q)$. Let $(a_0, ..., a_N) \in \mathbb{R}^{N+1}$ be the coefficients of P in the canonical basis $(1, X, ..., X^N)$ and $(b_0, ..., b_M) \in \mathbb{R}^{M+1}$ be the coefficients of Q in the canonical basis. Let $l \in \mathbb{R}_+^*$. We want to estimate the polynomial Q that best approximates P as such:

$$\min_{(b_0,...,b_M)} \frac{1}{2l} \int_{-l}^{l} (Q(x) - P(x))^2 \, dx = \min_{(b_0,...,b_M)} J_P(b_0, ..., b_M).$$

Determining the coefficients $(b_0, ..., b_M)$ can be achieved by solving the linear equation $\nabla J_P = 0$. However, solving the equation for these coefficients may prove to be difficult. Therefore, as an intermediate step, we suppose that there exists an orthonormal basis $(e_{0,l}, ..., e_{N,l})$ such that

$$\exists (\beta_0, ..., \beta_M) \in \mathbb{R}^{M+1}, Q = \sum_{n=0}^{M} \beta_n e_{n,l},$$

and

$$\forall m, n \in [\![0, N]\!], \frac{1}{2l} \int_{-l}^{l} e_{m,l}(x) e_{n,l}(x) dx = \delta_{mn},$$

where δ is the Kronecker delta. The objective is then to minimize $J'_P(\beta_0, ..., \beta_M) = \frac{1}{2l}\int_{-l}^{l}(Q(x) - P(x))^2 dx$ such that Q is expressed in the orthonormal basis. Solving the equation $\nabla J'_P = 0$ is equivalent to solving the following linear system:

$$\begin{aligned}
\forall m \in [\![0, M]\!], \frac{\partial J'_P}{\partial \beta_m}(\beta_0, ..., \beta_M) = 0 &\iff \frac{1}{2l}\int_{-l}^{l} e_{m,l}(x)\left(\sum_{n=0}^{M}\beta_n e_{n,l}(x) - P(x)\right)dx = 0 \\
&\iff \sum_{n=0}^{M}\beta_n \frac{1}{2l}\int_{-l}^{l} e_{m,l}(x)e_{n,l}(x)dx = \frac{1}{2l}\int_{-l}^{l} P(x)e_{m,l}(x)dx \\
&\iff \beta_m = \frac{1}{2l}\int_{-l}^{l} P(x)e_{m,l}(x)dx = \sum_{n=0}^{N} a_n \frac{1}{2l}\int_{-l}^{l} x^n e_{m,l}(x)dx.
\end{aligned}$$

The solution of the system is then

$$\begin{pmatrix}\beta_0 \\ \vdots \\ \beta_M\end{pmatrix} = T^{[M,N,l]}\begin{pmatrix}a_0 \\ \vdots \\ \vdots \\ a_N\end{pmatrix}, \tag{4.1}$$

where $T^{[M,N,l]}$ is a $(M+1)\times(N+1)$ matrix such that

$$\forall (m,n) \in [\![0, M]\!] \times [\![0, N]\!], T^{[M,N,l]}_{m,n} = \frac{1}{2l}\int_{-l}^{l} x^n e_{m,l}(x)dx.$$

$T^{[M,N,l]}$ is the orthonormal projection matrix of $\mathbb{R}_N[X]$ on $\mathbb{R}_M[X]$ from the canonical basis to the orthonormal basis. In order to determine the coefficients $(b_0, ..., b_M)$ in the canonical basis, we use the transition matrix from the orthogonal basis to the canonical basis which happens to be $\left(T^{[M,M,l]}\right)^{-1}$ as such:

$$\left(T^{[M,M,l]}\right)^{-1}\begin{pmatrix}\beta_0 \\ \vdots \\ \beta_M\end{pmatrix} = \begin{pmatrix}b_0 \\ \vdots \\ b_M\end{pmatrix} = \left(T^{[M,M,l]}\right)^{-1} T^{[M,N,l]}\begin{pmatrix}a_0 \\ \vdots \\ \vdots \\ a_N\end{pmatrix} = V^{[M,N,l]}\begin{pmatrix}a_0 \\ \vdots \\ \vdots \\ a_N\end{pmatrix}.$$

The objective is to determine $(b_0, ..., b_M)$ by finding the elements of $V^{[M,N,l]} = \left(T^{[M,M,l]}\right)^{-1} T^{[M,N,l]}$ without the need to calculate the matrix product $\left(T^{[M,M,l]}\right)^{-1} T^{[M,N,l]}$.

4.3 Determining the Orthonormal Projection of $\mathbb{R}_N[X]$ on $\mathbb{R}_M[X]$ in the Orthonormal Basis

In this section, we will define all the necessary notions that are needed to perform the orthonormal projection of $\mathbb{R}_N[X]$ on $\mathbb{R}_M[X]$. The notion of orthonormality is defined with respect to a scalar product. Thus, we first need to determine a suitable scalar product for the projection. We first define an intermediate scalar product that will serve as a basis for the remaining.

Proposition 4.3.1. *Let $\langle \cdot, \cdot \rangle$ be defined as such:*

$$\begin{aligned} \langle \cdot, \cdot \rangle \colon \mathbb{R}[X] \times \mathbb{R}[X] &\to \mathbb{R} \\ (A, B) &\mapsto \int_{-1}^{1} A(x)B(x)dx. \end{aligned}$$

$\langle \cdot, \cdot \rangle$ is a scalar product on $\mathbb{R}[X]$.

We now define the scalar product that is relevant to our problem.

Proposition 4.3.2. *Let $l \in \mathbb{R}_+^*$ and $\langle \cdot, \cdot \rangle_l$ be defined as such:*

$$\begin{aligned} \langle \cdot, \cdot \rangle_l \colon \mathbb{R}[X] \times \mathbb{R}[X] &\to \mathbb{R} \\ (A, B) &\mapsto \frac{1}{2l} \int_{-l}^{l} A(x)B(x)dx. \end{aligned}$$

$\langle \cdot, \cdot \rangle_l$ is a scalar product on $\mathbb{R}[X]$.

Proof. The proof can be derived from the proof of Proposition 4.3.1. ■

The scalar product defined in Proposition 4.3.1 is a well-known one that is related to a set of orthogonal polynomials called the Legendre polynomials.

Definition 4.3.1. *The set of polynomials $(L_m)_{m \in \mathbb{N}}$ called Legendre polynomials is defined using Rodrigues formula [75] as such:*

$$\forall m \in \mathbb{N}, \forall x \in \mathbb{R}, L_m(x) = \frac{1}{2^m m!} \frac{d^m}{dx^m} \left[\left(x^2 - 1 \right)^m \right].$$

Lemma 4.3.1. *Legendre polynomials are orthogonal for $\langle \cdot, \cdot \rangle$.*

Using Legendre polynomials, we want to define an orthonormal basis of $\mathbb{R}[X]$ for $\langle \cdot, \cdot \rangle_l$, where $l \in \mathbb{R}_+^*$.

Theorem 4.3.1. *Let $m \in \mathbb{N}, l \in \mathbb{R}_+^*$ and $e_{m,l} = \sqrt{2m+1} L_m \left(\frac{X}{l} \right)$. The set of polynomials $(e_{m,l})_{m \in \mathbb{N}}$ is an orthonormal basis of $\mathbb{R}[X]$ for $\langle \cdot, \cdot \rangle_l$.*

Proof. Let $l \in \mathbb{R}_+^*$. To prove that the set $(e_{m,l})_{m\in\mathbb{N}}$ is an orthonormal basis of $\mathbb{R}[X]$ for $\langle\cdot,\cdot\rangle_l$, we first need to prove that it is orthonormal for $\langle\cdot,\cdot\rangle_l$, then we need to prove that it is a basis of $\mathbb{R}_m[X]$, $\forall m \in \mathbb{N}$.
Let $m, n \in \mathbb{N}$ such that $m \neq n$. We have

$$\langle e_{m,l}, e_{n,l}\rangle_l = \frac{1}{2l}\int_{-l}^{l} e_{m,l}(x)e_{n,l}(x)dx = \frac{\sqrt{2m+1}\sqrt{2n+1}}{2l}\int_{-l}^{l} L_m(\frac{x}{l})L_n(\frac{x}{l})dx.$$

We can use the substitution $t = \dfrac{x}{l}$ since $x\colon \mapsto \dfrac{x}{l}$ is a diffeomorphism from $[-l, l]$ to $[-1, 1]$. Therefore, we obtain

$$\begin{aligned}\langle e_{m,l}, e_{n,l}\rangle_l &= \frac{\sqrt{2m+1}\sqrt{2n+1}}{2l}\int_{-1}^{1} L_m(t)L_n(t)ldt = \frac{\sqrt{2m+1}\sqrt{2n+1}}{2}\int_{-1}^{1} L_m(t)L_n(t)dt\\ &= \frac{\sqrt{2m+1}\sqrt{2n+1}}{2}\langle L_m, L_n\rangle = 0.\end{aligned}$$

This proves that $(e_{m,l})_{m\in\mathbb{N}}$ is an orthogonal set. Let us show that it is orthonormal. To do so, we need to prove that $\langle e_{m,l}, e_{m,l}\rangle_l = 1$. We have

$$\begin{aligned}\langle e_{m,l}, e_{m,l}\rangle_l &= \frac{2m+1}{2}\langle L_m, L_m\rangle = \frac{2m+1}{2^{2m+1}(m!)^2}\int_{-1}^{1}\frac{d^m}{dx^m}\left[(x^2-1)^2\right]\frac{d^m}{dx^m}\left[(x^2-1)^2\right]dx\\ &= \frac{2m+1}{2^{2m+1}(m!)^2}I_{m,m},\end{aligned}$$

where

$$I_{m,m} = \int_{-1}^{1}\frac{d^m}{dx^m}\left[(x^2-1)^2\right]\frac{d^m}{dx^m}\left[(x^2-1)^2\right]dx.$$

By iteratively using integration by parts, we obtain

$$\begin{aligned}I_{m,m} &= (-1)^m I_{2m,0} = (-1)^m(2m)!\int_{-1}^{1}(x^2-1)^m dx = (-1)^m(2m)!\int_{-1}^{1}(x-1)^m(x+1)^m dx\\ &= (-1)^m(2m)!J_{m,0},\end{aligned}$$

where

$$J_{m,0} = \int_{-1}^{1}(x-1)^m(x+1)^m dx.$$

By using integration by parts, we have

$$J_{m,0} = \left[(x-1)^m\frac{(x+1)^{m+1}}{m+1}\right]_{-1}^{1} - \frac{m}{m+1}\int_{-1}^{1}(x-1)^{m-1}(x+1)^{m+1}dx = -\frac{m}{m+1}J_{m,1}.$$

By iterating, we find that

$$\begin{aligned} J_{m,0} &= (-1)^m \frac{(m!)^2}{(2m)!} J_{m,m} = (-1)^m \frac{(m!)^2}{(2m)!} \int_{-1}^{1} (x+1)^{2m} dx = (-1)^m \frac{(m!)^2}{(2m)!} \left[\frac{(x+1)^{2m+1}}{2m+1} \right]_{-1}^{1} \\ &= (-1)^m 2^{2m+1} \frac{(m!)^2}{(2m+1)!}. \end{aligned} \tag{4.2}$$

Finally, we obtain

$$\begin{aligned} \langle e_{m,l}, e_{m,l} \rangle_l &= \frac{2m+1}{2^{2m+1}(m!)^2} I_{m,m} = (-1)^m \frac{(2m+1)!}{2^{2m+1}(m!)^2} J_{m,0} \\ &= (-1)^m \frac{(2m+1)!}{2^{2m+1}(m!)^2} (-1)^m 2^{2m+1} \frac{(m!)^2}{(2m+1)!} = 1. \end{aligned}$$

As a result, $(e_{m,l})_{m\in\mathbb{N}}$ is an orthonormal set for $\langle \cdot, \cdot \rangle_l$. In particular, $\forall m \in \mathbb{N}, (e_{0,l}, ..., e_{m,l})$ is an orthonormal set of $\mathbb{R}_m[X]$ for $\langle \cdot, \cdot \rangle_l$ and its size is $m+1 = dim(\mathbb{R}_m[X])$. Since $(e_{0,l}, ..., e_{m,l})$ is an orthonormal set, it is linearly independent. Hence, it is a basis of $\mathbb{R}_m[X]$. Finally, we conclude that the set of polynomials $(e_{m,l})_{m\in\mathbb{N}}$ is an orthonormal basis of $\mathbb{R}[X]$ for $\langle \cdot, \cdot \rangle_l$. ■

After determining the orthonormal basis, we want to determine how any polynomial expressed in the canonical basis can be expressed in the orthonormal basis.

Lemma 4.3.2. *Let $l \in \mathbb{R}_+^*$ and $m \in \mathbb{N}$. The degrees of all the monomials of $e_{m,l}$ have the same parity as m.*

Proof. Let $l \in \mathbb{R}_+^*, m \in \mathbb{N}$. According to the binomial theorem, $(X^2-1)^m = \sum_{n=0}^{m} \binom{m}{n} X^{2n} (-1)^{m-n}$. Thus, every monomial of $(X^2-1)^m$ has an even degree. Therefore, given that $L_m \propto \frac{d^m}{dX^m}[(X^2-1)^m]$, the degrees of every monomial of L_m will either be even or odd, depending on the parity of m, since $deg(L_m) = m$. Given that $e_{m,l} = \sqrt{2m+1} L_m\left(\frac{X}{l}\right)$, the degrees of all the monomials of $e_{m,l}$ also have the same parity as m. ■

Theorem 4.3.2. *Let $l \in \mathbb{R}_+^*$.*

$$\forall m, n \in \mathbb{N}, \begin{cases} \langle X^{2n}, e_{2m+1,l} \rangle_l & = \langle X^{2n+1}, e_{2m,l} \rangle_l = 0 \\ \langle X^{2n}, e_{2m,l} \rangle_l & = \sqrt{4m+1}.2^{2m+1} l^{2n} \frac{(2n)!(m+n+1)!}{(n-m)!(2(m+n+1))!} \mathbf{1}_{\mathbb{N}\setminus[\![0,m]\!]}(n) \\ \langle X^{2n+1}, e_{2m+1,l} \rangle_l & = \sqrt{4m+3}.2^{2m+1} l^{2n+1} \frac{(2n+1)!(m+n+1)!}{(n-m)!(2(m+n+1)+1)!} \mathbf{1}_{\mathbb{N}\setminus[\![0,m]\!]}(n) \end{cases}, \tag{4.3}$$

where $\mathbf{1}_{\mathbb{N}\setminus[\![0,m[\![}$ *is the indicator function on* $\mathbb{N}\setminus[\![0,m[\![$.

Proof. Let $l \in \mathbb{R}_+^*$ and $m, n \in \mathbb{N}$. According to Lemma 4.3.2, we know that the degrees of every monomial of $e_{2m+1,l}$ is odd, since $2m+1$ is odd. We also know that

$$\forall i \in [\![0,m]\!], \langle X^{2n}, X^{2i+1}\rangle_l = \frac{1}{2l}\int_{-l}^{l} x^{2n}x^{2i+1}dx = \frac{1}{2l}\int_{-l}^{l} x^{2(n+i)+1}dx = \left[\frac{x^{2(n+i+1)}}{2(n+i+1)}\right]_{-l}^{l} = 0.$$

Accordingly, since the degrees of every monomial of $e_{2m+1,l}$ are odd, we obtain $\langle X^{2n}, e_{2m+1,l}\rangle_l = 0$. The same can be deduced for $\langle X^{2n+1}, e_{2m,l}\rangle_l$.
Let us determine $\langle X^{2n}, e_{2m,l}\rangle_l$. We have

$$\langle X^{2n}, e_{2m,l}\rangle_l = \frac{1}{2l}\int_{-l}^{l} x^{2n}e_{2m,l}(x)dx = \frac{\sqrt{4m+1}}{2l}\int_{-l}^{l} x^{2n}L_{2m}\left(\frac{x}{l}\right)dx.$$

We can use the substitution $t = \dfrac{x}{l}$ since $x\colon \mapsto \dfrac{x}{l}$ is a diffeomorphism from $[-l,l]$ to $[-1,1]$. As a result, we obtain

$$\begin{aligned}\langle X^{2n}, e_{2m,l}\rangle_l &= \frac{\sqrt{4m+1}}{2l}\int_{-1}^{1}(lt)^{2n}L_{2m}(t)\,ldt = \frac{\sqrt{4m+1}}{2}l^{2n}\int_{-1}^{1}t^{2n}L_{2m}(t)\,dt \\ &= \frac{\sqrt{4m+1}}{2}l^{2n}\langle X^{2n}, L_{2m}\rangle.\end{aligned}$$

We have

$$\langle X^{2n}, L_{2m}\rangle = \frac{1}{2^{2m}(2m)!}\int_{-1}^{1} x^{2n}\frac{d^{2m}}{dx^{2m}}\left[(x^2-1)^{2m}\right]dx = \frac{1}{2^{2m}(2m)!}K_{2n,2m}, \tag{4.4}$$

where

$$K_{2n,2m} = \int_{-1}^{1} x^{2n}\frac{d^{2m}}{dx^{2m}}\left[(x^2-1)^{2m}\right]dx.$$

By using integration by parts, we obtain

$$\begin{aligned}K_{2n,2m} &= \left[x^{2n}\frac{d^{2m-1}}{dx^{2m-1}}\left[(x^2-1)^{2m}\right]\right]_{-1}^{1} - 2n\int_{-1}^{1}x^{2n-1}\frac{d^{2m-1}}{dx^{2m-1}}\left[(x^2-1)^{2m}\right]dx \\ &= -2nK_{2n-1,2m-1}.\end{aligned}$$

If $n < m$, we can iterate the previous result $2n$ times to find that

$$K_{2n,2m} = (-1)^{2n}(2n)!K_{0,2(m-n)} = (2n)!\int_{-1}^{1}\frac{d^{2(m-n)}}{dx^{2(m-n}}\left[(x^2-1)^{2m}\right]dx = 0.$$

Hence, $\langle X^{2n}, e_{2m,l}\rangle_l = 0$ when $n < m$. If $n \geq m$, we can iterate the integration by part of $K_{2n,2m}$ $2m$ times to obtain

$$\begin{aligned} K_{2n,2m} &= (-1)^{2m}\frac{(2n)!}{(2(n-m))!}K_{2(n-m),0} = \frac{(2n)!}{(2(n-m))!}\int_{-1}^{1} x^{2(n-m)}\left(x^2-1\right)^{2m}dx \\ &= \frac{(2n)!}{(2(n-m))!}H_{n-m,2m}, \end{aligned} \tag{4.5}$$

where

$$H_{n-m,2m} = \int_{-1}^{1} x^{2(n-m)}\left(x^2-1\right)^{2m}dx = \int_{-1}^{1} x^{2(n-m)-1}x\left(x^2-1\right)^{2m}dx.$$

By using integration by parts, we obtain

$$\begin{aligned} H_{n-m,2m} &= \left[\frac{x^{2(n-m)-1}\left(x^2-1\right)^{2m+1}}{2(2m+1)}\right]_{-1}^{1} - \frac{2(n-m)-1}{2(2m+1)}\int_{-1}^{1} x^{2(n-m-1)}\left(x^2-1\right)^{2m+1}dx \\ &= -\frac{2(n-m)-1}{2(2m+1)}H_{n-m-1,2m+1} = (-1)^{n-m}\frac{(2(n-m))!}{2^{n-m}(n-m)!}\frac{(2m)!}{2^{n-m}(m+n)!}H_{0,m+n} \\ &= (-1)^{n-m}\frac{(2(n-m))!}{2^{2(n-m)}(n-m)!}\frac{(2m)!}{(m+n)!}\int_{-1}^{1}(x^2-1)^{m+n}dx \\ &= (-1)^{n-m}\frac{(2(n-m))!}{2^{2(n-m)}(n-m)!}\frac{(2m)!}{(m+n)!}J_{m+n,0}. \end{aligned} \tag{4.6}$$

Using Eqs.(4.2), (4.6), (4.5), and (4.4), we find that

$$\begin{aligned} \langle X^{2n}, e_{2m,l}\rangle_l &= \frac{\sqrt{4m+1}}{2}l^{2n}\frac{(2n)!(-1)^{n-m}(2(n-m))!(2m)!(-1)^{m+n}2^{2(m+n)+1}\left((m+n)!\right)^2}{2^{2m}(2m)!(2(n-m))!2^{2(n-m)}(n-m)!(m+n)!((2(m+n)+1)!} \\ &= \sqrt{4m+1}.2^{2m+1}l^{2n}\frac{(2n)!(m+n+1)!}{(n-m)!(2(m+n+1))!}. \end{aligned}$$

Since $\langle X^{2n}, e_{2m,l}\rangle_l = 0$ when $n < m$, we finally conclude that

$$\langle X^{2n}, e_{2m,l}\rangle_l = \sqrt{4m+1}.2^{2m+1}l^{2n}\frac{(2n)!(m+n+1)!}{(n-m)!(2(m+n+1))!}\mathbf{1}_{\mathbb{N}\setminus[\![0,m[\![}(n).$$

The same steps can be followed to determine $\langle X^{2n+1}, e_{2m+1,l}\rangle_l$. ■

Using Theorem 4.3.2, we can fully determine $T^{[M,N,l]}$ defined in Eq. (4.1) such that $\forall(m,n) \in [\![0,M]\!] \times [\![0,N]\!], T^{[M,N,l]}_{m,n} = \langle X^n, e_{m,l}\rangle_l$.

4.4 Determining the Orthonormal Projection of $\mathbb{R}_N[X]$ on $\mathbb{R}_M[X]$ in the Canonical Basis

In this section, we use the theorems proposed in Section 4.3 to solve the problem defined in Section 4.2. In order to determine the canonical basis coefficients, we need to determine the transition matrix $\left(T^{[M,M,l]}\right)^{-1}$ from the orthonormal basis to the canonical basis as defined in Eq. (4.1). To do so, we need to express the set of orthonormal polynomials $(e_{m,l})_{m\in\mathbb{N}}$ in the canonical basis.

Theorem 4.4.1. *Let* $l \in \mathbb{R}_+^*$. $\forall m \in \mathbb{N}, e_{m,l} = \sum_{n=0}^{m} \epsilon_{m,n,l} X^n$ *where*

$$\forall m,n \in \mathbb{N}, \begin{cases} \epsilon_{2m,2n+1,l} & = \epsilon_{2m+1,2n,l} = 0 \\ \epsilon_{2m,2n,l} & = (-1)^{m-n}\dfrac{\sqrt{4m+1}}{2^{2m}l^{2n}}\dfrac{(2(m+n))!}{(m+n)!(m-n)!(2n)!}\mathbf{1}_{\mathbb{N}\setminus[\![0,n[\![}(m) \\ \epsilon_{2m+1,2n+1,l} & = (-1)^{m-n}\dfrac{\sqrt{4m+3}}{2^{2m}l^{2n}}\dfrac{(2(m+n)+1)!}{(m+n)!(m-n)!(2n+1)!}\mathbf{1}_{\mathbb{N}\setminus[\![0,n[\![}(m) \end{cases} . \tag{4.7}$$

Proof. Let $l \in \mathbb{R}_+^*, m \in \mathbb{N}$. We have

$$\begin{aligned} e_{2m,l} &= \sqrt{4m+1}L_{2m}\left(\frac{X}{l}\right) = \frac{\sqrt{4m+1}}{2^{2m}(2m)!}\frac{d^{2m}}{d\left(\frac{X}{l}\right)^{2m}}\left[\left(\left(\frac{X}{l}\right)^2-1\right)^{2m}\right] \\ &= \frac{\sqrt{4m+1}l^{2m}}{2^{2m}(2m)!}\frac{d^{2m}}{dX^{2m}}\left[\left(\left(\frac{X}{l}\right)^2-1\right)^{2m}\right]. \end{aligned}$$

According to the binomial theorem, we have

$$\left(\left(\frac{X}{l}\right)^2-1\right)^{2m} = \sum_{n=0}^{2m}\binom{2m}{n}\left(\frac{X}{l}\right)^{2n}(-1)^{2m-n}.$$

Consequently, we obtain

$$\begin{aligned} e_{2m,l} &= \frac{\sqrt{4m+1}l^{2m}}{2^{2m}(2m)!}\sum_{n=m}^{2m}\binom{2m}{n}\frac{(-1)^{2m-n}(2n)!}{(2(n-m))!l^{2n}}X^{2(n-m)} \\ &= \frac{\sqrt{4m+1}}{2^{2m}}\sum_{n=m}^{2m}(-1)^{2m-n}\frac{1}{l^{2(n-m)}}\frac{(2n)!}{n!(2m-n)!((2(n-m))!}X^{2(n-m)}. \end{aligned}$$

By using the index substitution $n \leftarrow n-m$, we have

$$e_{2m,l} = \sum_{n=0}^{m}(-1)^{m-n}\frac{\sqrt{4m+1}}{2^{2m}l^{2m}}\frac{(2(m+n))!}{(m+n)!(m-n)!(2n)!}X^{2n} = \sum_{n=0}^{m}\epsilon_{2m,2n,l}X^{2n}.$$

Therefore, by identification, we have $\epsilon_{2m,2n+1,l} = 0$ and

$$\epsilon_{2m,2n,l} = \begin{cases} 0 & \text{, if } n > m \\ (-1)^{m-n}\frac{\sqrt{4m+1}}{2^{2m}l^{2m}}\frac{(2(m+n))!}{(m+n)!(m-n)!(2n)!} & \text{, otherwise} \end{cases}.$$

Finally, we obtain

$$\epsilon_{2m,2n,l} = (-1)^{m-n}\frac{\sqrt{4m+1}}{2^{2m}l^{2m}}\frac{(2(m+n))!}{(m+n)!(m-n)!(2n)!}\mathbf{1}_{\mathbb{N}\setminus[\![0,n[\![}(m).$$

The same steps can be followed for $\epsilon_{2m+1,2n,l}$ and $\epsilon_{2m+1,2n+1,1}$. ■

We can now use Theorem 4.4.1 to determine $\left(T^{[M,M,l]}\right)^{-1}$ such that $\forall(m,n) \in [\![0,M]\!]^2, \left(T^{[M,M,l]}\right)^{-1}_{m,n} = \epsilon_{n,m}$. Having determined $T^{[M,N,l]}$ and $\left(T^{[M,M,l]}\right)^{-1}$, the canonical coefficients can be determined by calculating $V^{[M,N,l]} = \left(T^{[M,M,l]}\right)^{-1} T^{[M,N,l]}$. We can notice that $V^{[M,M,l]} = I_M$ where I_M is the identity matrix. In that case, there is only the need to determine the elements of $V^{[M,N,l]}$ whose columns are strictly greater than M. We denote by $v^{[M,N,l]}_{m,n}$ an element of $V^{[M,N,l]}$ at line m and column n, $\forall(m,n) \in [\![0,M]\!] \times [\![0,N]\!]$.

Theorem 4.4.2. *Let $l \in \mathbb{R}^*_+$ and $M, N \in \mathbb{N}$ such that $M < N$. Let $p = \left\lfloor \frac{M}{2} \right\rfloor, p_1 = \left\lfloor \frac{M-1}{2} \right\rfloor, q = \left\lfloor \frac{N}{2} \right\rfloor$ and $q_1 = \left\lfloor \frac{N-1}{2} \right\rfloor$. We have*

$$\forall(m,n) \in [\![0,p]\!] \times [\![p_1+1, q_1]\!], v^{[M,N,l]}_{2m,2n+1} = 0,$$
$$\forall(m,n) \in [\![0,p_1]\!] \times [\![p+1, q]\!], v^{[M,N,l]}_{2m+1,2n} = 0,$$
$$\forall(m,n) \in [\![0,p]\!] \times [\![p+1, q]\!], v^{[M,N,l]}_{2m,2n} = \frac{(-1)^{p-m}l^{2(n-m)}(2n)!}{2^{n-m}(n-m)!(p-m)!(2m)!}\prod_{r=n-p}^{n-m-1} r \prod_{r=p+m+1}^{p+n}\frac{1}{2r+1},$$

and

$$\forall(m,n) \in [\![0,p_1]\!] \times [\![p_1+1, q_1]\!],$$
$$v^{[M,N,l]}_{2m+1,2n+1} = \frac{(-1)^{p-m}l^{2(n-m)}(2n+1)!}{2^{n-m}(n-m)!(p-m)!(2m+1)!}\prod_{r=n-p}^{n-m-1} r \prod_{r=p+m+2}^{p+n+1}\frac{1}{2r+1}.$$

Proof. Let $l \in \mathbb{R}^*_+$ and $M, N \in \mathbb{N}$ such that $M < N$. Let $p = \left\lfloor \frac{M}{2} \right\rfloor, p_1 = \left\lfloor \frac{M-1}{2} \right\rfloor, q = \left\lfloor \frac{N}{2} \right\rfloor$ and $q_1 = \left\lfloor \frac{N-1}{2} \right\rfloor$. We will first prove that $\forall(m,n) \in [\![0,p]\!] \times [\![p_1+1, q_1]\!], v^{[M,N,l]}_{2m,2n+1} = 0$.

Let $(m,n) \in [\![0,p]\!] \times [\![p_1+1, q_1]\!]$. We have

$$v_{2m,2n+1}^{[M,N,l]} = \sum_{k=0}^{M} \epsilon_{k,2m,l} \langle X^{2n+1}, e_{k,l} \rangle_l.$$

According to Theorem 4.3.2 and Theorem 4.4.1, $\langle X^{2n+1}, e_{k,l} \rangle_l = 0$ when k is even and $\epsilon_{k,2m,l} = 0$ when k is odd. However, k can not be even and odd at the same time, thus, one of both terms in the multiplication will necessarily be nil. Thus, $v_{2m,2n+1}^{[M,N,l]} = 0$. The same reasoning can be used for $v_{2m+1,2n}^{[M,N,l]}$.
We now want to determine the value of $v_{2m,2n}^{[M,N,l]}, \forall (m,n) \in [\![0,p]\!] \times [\![p+1,q]\!]$. Let $(m,n) \in [\![0,p]\!] \times [\![p+1,q]\!]$. We have

$$v_{2m,2n}^{[M,N,l]} = \sum_{k=0}^{M} \epsilon_{k,2m,l} \langle X^{2n}, e_{k,l} \rangle_l.$$

According to Theorem 4.3.2 and Theorem 4.4.1, $\langle X^{2n}, e_{k,l} \rangle_l = 0$ when k is odd and $\epsilon_{k,2m,l} = 0$ when k is odd. Hence we obtain

$$v_{2m,2n}^{[M,N,l]} = \sum_{k=0}^{p} \epsilon_{2k,2m,l} \langle X^{2n}, e_{2k,l} \rangle_l.$$

According to Theorem 4.4.1, $\epsilon_{2k,2m,l} = 0$ when $k < m$. As a result, we have

$$v_{2m,2n}^{[M,N,l]} = \sum_{k=m}^{p} \epsilon_{2k,2m,l} \langle X^{2n}, e_{2k,l} \rangle_l.$$

In order to determine $v_{2m,2n}^{[M,N,l]}$, we can first determine $v_{2(p-m),2n}^{[M,N,l]}$ as such:

$$\begin{aligned} v_{2(p-m),2n}^{[M,N,l]} &= \sum_{k=p-m}^{p} \epsilon_{2k,2(p-m),l} \langle X^{2n}, e_{2k,l} \rangle_l \\ &= \sum_{k=p-m}^{p} \frac{(-1)^{k+m-p} 2(4k+1) l^{2(n+m-p)}}{(k+m-p)!(n-k)!} \frac{(2n)!}{(2(p-m))!} \frac{(k+n+1)!}{(2(k+n+1))!} \frac{(2(k+p-m))!}{(k+p-m)!}. \end{aligned} \tag{4.8}$$

We have

$$(2(k+p-m))! = \prod_{r=1}^{2(k+p-m)} r = \prod_{r=1}^{k+p-m} 2r \prod_{r=0}^{k+p-m-1} (2r+1) = 2^{k+p-m}(k+p-m)! \prod_{r=0}^{k+p-m-1} (2r+1).$$

Thus,

$$\frac{(2(k+p-m))!}{(k+p-m)!} = 2^{k+p-m} \prod_{r=0}^{k+p-m-1} (2r+1). \tag{4.9}$$

The same steps can be followed to show that

$$\frac{(k+n+1)!}{(2(k+n+1))!} = \frac{1}{2^{k+n+1}} \prod_{r=0}^{k+n} \frac{1}{2r+1}. \tag{4.10}$$

From Eqs. (4.9) and (4.10), we can derive the following expression:

$$\frac{(k+n+1)!}{(2(k+n+1))!} \frac{(2(k+p-m))!}{(k+p-m)!} = \frac{1}{2^{n+m-p+1}} \prod_{r=k+p-m}^{k+n} \frac{1}{2r+1}. \tag{4.11}$$

By injecting Eq. (4.11) in Eq. (4.8), and by factorizing the terms of the summation that do not depend on k, we obtain

$$v_{2(p-m),2n}^{[M,N,l]} = \frac{l^{(2(n+m-p)}(2n)!}{2^{n+m-p}(2(p-m))!} \sum_{k=p-m}^{p} (-1)^{k-p+m} \frac{(4k+1)}{(k+m-p)!(n-k)!} \prod_{r=k+p-m}^{k+n} \frac{1}{2r+1}. \tag{4.12}$$

Since Eq. (4.12) consists in the summation of different fractions, we express the fractions with respect to the same common denominator as such:

$$v_{2(p-m),2n}^{[M,N,l]} = \frac{l^{2(n+m-p)}(2n)!}{2^{n+m-p}(2(p-m))!(n-p+m)!m!} \prod_{r=2(p-m)}^{p+n} \frac{1}{2r+1} S_{m,n,p}, \tag{4.13}$$

where

$$S_{m,n,p} = \sum_{k=p-m}^{p} (-1)^{k+m-p}(4k+1) \prod_{n-k+1}^{n-p+m} r \prod_{k+m-p+1}^{m} r \prod_{2(p-m)}^{k+p-m-1} (2r+1) \prod_{k+n+1}^{p+n} (2r+1).$$

The objective is now to prove that

$$S_{m,n,p} = (-1)^m \prod_{2(p-m)}^{2p-m} (2r+1) \prod_{n-p}^{n-p+m-1} r.$$

To do so, we first need to rearrange the terms of $S_{m,n,p}$. We use the index substitution $k \leftarrow p-k$ to obtain

$$S_{m,n,p} = \sum_{k=0}^{m} (-1)^{m-k}(4(p-k)+1) \prod_{r=n-p+k+1}^{n-p+m} r \prod_{r=m+1-k}^{m} r \prod_{r=2(p-m)}^{2p-m-k-1} (2r+1) \prod_{r=p+n-k+1}^{p+n} (2r+1).$$

We then incorporate the common terms of each boundary of each product into each product index to obtain

$$S_{m,n,p} = \sum_{k=0}^{m}(-1)^{k+m}(4(p-k)+1)\prod_{r=k+1}^{m}(r+n-p)\prod_{r=-k+m+1}^{m}r\prod_{r=-2m}^{-k-m-1}(4p+2r+1)$$
$$\times\prod_{r=-k+1}^{0}(2(r+p+n)+1).$$

Let $S_m(X,Y) \in \mathbb{R}[X,Y]$ be a polynomial in 2 indeterminates X and Y over $\mathbb{R}$ defined as such:

$$S_m = \sum_{k=0}^{m}(-1)^{k+m}(4(Y-k)+1)\prod_{r=k+1}^{m}(r+X-Y)\prod_{r=-k+m+1}^{m}r\prod_{r=-2m}^{-k-m-1}(4Y+2r+1)$$
$$\times\prod_{r=-k+1}^{0}(2(r+Y+X)+1).$$

This makes $S_{m,n,p} = S_m(n,p)$. Since our objective is to show that

$$S_{m,n,p} = (-1)^m\prod_{2(p-m)}^{2p-m}(2r+1)\prod_{n-p}^{n-p+m-1}r = (-1)^m\prod_{r=0}^{m}(4(p-m)+2r+1)\prod_{r=0}^{m-1}(r+n-p),$$

we can show that the polynomial $S_m(n,X)$ in one indeterminate X can be factorized as such:

$$S_m(n,X) = (-1)^m\prod_{r=0}^{m}(4(X-m)+2r+1)\prod_{r=0}^{m-1}(r+n-X).$$

Therefore, the objective is to show that $\forall t \in [\![0,m]\!], m-\frac{1}{4}(2t+1)$ is a root of $S_m(n,X)$ and that $\forall t \in [\![0,m-1]\!], t+n$ is also a root of $S_m(n,X)$. We begin by showing that $m-\frac{1}{4}(2t+1)$ is a root of $S_m(n,X), \forall t \in [\![0,m]\!]$. Let $t \in [\![0,m]\!]$. We want to show that $S_m\left(n, m-\frac{1}{4}(2t+1)\right) = 0$. We have

$$\begin{aligned}S_m\left(n, m-\frac{1}{4}(2t+1)\right) &= \sum_{k=0}^{m}(-1)^{k+m}(4(m-k)-2t)\prod_{r=k+1}^{m}\left(r+n-m+\frac{t}{2}+\frac{1}{4}\right)\prod_{r=-k+m+1}^{m}r\\ &\quad\prod_{r=-2m}^{-k-m-1}(4m+2(r-t))\prod_{r=-k+1}^{0}\left(2(r+n+m)-t+\frac{1}{2}\right)\\ &= \sum_{k=0}^{m}U_{m,n,t,k},\end{aligned}$$

where $U_{m,n,t,k}$ represents the k-th term of the summation. The term $\prod_{r=-2m}^{-k-m-1}(4m+2(r-t))$ is nil when $\exists r' \in \mathbb{N}$ such that $-2m \leq r' \leq -k-m-1$ and $4m+2(r'-t)=0$. This means

that $r' = -2m + t$. However, since $r' \leq -k - m - 1$, we find that, when $k \leq m - t - 1$, $\prod_{r=-2m}^{-k-m-1}(4m + 2(r - t)) = 0$. Moreover, the term $4(m - k) - 2t$ is nil when $k = m - \frac{t}{2}$. This means that $4(m - k) - 2t = 0$ when t is even. We suppose that t is even. Then, we can rewrite $S_m\left(n, m - \frac{1}{4}(2t + 1)\right)$ by splitting the sum in two parts as such:

$$S_m\left(n, m - \frac{1}{4}(2t + 1)\right) = \sum_{k=m-t}^{m-\left\lfloor \frac{t}{2} \right\rfloor - 1} U_{m,n,t,k} + \sum_{k=m-\left\lfloor \frac{t}{2} \right\rfloor + 1}^{m} U_{m,n,t,k}. \tag{4.14}$$

By using the substitution $k \leftarrow m - \left\lfloor \frac{t}{2} \right\rfloor - k$ on the first sum of Eq. (4.14), and the substitution $k \leftarrow -m + \left\lfloor \frac{t}{2} \right\rfloor + k$ on the second sum of Eq. (4.14), we can regroup both sums to obtain

$$S_m\left(n, m - \frac{1}{4}(2t + 1)\right) = \sum_{k=1}^{\left\lfloor \frac{t}{2} \right\rfloor} \left(U_{m,n,t,m-\left\lfloor \frac{t}{2} \right\rfloor - k} + U_{m,n,t,m-\left\lfloor \frac{t}{2} \right\rfloor + k} \right). \tag{4.15}$$

We have

$$\begin{aligned} U_{m,n,t,m-\left\lfloor \frac{t}{2} \right\rfloor - k} =& (-1)^{2m-\left\lfloor \frac{t}{2} \right\rfloor - k} 4k \prod_{r=m-\left\lfloor \frac{t}{2} \right\rfloor - k + 1}^{m} \left(r + n - m + \frac{t}{2} + \frac{1}{4} \right) \prod_{r=k+\left\lfloor \frac{t}{2} \right\rfloor + 1}^{m} r \\ & \prod_{r=-2m}^{k+\left\lfloor \frac{t}{2} \right\rfloor - 2m - 1} (4m + 2(r - t)) \prod_{r=k+\left\lfloor \frac{t}{2} \right\rfloor - m + 1}^{0} \left(2(r + n + m) - t + \frac{1}{2} \right), \end{aligned} \tag{4.16}$$

and

$$\begin{aligned} U_{m,n,t,m-\left\lfloor \frac{t}{2} \right\rfloor + k} =& (-1)^{2m-\left\lfloor \frac{t}{2} \right\rfloor + k} (-4k) \prod_{r=m-\left\lfloor \frac{t}{2} \right\rfloor + k + 1}^{m} \left(r + n - m + \frac{t}{2} + \frac{1}{4} \right) \prod_{r=-k+\left\lfloor \frac{t}{2} \right\rfloor + 1}^{m} r \\ & \prod_{r=-2m}^{-k+\left\lfloor \frac{t}{2} \right\rfloor - 2m - 1} (4m + 2(r - t)) \prod_{r=-k+\left\lfloor \frac{t}{2} \right\rfloor - m + 1}^{0} \left(2(r + n + m) - t + \frac{1}{2} \right). \end{aligned} \tag{4.17}$$

By factorizing the common terms of $U_{m,n,t,m-\left\lfloor\frac{t}{2}\right\rfloor-k}$ and $U_{m,n,t,m-\left\lfloor\frac{t}{2}\right\rfloor+k}$, we obtain

$$\begin{aligned}
&U_{m,n,t,m-\left\lfloor\frac{t}{2}\right\rfloor-k}+U_{m,n,t,m-\left\lfloor\frac{t}{2}\right\rfloor+k}\\
=&(-1)^{2m-\left\lfloor\frac{t}{2}\right\rfloor+k}4k\prod_{r=m-\left\lfloor\frac{t}{2}\right\rfloor+k+1}^{m}\left(r+n-m+\frac{t}{2}+\frac{1}{4}\right)\prod_{r=+k+\left\lfloor\frac{t}{2}\right\rfloor+1}^{m}r\prod_{r=-2m}^{-k+\left\lfloor\frac{t}{2}\right\rfloor-2m-1}(4m+2(r-t))\\
&\prod_{r=+k+\left\lfloor\frac{t}{2}\right\rfloor-m+1}^{0}\left(2(r+n+m)-t+\frac{1}{2}\right)W_{m,n,t,k},
\end{aligned}\tag{4.18}$$

where,

$$\begin{aligned}
W_{m,n,t,k}=&\prod_{r=m-\left\lfloor\frac{t}{2}\right\rfloor-k+1}^{m-\left\lfloor\frac{t}{2}\right\rfloor+k}\left(r+n-m+\frac{t}{2}+\frac{1}{4}\right)\prod_{r=-k+\left\lfloor\frac{t}{2}\right\rfloor-2m}^{+k+\left\lfloor\frac{t}{2}\right\rfloor-2m-1}(4m+2(r-t))\\
&-\prod_{r=-k+\left\lfloor\frac{t}{2}\right\rfloor+1}^{k+\left\lfloor\frac{t}{2}\right\rfloor}r\prod_{r=-k+\left\lfloor\frac{t}{2}\right\rfloor-m+1}^{k+\left\lfloor\frac{t}{2}\right\rfloor-m}\left(2(r+n+m)-t+\frac{1}{2}\right).
\end{aligned}\tag{4.19}$$

By using the index substitutions $r\leftarrow -r$, $r\leftarrow r+m$, $r\leftarrow r-m-1$ and $r\leftarrow r-1$ respectively on the 4 products in Eq. (4.19), we obtain

$$\begin{aligned}
W_{m,n,t,k}=&\prod_{r=-m+\left\lfloor\frac{t}{2}\right\rfloor-k}^{-m+\left\lfloor\frac{t}{2}\right\rfloor+k-1}\left(-r+n-m+\frac{t}{2}+\frac{1}{4}\right)(2m+2(r-t))\\
&-\prod_{r=-k+\left\lfloor\frac{t}{2}\right\rfloor-m}^{k+\left\lfloor\frac{t}{2}\right\rfloor-m-1}(r+m+1)\left(2(r+n+m+1)-t+\frac{1}{2}\right).
\end{aligned}\tag{4.20}$$

By using the index substitution $r \leftarrow -2m + t - 1 - r$ on the second product in Eq. (4.20), we obtain

$$\begin{aligned}
&\prod_{r=-k+\left\lfloor\frac{t}{2}\right\rfloor-m}^{k+\left\lfloor\frac{t}{2}\right\rfloor-m-1} (r+m+1)\left(2(r+n+m+1)-t+\frac{1}{2}\right) \\
&= \prod_{r=-k+\left\lfloor\frac{t}{2}\right\rfloor-m}^{k+\left\lfloor\frac{t}{2}\right\rfloor-m-1} (t-r-m)\left(-2r+2n-2m+t+\frac{1}{2}\right) \\
&= \prod_{r=-k+\left\lfloor\frac{t}{2}\right\rfloor-m}^{k+\left\lfloor\frac{t}{2}\right\rfloor-m-1} (2t-2r-2m)\left(-r+n-m+\frac{t}{2}+\frac{1}{4}\right) \\
&= \prod_{r=-k+\left\lfloor\frac{t}{2}\right\rfloor-m}^{k+\left\lfloor\frac{t}{2}\right\rfloor-m-1} (-(2m+2(r-t)))\left(-r+n-m+\frac{t}{2}+\frac{1}{4}\right)
\end{aligned} \tag{4.21}$$

$$\begin{aligned}
&=(-1)^{2k} \prod_{r=-k+\left\lfloor\frac{t}{2}\right\rfloor-m}^{k+\left\lfloor\frac{t}{2}\right\rfloor-m-1} (2m+2(r-t))\left(-r+n-m+\frac{t}{2}+\frac{1}{4}\right) \\
&= \prod_{r=-k+\left\lfloor\frac{t}{2}\right\rfloor-m}^{k+\left\lfloor\frac{t}{2}\right\rfloor-m-1} (2m+2(r-t))\left(-r+n-m+\frac{t}{2}+\frac{1}{4}\right).
\end{aligned}$$

By injecting Eq. (4.21) in Eq. (4.20), we find that $W_{m,n,t,k} = 0$. Consequently, using Eq. (4.18), we have $U_{m,n,t,m-\left\lfloor\frac{t}{2}\right\rfloor-k} + U_{m,n,t,m-\left\lfloor\frac{t}{2}\right\rfloor+k} = 0$. Using Eq. (4.15), we find that $S_m\left(n, m-\frac{1}{4}(2t+1)\right) = 0$ when t is even. The same can be concluded when t is odd by rewriting $S_m\left(n, m-\frac{1}{4}(2t+1)\right)$ as such:

$$S_m\left(n, m-\frac{1}{4}(2t+1)\right) = \sum_{k=m-t}^{m-\left\lfloor\frac{t}{2}\right\rfloor-1} U_{m,n,t,k} + \sum_{k=m-\left\lfloor\frac{t}{2}\right\rfloor}^{m} U_{m,n,t,k}. \tag{4.22}$$

The same steps can then be performed on Eq.(4.22) to show that it is nil. The only difference

lies in applying the substitution $k \leftarrow -m + \left\lfloor \frac{t}{2} \right\rfloor + k + 1$ on the second sum of Eq.(4.22). We can subsequently conclude that $m - \frac{1}{4}(2t+1)$ is a root of $S_m(n, X), \forall t \in [\![0, m]\!]$. Moreover, we notice that n has no influence on whether $m - \frac{1}{4}(2t+1)$ is a root of $S_m(n, X), \forall t \in [\![0, m]\!]$, thus we can induce that $S_m\left(X, m - \frac{1}{4}(2t+1)\right) = 0, \forall t \in [\![0, m]\!]$. Furthermore, since $p+1 \leq n \leq q, \exists n' \in [\![1, q-p]\!]$ such that $n = n' + p$. We then have $S_{m,n,p} = S_{m,n'+p,p} = S_m(n'+p, p)$. Hence, we can study the polynomial $S_m(n'+X, X)$ to prove that

$$S_m(n'+X, X) = (-1)^m \prod_{r=-2m}^{-m} (4X + 2r + 1) \prod_{r=0}^{m-1} (r + n').$$

In order to do so, we need to prove that $deg\left(S_m(n'+X, X)\right) = m+1$ and that the leading coefficient of $S_m(n'+X, X)$ is $(-1)^m 4^{m+1} \prod_{r=0}^{m-1} (r+n')$. We have

$$\begin{aligned} S_m(n'+X, X) = \sum_{k=0}^{m} (-1)^{k+m} (4(X-k)+1) \prod_{r=k+1}^{m} (r+n') \prod_{r=-k+m+1}^{m} r \prod_{r=-2m}^{-k-m-1} (4X+2r+1) \\ \prod_{r=-k+1}^{0} (2(r+n'+2X)+1). \end{aligned} \tag{4.23}$$

By examining the monomials of $S_m(n'+X, X)$, we find that $deg\left(S_m(n'+X, X)\right) \leq 1 + (-k - m - 1 + 2m + 1) + k \leq m+1$. Since we proved that $S_m(n'+X, X)$ has at least $m+1$ distinct roots, $deg\left(S_m(n'+X, X)\right) = m+1$. To determine the leading coefficient $c_{m,n'}$ of $S_m(n'+X, X)$, we observe that every term of the summation in Eq. (4.23) is a polynomial of degree $m+1$. Then, we identify the leading coefficient of every term, and sum it with the other coefficients to obtain

$$\begin{aligned} c_{m,n'} &= \sum_{k=0}^{m} (-1)^{k+m} 4 \prod_{r=k+1}^{m} (r+n') \left(\prod_{r=-k+m+1}^{m} r \right) 4^{m-k} 4^k. \\ &= (-1)^m 4^{m+1} \sum_{k=0}^{m} (-1)^k \frac{(m+n')!}{(k+n')} \frac{m!}{(m-k)!}. \end{aligned}$$

By using the index substitution $k \leftarrow m - k$, we obtain

$$c_{m,n'} = 4^{m+1} m! \sum_{k=0}^{m} (-1)^k \frac{(m+n')!}{k!(m+n'-k)!} = 4^{m+1} m! \sum_{k=0}^{m} (-1)^k \binom{m+n'}{k}. \tag{4.24}$$

Henceforth, the objective is to show that

$$\sum_{k=0}^{m}(-1)^k\binom{m+n'}{k} = (-1)^m\frac{1}{m!}\prod_{r=0}^{m-1}(r+n') = (-1)^m\frac{(m+n'-1)}{m!(n'-1)!} = (-1)^m\binom{m+n'-1}{m}.$$

To do so, we use Pascal's rule to obtain

$$\begin{aligned}\sum_{k=0}^{m}(-1)^k\binom{m+n'}{k} &= \sum_{k=0}^{m}(-1)^k\left(\binom{m+n'-1}{k}+\binom{m+n'-1}{k-1}\right)\\ &= \sum_{k=0}^{m-1}(-1)^k\binom{m+n'-1}{k} + (-1)^m\binom{m+n'-1}{m} + \sum_{k=0}^{m-1}(-1)^{k+1}\binom{m+n'-1}{k}\\ &= (-1)^m\binom{m+n'-1}{m}.\end{aligned} \tag{4.25}$$

As a result, by using Eq. (4.25) in Eq. (4.24), the leading coefficient $c_{m,n'}$ of $S_m(n'+X,X)$ is

$$c_{m,n'} = (-1)^m 4^{m+1} m!\binom{m+n'-1}{m} = (-1)^m 4^{m+1}\frac{(m+n'-1)!}{(n'-1)!} = (-1)^m 4^{m+1}\prod_{r=0}^{m-1}(r+n').$$

Thus, since $\forall r \in [\![0,m]\!], m-\frac{1}{4}(2r+1)$ is a root of $S_m(n'+X,X)$, and $deg(S_m(n'+X,X)) = m+1$, we have

$$\begin{aligned}S_m(n'+X,X) &= (-1)^m 4^{m+1}\prod_{r=0}^{m}(X-m+\frac{1}{4}(2r+1))\prod_{r=0}^{m-1}(r+n')\\ &= (-1)^m 4^{m+1}\prod_{r=0}^{m}\left(\frac{4(X-m)+2r+1}{4}\right)\prod_{r=0}^{m-1}(r+n')\\ &= (-1)^m\prod_{r=0}^{m}(4(X-m)+2r+1)\prod_{r=0}^{m-1}(r+n').\end{aligned}$$

We then have $S_m(n'+p,p) = S_{m,n'+p,p}$ and since $n = n'+p$, we have $n' = n-p$ and we obtain

$$S_{m,n,p} = (-1)^m\prod_{r=0}^{m}(4(p-m)+2r+1)\prod_{r=0}^{m-1}(r+n-p). \tag{4.26}$$

By using Eq. (4.26) in Eq. (4.13), we obtain

$$\begin{aligned} v_{2(p-m),2n}^{[M,N,l]} &= \frac{(-1)^m l^{2(n+m-p)}(2n)!}{2^{n+m-p}(2(p-m))!(n-p+m)!m!} \prod_{r=2(p-m)}^{p+n} \frac{1}{2r+1} \prod_{r=0}^{m} (4(p-m)+2r+1) \prod_{r=0}^{m-1} (r+n-p) \\ &= \frac{(-1)^m l^{2(n+m-p)}(2n)!}{2^{n+m-p}(2(p-m))!(n-p+m)!m!} \prod_{r=2(p-m)}^{p+n} \frac{1}{2r+1} \prod_{r=2(p-m)}^{2p-m} (2r+1) \prod_{r=0}^{m-1} (r+n-p) \\ &= \frac{(-1)^m l^{2(n+m-p)}(2n)!}{2^{n+m-p}(2(p-m))!(n-p+m)!m!} \prod_{r=2p-m+1}^{p+n} \frac{1}{2r+1} \prod_{r=n-p}^{n-p+m-1} r. \end{aligned}$$

Hence, we finally conclude that

$$v_{2m,2n}^{[M,N,l]} = \frac{(-1)^{p-m} l^{2(n-m)}(2n)!}{2^{n-m}(n-m)!(p-m)!(2m)!} \prod_{r=n-p}^{n-m-1} r \prod_{r=p+m+1}^{p+n} \frac{1}{2r+1}.$$

The same steps can be followed to determine $v_{2m+1,2n+1}^{[M,N,l]}$. ■

Since the elements of $V^{[M,N,l]}$ are defined using products, recurrence relationships are more interesting for computation purposes.

Corollary 4.4.2.1. *Let $l \in \mathbb{R}_+^*$ and $M, N \in \mathbb{N}$ such that $M < N$. Let $p = \left\lfloor \frac{M}{2} \right\rfloor, p_1 = \left\lfloor \frac{M-1}{2} \right\rfloor, q = \left\lfloor \frac{N}{2} \right\rfloor$ and $q_1 = \left\lfloor \frac{N-1}{2} \right\rfloor$. We have*

$$\forall (m,n) \in [\![0, p-1]\!] \times [\![p+1, q]\!], v_{2m+2,2n}^{[M,N,l]} = -\frac{(n-m)(p-m)(2(p+m)+3)}{l^2(n-m-1)(2m+1)(m+1)} v_{2m,2n}^{[M,N,l]},$$

$$\forall (m,n) \in [\![0, p]\!] \times [\![p+1, q-1]\!], v_{2m,2n+2}^{[M,N,l]} = \frac{l^2(2n+1)(n+1)(n-m)}{(n-m+1)(n-p)(2(p+n)+3)} v_{2m,2n}^{[M,N,l]},$$

$$\forall (m,n) \in [\![0, p_1-1]\!] \times [\![p_1+1, q_1]\!], v_{2m+3,2n+1}^{[M,N,l]} = -\frac{(n-m)(p-m)(2(p+m)+5)}{l^2(n-m-1)(2m+3)(m+1)} v_{2m+1,2n+1}^{[M,N,l]},$$

$$\forall (m,n) \in [\![0, p_1]\!] \times [\![p_1+1, q_1-1]\!], v_{2m+1,2n+3}^{[M,N,l]} = \frac{l^2(2n+3)(n+1)(n-m)}{(n-m+1)(n-p)(2(p+n)+5)} v_{2m+1,2n+1}^{[M,N,l]}.$$

Proof. All the formulas can be derived directly from Theorem 4.4.2. ■

Using the fact that $V^{[M,M,l]} = I_M$ and Theorem 4.4.2, we can finally determine the coefficients $(b_0, ..., b_M)$ in the canonical basis for the polynomial degree reduction of P in $\langle \cdot, \cdot \rangle_l$ as such:

$$\begin{cases} \forall m \in [\![0, \left\lfloor \frac{M}{2} \right\rfloor]\!], b_{2m} = a_{2m} + \sum_{n=\left\lfloor \frac{M}{2} \right\rfloor + 1}^{\left\lfloor \frac{N}{2} \right\rfloor} v_{2m,2n}^{[M,N,l]} a_{2n} \\ \forall m \in [\![0, \left\lfloor \frac{M-1}{2} \right\rfloor]\!], b_{2m+1} = a_{2m+1} + \sum_{n=\left\lfloor \frac{M-1}{2} \right\rfloor + 1}^{\left\lfloor \frac{N-1}{2} \right\rfloor} v_{2m+1,2n+1}^{[M,N,l]} a_{2n+1} \end{cases} \tag{4.27}$$

4.5 Computational Complexity and Examples

In this section, we will compare the computational complexity of determining the canonical coefficients $(b_0, ..., b_M)$ using Eq. (4.27), and the computational complexity of determining them by calculating $V^{[M,N,l]}$ as a matrix product. We will also present an example of how Eq. (4.27) can be used and an example showing its numerical stability compared to the classical approach.

Proposition 4.5.1. *The computational complexity of using Eq. (4.27) is* $\mathcal{O}((N-M)M)$.

Proof. According to Eq. (4.27), the number of summations and products that are required to determine one coefficient b_m is at most $2\left(\left\lfloor \frac{N}{2} \right\rfloor - \left\lfloor \frac{M}{2} \right\rfloor\right)$. However, this is without counting how many operations are needed to determine $v_{m,n}^{[M,N,l]}$. According to Corollary 4.4.2.1, the number of operations to determine $v_{m,n}^{[M,N,l]}$ is constant and does not depend on M or N. Therefore, it can be ignored for complexity calculations. Since we need to calculate M coefficients, the computational complexity is then of $\mathcal{O}\left(2M\left(\left\lfloor \frac{N}{2} \right\rfloor - \left\lfloor \frac{M}{2} \right\rfloor\right)\right)$. We have $2\left(\left\lfloor \frac{N}{2} \right\rfloor - \left\lfloor \frac{M}{2} \right\rfloor\right) \leq N - M + 1$. Thus, we conclude that the computational complexity is $\mathcal{O}((N-M)M)$. ■

Proposition 4.5.2. *The computational complexity of determining* $(b_0, ..., b_M)$ *by calculating* $V^{[M,N,l]}$ *using a matrix product is* $\mathcal{O}((N-M)M^2)$.

Proof. We can assume that to calculate $V^{[M,N,l]}$, we only need to determine the elements of $V^{[M,N,l]}$ whose columns are greater than M, since $V^{[M,M,l]} = I_M$. This is equivalent to performing a matrix product between a $(M+1)\times(M+1)$ matrix and a $(M+1)\times(N-M)$ matrix which has a $\mathcal{O}((N-M)M^2)$ complexity. The complexities of determining $T^{[M,N,l]}$ and $\left(T^{[M,M,l]}\right)^{-1}$ using Eqs. (4.3) and (4.7) are negligible compared to a matrix product. Therefore, the computational complexity of determining $(b_0, ..., b_M)$ by calculating $V^{[M,N,l]}$ using a matrix product is $\mathcal{O}((N-M)M^2)$. ■

According to Proposition 4.5.1 and Proposition 4.5.2, directly computing $(b_0, ..., b_M)$ using Eq. (4.27) is at least M times less complex than calculating $V^{[M,N,l]}$. What follow are two examples showing the use of Theorem 4.4.2 and Eq. (4.27).

Example 3. $N = 7, M = 5$

Let $l \in \mathbb{R}_+^*$. Using Theorem 4.4.2, we find that

$$V^{[5,7,l]} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & l^6\frac{1440}{66528} & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & l^6\frac{10080}{123552} \\ 0 & 0 & 1 & 0 & 0 & 0 & -l^4\frac{720}{1584} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -l^4\frac{5040}{6864} \\ 0 & 0 & 0 & 0 & 1 & 0 & l^2\frac{720}{528} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & l^2\frac{5040}{3120} \end{pmatrix}$$

For a polynomial $P = \sum_{k=0}^{7} a_k X^k$ where $(a_0, ..., a_7) \in \mathbb{R}^8$, the canonical coefficients $(b_0, ..., b_5) \in \mathbb{R}^6$ of the polynomial of degree 5 that best approximates P with respect to $\langle \cdot, \cdot \rangle_l$ are determined using Eq. (4.27) as such:

$$\begin{cases} b_0 &= a_0 + l^6\frac{1440}{66528}a_6 \\ b_1 &= a_1 + l^6\frac{10080}{123552}a_7 \\ b_2 &= a_2 - l^4\frac{720}{1584}a_6 \\ b_3 &= a_3 - l^4\frac{5040}{6864}a_7 \\ b_4 &= a_4 + l^2\frac{720}{528}a_6 \\ b_5 &= a_5 + l^2\frac{5040}{3120}a_7 \end{cases}$$

Figure 4.1 shows an example of a polynomial of degree 7 approximated by a polynomial of degree 5 on the interval $[-5, 5]$.

Example 4. $N = 150, M = 40, l = 1$

In this example, we want to reduce a polynomial of degree 150 to a polynomial of degree 40

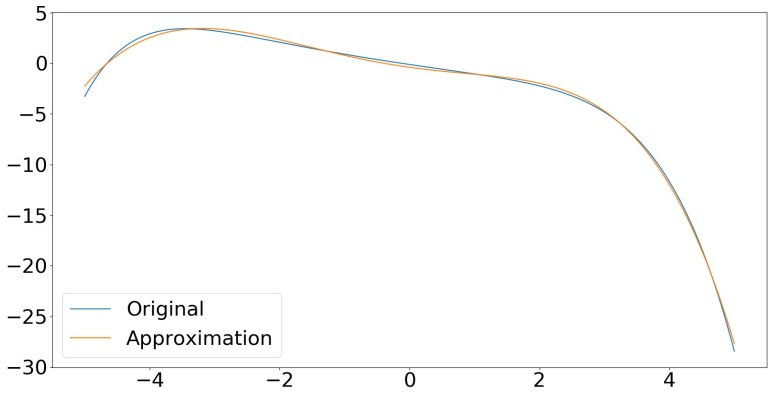
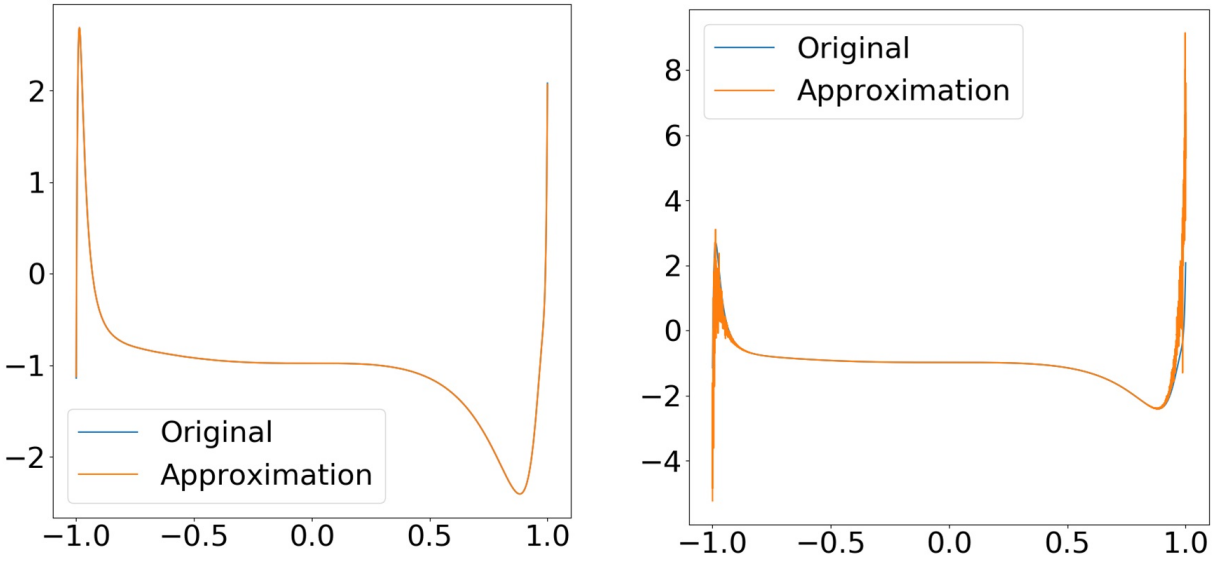


FIGURE 4.1: A polynomial of degree 7 (in blue) approximated by a polynomial of degree 5 (in orange) on the interval $[-5, 5]$.

on the interval $[-1, 1]$. *Figure 4.2.(a) shows the original polynomial in blue and its approximation in orange using the direct formula while figure 4.2.(b) shows the original polynomial in blue and its approximation in orange using the matrix multiplication approach. We notice that there are artifacts that corrupt the approximation using the matrix multiplication approach and this is mainly due to the floating point precision of the computation. Indeed, the matrix multiplication approach involves additions and multiplications between floating point quantities in order to determine one element of* $V^{[M,N,l]}$. *As a result, the direct formula is numerically more stable since, according to Theorem 4.4.2, every element of* $V^{[M,N,l]}$ *is the multiplication between a floating point quantity* l *and a fraction. In fact, a fraction is defined as a floating point quantity but it results from the division of two integer quantities. Consequently, the numerical floating point representation of the fraction will be better preserved than when using the matrix multiplication approach, hence the smooth approximation that is observed in Figure 4.2.(a).*

4.6 N-Dimensional Polynomial Neural Networks and Polynomial Degree Reduction Heuristic

1DPNNs are an extension of 1DCNNs such that each neuron creates a polynomial approximation of a kernel that is applied to its input. Given a network with L layers, such that a layer $l \in [\![1, L]\!]$ contains N_l neurons, a neuron $i \in [\![1, N_l]\!]$ in a layer l produces an output $y_i^{(l)}$ from its previous layer's output Y_{l-1}. The neuron possess a bias $b_i^{(l)}$, an activation function $f_i^{(l)}$, and D_l weight vectors $W_{id}^{(l)}$ corresponding to every exponentiation d of its previous layer's output up to a degree D_l. Eq. (3.1) shows that the weight $W_{id}^{(l)}$ corresponding to Y_{l-1}^d can be of any dimension as long as the convolution with Y_{l-1}^d remains valid. For instance, if Y_{l-1}^d is a list of 2D feature maps, $W_{id}^{(l)}$ has to be a list of 2D filter masks. Therefore, the equation of a 1DPNN neuron



(a) Polynomial reduction using the direct formula.

(b) Polynomial reduction using the classical matrix multiplication approach.

FIGURE 4.2: Comparison between the use of the direct formula and the use of the matrix multiplication approach on polynomial reduction.

can be applied on images and videos as long as the dimension of the weights are adjusted accordingly. Hence, the same equation governs the behavior of a 2DPNN, a 3DPNN and an NDPNN in general. Furthermore, this also applies to the gradient estimation of an NDPNN detailed in Section 3.2.3, so even the backpropagation remains unchanged. Furthermore, we propose a method to reduce the degree of each layer after an NDPNN network is fully trained on a given dataset. This makes use of the fast polynomial degree reduction formula that we proposed in Section 4.4.2, which can generate a polynomial of low degree that behaves the same as a given polynomial of higher degree on a symmetric interval. The method that we propose is a post-processing method that can compress a fully trained NDPNN, thus making it faster and lighter, without sacrificing its performance on the dataset it was trained on. The memory and computational efficiency gain mainly depend on the topology of the NDPNN and the performance loss tolerance. Although the polynomial degree reduction is performed on a symmetric interval and an NDPNN can use unbounded activation functions in general, we use the fact that, after training, the NDPNN weights do not change. So the input of each layer will be bounded in a certain interval when the NDPNN is fed with samples from the training set. The bounding interval may not be symmetric, but every interval is contained within a symmetric one, which allows us to properly use the polynomial degree reduction formula. Eq. (3.1) does not clearly show how the polynomial degree reduction can be achieved, which is why we consider the case of 1DPNNs to demonstrate the principle.

In the 1D case, the output of a given layer l is Y_l which is a list containing the outputs of every

neuron in that layer. The output of a neuron i in that layer, $y_i^{(l)} = f_i^{(l)}\left(x_i^{(l)}\right)$, is a vector of size M_l. Similarly, the weights of that neuron with respect to the exponentiation degree d, $W_{id}^{(l)}$, is a list of N_{l-1} vectors of size K_l. Therefore, expanding $x_i^{(l)}$ from Eq. (3.1) to compute each of its vector element independently produces:

$$\begin{aligned}
\forall m \in [\![0, M_l - 1]\!], x_i^{(l)}(m) &= \sum_{d=1}^{D_l} \sum_{j=1}^{N_{l-1}} \sum_{k=0}^{K_l-1} w_{ijd}^{(l)}(k) \left(y_j^{(l-1)}(m+k)\right)^d + b_i^{(l)} \\
&= \sum_{j=1}^{N_{l-1}} \sum_{k=0}^{K_l-1} \left(\sum_{d=1}^{D_l} w_{ijd}^{(l)}(k) \left(y_j^{(l-1)}(m+k)\right)^d + \frac{b_i^{(l)}}{N_{l-1}K_l} \right) \\
&= \sum_{j=1}^{N_{l-1}} \sum_{k=0}^{K_l-1} P_{ijk}^{(l)}\left(y_j^{(l-1)}(m+k)\right),
\end{aligned}$$

such that

$$P_{ijk}^{(l)}(X) = \sum_{d=1}^{D_l} w_{ijd}^{(l)}(k) X^d + \frac{b_i^{(l)}}{N_{l-1}K_l},$$

and X is an indeterminate. This shows that the output of a neuron i in a layer l is the result of the summation of $N_{l-1}K_l$ distinct polynomials, and that the layer consists of $N_l N_{l-1} K_l$ distinct polynomials. In the general case of an NDPNN, the number of polynomials in a layer l would be $N_l N_{l-1}$ multiplied by the receptive field of that layer. Figure 4.3 shows an overview of the heuristic algorithm and Algorithm 1 describes the process of compressing an NDPNN by means of layer-wise polynomial degree reduction. F represents the trained model which takes as input samples from a given training set T and processes them into an output. Note that given a model F, one can access the output of its layer l by using F_l. The algorithm needs a performance evaluation function ϵ which takes a model, a dataset and its labels T_{true} as input and needs a reduction tolerance ϵ_0 which stops the algorithm when the performance evaluation of the reduced model is below ϵ_0. ϵ can be any performance evaluation metric which outputs a higher score when the performance of the model is better, such as the accuracy. The algorithm starts by initializing the reduction of each layer R to 0 and by determining the smallest symmetric interval $[-A_l, A_l]$ where the values of the input of each layer l are bounded. Then, it goes through each layer l', it creates a copy $\tilde{F}$ of the initial model F, and reduces every layer's degree D_l of the copy $\tilde{F}$ by the last degree reduced R on the symmetric interval $[-A_l, A_l]$, except for layer l' which has its degree reduced by $R+1$ on the symmetric interval $[-A_{l'}, A_{l'}]$ as expressed by the instruction $\tilde{D}_l \leftarrow D_l - \left(R_l + \mathbb{1}_{\{l'\}}(l)\right)$ where $\mathbb{1}$ is the indicator function. The weights of each layer of the initial model copy $\tilde{F}$ are therefore replaced during this process by reduced weights calculated via the polynomial degree reduction formula. If a layer l' has been reduced to the degree 1, the algorithm does not attempt to reduce it further and ignores it by assigning a nil

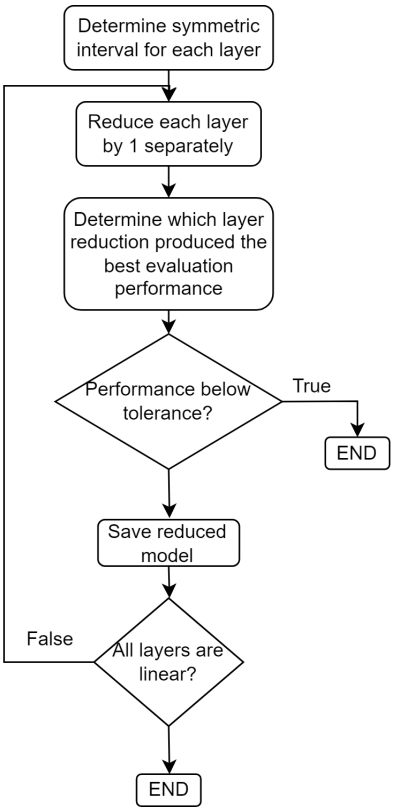


FIGURE 4.3: Overview of the layer-wise degree reduction algorithm.

performance $P_{l'} = 0$. It then evaluates the copy $\tilde{F}$ using the performance evaluation function ϵ and stores the score $P_{l'}$ in a list. After performing these steps for every layer l' in the model, the algorithm determines the layer $\tilde{l}$ whose reduction impacted the performance of the model the least and increases its reduction $R_{\tilde{l}}$ by 1. These steps are repeated until all the layers are reduced to a degree of 1 or until the best performance of the current reduction is below ϵ_0. Following that, the algorithm creates a final copy $\tilde{F}$ of the model F and reduces the degree of each layer l according to the reduction limit R_l determined in the previous steps. The algorithm then returns the most reduced model within the limit of ϵ_0.

4.7 Chapter Summary

In this chapter, we developed a fast and stable mathematical formula for polynomial degree reduction on a symmetric interval for the canonical basis which can not only be used on NDPNNs, but on any model that either uses polynomial kernels or whose core is built on a polynomial approximation. We also showed that the equations governing the behavior of a 1DPNN's forward and backward propagation are invariant with respect to the dimension of the signal which allowed us to generalize the 1DPNN to an NDPNN that can process 2D and 3D signals. In addition, we used the polynomial degree reduction formula to develop a heuristic algorithm that

Algorithm 1: NDPNN layer-wise polynomial degree reduction

Data:

- $L, N_l, D_l, W_{id}^{(l)}, b_i^{(l)}, \forall l \in [\![1, L]\!], \forall (i, d) \in [\![1, N_l]\!] \times [\![1, D_l]\!]$
- Trained model F
- Training set T and training labels T_{true}
- Evaluation function $\epsilon(F, T, T_{true})$
- Reduction tolerance ϵ_0

Result:

- $\tilde{D}_l, \tilde{W}_{id}^{(l)}, \tilde{b}_i^{(l)}, \forall l \in [\![1, L]\!], \forall (i, d) \in [\![1, N_l]\!] \times [\![1, \tilde{D}_l]\!]$
- Reduced model $\tilde{F}$

$A \leftarrow (0, ..., 0)_L$
$R \leftarrow (0, ..., 0)_L$
$P \leftarrow (\epsilon_0, ..., \epsilon_0)_L$
$A_1 \leftarrow \max|T|$
for $l \in [\![2, L]\!]$ **do**
  $A_l \leftarrow \max\left(|F_{l-1}(T)|\right)$
end
$\tilde{l} \leftarrow 0$
while $P_{\tilde{l}} \geq \epsilon_0 \wedge \max_{l \in [\![1,L]\!]} (D_l - R_l) > 1$ **do**
  for $l' \in [\![1, L]\!]$ **do**
    $\tilde{F} \leftarrow F$
    for $l \in [\![1, L]\!]$ **do**
      $\tilde{D}_l \leftarrow D_l - \left(R_l + \mathbb{1}_{\{l'\}}(l)\right)$
      if $\tilde{D}_l \geq 1$ **then**
        for $i \in [\![1, N_l]\!]$ **do**
          $\left(\left(\tilde{W}_{id}^{(l)}\right)_{[\![1,\tilde{D}_l]\!]}, \tilde{b}_i^{(l)}\right) \leftarrow$reduce_degree_to_n$\left(\left(W_{id}^{(l)}\right)_{[\![1,D_l]\!]}, b_i^{(l)}, D_l, \tilde{D}_l, A_l\right)$
          $\tilde{F} \leftarrow$replace_neuron_weights$\left(\tilde{F}, \tilde{b}_i^{(l)}, \left(\tilde{W}_{id}^{(l)}\right)_{[\![1,\tilde{D}_l]\!]}, \tilde{D}_l, i, l\right)$
        end
      else
        $P_{l'} \leftarrow 0$
        Break the innermost For loop
      end
    end
    $P_{l'} \leftarrow \epsilon(\tilde{F}, T, T_{true}) * \left(1 - \mathbb{1}_{\{0\}}(P_{l'})\right)$
  end
  $\tilde{l} \leftarrow \underset{l \in [\![1,L]\!]}{argmax} P$
  if $P_{\tilde{l}} \geq \epsilon_0$ **then**
    $R_{\tilde{l}} \leftarrow R_{\tilde{l}} + 1$
  end
end
$\tilde{F} \leftarrow F$
for $l \in [\![1, L]\!]$ **do**
  $\tilde{D}_l \leftarrow D_l - R_l$
  for $i \in [\![1, N_l]\!]$ **do**
    $\left(\left(\tilde{W}_{id}^{(l)}\right)_{[\![1,\tilde{D}_l]\!]}, \tilde{b}_i^{(l)}\right) \leftarrow$reduce_degree_to_n$\left(\left(W_{id}^{(l)}\right)_{[\![1,D_l]\!]}, b_i^{(l)}, D_l, \tilde{D}_l, A_l\right)$
    $\tilde{F} \leftarrow$replace_neuron_weights$\left(\tilde{F}, \tilde{b}_i^{(l)}, \left(\tilde{W}_{id}^{(l)}\right)_{[\![1,\tilde{D}_l]\!]}, \tilde{D}_l, i, l\right)$
  end
end

performs layer-wise degree reduction on a pre-trained NDPNN while preserving its performance on the dataset it was trained on.

Chapter 5

Plant Species Recognition with Optimized 3DPNN and Variably Overlapping Time–Coherent Sliding Window

5.1 Introduction

Agriculture is a sector that requires multi-disciplinary knowledge to steadily evolve [76, 77, 78] since large-scale food production necessitates a deep understanding of every relevant plant species [79, 80, 81] and highly advanced machinery [82, 83, 84] to ensure an optimized yield. With the emergence and the recent practical successes of the Internet of Things, robotics and artificial intelligence [85], the sector has observed a surge in smart farming solutions which has led the march to the fourth agricultural revolution [86]. Consequently, new fields such as digital agriculture [87] and precision agriculture [88] have become intensively researched which has led to an ever-increasing pace in innovation. Although the integration of artificial intelligence is making its way to digital agricultural applications such as crop planting [89, 90, 91] and harvesting [92, 93, 94], it is still limited to mechanical tasks that mainly require environmental awareness [82]. A number of issues that are highly concerning for farmers such as crop disease detection, plant-growth monitoring and need-based irrigation have only recently started to gain an interest in the machine learning community [95]. Despite the efforts to create automated systems to resolve such issues [96, 97], only highly specialized systems are created which only work with specific species or variants under constrained conditions [98, 99, 100]. Furthermore, due to the lack of voluminous labeled data for each species and for each specific issue [95], automated systems mainly rely on advanced visual feature engineering that requires a team of plant specialists, and only partially rely on machine learning for using these features to extract useful information [101, 102, 103]. In fact, the sheer diversity of plant species, variants,

diseases, growth stages and growing conditions makes manual feature engineering unfeasible to cover every individual farmer's need. While some attempts at creating massive plants datasets have been successful [104, 105, 106], their usefulness was limited by the species present in them and the initial task they were created for. For instance, a dataset containing species grown in a tropical biome for the purpose of species recognition has limited to no usefulness for farmers working in a grassland biome who want to establish the presence of a disease in certain plants. As a result, the EAGL–I system [107] has been recently proposed to automatically generate a high number of labeled images in a short time (1 image per second) in an effort to circumvent the problem of the lack of data for specific needs. Consequently, two massive datasets were created with this system [108]; one that contains 1.2 million images of indoor-grown crops and weeds common to Canadian prairies, and one that contains 540,000 images of plants imaged in farmland. Also, a publicly available dataset called "Weed seedling images of species common to Manitoba, Canada" (WSISCMC) [23] which contains 40,000 images of 8 species that are very rarely represented in plants datasets was created with the EAGL–I system. The purpose of the creation of the dataset was to demonstrate the capability of the EAGL–I system to rapidly generate large amounts of data that are suitable for machine learning applications and that can be used to solve specific digital agriculture problems, such as the ability to recognize several species of invasive species which are responsible for the loss of hundreds of millions of dollars. However, in [107], the validity of the dataset was only tested for a binary classification problem consisting in differentiating between grasses and non-grasses without determining the species of the given plant itself. While the model that solves this problem gives an indication on how well the samples of the dataset are distributed to allow for grass differentiation, it does not provide enough information on how detailed the samples of the dataset are at a granular level to allow the identification of the species they belong to. Indeed, the higher the number of mutually exclusive classes, the more distinctive and detailed the spatial features of the plants need to be. Therefore, we chose to tackle the problem of plant species recognition on the WSISCMC dataset leading to a solution that can help the early eradication of grasses that are harmful to the growth of certain crops. Furthermore, this constitutes a step further in validating the dataset for machine learning applications, and by extension, the EAGL–I system for the creation of meaningful datasets. Thus, this work aims at maximizing the plant recognition accuracy on the WSISCMC dataset by creating a highly reliable and accurate network using the model development framework shown in Figure 5.1 in order to provide insight on how to improve the data acquisition process to produce cleaner samples for future massive datasets.

The outline of this chapter is as follows. Section 5.2 discusses the most recent works in plant species recognition while Section 5.3 formulates the theoretical foundation of the VOTCSW method. Section 5.4 presents the model development framework established to produce a highly accurate model for the WSISCMC dataset and discusses the results obtained while providing

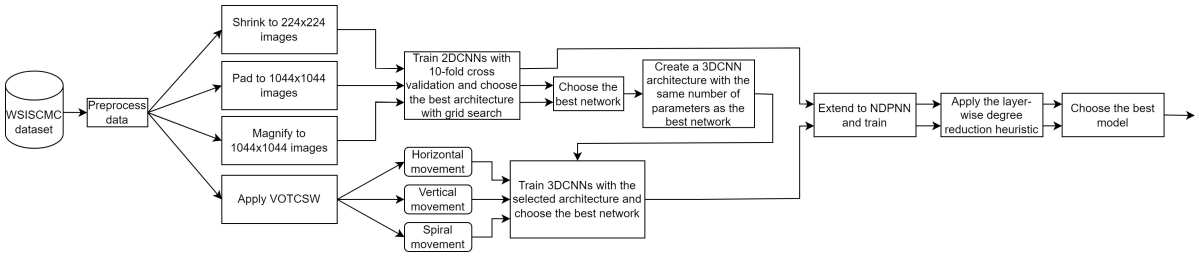


FIGURE 5.1: Block diagram for producing a reliable plant species classification model.

insights on how the methods developed in Section 4.6 and Section 5.3 influenced them.

5.2 Related Work

Plant species recognition is one of the most important tasks in the application of machine learning to digital agriculture. In this context, behaviour specific to a species will inform the identification of a plant's disease or the plant's need for resources such as water or light. The most common approach to identify the species of a given plant is to analyze its leaves [109]. In fact, many public datasets [104, 110] are only composed of leaves scanned in a uniform background. The methods that are mostly used rely either on combining feature engineering and machine learning classifiers or using deep learning models for both feature extraction and classification.

Indeed, Purohit *et al.* [111] created morphological features based on the geometric shape of any given leaf and used these features to discriminate between 33 species of plants using different classifiers. They achieved a state-of-the-art 95.42% accuracy on the Flavia dataset [104] and they demonstrated that their morphological features are superior to color features or texture features. However, they did not compare the efficiency of their features to ones that are fully determined using deep learning models. On the contrary, Wang *et al.* [112] created a novel multiscale convolutional neural network (CNN) with an attention mechanism that can filter out the global background information of the given leaf image and highlight its saliency which allows it to consistently outperform classification models based on morphological feature extraction, as well as classification models based on well-known CNN architectures. They explain that CNNs are better at extracting high-level and low-level features without the need to perform image preprocessing and that their model which combines both these types of features in an attempt to discover estimatable relationships outperforms regular CNNs. However, the models that were developed were only trained on the ICL dataset [110] which only contains leaf images, and can therefore be hard to use on images of entire plants.

Mehdipour Ghazi *et al.* [113] have considered training versatile architectures of CNNs on the LifeCLEF 2015 [106] which contains $100,000$ images of plant organs from 1000 species that are mostly captured outdoors. They proposed a data augmentation approach that randomly

extracts and scales a number of random square patches from any given image before applying a rotation. These patches and the original image are then resized to a fixed size and the mean image is substracted from them in order to keep the most relevant features. The resulting images are then fed to a CNN model which outputs a prediction for each image. The prediction of the original image is then determined by summing these predictions together. This aggregation as well as fine-tuning pre-trained networks such as VGGNet [114], GoogLeNet [115], and AlexNet [116] with different hyperparameters controlling the number of weight updates, the batch size and the number of patches, allowed them to achieve state-of-the-art results on the dataset, notably by fusing VGGNet and GoogLeNet. The authors observe that, when training networks from scratch, simpler architectures are preferred to the kind of architectures they used in their work. Indeed, they could not train any network from scratch to produce satisfactory results, and they did not attempt to create simpler and more specialized architectures for the problem. The work presented in this thesis differs from the related papers in two ways:

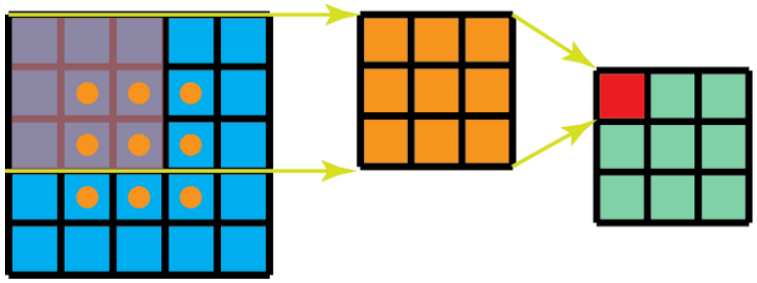
1. The WSISCMC dataset contains images of entire plants that trace different growth stages such that a trained classifier may be able to integrate the temporal evolution of a species organs in its inference, and may produce features richer than the ones that are only determined from separate organs such as leaves.

2. Concomitantly, the neural network architectures that we used are constructed in various stages of simplicity. Starting from a simple 2DCNN architecture, a different data representation suitable for a simple 3DCNN architecture is built, then both architectures are extended to NDPNNs. Finally, highly complex architectures such as InceptionV3 and ResNet50V2 are considered.

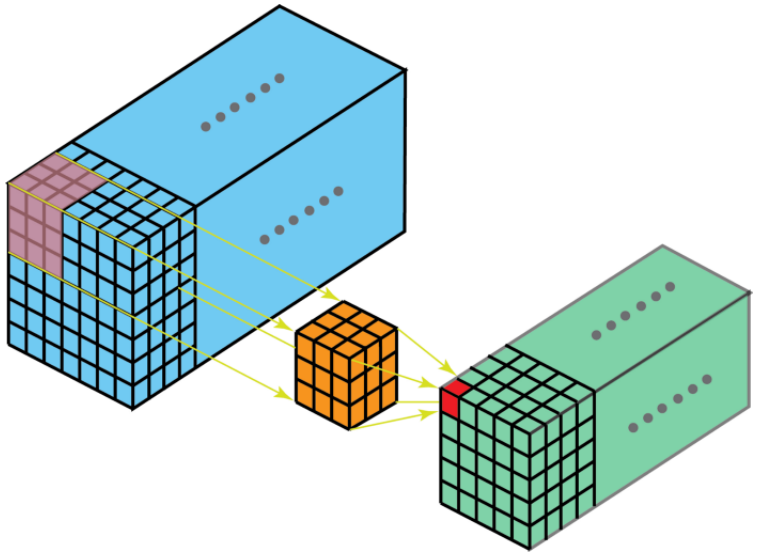
5.3 Variably Overlapping Time–Coherent Sliding Window

The WSISCMC plant species classification dataset used in this work contains images with varying sizes which are not suitable for neural networks that only accept an input with a predetermined size. As a result, there is a need to transform these images into a representation with a fixed size. In most cases, shrinking the images to the smallest size present in the dataset is enough to train a network to produce very accurate results. However, image resizing comes at the cost of either losing important details that may be detrimental for the performance of the network when shrinking, or adding synthetic pixels when padding or magnifying. Thus, we created the Variably Overlapping Time–Coherent Sliding Window (VOTCSW) technique, which transforms each image, regardless of its size, to a 3-dimensional representation with fixed size (h, w, M). The VOTCSW allows this representation to be interpreted as a video of size (h, w) containing M frames by ensuring that two consecutive frames are spatially correlated hence the "Time–Coherent" term. Therefore, the 3-dimensional representation can be fed into 3DCNNs

and 3DPNNs as a tensor of shape (h, w, M, C) where C is the number of channels present in the original image. Figure 5.2 (adapted from [117]) shows the difference between a 2D convolution applied on an image and a 3D convolution applied on a video similar to the ones that can be created using the VOTCSW method. The difference resides in the way a filter is defined and the way it slides. To apply 2D convolution on an image, a 2D filter – of size 3×3 in this example – slides over the 2 dimensions that define an image, namely, height and width, to produce a feature map. To apply 3D convolution on a video, a 3D filter – of size $3 \times 3 \times 3$ in this example – slides over the 3 dimensions that define a video, namely, height, width and time, to produce a feature map. The VOTCSW method is a safe alternative to resizing as it does not add nor remove any



(a) 2D convolution on an image.



(b) 3D convolution on a video.

FIGURE 5.2: Difference between 2D convolution and 3D convolution [117].

pixel from the original images, or at least, it minimizes the need to do so under certain conditions that will be discussed below. It is based on the sliding window technique (hence the "Sliding Window" term) which is a powerful signal processing tool that is used to decompose a signal containing a high number of samples into small chunks called windows containing a smaller number of samples that can be processed faster. Consecutive windows overlap with a certain

ratio $\alpha \in [0, 1[$ in order to ensure a correlation between them. The classical use of this technique consists in determining a window size that ensures enough representative samples to be present in a single window and an overlap that allows better processing performance. However, there is no consideration as to how many windows are extracted for each different signal length. In constrast, our proposed VOTCSW method aims to extract exactly M windows of a fixed size from any signal regardless of its length. This is achieved by calculating an overlap for each signal length, hence the Variably Overlapping term.
Given an image of size (H, W) and a desired 3-dimensional representation (h, w, M), we define the following relationships:

$$\begin{cases} H &= h + N_h(1-\alpha)h \\ W &= w + N_w(1-\alpha)w \end{cases}, \tag{5.1}$$

where $\alpha \in [0, 1[$ is the window overlap, N_h is the number of windows that overlap with their predecessors and that are needed to cover the height of the image, and N_w is the number of windows that overlap with their predecessors and that are needed to cover the width of the image. Following that, the total number of windows M needed to cover the image in its entirety is

$$M = (1 + N_h)(1 + N_w) = \left(\frac{H-h}{(1-\alpha)h} + 1\right)\left(\frac{W-w}{(1-\alpha)w} + 1\right). \tag{5.2}$$

Eq. (5.2) establishes a relationship between the overlap α, the size of the image (H, W) which is fixed, the size of the sliding window (h, w) which is fixed and the total number of windows M which is also fixed.

Proposition 5.3.1. *The value of α with respect to H, h, W, w and M is*

$$\alpha = 1 - \frac{((H-h)w + (W-w)h) + \sqrt{((H-h)w + (W-w)h)^2 + 4hw(H-h)(W-w)(M-1)}}{2hw(M-1)}. \tag{5.3}$$

Proof. From Eq.(5.2), we have

$$\begin{aligned} M &= \left(\frac{H-h}{(1-\alpha)h} + 1\right)\left(\frac{W-w}{(1-\alpha)w} + 1\right) \\ &\iff hwM(1-\alpha)^2 = (H - h + h(1-\alpha))\,(W - w + w(1-\alpha)) \\ &\iff hwM(1-\alpha)^2 = (H-h)(W-w) + ((H-h)w + (W-w)h)(1-\alpha) + hw(1-\alpha)^2 \\ &\iff hw(M-1)(1-\alpha)^2 - ((H-h)w + (W-w)h)(1-\alpha) - (H-h)(W-w) = 0. \end{aligned}$$

This is a second degree equation with $(1-\alpha)$ as unknown. Therefore, the discriminant is

$$\Delta = ((H-h)w + (W-w)h)^2 + 4hw(H-h)(W-w)(M-1).$$

Since M is the number of windows, it is necessarily greater than 1. Moreover, $H > h$ and $W > w$ because the window size is always smaller than the image size. Consequently $\Delta > 0$ and we have two solutions expressed as such:

$$1-\alpha_{1,2} = \frac{((H-h)w + (W-w)h) \pm \sqrt{\Delta}}{2hw(M-1)}.$$

However, $\sqrt{\Delta} > ((H-h)w + (W-w)h)$ and since $\alpha \in [0,1[$ by definition, then $1-\alpha > 0$ and only the following valid solution remains:

$$\alpha = 1 - \frac{((H-h)w + (W-w)h) + \sqrt{\Delta}}{2hw(M-1)}.$$

■

Eq. (5.3) was determined from the fact that $\alpha < 1$. However, α needs to be positive as well. Therefore, there is a need to determine a condition on the choice of h, w and M with respect to H and W to ensure the positivity of α.

Proposition 5.3.2.

$$\alpha \geq 0 \iff hwM \geq HW.$$

Proof. We suppose that $\alpha \geq 0$.

$$\alpha \geq 0 \iff 1-\alpha \leq 1 \iff$$
$$\frac{((H-h)w + (W-w)h) + \sqrt{((H-h)w + (W-w)h)^2 + 4hw(H-h)(W-w)(M-1)}}{2hw(M-1)} \leq 1$$
$$\iff$$
$$\sqrt{((H-h)w + (W-w)h)^2 + 4hw(H-h)(W-w)(M-1)}$$
$$\leq 2hw(M-1) - ((H-h)w + (W-w)h)$$
$$\iff$$
$$((H-h)w + (W-w)h)^2 + 4hw(H-h)(W-w)(M-1)$$
$$\leq 4h^2w^2(M-1)^2 + ((H-h)w + (W-w)h)^2 - 4hw((H-h)w + (W-w)h)(M-1)$$

$$\iff 4hw(M-1)\left((H-h)(W-w)+(H-h)w+(W-w)h-hw(M-1)\right)\leq 0$$
$$\underset{M>1}{\iff} ((H-h)(W-w)+(H-h)w+(W-w)h-hw(M-1))\leq 0$$
$$\iff HW-Hw-hW+hw+Hw-hw+hW-hw-hwM+hw\leq 0$$
$$\iff HW-hwM\leq 0$$
$$\iff hwM\geq HW$$

■

Proposition 5.3.2 implies that h, w and M can not be chosen arbitrarily. Furthermore, since h, w and M need to be fixed before transforming the images of the dataset, they have to verify this condition for every image of size (H, W). As a result, we need to determine tighter conditions to be able to determine h, w and M consistently. Let β be the aspect ratio of an image of size (H, W) such that $W = \beta H$ and let γ be the aspect ratio of a window of size (h, w) such that $w = \gamma h$. We then derive from Eq. (5.1)

$$W = w + (1-\alpha)wN_w \iff \beta H = \gamma h + (1-\alpha)\beta h N_w \iff N_w = \frac{\beta H - \gamma h}{(1-\alpha)\gamma h}.$$

And since from Eq. (5.1), we have $N_h = \dfrac{H-h}{(1-\alpha)h}$, we deduce by dividing N_w by N_h that $N_w = \dfrac{\beta H - \gamma h}{\gamma(H-h)}N_h$ and thus

$$M = (N_h+1)\left(\frac{\beta H - \gamma h}{\gamma(H-h)}N_h + 1\right). \tag{5.4}$$

Since M is an integer constant, and N_h is an integer constant, $\dfrac{\beta H - \gamma h}{\gamma(H-h)}N_h$ should also be an integer constant. As a result, we can define a positive constant p such that $p = \dfrac{\beta H - \gamma h}{\gamma(H-h)}$. The only variables in p are β and H since they depend on the image being processed. γ and h are not supposed to change with the size of the image being processed. Consequently, we can write

$$p = \frac{\beta H - \gamma h}{\gamma(H-h)} \iff \gamma(H-h)p + \gamma h = \beta H \iff H(\beta - \gamma p) = \gamma h(1-p). \tag{5.5}$$

This means that when $p \neq 1$ and $\beta \neq \gamma p$, we have $H = \dfrac{\gamma h(1-p)}{\beta - \gamma p}$. This implies that, in order for M to be a valid integer, every image height has to be resized to H, which contradicts the purpose of the VOTCSW. Therefore, we consider the case where $\beta = \gamma p$. We can deduce from Eq. (5.5) that, under this condition, p will be equal to 1 and $\beta = \gamma$. As a result, the aspect ratio of each image should be equal to that of the sliding window for M to be a valid integer,

which is a simpler condition than the previous one. In the following, we will only consider the case where $\beta = \gamma$ since the dataset used in this work contains images that have the same aspect ratio. When $\beta = \gamma$, $M = (N_h + 1)^2$ which means that M should be a square number. This is yet another condition on how to choose M and this narrows down the possibilities even further. Under this assumption, the calculation of α can be simplified.

Proposition 5.3.3. *When $\beta = \gamma$, α only depends on M, H and h and its value is*

$$\alpha = \frac{\sqrt{M}h - H}{h(\sqrt{M} - 1)}. \tag{5.6}$$

Proof. By combining Eq. (5.1) and Eq. (5.4), we have

$$\begin{aligned} M = (N_h + 1)^2 = \left(\frac{H - h}{(1 - \alpha)h} + 1\right)^2 &\iff (1 - \alpha)^2 h^2 M = (H - \alpha h)^2 \\ &\iff (1 - \alpha)h\sqrt{M} = H - \alpha h \\ &\iff \alpha = \frac{\sqrt{M}h - H}{h(\sqrt{M} - 1)}. \end{aligned}$$

The positivity of α is verified when $h \geq \dfrac{H}{\sqrt{M}}$. ■

When extracting the windows from an image, we should limit the overlap α to not reach its extremum in order to obtain consistent windows. Therefore, we impose on α two limits α_{min} and α_{max} such that $0 \leq \alpha_{min} \leq \alpha \leq \alpha_{max} < 1$. Given H_{max}, the height of the biggest image in the dataset and H_{min}, the height of the smallest image in the dataset, we can formulate a condition on h.

Proposition 5.3.4. *The height h of the sliding window has to verify the following condition:*

$$\frac{H_{max}}{\sqrt{M} - \alpha_{min}(\sqrt{M} - 1)} \leq h \leq \frac{H_{min}}{\sqrt{M} - \alpha_{max}(\sqrt{M} - 1)}.$$

Proof. We know that $H_{min} \leq H \leq H_{max}$. Consequently, by using Eq. (5.6) we have

$$\alpha_{min} \leq \frac{\sqrt{M}h - H_{max}}{h(\sqrt{M} - 1)} \leq \alpha \leq \frac{\sqrt{M}h - H_{min}}{h(\sqrt{M} - 1)} \leq \alpha_{max}. \tag{5.7}$$

Therefore, by extracting h from Eq. (5.7) we obtain

$$\frac{H_{max}}{\sqrt{M} - \alpha_{min}(\sqrt{M} - 1)} \leq h \leq \frac{H_{min}}{\sqrt{M} - \alpha_{max}(\sqrt{M} - 1)}.$$

■

Although a condition on the choice h is important, the parameter that will most likely be chosen first when using the VOTCSW technique is M. However, α_{min} and α_{max} are as important as M because they specify the maximum and minimum amount of correlation between two consecutive windows. Hence, a condition on their choice is also important.

Theorem 5.3.1. *The parameters M, α_{min} and α_{max} can be determined in 6 different ways. For each way, there are conditions that need to be verified in order to extract the windows correctly.*

- *When determining M then α_{min} then α_{max}, the following conditions apply:*

$$\begin{cases} \sqrt{M} & \geq \dfrac{H_{max}}{H_{min}} \\ \alpha_{min} & \leq \dfrac{H_{max}}{H_{min}} + \left(1 - \dfrac{H_{max}}{H_{min}}\right) \dfrac{\sqrt{M}}{\sqrt{M}-1} \\ \alpha_{max} & \geq \left(1 - \dfrac{H_{min}}{H_{max}}\right) \dfrac{\sqrt{M}}{\sqrt{M}-1} + \dfrac{H_{min}}{H_{max}} \alpha_{min} \end{cases}$$

- *When determining M then α_{max} then α_{min}, the following conditions apply:*

$$\begin{cases} \sqrt{M} & \geq \dfrac{H_{max}}{H_{min}} \\ \alpha_{max} & \geq \left(1 - \dfrac{H_{min}}{H_{max}}\right) \dfrac{\sqrt{M}}{\sqrt{M}-1} \\ \alpha_{min} & \leq \dfrac{H_{max}}{H_{min}} \alpha_{max} + \left(1 - \dfrac{H_{max}}{H_{min}}\right) \dfrac{\sqrt{M}}{\sqrt{M}-1} \end{cases}$$

- *When determining α_{min} then M then α_{max}, the following conditions apply:*

$$\begin{cases} \sqrt{M} & \geq \dfrac{H_{max} - H_{min}\alpha_{min}}{(1-\alpha_{min})H_{min}} \\ \alpha_{max} & \geq \left(1 - \dfrac{H_{min}}{H_{max}}\right) \dfrac{\sqrt{M}}{\sqrt{M}-1} + \dfrac{H_{min}}{H_{max}} \alpha_{min} \end{cases} \tag{5.8}$$

- *When determining α_{min}, then α_{max} then M, the following conditions apply:*

$$\begin{cases} \alpha_{max} & \geq 1 - (1-\alpha_{min}) \dfrac{H_{min}}{H_{max}} \\ \sqrt{M} & \geq \dfrac{H_{max}\alpha_{max} - H_{min}\alpha_{min}}{H_{min}(1-\alpha_{min}) - H_{max}(1-\alpha_{max})} \end{cases} \tag{5.9}$$

- *When determining α_{max} then M then α_{min}, the following conditions apply:*

$$\begin{cases} \alpha_{max} & \geq 1 - \dfrac{H_{min}}{H_{max}} \\ \sqrt{M} & \geq \dfrac{H_{max}\alpha_{max}}{H_{min} - (1-\alpha_{max})H_{max}} \\ \alpha_{min} & \leq \dfrac{H_{max}}{H_{min}}\alpha_{max} + \left(1 - \dfrac{H_{max}}{H_{min}}\right)\dfrac{\sqrt{M}}{\sqrt{M}-1} \end{cases}$$

- *When determining α_{max} then α_{min} then M, the following conditions apply:*

$$\begin{cases} \alpha_{max} & \geq 1 - \dfrac{H_{min}}{H_{max}} \\ \alpha_{min} & \leq 1 - \dfrac{H_{max}}{H_{min}}(1-\alpha_{max}) \\ \sqrt{M} & \geq \dfrac{H_{max}\alpha_{max} - H_{min}\alpha_{min}}{H_{min}(1-\alpha_{min}) - H_{max}(1-\alpha_{max})} \end{cases}$$

Proof. The proof of this theorem is based solely on the result of Proposition 5.3.4. The condition for this Proposition to be valid is

$$\frac{H_{max}}{\sqrt{M} - \alpha_{min}(\sqrt{M}-1)} \leq \frac{H_{min}}{\sqrt{M} - \alpha_{max}(\sqrt{M}-1)} \quad .$$

Therefore we get

$$H_{max}\left(\sqrt{M} - \alpha_{max}(\sqrt{M}-1)\right) \leq H_{min}\left(\sqrt{M} - \alpha_{min}(\sqrt{M}-1)\right). \tag{5.10}$$

This equation establishes a relationship between M, α_{min} and α_{max} and the conditions stated in the theorem are all derived from it. We will only prove the case where α_{max} is determined first, then α_{min} then M because it illustrates how the 5 other cases are proved. We begin by isolating $\sqrt{M}$ from Eq. (5.10) as such:

$$\sqrt{M}\left(H_{max}(1-\alpha_{max}) - H_{min}(1-\alpha_{min})\right) \leq H_{min}\alpha_{min} - H_{max}\alpha_{max}.$$

The right term of the inequality is negative since $H_{min} \leq H_{max}$ and $\alpha_{min} \leq \alpha_{max}$ by definition. Consequently, if the term $(H_{max}(1-\alpha_{max}) - H_{min}(1-\alpha_{min}))$ was positive, it would result in $\sqrt{M} \leq 0$ which is false by definition. Therefore, it must be negative for M to exist. As a result, we obtain the following condition:

$$\sqrt{M} \geq \frac{H_{max}\alpha_{max} - H_{min}\alpha_{min}}{H_{min}(1-\alpha_{min}) - H_{max}(1-\alpha_{max})}.$$

The fact that $(H_{max}(1-\alpha_{max}) - H_{min}(1-\alpha_{min})) \leq 0$ leads to

$$\alpha_{min} \leq 1 - \frac{H_{max}}{H_{min}}(1-\alpha_{max}).$$

However, $\alpha_{min} \geq 0$ by definition, so the right term of the inequality has to be positive for α_{min} to exist. Thus, we obtain the following condition:

$$\alpha_{max} \geq 1 - \frac{H_{min}}{H_{max}}.$$

The other cases are proved by using the same reasoning. ■

Theorem 5.3.1 shows that in 2 particular cases described by Eqs. (5.8) and (5.9) , when α_{min} is determined first, there is no constraint on its value other than that it should be positive and lower than 1. These 2 cases should be preferred over the other more constrained ones when choosing the values of α_{min}, α_{max} and M. The theorem also ensures the choice of well defined parameters, given H_{min} and H_{max} only. Nevertheless, in some datasets, the difference between H_{min} and H_{max} is considerable and may lead to the choice of a high number of windows M, a maximum overlap α_{max} close to 1, or a minimum overlap α_{min} close to 0. Therefore, one can also arbitrarily choose the parameters that are deemed appropriate and then choose a maximum height $\tilde{H}_{max}$ and a minimum height $\tilde{H}_{min}$ such that:

$$\frac{\tilde{H}_{max}}{\tilde{H}_{min}} \leq \frac{\sqrt{M} - \alpha_{min}(\sqrt{M}-1)}{\sqrt{M} - \alpha_{max}(\sqrt{M}-1)}. \tag{5.11}$$

After performing this choice, each image in the dataset whose height exceeds $\tilde{H}_{max}$ should be shrinked to $\tilde{H}_{max}$, and each image whose height is less than $\tilde{H}_{min}$ should be padded or magnified to $\tilde{H}_{min}$. Although this defeats the purpose of the VOTCSW method, it is still better than resizing all the images in the dataset since the images whose heights are within $[\tilde{H}_{min}, \tilde{H}_{max}]$ will remain the same. Using Proposition 5.3.3, Proposition 5.3.4 and Theorem 5.3.1 enables any image whose height is between the limits H_{min} and H_{max} and whose aspect ratio β is the same as that of the sliding window γ to be transformed in a fixed 3-dimensional representation $(h, \gamma h, M)$. However, the order of the windows in the 3-dimensional sequence should not be arbitrary as it should ensure a correlation between every two consecutive windows. The VOTCSW method is based on the sliding window technique which is widely used on 1-dimensional signals because the inter-window correlation is always ensured. However, this is no longer guaranteed for higher dimensional signals such as images as shown in Figure 5.3 which describes the sliding window technique on an image represented by a rectangle where the the light blue color represents a window a the dark blue color shows the overlap between two windows. As represented by the arrows, the window slides from left to right and resumes

to the initial position when it reaches the boundaries of the image. It then slides one step downward and continues sliding from left to right until it covers the whole image. This sliding pattern does not ensure that every two consecutive windows are correlated and this can be noticed in the aforementioned figure where window n and window $n+1$ are totally separated. Nevertheless, this usually does not matter in machine learning applications where each window

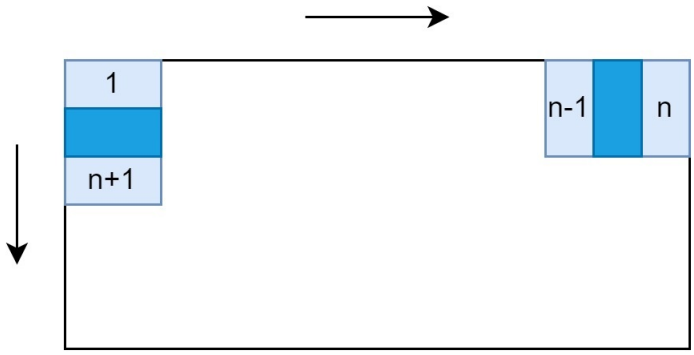


FIGURE 5.3: Sliding windows on a rectangle representing an image. The arrows represent the sliding pattern and the dark blue color shows the overlap between two windows.

can be treated independently and the result is determined by aggregating the results obtained on each individual window. However, the VOTCSW method consists in creating a causal 3-dimensional representation that is analogous to a video and which can be processed as a video in the sense that there is a time coherence between two consecutive windows meaning that one necessarily appears before the other and is spatially correlated with it. To ensure that, we define, given a matrix I of size $H \times W$ representing an image, the following sequences describing a time–coherent left-to-right, top-to-bottom sliding pattern:

$$
\begin{aligned}
&\forall n \in [\![0, M-1]\!], \\
&\delta_n = \left\lfloor \frac{n}{\sqrt{M}} \right\rfloor, \\
&a_n = (1-\alpha) h \delta_n, \\
&b_n = a_n + h, \\
&c_n = \left(\left(1 - (-1)^{\delta_n}\right) \frac{\sqrt{M}-1}{2} + (-1)^{\delta_n} \left(n - \delta_n \sqrt{M}\right) \right) (1-\alpha)\gamma h, \\
&d_n = c_n + \gamma h, \\
&\mathcal{W}_n = \begin{bmatrix} I_{a_n c_n} & I_{a_n c_{n+1}} & \dots \\ \vdots & \ddots & \\ I_{b_n c_n} & & I_{b_n d_n} \end{bmatrix},
\end{aligned}
\tag{5.12}
$$

where $\mathcal{W}_n$ is the n-th window extracted from the image I, h is a constant determined using Theorem 5.3.1 and Proposition 5.3.4, and α is calculated using Proposition 5.3.3. This pattern is a modification of the one described in Figure 5.3 such that when the window slides to the right edge of the rectangle, it slides downward by a step, and slides back to the left until reaching the left edge before it slides downward again and slides back to the right edge. As a result, this pattern ensures the time–coherence of the 3-dimensional representation created by the VOTCSW method. Another property of the VOTCSW method is that it performs an oversampling of the pixels due to the window overlap that enables the window to cover the same pixel more than once.

Proposition 5.3.5. *The maximum oversampling factor for a pixel in an image that is processed with the VOTCSW method with an overlap α is* $\dfrac{1}{(1-\alpha)^2}$.

Proof. We designate by (x, y) a pixel in a given image. We suppose that h, M, and γ were chosen prior to the use of the VOTCSW method and that the overlap α was calculated for the image. For the pixel (x, y) to be contained in a given window (n, m) such that $(n, m) \in [\![0, \sqrt{M} - 1]\!]^2$, the following conditions need to be verified:

$$\begin{cases} (1-\alpha)hn & \leq x \leq (1-\alpha)hn + h \\ (1-\alpha)\gamma hm & \leq y \leq (1-\alpha)\gamma hm + \gamma h \end{cases}.$$

By extracting n and m from the previous conditions, we obtain

$$\begin{cases} \dfrac{x}{(1-\alpha)h} - \dfrac{1}{1-\alpha} & \leq n \leq \dfrac{x}{(1-\alpha)h} \\ \dfrac{y}{(1-\alpha)\gamma h} - \dfrac{1}{1-\alpha} & \leq m \leq \dfrac{y}{(1-\alpha)\gamma h} \end{cases},$$

which determines what values of n and m are valid for a window to contain the pixel (x, y). Since n is an integer, it can take at most $\dfrac{x}{(1-\alpha)h} - \left(\dfrac{x}{(1-\alpha)h} - \dfrac{1}{1-\alpha}\right) = \dfrac{1}{1-\alpha}$ different values in $[\![0, \sqrt{M} - 1]\!]$, and the same can be inferred for m. Since every combination of n and m that follows the previous conditions can determine a window that contains the pixel (x, y), the total number of times the pixel (x, y) is present in a window is at most $\dfrac{1}{(1-\alpha)^2}$. ■

Proposition 5.3.5 implies that α_{min} and α_{max} are means to control the maximum oversampling factor of a pixel. Moreover, it implies that, the smaller the image, the more its pixels will be oversampled which will definitely alter the class distribution for a classification problem if the size distribution in each class is uneven. This may prove useful in certain cases of image

classification where the images representing the least represented class happen to be the smallest in size. Finally, the VOTCSW method can be summarized in the following steps:

1. Ensure that the image dataset has a single aspect ratio β and determine H_{max} and H_{min}.
2. Choose α_{min}, α_{max}, M and h as recommended in Theorem 5.3.1 and Proposition 5.3.4. Define $\gamma = \beta$.
3. Choose a time–coherent sliding pattern such as the one described in Eq. (5.12) and use it for each image in the dataset.

5.4 Experiments and results

In order to produce meaningful results and to reliably choose a model over the other, the framework shown in Figure 5.1 was designed. This section details and discusses each block in the diagram and provides an in-depth analysis of the results obtained on the WSISCMC dataset.

5.4.1 Preprocessing

The WSISCMC plant species classification dataset that is used in this work was mainly constructed to specifically overcome the limitations of the current state-of-the-art datasets used in machine learning which consist in a lack of sufficient variety and quantity. It contains $38,680$ high-quality square photos with different sizes of 8 different species of plants taken from different angles, and at various growth stages, which can theoretically enable a well-trained classification model to recognize plants at any stage in their growth. Moreover, the plants are potted and placed in front of a uniform background that can easily be substituted with field images, for example. There are also images which are taken with a mobile phone in various backgrounds to enable trained models to be tested on unprocessed images. In [107], it was shown that the WSISCMC dataset allows the production of a reliable binary classifier that differentiates between grasses and non-grasses. However, the dataset was not used to train a more complex model of plant species classification. A first attempt to evaluate the difficulty of this task on this dataset was to create a baseline 2DCNN that takes images resized to 224×224 and attempts to predict the species of the plant present in the dataset. The tested accuracy of that baseline model was 49.23%, which was mediocre.

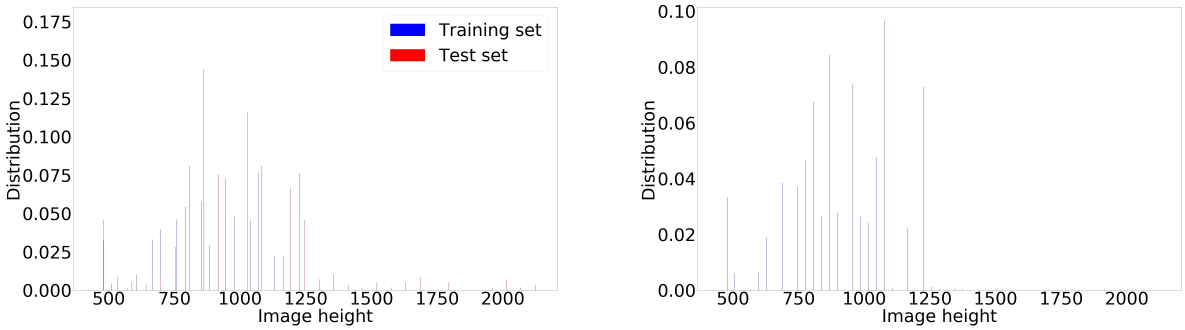
It was then determined, after an analysis of the dataset, that there was a significant imbalance in the class distribution. Moreover, this distribution was radically different between the training set and the test set. Furthermore, the size distribution was also very different between the training set and test set, such that the maximum image size in the training set was 1226×1226 and the maximum image size in the test set was 2346×2346. In addition, the distribution of

size per class also varied between the training set and the test set. Therefore, the dataset was entirely redistributed into a training set and a test set that have the same class distribution and the same size per class distribution while keeping the same train-test ratio as the initial dataset. Table 5.1 shows the initial training set and test set class distributions as well as the class distribution of the reworked (redistributed). We notice that the "Wild Oat" class is not

Species	Training set distribution	Test set distribution	Reworked distribution
Smartweed	0.03	0.14	0.04
Yellow Foxtail	0.10	0.22	0.11
Barnyard Grass	0.25	0.12	0.23
Wild Buckwheat	0.12	0.14	0.12
Canola	0.19	0.14	0.19
Canada Thistle	0.14	0.14	0.14
Dandelion	0.14	0.10	0.13
Wild Oat	0.03	0	0.03
Total	1	1	1

TABLE 5.1: Training set, test set and reworked distributions of the 8 species in the dataset.

even present in the test set and that the "Smartweed" and "Yellow Foxtail" classes are the most represented classes in the test set, and the least represented ones in the training set. Figure 5.4.(a) shows the size distribution of the "Canola" class in the training set and the test set whereas Figure 5.4.(b) shows the size distribution of the "Canola" class in the reworked dataset. We notice that the size distribution of the "Canola" class in the training set is widely different from the test set and that the reworked dataset ensures that every size is present with the same proportion in the training set and the test set.



(a) Training set and test set distribution.

(b) Reworked dataset distribution.

FIGURE 5.4: Size distributions for the "Canola" class in the training set, the test set and the reworked dataset.

These factors, as well as the aforementioned ones, explain why the baseline model failed to

accurately determine the species of the images it processed. Nevertheless, after reworking the dataset, the baseline model achieved an accuracy of 95.1%. Although this was a noticeable improvement, there was still room to achieve better accuracy with further preprocessing. One of which is to modify the bias initialization of the classifier's last layer to take into account the inherent data imbalance. Since the last layer uses a softmax activation function, we need to solve the following system of equations:

$$\forall k \in [\![1, N]\!], p_k = \frac{e^{b_k}}{\sum_{i=1}^{N} e^{b_i}},$$

where N is the number of classes, p_k is the presence of class k in the dataset, and b_k is the bias in the neuron k that will predict the probability of an image to belong to class k. This system is linear in e^{b_k} and is easily solvable by noticing that $\frac{p_j}{p_k} = e^{b_j - b_k}$. This bias modification improved the accuracy of the baseline model by 1.58% which reached 96.68% accuracy. Consequently, we created 6 different versions of the same dataset. The first three versions are produced by the VOTCSW method with the following parameters $\alpha_{min} = 0.1$, $\alpha_{max} = 0.9$ and $M = 9$. These parameters impose that we shrink the images whose height is greater than 973 to 973. This value was determined using Eq. (5.11) for $H_{min} = 418$ which represents the lowest size present in the dataset. The difference between the first three versions produced by the VOTCSW method is the sliding patterns which are horizontal (refer to Eq. (5.12)), vertical and spiral, respectively. The images were all transformed to 3D tensors of size $348 \times 348 \times 9$, meaning that they are equivalent to videos of size 348×348 containing 9 frames. The fourth version and fifth version of the dataset consist in resizing the images to a size that is equivalent, in terms of the number of samples, to the size of the "videos" generated by the VOTCSW method. This size is $\sqrt{348 \times 348 \times 9} = 1044$ and the images that are larger than 1044 are shrinked to 1044 whereas the ones that are smaller than 1044 are zero-padded to that size in the fourth version, and magnified to that size in the fifth version. The sixth version of the dataset consists in shrinking all the images to a size of 224×224 which is suitable for using well-known architectures such as ResNet or Inception. The 6 versions of the dataset are referred to as WSISCMC–H, WSISCMC–V, WSISCMC–S, WSISCMC–1044P, WSISCMC–1044M, and WSISCMC–224 following the order in which they were introduced above.

5.4.2 Model development

After performing the preprocessing described above, we created multiple variations for each kind of model. All the networks used share in common a 3×3 ($3 \times 3 \times 3$ for 3D) filter mask size for the convolution layers, an initial layer composed of 32 neurons, a last feature extraction layer composed of 64 neurons, the use of ReLU [32] as activation function, the shape of the output of their feature extractor which is 576 features and their densely connected layers which

are composed of 2 layers, one containing 128 neurons, followed by one containing 8 neurons corresponding to the 8 classes of the dataset with a softmax activation. For each dataset version, there is a predetermined depth for the networks created as shown in Figure 5.5. We performed a grid search on the number of neurons for the 4 penultimate feature extraction layers of every 2D network with 10-fold cross validation and we ensure that the validation set always has the same class distribution and size distribution as the train set. The possible values used for the number of neurons in these layers were 16, 32 and 64 to keep the experiments feasible to be completed in a reasonable amount of time. The layers that were not searched were composed of 64 neurons by default. This search allowed us to select the best architecture among 81 variations for each of the 3 2D dataset versions, WSISCMC–224, WSISCMC–1044P and WSISCMC–1044M.

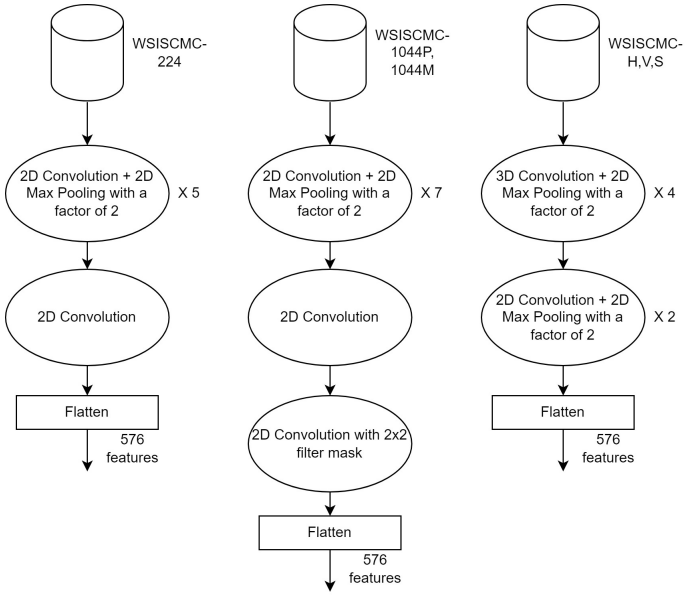


FIGURE 5.5: Network architecture for each version of the WSISCMC dataset.

Following that, for each 2D version of the dataset, a network was created and trained with the best determined architecture 10 times with different initial weights and evaluated on its corresponding test set such that only the best one is chosen. Then, the 2DCNN architecture of the best network among the one trained on the WSISCMC–1044P and the one trained on the WSISCMC–1044M was chosen to create a 3DCNN architecture that has the same number of parameters as the chosen architecture. This is to ensure a fair evaluation of the effect of the VOTCSW method on the performance of the 3DCNN. Indeed, if a 3DCNN with more parameters than a 2DCNN trained on WSISCMC–1044P or WSISCMC–1044M achieves better results than the 2DCNN, then this might be due to the difference in the number of parameters, and not the difference in data representation. Since 2DCNNs and 3DCNNs perform the same fundamental operation, if both have the same number of parameters and one of them performs

better than the other, then this is most likely due to the difference in data representation. Naturally, the weight initialization also plays a role in this difference in performance, therefore, we always train the same network 10 times with different initial weights and choose the one that achieves the best accuracy on the test set. The architecture of the 3DCNN that is calculated from the best 2DCNN architecture has to be consistent with the above description in the sense that the initial layer contains 32 neurons, and the last feature extraction layer contains 64 neurons. We assume that all the inner layers of the 3DCNN have the same number of neurons N. If we denote the number of parameters of the feature extractor of the best network trained on WSISCMC–1044P or WSISCMC–1044M by N_{1044}, the receptive field of the 3D convolution layers by R_3 (27 in this case), and the receptive field of the 2D convolution layers by R_2 (9 in this case), then the number of neurons N of each inner layer of the 3DCNN is determined by the following equation:

$$3 \times 32 \times R_3 + 32R_3N + 2R_3N^2 + R_2N^2 + 64R_2N + 32 + 64 + 4N = N_{1044},$$

which is equivalent to

$$(2R_3 + R_2)N^2 + 4(8R_3 + 16R_2 + 1)N + 96(R_3 + 1) - N_{1044} = 0. \tag{5.13}$$

This second degree equation with unknown N has a unique positive solution because $96(R_3 + 1) - N_{1044}$ is negative and the other coefficients are positive. Since N has to be an integer, the determined solution is rounded before it is used.
The resulting architecture was then used to create and train 3DCNNs on WSISCMC–H, WSISCMC–V and WSISCMC–S. The best 3DCNN network and the best 2DCNN network were then selected to be extended to NDPNNs. Consequently, each convolution layer of each network was changed to an NDPNN layer with a degree 7, such that a 2DPNN and a 3DPNN were trained and evaluated on their proper respective datasets. Finally, both of them were reduced using the layer-wise degree reduction heuristic described in Algorithm 1 with 0 tolerance to gain computational efficiency and reduce memory usage without sacrificing their accuracy. We also fine-tuned a ResNet50V2, an InceptionV3 and a Xception network on the WSISCMC–224 dataset. These networks were also trained from scratch and only the best among the fine-tuned and the custom network of each model was selected for evaluation. Every model was trained with the Adam optimizer [57], a batch size of 128 and a learning rate of 10^{-3} for 100 epochs.

5.4.3 Results and discussion

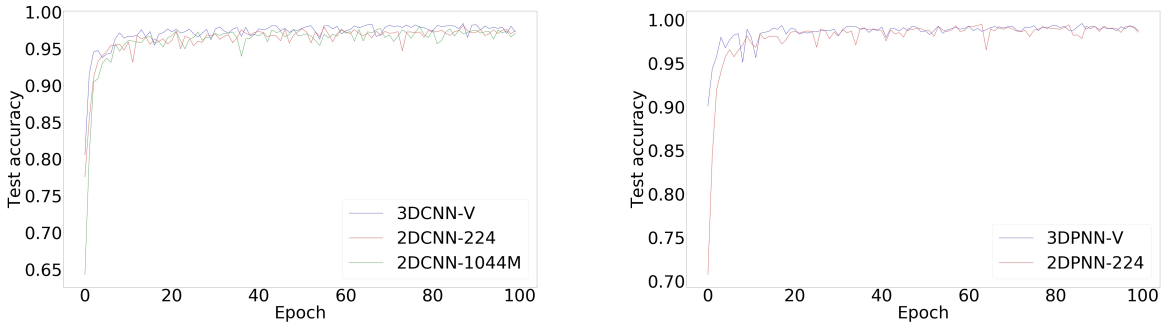
The grid search determined that, for all the networks, 64 neurons in every layer (except the first) produces the best average results. The number of neurons for the 3DCNN layers was therefore determined to be 53 according to Eq. (5.13) with $N_{1044} = 257408$ which corresponds

to the number of parameters of the feature extractor of the 2DCNN–1044P and the 2DCNN–1044M. Furthermore, ResNet50V2, InceptionV3 and Xception failed to produce decent results when they were trained from scratch. Therefore, only the fine-tuned networks were considered. Table 5.2 shows the best test accuracy, aggregated precision, aggregated recall, aggregated F1 score, the average inference time per sample and the number of trainable parameters of every model described above. The experiments show that the best 2DCNN model is the one trained on WSISCMC–224, and the best 3DCNN is the one trained with the vertical sliding pattern. Therefore, they were chosen to be extended to NDPNNs and the 2DPNN achieved a 99.48% accuracy while the 3DPNN achieved a state-of-the-art 99.58% accuracy. Their execution times and the number of trainable parameters were measured after the polynomial degree reduction described in Algorithm 1 which determined that the first two layers of the 2DPNN could be reduced to a degree of 7 and 2 respectively while the remaining ones could be reduced to 1 which represents 4.67 times less parameters, and that the first three layers of the 3DPNN could be reduced to 6, 2 and 2 respectively, while the remaining ones could be reduced to 1 which represents 4.01 times less parameters. The results also show that ResNet50V2, InceptionV3 and Xception failed to match the performance of the 2DPNN and the 3DPNN despite having more than 30 times the number of parameters. Furthermore, even though the accuracy of the 3DPNN is unmatched, the data generated by the VOTCSW method came with an increase in the spatio-temporal complexity of the model as it runs 3.28 times slower than its 2D counterpart, and has 1.75 times more parameters. However, both 2DCNN–1044P and 2DCNN–1044M did not provide satisfactory results compared to the 3DCNN models that run faster and have less parameters, which tends to show that the 3D representation created with the VOTCSW method is better than padding, magnifying and shrinking. Moreover, the 3DCNN models achieve better performance overall than the 2DCNN–224 model but they have more parameters and run slower than the 2DCNN–224 model which suggests that the VOTCSW method comes with the cost of slightly heavier but better models.

An analysis of the generalization behavior of the networks shows that the 3DCNN–V model learns to generalize faster than the equally complex 2DCNN–1044M model and the 2DCNN–224 model as represented in Figure 5.6.(a). Furthermore, a more stable convergence is observed for the 3DCNN–V model which may be explained by the fact that the oversampling inherent to the VOTCSW method (refer to Proposition 5.3.5) has a regularization effect that smoothes the weight updates and enables a steadier training with a potential reduction of overfitting. These effects are also observed in the convergence of the NDPNN models represented in Figure 5.6.(b) where there is a clear gap between the convergence speed and stability of the 3DPNN–V and that of the 2DPNN–224. A further analysis of the performance of the 3DPNN–V model shows that it was able to encapsulate enough information to perfectly recognize the least represented class in the dataset which is "Wild Oat" as shown in Table 5.3 outlining the confusion matrix of the 3DPNN–V model. Since only 17 images were wrongly classified, an in-depth investigation

	Performance					
Model	Accuracy	Precision	Recall	F1 Score	Inference time (ms)	Parameters
2DCNN–224	98.15	98.2	98.08	98.14	4.7	241,992
2DCNN–1044P	97.8	97.83	97.78	97.8	14.49	332,296
2DCNN–1044M	98.03	97.96	98.11	98.03	14.73	332,296
3DCNN–H	98.28	98.37	98.16	98.26	14.26	331,075
3DCNN–V	98.43	98.64	98.3	98.46	14.15	331,075
3DCNN–S	98.28	98.37	98.16	98.26	14.31	331,075
2DPNN–224	99.48	99.53	99.33	99.42	4.98	265,608
3DPNN–V	**99.58**	**99.69**	**99.36**	**99.52**	16.34	465,670
ResNet50V2	98.25	98.17	98.28	98.22	28	23,581,192
InceptionV3	97.78	97.64	97.83	97.73	38	21,819,176
Xception	97.9	97.85	98.03	97.93	31	21,819,176

TABLE 5.2: Accuracy, precision, recall, F1 score, inference time and number of trainable parameters of all the models trained on the reworked WSISCMC dataset. Bold values represent the highest value in their respective column.



(a) CNNs accuracy over time.

(b) NDPNNs accuracy over time.

FIGURE 5.6: Evolution of the networks' test accuracies per epoch.

was performed. This investigation revealed that 10 images were showing a blue background as illustrated in Figure 5.7.(a) and that 3 images were containing multiple plants in one image as shown in Figure 5.7.(b). Moreover, upon further investigation, it was determined that the 3DPNN–V model correctly recognizes one of the plants present in all of the 3 multiple-plant images. Therefore, we can consider that the model is only wrong on 4 images since the problematic images contradict the task of single plant classification by either showing no plant or multiple ones. As a result, the 3DPNN–V model achieved an effective accuracy of $\frac{4004-17}{4004-17+4} = 99.9\%$ when we removed the aberrant samples from the test set.

We now investigate the reason why the VOTCSW method enables the creation of a model that generalizes better than one trained on resized images aside from its regularization-like behavior. The intuition behind the improved generalization comes from the fact that the VOTCSW

Actual–Predicted	Canola	Dandelion	Canada Thistle	Wild Oat	Wild Buckwheat	Smartweed	Barnyard Grass	Yellow Foxtail
Canola	752	1	1	0	1	0	0	0
Dandelion	0	540	0	0	0	0	0	0
Canada Thistle	0	0	545	0	0	0	0	0
Wild Oat	0	0	0	126	0	0	0	0
Wild Buckwheat	0	1	0	0	488	0	1	0
Smartweed	0	1	1	0	0	145	2	0
Barnyard Grass	1	0	0	0	0	0	939	0
Yellow Foxtail	0	1	1	0	0	0	5	452

TABLE 5.3: Confusion matrix of the 3DPNN–V model.

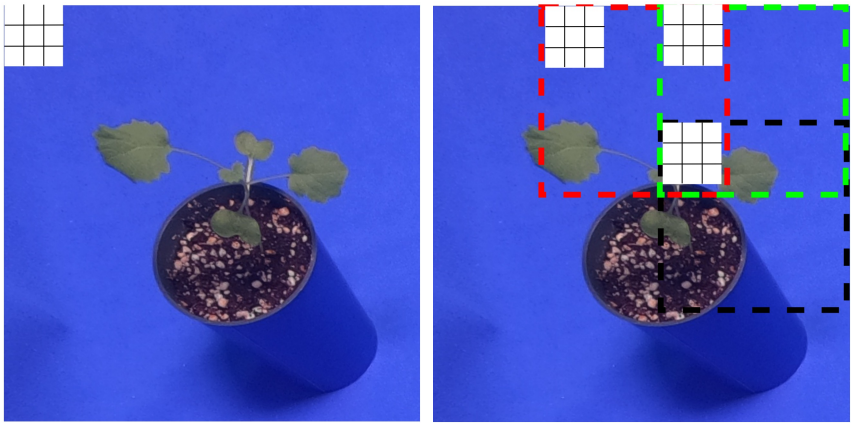
(a) Empty image. (b) Multiple plants in one image.

FIGURE 5.7: Examples of images wrongly classified by the 3DPNN-V model.

method enables the model using a 3D convolution kernel to have a larger effective field of view than a 2D convolution kernel as illustrated in Figure 5.8 where the white squares represents a 3×3 convolution kernels, and the red, green and black dashed squares represent 3 consecutive overlapping windows generated by the VOTCSW method. The VOTCSW convolution kernel has three times more parameters and is more spatially dilated which enables it to take into account three distinct informative areas that may be distant such as leaves. Hence, this helps in creating "spatially aware" models that can, not only achieve what regular 2D convolution models already do, but also establish a map of more complex spatial features.

5.5 Chapter Summary

In this chapter, we used 2DPNNs and 3DPNNs to tackle the problem of plant species recognition on the WSISCMC dataset which contains plant images with variable size. We therefore formally developed the VOTCSW method which transforms any image from the dataset to a 3D representation with fixed size that is suitable for 3DCNNs and 3DPNNs. Furthermore, we designed a model development framework that makes use of the NDPNN layer-wise degree reduction heuristic and the VOTCSW to create a highly reliable NDPNN architecture for plant species recognition. Moreover, we resampled the dataset with respect to the class distribution



(a) Regular 2D convolution kernel. (b) VOTCSW 3D convolution kernel.

FIGURE 5.8: Comparison between a regular 2D convolution kernel and VOTCSW 3D convolution kernel.

and the size distribution to make it more suitable for machine learning models. In addition, we evaluated the gain of using the VOTCSW method and we fine-tuned complex architectures such as ResNetV2, InceptionV3 and Xception. We also showed that the 3DPNN used in conjunction with the VOTCSW method and the NDPNN layer-wise degree reduction outperformed all the considered models and achieved a state-of-the-art 99.9% accuracy on the WSISCMC dataset after determining the existence of aberrant samples in the dataset.

Chapter 6

Conclusions and Future Work

In this thesis, we developed a novel deep learning model called NDPNN in an attempt to solve complex problems more efficiently than the NDCNN model by encapsulating more non-linearity. We began by formalizing the 1D version of the NDPNN and we derived the forward and backward propagation equations that govern its learning process. Following that, we theoretically analyzed the computational complexity of the 1DPNN with respect to that of the 1DCNN and we empirically showed the linearization of the 1DPNN computational complexity with a GPU implementation. We then developed a theorem that enables the transformation of a 1DPNN to a 1DCNN with the same number of parameters. Consequently, we derived 3 comparison strategies to estimate fairly the gain of using 1DPNNs over 1DCNNs and we evaluated the 1DPNN's performance with respect to that of the 1DCNN on two classification problems and one regression problem related to audio signals. We also tested various activation functions to estimate their influence on the 1DPNN. As a result, we determined that the 1DPNN could encapsulate more information with less spatio-temporal complexity than a 1DCNN on the audio processing problems that were considered. Furthermore, we empirically determined that the 1DPNN had an affinity for bounded activation functions which ensured a stable convergence. Following that, we showed that the forward and backward propagation equations were invariant with respect to the dimension of the signal and we generalized 1DPNNs to NDPNNs. Consequently, we formally developed a general polynomial degree reduction formula which we used to design a heuristic algorithm that reduces the degrees of the layers of a pre-trained NDPNN which makes it lighter and faster while maintaining its performance on the dataset it was trained on. Then, we considered the problem of plants species recognition on the WSISCMC dataset created by the EAGL–I system [107] and for which we developed the VOTCSW method. The experiments conducted on the WSISCMC dataset demonstrate that the VOTCSW method coupled with 3DPNNs outperform fine-tuned ResNetV2, InceptionV3 and Xception with far less spatio-temporal complexity and achieve a 99.9% state-of-the-art accuracy. This confirms the intuition proposed by Mehdipour Ghazi *et al.* [113] who states that models with simpler architectures

have the tendency to better learn from scratch than complex architectures. Moreover, we determined that the NDPNN layer-wise degree reduction heuristic would be able to significantly compress a pre-trained NDPNN without altering its performance on the test set, which makes it a necessary postprocessing tool that has to be used in conjunction with NDPNNs. Furthermore, we also demonstrated that the VOTCSW method offers a better alternative than resizing when using the WSISCMC dataset which contains images with variable sizes and that the 3D representation it creates is more informative than a resized 2D representation. In addition, we discovered that the current publicly available WSISCMC dataset can not be used with machine learning models without a mandatory preprocessing consisting in redistributing the samples of each class by occurrence and size to create a test set that has the same distribution as the training set. Besides, we also discovered that there were aberrant samples in the dataset which contradict the fact that the dataset should only contain single-plant images. However, despite these minor issues, we can safely declare that the EAGL–I system has the potential to produce highly relevant massive datasets, provided that the authors impose a stricter control on the data acquisition process and ensure that classes are balanced.

6.1 Discussion

Our experiments are not sufficient to claim that our proposed NDPNN surpasses NDCNN on all class of complex classification and regression problems. In addition, there is still no mathematical proof that the NDPNN is a convergent model. Moreover, the stability of the model is not ensured as stacked layers with high degrees can lead to the model becoming unstable, thus, losing its generalization capability. Therefore, there is also a need to estimate an upper bound limit for the degree of each layer so that the overall network stably learns from the given data. Furthermore, due to computational limits, the model could not be tested with deep topologies that are used to solve very complex classification and regression problems involving a huge amount of data. Also, despite its effectiveness, the NDPNN layer-wise degree reduction heuristic was only applied after the training of an NDPNN was completed. This is the most time consuming process and the most risky – in terms of stability – as 1DPNNs were proven to show some instability when trained with unbounded activation functions and this instability is expected to be observed on 2DPNNs and 3DPNNs. The instability can potentially be reduced by lowering the degree of each layer of the model either before or during training (hence the potential to use this heuristic in the model validation process) instead of after a model is trained. Furthermore, the heuristic is based on determining the smallest symmetric interval that contains the values of a given layer's output regardless of the channel/neuron. This implies that some polynomials are over-reduced, meaning that they are reduced on a bigger interval than the one they produce their output from. A potential solution to this is to determine the smallest symmetric interval that contains the values of a given neuron's output to produce a finer

and more accurate reduction. As for the VOTCSW method, although it enabled the creation of highly accurate models on the WSISCMC dataset, there is not enough evidence to claim that it can improve the results on any dataset that contains images with variable size. Additionally, the increase in model performance may not always justify the size and parameter overhead that it introduces compared to simply shrinking images for applications that are memory bound. Besides, there is no clear indication on how to determine the minimum ratio $\frac{\tilde{H}_{max}}{\tilde{H}_{min}}$ or the adequate parameters M, α_{min}, α_{max} and h that can maximize the performance of a model on a given dataset.

6.2 Future Work

Future work may be done on demonstrating the conditions of the stability of the model as well as its convergence. In addition, a more specific gradient descent optimization scheme may be developed to avoid gradient explosion. Moreover, deeper NDPNN topologies can be created to compete with state-of-the-art models on any multimedia processing related problem. Furthermore, we will focus on applying the VOTCSW method on bigger and more varied datasets in order to determine if the performance improvement that it introduces to models has any statistical significance. We will also aim to determine how the choice of the VOTCSW parameters influence the performance of the trained models, with an emphasis on creating a more dataset-specific set of rules for the method's use. We also plan on exploiting the NDPNN layer-wise degree reduction heuristic for validating models before training, which can be a safer alternative to the one it was used in this thesis. In addition, we intend to apply the polynomial reduction on the neuronal level in order to obtain more stable and accurate polynomials than the ones obtained with the layer-wise reduction, which may enable the degree of a layer to be further reduced. Finally, more thorough experiments and further mathematical analysis can help the NDPNN model to thrive and find its place as a standard model in the deep learning field.

Bibliography

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015. https://doi.org/10.1016/j.neunet.2014.09.003.

[2] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object Recognition with Gradient-Based Learning," in *Shape, Contour and Grouping in Computer Vision*, (Berlin, Heidelberg), p. 319, Springer-Verlag, 1999.

[3] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybernetics*, vol. 36, p. 193–202, 1980. https://doi.org/10.1007/BF00344251.

[4] Pearlmutter, "Learning state space trajectories in recurrent neural networks," in *International 1989 Joint Conference on Neural Networks*, pp. 365–372 vol.2, 1989. https://doi.org/10.1109/IJCNN.1989.118724.

[5] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite State Automata and Simple Recurrent Networks," *Neural Computation*, vol. 1, no. 3, pp. 372–381, 1989. https://doi.org/10.1162/neco.1989.1.3.372.

[6] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pp. 730–734, 2015. https://doi.org/10.1109/ACPR.2015.7486599.

[7] J. Salamon and J. P. Bello, "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017. https://doi.org/10.1109/LSP.2017.2657381.

[8] S. Oeda, I. Kurimoto, and T. Ichimura, "Time Series Data Classification Using Recurrent Neural Network with Ensemble Learning," in *Knowledge-Based Intelligent Information and Engineering Systems* (B. Gabrys, R. J. Howlett, and L. C. Jain, eds.), (Berlin, Heidelberg), pp. 742–748, Springer Berlin Heidelberg, 2006.

[9] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent Neural Networks for Multivariate Time Series with Missing Values," *Scientific Reports*, vol. 8, p. 6085, Apr 2018. https://doi.org/10.1038/s41598-018-24271-9.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[11] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[12] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989. https://doi.org/10.1016/0893-6080(89)90020-8.

[13] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *Ann. Statist.*, vol. 36, pp. 1171–1220, 06 2008. https://doi.org/10.1214/009053607000000677.

[14] Y. Cho and L. K. Saul, "Kernel Methods for Deep Learning," in *Advances in Neural Information Processing Systems 22* (Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, eds.), pp. 342–350, Curran Associates, Inc., 2009.

[15] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid, "Convolutional Kernel Networks," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2627–2635, Curran Associates, Inc., 2014.

[16] D. Chen, L. Jacob, and J. Mairal, "Recurrent Kernel Networks," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, eds.), pp. 13431–13442, Curran Associates, Inc., 2019.

[17] A. G. Ivakhnenko, "Polynomial Theory of Complex Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-1, no. 4, pp. 364–378, 1971.

[18] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[19] S.-K. Oh, W. Pedrycz, and B.-J. Park, "Polynomial neural networks architecture: analysis and design," *Computers & Electrical Engineering*, vol. 29, no. 6, pp. 703 – 725, 2003.

[20] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, and K. Simonyan, "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 1068–1077, JMLR.org, 2017.

[21] [**dataset**] Zohar Jackson, C. Souza, J. Flaks, and H. Nicolas, "Jakobovski/free-spoken-digit-dataset v1.0.6," Oct. 2017. https://doi.org/10.5281/zenodo.1039893.

[22] [**dataset**] Z. Rafii, A. Liutkus, F.-R. Stöter, S. I. Mimilakis, and R. Bittner, "The MUSDB18 corpus for music separation," Dec. 2017. https://doi.org/10.5281/zenodo.1117372.

[23] [**dataset**] Michael A. Beck, C.-Y. Liu, C. P. Bidinosti, C. J. Henry, C. M. Godee, and M. Ajmani, "Weed seedling images of species common to Manitoba, Canada," 2021. https://doi.org/10.5061/dryad.gtht76hhz.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision – ECCV 2016* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), pp. 630–645, Springer International Publishing, 2016.

[25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

[26] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, 2017.

[27] H. Ben Abdallah, C. J. Henry, and S. Ramanna, "1-dimensional polynomial neural networks for audio signal related problems," *Knowledge-Based Systems*, vol. 240, p. 108174, 2022.

[28] H. Ben Abdallah, C. J. Henry, and S. Ramanna, "Polynomial degree reduction in the l2-norm on a symmetric interval for the canonical basis," *Results in Applied Mathematics*, vol. 12, p. 100185, 2021.

[29] H. Ben Abdallah, C. J. Henry, and S. Ramanna, "Plant species recognition with optimized 3d polynomial neural networks and variably overlapping time-coherent sliding window," *ArXiv*, vol. abs/2203.02611, 2022.

[30] J. Turian, J. Bergstra, and Y. Bengio, "Quadratic Features and Deep Architectures for Chunking," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, (Boulder, Colorado), pp. 245–248, Association for Computational Linguistics, June 2009.

[31] D. L. Elliott, "A better activation function for artificial neural networks," 1993.

[32] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *2009 IEEE 12th International Conference on Computer Vision*, pp. 2146–2153, 2009.

[33] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[34] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *CoRR*, vol. abs/1710.05941, 2017.

[35] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature Neuroscience*, vol. 2, pp. 1019–1025, Nov 1999.

[36] B. Graham, "Fractional Max-Pooling," *CoRR*, vol. abs/1412.6071, 2014.

[37] C.-Y. Lee, P. W. Gallagher, and Z. Tu, "Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (A. Gretton and C. C. Robert, eds.), vol. 51 of *Proceedings of Machine Learning Research*, (Cadiz, Spain), pp. 464–472, PMLR, 09–11 May 2016.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *J. Mach. Learn. Res.*, vol. 15, p. 1929–1958, Jan. 2014.

[39] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 448–456, JMLR.org, 2015.

[40] W. M. Campbell and C. C. Broun, "Using polynomial networks for speech recognition," in *Neural Networks for Signal Processing X. Proceedings of the 2000 IEEE Signal Processing Society Workshop (Cat. No.00TH8501)*, vol. 2, pp. 795–803 vol.2, 2000.

[41] R. Livni, S. Shalev-Shwartz, and O. Shamir, "A Provably Efficient Algorithm for Training Deep Networks," *CoRR*, vol. abs/1304.7045, 2013.

[42] T. Hughes and K. Mierle, "Recurrent neural networks for voice activity detection," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 7378–7382, 2013.

[43] C. Wang, J. Yang, L. Xie, and J. Yuan, "Kervolutional Neural Networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 31–40, 2019. https://doi.org/10.1109/CVPR.2019.00012.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. https://doi.org/10.1109/CVPR.2016.90.

[45] J. Mairal, "End-to-End Kernel Learning with Supervised Convolutional Kernel Networks," in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 1399–1407, Curran Associates, Inc., 2016.

[46] N. Aronszajn, "Theory of reproducing kernels," *Trans. Amer. Math. Soc.*, vol. 68, pp. 337–404, 1950.

[47] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a Deep Convolutional Network for Image Super-Resolution," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 184–199, Springer International Publishing, 2014.

[48] D. T. Tran, S. Kiranyaz, M. Gabbouj, and A. Iosifidis, "Heterogeneous Multilayer Generalized Operational Perceptron," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 3, pp. 710–724, 2020. https://doi.org/10.1109/TNNLS.2019.2914082.

[49] S. Kiranyaz, T. Ince, A. Iosifidis, and M. Gabbouj, "Operational neural networks," *Neural Computing and Applications*, vol. 32, p. 6645–6668, 2020.

[50] S. Kiranyaz, J. Malik, H. B. Abdallah, T. Ince, A. Iosifidis, and M. Gabbouj, "Self-organized operational neural networks with generative neurons," *Neural Networks*, vol. 140, pp. 294–308, 2021.

[51] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A. rahman Mohamed, G. Dahl, and B. Ramabhadran, "Deep Convolutional Neural Networks for Large-scale Speech Tasks," *Neural Networks*, vol. 64, pp. 39 – 48, 2015. Special Issue on "Deep Learning of Representations".

[52] A. CAUCHY, "Methode generale pour la resolution des systemes d'equations simultanees," *C.R. Acad. Sci. Paris*, vol. 25, pp. 536–538, 1847.

[53] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 13–15 May 2010.

[54] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems." https://www.tensorflow.org/, 2015. (accessed 27 July 2020).

[55] NVIDIA, "CUDA, release: 11.4.0." https://docs.nvidia.com/cuda/index.html, 2021.

[56] F. Chollet *et al.*, "Keras." https://github.com/fchollet/keras, 2015. (accessed 27 July 2020).

[57] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2017.

[58] D. Gabor, "Theory of communication. Part 1: The analysis of information," *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 429–441, 1946. https://doi.org/10.1049/ji-3-2.1946.0074.

[59] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[60] P. Refaeilzadeh, L. Tang, and H. Liu, *Cross-Validation*, pp. 532–538. Boston, MA: Springer US, 2009. https://doi.org/10.1007/978-0-387-39940-9_565.

[61] P. Mermelstein, "Evaluation of a segmental snr measure as an indicator of the quality of adpcm coded speech," *The Journal of the Acoustical Society of America*, vol. 66, no. 6, pp. 1664–1667, 1979.

[62] A. W. Rix, J. G. Beerends, M. P. Hollier, and A. P. Hekstra, "Perceptual Evaluation of Speech Quality (PESQ)-a New Method for Speech Quality Assessment of Telephone Networks and Codecs," in *Proceedings of the Acoustics, Speech, and Signal Processing, 200. on IEEE International Conference - Volume 02*, ICASSP '01, (USA), p. 749–752, IEEE Computer Society, 2001.

[63] E. López-Rubio, F. Ortega-Zamorano, E. Domínguez, and J. Muñoz-Pérez, "Piecewise polynomial activation functions for feedforward neural networks," *Neural Processing Letters*, vol. 50, pp. 121–147, Aug 2019.

[64] A. Patle and D. S. Chouhan, "Svm kernel functions for classification," in *2013 International Conference on Advances in Technology and Engineering (ICATE)*, pp. 1–9, 2013.

[65] E. Ostertagová, "Modelling using polynomial regression," *Procedia Engineering*, vol. 48, pp. 500–506, 2012. Modelling of Mechanical and Mechatronics Systems.

[66] W. J. Cody and L. Stoltz, "The use of taylor series to test accuracy of function programs," *ACM Trans. Math. Softw.*, vol. 17, p. 55–63, Mar. 1991.

[67] J. Zhou, H. Qian, X. Lu, Z. Duan, H. Huang, and Z. Shao, "Polynomial activation neural networks: Modeling, stability analysis and coverage bp-training," *Neurocomputing*, vol. 359, pp. 227–240, 2019.

[68] M. Eck, "Degree reduction of bézier curves," *Computer Aided Geometric Design*, vol. 10, no. 3, pp. 237–251, 1993.

[69] B.-G. Lee and Y. Park, "Distance for bézier curves and degree reduction," *Bulletin of the Australian Mathematical Society*, vol. 56, no. 3, p. 507–515, 1997.

[70] H. Kim and S. Moon, "Degree reduction of bézier curves by l1-approximation with endpoint interpolation," *Computers & Mathematics with Applications*, vol. 33, no. 5, pp. 67–77, 1997.

[71] M. E. Mortenson, *Mathematics for Computer Graphics Applications: An Introduction to the Mathematics and Geometry of CAD/Cam, Geometric Modeling, Scientific Visualizati*. USA: Industrial Press, Inc., 2nd ed., 1999.

[72] C. Wang, J. Yang, L. Xie, and J. Yuan, "Kervolutional neural networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 31–40, 2019.

[73] G. G. Chrysos, S. Moschoglou, G. Bouritsas, Y. Panagakis, J. Deng, and S. Zafeiriou, "P–nets: Deep polynomial neural networks," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7323–7333, 2020.

[74] Hussain and Liatsis, "A new recurrent polynomial neural network for predictive image coding," in *Image Processing And Its Applications, 1999. Seventh International Conference on (Conf. Publ. No. 465)*, vol. 1, pp. 82–86 vol.1, 1999.

[75] O. Rodrigues, "De l'attraction de sphéroïdes," in *Correspondence sur l'École Impériale Polytechnique*, vol. 3, pp. 361–385, 1816.

[76] C. R. Stark, "Adopting multidisciplinary approaches to sustainable agriculture research: Potentials and pitfalls," *American Journal of Alternative Agriculture*, vol. 10, no. 4, p. 180–183, 1995.

[77] M. Barbercheck, N. E. Kiernan, A. G. Hulting, S. Duiker, J. Hyde, H. Karsten, and E. Sanchez, "Meeting the 'multi-' requirements in organic agriculture research: Successes, challenges and recommendations for multifunctional, multidisciplinary, participatory projects," *Renewable Agriculture and Food Systems*, vol. 27, no. 2, p. 93–106, 2012.

[78] A. Luca, G. Molari, G. Seddaiu, A. Toscano, G. Bombino, L. Ledda, M. Milani, and M. Vittuari, "Multidisciplinary and innovative methodologies for sustainable management in agricultural systems," *Environmental Engineering and Management Journal*, vol. 14, no. 7, pp. 1571–1581, 2015.

[79] M. J. Paul, A. Watson, and C. A. Griffiths, "Linking fundamental science to crop improvement through understanding source and sink traits and their integration for yield enhancement," *Journal of Experimental Botany*, vol. 71, pp. 2270–2280, 10 2019.

[80] N. K. Fageria, V. C. Baligar, and Y. C. Li, "The role of nutrient efficient plants in improving crop yields in the twenty first century," *Journal of Plant Nutrition*, vol. 31, no. 6, pp. 1121–1157, 2008.

[81] N. Senapati, H. E. Brown, and M. A. Semenov, "Raising genetic yield potential in high productive countries: Designing wheat ideotypes under climate change," *Agricultural and Forest Meteorology*, vol. 271, pp. 33–45, 2019.

[82] L. F. P. Oliveira, A. P. Moreira, and M. F. Silva, "Advances in agriculture robotics: A state-of-the-art review and challenges ahead," *Robotics*, vol. 10, no. 2, 2021.

[83] T. Duckett, S. Pearson, S. Blackmore, and B. Grieve, "Agricultural robotics: The future of robotic agriculture," *CoRR*, vol. abs/1806.06762, 2018.

[84] J. Relf-Eckstein, A. T. Ballantyne, and P. W. Phillips, "Farming reimagined: A case study of autonomous farm equipment and creating an innovation opportunity space for broadacre smart farming," *NJAS - Wageningen Journal of Life Sciences*, vol. 90-91, p. 100307, 2019.

[85] K. Jha, A. Doshi, P. Patel, and M. Shah, "A comprehensive review on automation in agriculture using artificial intelligence," *Artificial Intelligence in Agriculture*, vol. 2, pp. 1–12, 2019.

[86] D. C. Rose and J. Chilvers, "Agriculture 4.0: Broadening responsible innovation in an era of smart farming," *Frontiers in Sustainable Food Systems*, vol. 2, 2018.

[87] A. G. Green, A.-R. Abdulai, E. Duncan, A. Glaros, M. Campbell, R. Newell, P. Quarshie, K. B. KC, L. Newman, E. Nost, and E. D. G. Fraser, "A scoping review of the digital agricultural revolution and ecosystem services: implications for canadian policy and research agendas," *FACETS*, vol. 6, pp. 1955–1985, 2021.

[88] I. Cisternas, I. Velásquez, A. Caro, and A. Rodríguez, "Systematic literature review of implementations of precision agriculture," *Computers and Electronics in Agriculture*, vol. 176, p. 105626, 2020.

[89] N. Srinivasan, P. Prabhu, S. S. Smruthi, N. V. Sivaraman, S. J. Gladwin, R. Rajavel, and A. R. Natarajan, "Design of an autonomous seed planting robot," in *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pp. 1–4, 2016.

[90] S. Sukkarieh, "Mobile on-farm digital technology for smallholder farmers," no. 2059-2018-203, p. 9, 2017.

[91] M. U. Hassan, M. Ullah, and J. Iqbal, "Towards autonomy in agriculture: Design and prototyping of a robotic vehicle with seed selector," in *2016 2nd International Conference on Robotics and Artificial Intelligence (ICRAI)*, pp. 37–44, 2016.

[92] S. Birrell, J. Hughes, J. Y. Cai, and F. Iida, "A field-tested robotic harvesting system for iceberg lettuce," *Journal of Field Robotics*, vol. 37, no. 2, pp. 225–245, 2020.

[93] Y. Ge, Y. Xiong, G. L. Tenorio, and P. J. From, "Fruit localization and environment perception for strawberry harvesting robots," *IEEE Access*, vol. 7, pp. 147642–147652, 2019.

[94] D. SepúLveda, R. Fernández, E. Navas, M. Armada, and P. González-De-Santos, "Robotic aubergine harvesting using dual-arm manipulation," *IEEE Access*, vol. 8, pp. 121889–121904, 2020.

[95] V. Meshram, K. Patil, V. Meshram, D. Hanchate, and S. Ramkteke, "Machine learning in agriculture domain: A state-of-art survey," *Artificial Intelligence in the Life Sciences*, vol. 1, p. 100010, 2021.

[96] D. I. Patrício and R. Rieder, "Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review," *Computers and Electronics in Agriculture*, vol. 153, pp. 69–81, 2018.

[97] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018.

[98] R. Sujatha, J. M. Chatterjee, N. Jhanjhi, and S. N. Brohi, "Performance of deep learning vs machine learning in plant leaf disease detection," *Microprocessors and Microsystems*, vol. 80, p. 103615, 2021.

[99] D. Sivakumar, K. SuriyaKrishnaan, P. Akshaya, G. Anuja, and G. Devadharshini, "Computerized growth analysis of seeds using deep learning method," *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 7, no. 6S5, 2019.

[100] A. Morellos, X.-E. Pantazi, D. Moshou, T. Alexandridis, R. Whetton, G. Tziotzios, J. Wiebensohn, R. Bill, and A. M. Mouazen, "Machine learning based prediction of soil total nitrogen, organic carbon and moisture content by using vis-nir spectroscopy," *Biosystems Engineering*, vol. 152, pp. 104–116, 2016. Proximal Soil Sensing – Sensing Soil Condition and Functions.

[101] J. Wäldchen, M. Rzanny, M. Seeland, and P. Mäder, "Automated plant species identification—trends and future directions," *PLOS Computational Biology*, vol. 14, pp. 1–19, 04 2018.

[102] T. Jin, X. Hou, P. Li, and F. Zhou, "A novel method of automatic plant species identification using sparse representation of leaf tooth features," *PLOS ONE*, vol. 10, pp. 1–20, 10 2015.

[103] P. Barré, B. C. Stöver, K. F. Müller, and V. Steinhage, "Leafnet: A computer vision system for automatic plant species identification," *Ecological Informatics*, vol. 40, pp. 50–56, 2017.

[104] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. Lopez, and J. V. B. Soares, "Leafsnap: A computer vision system for automatic plant species identification," in *The 12th European Conference on Computer Vision (ECCV)*, October 2012.

[105] A. Olsen, D. A. Konovalov, B. Philippa, P. Ridd, J. C. Wood, J. Johns, W. Banks, B. Girgenti, O. Kenny, J. Whinney, B. Calvert, M. R. Azghadi, and R. D. White, "Deepweeds: A multiclass weed species image dataset for deep learning," *Scientific Reports*, vol. 9, p. 2058, Feb 2019.

[106] *Plant Identification in an Open-world (LifeCLEF 2016)*, vol. CEUR Workshop Proceedings of *CLEF: Conference and Labs of the Evaluation Forum*, (Évora, Portugal), Sept. 2016.

[107] M. A. Beck, C.-Y. Liu, C. P. Bidinosti, C. J. Henry, C. M. Godee, and M. Ajmani, "An embedded system for the automated generation of labeled plant images to enable machine learning applications in agriculture," *PLOS ONE*, vol. 15, pp. 1–23, 12 2020.

[108] M. A. Beck, C. Liu, C. P. Bidinosti, C. J. Henry, C. M. Godee, and M. Ajmani, "Presenting an extensive lab- and field-image dataset of crops and weeds for computer vision tasks in agriculture," *CoRR*, vol. abs/2108.05789, 2021.

[109] S. Zhang, W. Huang, Y. an Huang, and C. Zhang, "Plant species recognition methods using leaf image: Overview," *Neurocomputing*, vol. 408, pp. 246–272, 2020.

[110] S. G. Wu, F. S. Bao, E. Y. Xu, Y.-X. Wang, Y.-F. Chang, and Q.-L. Xiang, "A leaf recognition algorithm for plant classification using probabilistic neural network," in *2007*

IEEE International Symposium on Signal Processing and Information Technology, pp. 11–16, 2007.

[111] S. Purohit, R. Viroja, S. Gandhi, and N. Chaudhary, "Automatic plant species recognition technique using machine learning approaches," in *2015 International Conference on Computing and Network Communications (CoCoNet)*, pp. 710–719, 2015.

[112] X. Wang, C. Zhang, and S. Zhang, "Multiscale convolutional neural networks with attention for plant species recognition," *Computational intelligence and neuroscience*, vol. 2021, pp. 5529905–5529905, Jul 2021.

[113] M. Mehdipour Ghazi, B. Yanikoglu, and E. Aptoula, "Plant identification using deep neural networks via optimization of transfer learning parameters," *Neurocomputing*, vol. 235, pp. 228–235, 2017.

[114] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[115] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[116] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[117] K. Bai, "A comprehensive introduction to different types of convolutions in deep learning." https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215, 2019. (accessed 11 April 2022).