KTH Electrical Engineering

Training of a Neural Network with Using Deterministic Transforms

POL GRAU JURADO

Master Thesis in Electrical Engineering
Stockholm, Sweden 2021

Author

Pol Grau Jurado <polgj@kth.se>
Master's programme in Information and Network Engineering, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology
Master's programme in Telecommunications Engineering, ETSETB, UPC

Place of Project

Stockholm, Sweden

Examiner

Saikat Chatterjee
Stockholm, Sweden
Division of information science and engineering, KTH Royal Institute of Technology

Supervisor KTH

Alireza Mahdavi Javid
Stockholm, Sweden
Division of information science and engineering, KTH Royal Institute of Technology

Supervisor UPC

Jose Antonio Lazaro Villa
Barcelona, Spain
ETSETB, UPC

Abstract

Deep neural networks have been a leading research topic within the machine learning field for the past few years. The introduction of graphical processing units (GPUs) and hardware advances made possible the training of deep neural networks. Previously the training procedure was impossible due to the huge amount of training samples required. The new trained introduced architectures have outperformed the classical methods in different classification and regression problems. With the introduction of 5G technology, related to low-latency and online applications, the research on decreasing the computational cost of deep learning architectures while maintaining state-of-art performance has gained huge interest.

This thesis focuses on the use of Self Size-estimating Feedforward Network (SSFN), a feedforward multilayer network. SSFN presents low complexity on the training procedure due to a random matrix instance used in its weights. Its weight matrices are trained using a layer-wise convex optimization approach (a supervised training) combined with a random matrix instance (an unsupervised training). The use of deterministic transforms is explored to replace random matrix instances on the SSFN weight matrices. The use of deterministic transforms automatically reduces the computational complexity, as its structure allows to compute them by fast algorithms. Several deterministic transforms such as discrete cosine transform, Hadamard transform and wavelet transform, among others, are investigated. To this end, two methods based on features' statistical parameters are developed. The proposed methods are implemented on each layer to decide the deterministic transform to use.

The effectiveness of the proposed approach is illustrated by SSFN for object classification tasks using several benchmark datasets. The results show a proper performance, similar to the original SSFN, and also consistency across the different datasets. Therefore, the possibility of introducing deterministic transformations in machine learning research is demonstrated.

Keywords

Multilayer neural network, deterministic transforms, weight matrices, computational cost, machine learning.

Sammanfattning

Under de senaste åren har djupa neurala nätverk varit det huvudsakliga forskningsområdet inom maskininlärning. Införandet av grafiska processorenheter (GPU:er) och hårdvaruutveckling möjliggjorde träning av djupa neurala nätverk. Tidigare var träningsförfarandet omöjligt på grund av den enorma mängd datapunkter som krävs. De nya tränade arkitekturerna har överträffat de klassiska metoderna i olika klassificerings- och regressionsproblem. Med introduktionen av 5G-teknik, som hör samman med låg fördröjning och onlineapplikationer, har forskning om att minska beräkningskostnaderna för djupinlärningsarkitekturer utan att tappa prestandan, fått ökat intresset.

Denna avhandling fokuserar på användningen av Self Size Estimating Feedforward Network (SSFN), ett feedforward multilayer-nätverk. SSFN har låg komplexitet i träningsproceduren på grund av en slumpmässig matrisinstans som används i dess vikter. Dess viktmatriser tränas med hjälp av en lagervis konvex optimeringsstrategi (en övervakad träning) i kombination med en slumpmässig matrisinstans (en oövervakad träning). Användningen av deterministiska transformationer undersöks för att ersätta slumpmässiga matrisinstanser på SSFN-viktmatriserna. Användningen av deterministiska transformationer ger automatiskt en minskning av beräkningskomplexiteten, eftersom dess struktur gör det möjligt att beräkna dem med snabba algoritmer. Flera deterministiska transformationer som diskret cosinustransformation, Hadamardtransformation och wavelettransformation undersöks bland andra. För detta ändamål utvecklas två metoder som baseras på statistiska parametrar i indatans olika dimensioner. De föreslagna metoderna implementeras på varje lager för att bestämma den deterministiska transform som ska användas.

Effektiviteten av det föreslagna tillvägagångssättet illustreras med SSFN för objektklassificering med hjälp av flera dataset. Resultatet visar ett korrekt beteende, likt den ursprungliga SSFN, och konsistenta resultat över de olika dataseten. Därmed demonstreras möjligheten att införa deterministiska transformationer i maskininlärningsforskning.

Keywords

Neurala nätverk i flera lager, deterministiska transformationer, viktmatriser, beräkningskostnad, maskininlärning.

Acknowledgements

I want to thank all the people who helped me complete this thesis and supported and cheered me during my studies.

First of all, I would like to thank my examiner, Prof. Saikat Chatterjee, for all your support, patience and talks, who in the very beginning advised me that the research road would be difficult, and although it was, it has been the project with which I have learned the most. I would also like to thank my supervisor, Dr Alireza M. Javid; thanks a lot for your patience, guidance, and constant availability. You have helped me a lot.

On a personal level, first, I want to thank my parents, Inma and Xavi, for always believing in me, even when it felt difficult. Special thanks to my girlfriend Aina, who has encouraged me relentlessly. I want to express my esteem to all the friends I made during my time in Sweden, primarily to Johan and Enric, and to my friends that I left in Barcelona but are still there. I cannot mention everyone I would like to thank personally. It would not have resulted in the same without all of you.

Contents

List of Figures

List of Tables

List of Acronyms

ADMM	Alternating direction method of multipliers
ANN	Artificial neural network
CNN	Convolutional neural network
DB"X"	Daubechies "X" transform
DCT	Discrete cosine transform
DHT	Discrete Hartley transform
DLT	Deterministic linear transform
DNN	Deep neural network
DST	Discrete sine transform
DWT	Discrete wavelet transform
ELM	Extreme learning machine
FNN	Feed-forward neural network
FWHT	Fast Walsh-Hadamard transform
LFP	Loseless flow property
LS	Least-squares algorithm
LT	Linear transform
NLT	Non-linear transform
SLFN	Single-layer feed-forward network
SSFN	Self size-estimating feed-forward neural network
SVD	Singular value decomposition

Chapter 1

Introduction

To develop artificial computing systems capable of learning and being closest to human intelligence in decision terms has been the aspiration of researchers since digital technology appearance. It was in the late 1950s, by the hand of Rosenblatt, that the first considered artificial neural network (ANN), named perceptron, was invented [3]. This perceptron constituted the base for multi-layer neural networks and, consequently, deep learning (DL), with its first appearance in [4]. Unfortunately, training an RNA is computationally expensive, limiting its research and use for a few decades.

It has not been until the last decade, with technology evolution and hardware advances, that ANNs have received enormous attention. Newly neural network architectures have been created and improved. Examples such as deep neural networks (DNNs) [5] and convolutional neural networks (CNNs) [6], have outperformed the classical methods in different classification and regression problems [7, 8]. However, this impressive progress has come with a cost–it has made machine learning (ML) training strictly reliant on extensive computational resources due to the huge amount of data needed to train a network, as well as the enormous amount of operations and parameters to train. In order to increase the training speed, clusters of CPU or parallel computations using graphical processing unit (GPUs) have been applied. However, even many optimizations have been achieved, the process complexity and time consumption are huge, requiring high data processing capabilities [9].

Nowadays, with the 5G evolution, many applications demand efficient and fast algorithms, such as low-latency communication devices [10] and artificial intelligence in autonomous vehicle driving [11]. Several algorithms and architectures have been developed to approach these requirements. There are two main classes of DNNs based algorithms that address modern neural networks' high computational complexity requirements. The first class of algorithms tries to preserve the state-of-the-art performance of famous architectures, such as ResNet [12] and AlexNet [6], while reducing the size of the network as much as possible. EfficientNet [13], SqueezeNet [14], and MobileNet [15] are examples of this kind. However, training of the above architecture is still quite expensive due to the use of stochastic gradient descent and backpropagation [16]. The second class of algorithms tries to resolve this issue by using a gradient-free training approach. To this end, one popular technique is that some of the weight matrices of the network are set to instances of random matrices, and only the rest of the weight matrices are updated during training. This leads to a convex relaxation of the training cost and eliminates the need for error backpropagation throughout the layers. Extreme learning machine (ELM) [17], random vector functional link (RVFL) and its variants [18, 19], progressive learning network (PLN) [1], and self size-estimating feed-forward network (SSFN) [2] are examples of this class that are shown to provide competitive performance with very low computational requirements in various applications.

1.1 Problem Statement

To solve the expensive computational cost of deep neural networks, two algorithm approaches have been developed. The first class tries to reduce the network size as much as possible to reduce its operational complexity while keeping the performance. The second class also deals with the training complexity.

The intends to reduce the network operation complexity always have been similar, proposing different architectures and learning methods. Although there are uncountable different constructed network architectures, they are usually based on the same principle: a list of concatenated layers connecting the network input-output. Each layer consists basically of linear matrix multiplications followed by a non-linear transform.

The possibility to reduce neural network operation complexity without changing the "linear transform - non-linear transform" structure but changing the computation process could be explored.

1.2 Research Question and Goals

From the end of the above section, a question is raised

Is it possible to find a different network implementation approach to reduce the network operational complexity?

In other words, propose a new network operation paradigm, conserving the "linear transform - non-linear transform" structure, to reduce the network computational complexity after trained.

Thus, the main goal of the project is

Propose a method to achieve a reduction of the computational cost of a neural network on the operational stage, while preserving similar performances and training complexity compared to the original network.

In order to preserve network low training complexity, the second-class of neural network algorithms presented before will be considered. This class of networks uses a random matrix instance in the linear transform, which may give versatility to explore linear transforms atypical in neural networks.

1.3 Research Methodology

The second class of DNNs algorithms is chosen to consider and modify in the project. The structure of this class of networks is mainly based on multilayer perceptron (MLP) [20], more specifically on a feed-forward neural network (FNN) structure, presented in Figure 1.1. It consists of the input-output connected by a box. The box is filled by one or more layers, interconnected by a linear transform (LT), called weights, followed by a non-linear transform (NLT). The networks within this class of algorithms have in common that a part of their LT consists of random weights. The neural network decided to take into consideration is SSFN [2].

When the network is already trained and the network is operating, unseen samples pass through the network. When an input sample enters the network, at every layer is performed a vector-matrix product followed by a non-linear transform. If MLP structure is wanted to be kept, two options remain to improve the computational cost:

1. Modify the linear transform block, weights matrix.

2. Modify the non-linear transform block, activation function.

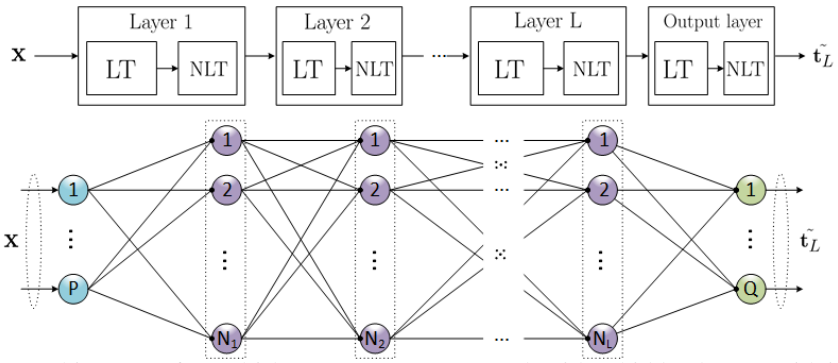


Figure 1.1: Architecture of a multi-layer perceptron network with L hidden layers, with input $\mathbf{x}$ and prediction $\tilde{\mathbf{t}}_L$.

In the project, the first option is approached. The linear transform block will be modified, and therefore the solutions might be either develop an efficient way to compute vector-matrix multiplication, which is not an option or find a more efficient way to compute the linear transform.

Following the second approach, it is proposed to use signal processing linear transforms, also referred to as deterministic or orthogonal transforms. In signal processing, a large variety of discrete linear transforms have an essential role in spectral analysis, image processing, and data compression, among others, representing different signal properties and features. A property of these transforms is their orthogonality or semi-orthogonality, making it possible to be computed by fast computation algorithms, reducing the cost.

The neural network decided to work with is Self Size-estimating Feed-forward Network (SSFN) [2]. It is built following a layer-wise approach with convex optimization, i.e., a new layer is added in top of the others and the corresponding output matrix is optimized. The network also presents low training complexity, achieved by using a random matrix instance, and ensuring monotonically decreasing training cost. Three main reasons are present to consider and work implementing this architecture:

1. Self size-estimating. It estimates itself its size, hence the network builds itself accordingly to its optimal architecture.

2. Random matrix instance. A random matrix is whatever matrix, thus it can be easily replaced by a deterministic transform.

3. Monotonically decreasing training cost. Even SSFN does not use gradient descent algorithms, the training cost is assured to decrease.

The project intends to assemble the concepts presented in this section in order to develop a new framework for the linear transform of neural network architectures.

1.4 Main Contributions

The author explored different deterministic linear transforms, such as Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT), widely applied in diverse signal processing fields. Deterministic transforms replace the random matrices instance of SSFN, the chosen neural network. Therefore, SSFN algorithm is modified and adapted to introduce the new working approach. Some statistical signal properties are summarized and used in the different approaches. Three different experiments are carried to perform a complete comparison and study of the implementation with

the original network in [2]. The studies show to provide similar performances to the original case over several bench-mark classification datasets. It offers a new method to compute linear transforms weights for different neural network architectures belonging to the second class of algorithms, based on deterministic transforms use.

Chapter 2

Background

In this chapter necessary theoretical background used along the thesis work is covered. In Section 2.1 the concept of signal processing deterministic linear transforms is introduced. Section 2.2 follows with a brief definition and explanation of three different signal statistical properties widely used in the field of signal processing. In Section 2.3 feed-forward neural networks (FNNs) are introduced, starting with a brief ANNs introduction and followed by its structure definition and training. Two classes of FNNs are distinguished but focusing on the second class by briefly explaining extreme learning machine (ELM). The section ends with a detailed introduction to SSFN, network structure evaluated in this thesis.

2.1 Deterministic Linear Transforms

Data collection has grown exponentially in the last decades with different infinite purposes such as image and audio analysis, noise reduction, and data transmission. Collected data is acquired from signals in nature and it must be meaningful for each specific purpose. Signals in their nature can be observed in spatial and temporal dimensions, which typically are correlated and with uniformly distributed energy. These facts, in general, are not usually desirable when pretending to analyze and extract information from the captured data. Signal correlation and uniformly distributed energy hinder signal decomposition, needed in applications such as information extraction, noise reduction and data compression [21]. Data analysis and signal processing become necessary to cope with this problem and represent data in a meaningful representation. At this point is where orthogonal linear transforms had become essential in signal processing.

Applying an orthogonal linear transform to an input signal present two main advantages of interest to this thesis:

- Decorrelates the signal and concentrates its energy in fewer coefficients, allowing a more compact information representation.

- Represent the signal in an alternative dimension than time and space, as frequency dimension, where more features can be present depending on the purpose. Fourier transform is an example, being the most famous and worldwide used transform illustrated in Figure 2.1.

An orthogonal or semi-orthogonal transform is equivalent to a simple rotation of the data in its dimensions, meaning that its energy and information are kept untouched. Biorthogonal, as some discrete wavelet transforms are [22], can be also as efficient in terms of energy preservation as orthogonal transforms [23] under some conditions. Different transforms differ among them on the signal properties they represent and how it is done, as well as how uncorrelated the resulting

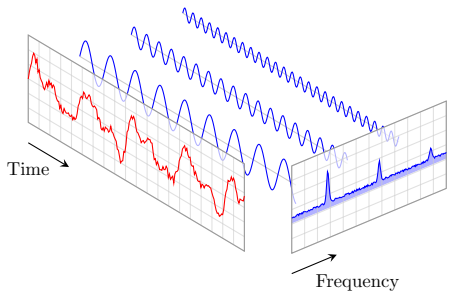


Figure 2.1: Discrete Fourier transform illustration.

coefficients are and its new information distribution. Depending on the application is intended, a transform will be optimal for the specified purposes.

Orthogonal linear transforms are fixed matrices for a determined input dimension N, i.e. are signal independent. To emphasize its constant structure, in the project they will be referred as Deterministic Linear Transforms (DLTs), denoted as $\mathbf{W}_{DLT}$. DLTs can be either orthogonal transforms or semi-orthogonal. Consider a raw input signal N-dimensional $\mathbf{x} \in \mathbb{R}^N$, to which a DLT $\mathbf{W}_{DLT} \in \mathbb{R}^{M \times N}$ is applied by matrix product

$$\mathbf{y} = \mathbf{W}_{DLT}\mathbf{x}. \tag{2.1}$$

The resulting output vector $\mathbf{y} \in \mathbb{R}^M$ is computed with $\mathcal{O}(M \times N)$ computational complexity. It is important to mention that most of the deterministic transforms are squared transforms, $N = M$, and even in some cases squared by a power of 2 [24].

Computational complexity when implementing DLTs can be reduced by using fast algorithms developed for each of them, as Fast Fourier Transform (FFT) can be for the DFT. These fast algorithms make use of matrix factorization (as FFT algorithm), cascade filterbanks (as DWT algorithms), and divide and conquer recursive implementations (as FWHT algorithm). By fast algorithms implementation, it is possible to reduce the computational complexity, for a square matrix transform, from $\mathcal{O}(N^2)$ to $\mathcal{O}(Nlog_2N)$ [21, 24], or even $\mathcal{O}(N)$ [25, 26]. To highlight when a fast algorithm is used instead of matrix products, the previous equation will be expressed in function notation as

$$\mathbf{y} = w_{DLT}(\mathbf{x}), \tag{2.2}$$

expressing the DLT as $w(\,\cdot\,) \in \mathbb{R}^N \rightarrow \mathbb{R}^M$.

2.2 Signal properties

Three different statistical signal properties, standard deviation and variance, cross-correlation matrix and singular value decomposition (SVD), are introduced in this section. It is crucial to fully understand them as the proposed methodology approaches rely on these properties.

2.2.1 Standard deviation and variance

In statistics given a signal $X = \left[x^{(1)}, ..., x^{(J)}\right]$, with J samples, its variance σ_X^2 or standard deviation σ_X measure the amount of variation or dispersion of the signal around its mean value. Standard deviation preserves the signal units, being useful if a direct comparison is intended.

$$\sigma_X = \sqrt{\frac{\sum_{j=1}^{J}(x^{(j)} - \mu_X)^2}{J}}, \qquad \sigma_Y = \sqrt{\frac{\sum_{j=1}^{J}(y^{(j)} - \mu_Y)^2}{J}}. \tag{2.3}$$

A low variance indicates that the samples of a set are close to the mean, while a large variance indicates that samples are spread from the mean, illustrated in Figure 2.2. In a neural network, the variance of the nodes can refer to the information a node is carrying [27]. If a node presents low variance across the training dataset, it means that the node presents similar values for all training samples and therefore, it carries a small amount of information, acting as a bias term.

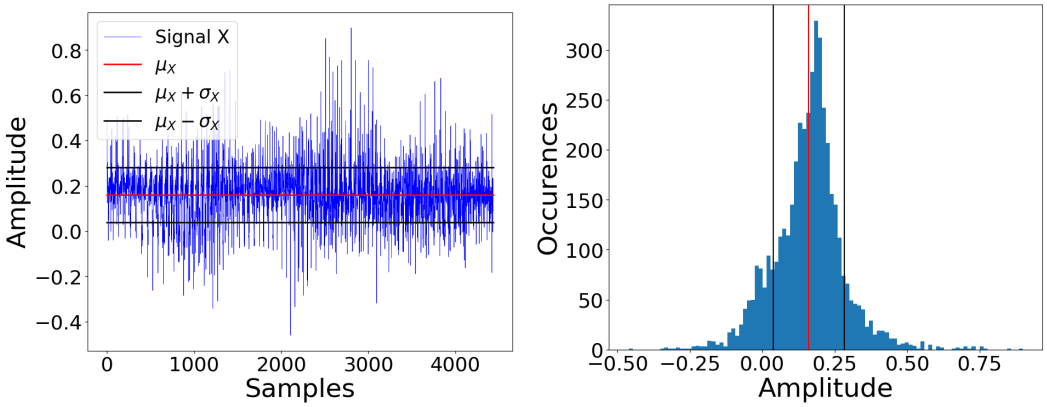


Figure 2.2: Signal plot on the left and same signal histogram on the right. Mean value indicated as a red line and standard deviation as black lines.

2.2.2 Cross-Correlation

In signal processing cross-correlation is a similarity measure among two signals as a function of time delay and relativity. If there is statistical relationship between two signals it means that a dependency exist between them. Cross-correlation between two signal variables $X = \left[x^{(1)}, ..., x^{(J)}\right]$ and $Y = \left[y^{(1)}, ..., y^{(J)}\right]$, both of J observations, is computed as

$$Cov(X, Y) = \frac{1}{J} \sum_{j=1}^{J} (x^{(j)} - \mu_X)(y^{(j)} - \mu_Y) \tag{2.4}$$

$$\rho_{X,Y} = Corr(X, Y) = \frac{Cov(X, Y)}{\sigma_X \sigma_Y} = \frac{E\left[(X - \mu_X)(Y - \mu_Y)\right]}{\sigma_X \sigma_Y}, \tag{2.5}$$

where $Cov(\cdot)$ and $E(\cdot)$ are the covariance and expected value operators respectively, μ and σ are the signal mean and standard deviation respectively and $\rho_{X,Y}$ is the correlation coefficient. The correlation coefficient is a dimensionless measure being $-1 \leq \rho_{X,Y} \leq 1$. The closer to the extreme values ± 1, stronger is the statistical relation between the signals, where the sign indicates similarities direction.

Now, consider two different vectors of variables expressed as $\mathbf{X} = [X_1, ..., X_P]^T$ and $\mathbf{Y} = [Y_1, ..., Y_N]^T$ with different amount of variables. Each variable X_1, X_2, Y_1, etc, is defined as before with J observations or samples. The correlation measure between two different variables is commonly summarized in the form of the correlation matrix $\mathbf{R}_{\mathbf{X},\mathbf{Y}}$.

$$\mathbf{R}_{\mathbf{X},\mathbf{Y}} = \begin{bmatrix} \rho_{X_1,Y_1} & \rho_{X_1,Y_2} & \cdots & \rho_{X_1,Y_Q} \\ \rho_{X_2,Y_1} & \rho_{X_2,Y_2} & \cdots & \rho_{X_2,Y_Q} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{X_P,Y_1} & \rho_{X_P,Y_2} & \cdots & \rho_{X_P,Y_Q} \end{bmatrix}. \tag{2.6}$$

2.2.3 Singular Value Decomposition (SVD)

SVD is a factorization method for real and imaginary matrices widely used in signal processing tasks. The two most important properties are the following:

- Decorrelates variables into a set of uncorrelated to express and make use of different relationships of the original data.
- Identify and order the main variation of points among different dimensions the data have. This allows to reduce data dimensions while keeping the maximum information amount.

A matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ is decomposed as

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T, \tag{2.7}$$

where $\mathbf{U} \in \mathbb{R}^{M \times M}$ and $\mathbf{V} \in \mathbb{R}^{N \times N}$ are orthogonal matrices which its column vectors are referred as left-singular vectors and right-singular vectors respectively. The singular values matrix $\mathbf{\Lambda} \in \mathbb{R}^{M \times N}$ is a diagonal matrix of the form

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \lambda_M \\ \vdots & & & 0 \\ \vdots & & & \vdots \\ 0 & \dots & \dots & 0 \end{bmatrix}. \tag{2.8}$$

When $\mathbf{A}$ is a squared matrix, singular values correspond to eigenvalues and singular vectors correspond to eigenvectors.

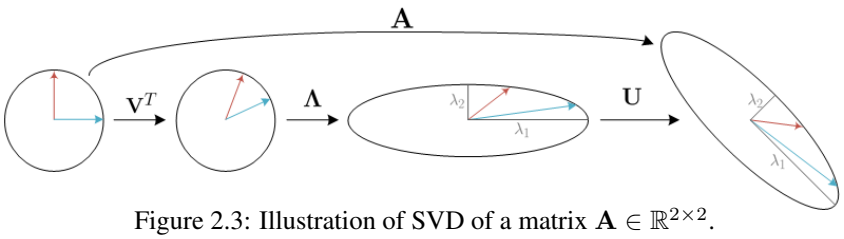


Figure 2.3: Illustration of SVD of a matrix $\mathbf{A} \in \mathbb{R}^{2 \times 2}$.

2.3 Feedforward Neural Networks

In this section first, it is explained an overview of the concept of Artificial Neural Networks (ANNs). Then the focus is set on Feedforward Neural Networks (FNNs), a type of Deep Neural Network (DNN) whose connections are connected only to forward layers. After DNNs structure is summarized its training procedure is approached. The training of a DNN is divided into two groups. The first group makes use of backpropagation and stochastic gradient descent, it is summarized and some disadvantages are listed. The second group solves some of those problems and it is illustrated by summarizing the training of extreme learning machine (ELM). To end the section, SSFN is summarized.

2.3.1 Overview

ANNs are computation algorithms based on brain neural network systems, although much less complex. Unlike traditional algorithms which are created for a specific purpose, neural networks are capable to learn to work for a specific task by themselves, by carrying a training procedure, analogously how humans do.

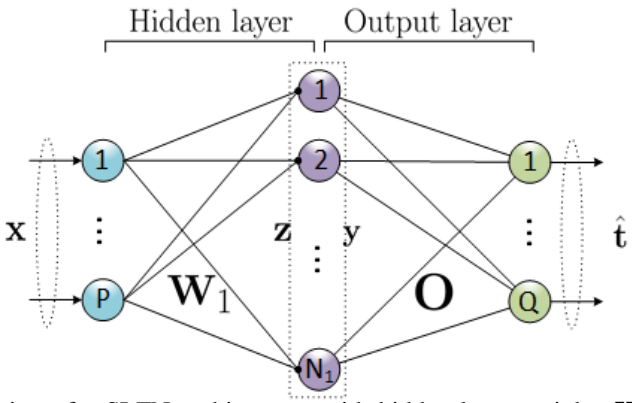


Figure 2.4: Illustration of a SLFN architecture, with hidden layer weights $\mathbf{W}_1$ and output layer weights $\mathbf{O}$.

An ANN architecture consists basically of an input layer with several input neurons P equal to the dimension of the input signal, its body which consists of one or more hidden layers, and the output neurons Q, with the number of neurons equal to the dimension of the output. When the layer connections are connected only to the next layers without forming a loop the network is named feed-forward neural network (FNN). In 2.4 is illustrated an FNN with only one hidden layer, named single-layer feed-forward network (SLFN). Provided enough number of neurons in an SLFN hidden layer any function of a compact set can be approximated [28]. However, it is a difficult task to achieve, almost impossible, as it requires a huge computational cost. To achieve a similar representation with less computational cost, deep neural networks (DNNs) are implemented. Instead of using only one wide layer to approximate the functions, DNNs are built with more than one and narrower hidden layers.

2.3.2 DNNs Overall structure

As mentioned above, DNNs are composed of an input layer, hidden layers and output layer. Consider a dataset containing J samples of P-dimensional input data $\mathbf{x}^{(j)} \in \mathbb{R}^P$ and its corresponding Q-dimensional target vector $\mathbf{t}_j \in \mathbb{R}^Q$, defined as $\mathcal{D} = \left\{\mathbf{x}^{(j)}, \mathbf{t}^{(j)}\right\}_{j=1}^{J}$. The j'th first layer linear transform (LT) output, with weight matrix $\mathbf{W}_1 \in \mathbb{R}^{N_1 \times P}$, is described as

$$\mathbf{z}_1^{(j)} = \mathbf{W}_1 \mathbf{x}^{(j)} + \mathbf{b}_1, \tag{2.9}$$

with output $\mathbf{z}_1^{(j)} \in \mathbb{R}^{N_1}$ and $\mathbf{b}_1 \in \mathbb{R}^{N_1}$ corresponding to the bias term used to adjust the output. The LT is always followed by a non-linear transform, named activation function $g(\,\cdot\,)$, to add non-linearity to the model in each node output, producing the feature vector $\mathbf{y}_1^{(j)} = g(\mathbf{z}_1^{(j)}) \in \mathbb{R}^{N_1}$. Real data is usually non-linear and with high dimensionality and/or complexity. The implementation of activation functions in the network allows it to create more complex mappings within the signal flow and therefore approximate and model more accurately the input data.

Convolutional neural networks (CNNs) are a type of DNNs with different architecture and layer variations. They work in a multi-dimensional space where the weights are a filter and the computation is carried through convolutions, therefore sharing the weights across the outputs. After the weight convolution and before the activation function, there is an extra stage to reduce the dimension named pooling layer. They are mainly used in image processing [29].

2.3.3 DNNs training

When training a neural network is intended to fit to the training data while keeping a generalization for when using the network with non-known samples. The training is carried by optimizing the parameters to minimize the loss cost $\mathcal{L}$ as

$$\mathbf{W}^*, \mathbf{O}_L^* = \arg\min_{\mathbf{W},\mathbf{O}_L} \sum_{j=1}^{J} \mathcal{L}(\mathbf{t}^{(j)}, \mathbf{O}_L f_{\mathbf{W}}(\mathbf{x}^{(j)})). \tag{2.10}$$

In the equation network hidden layer's parameters are represented by the function $f_{\mathbf{W}}(\mathbf{x}^{(j)}) = g(\mathbf{W}_{L-1}g(...g(\mathbf{W}_1\mathbf{x}^{(j)})))$, where the biases are obviated but present and $\mathbf{O}_L$ references to the network output matrix at last layer L. The loss function $\mathcal{L}$ of a sample pair $\{\mathbf{x}^{(j)}, \mathbf{t}^{(j)}\}$ can be mean-squared error loss, Huber loss or log-cosh loss among others. The implemented cost in this thesis is mean square error loss defined by

$$\mathcal{C} = \mathcal{L}(\mathbf{O}_L, \mathbf{W}) = \frac{1}{J}\sum_{j=1}^{J} ||\mathbf{t}^{(j)} - \hat{\mathbf{t}}^{(j)}||^2 = \frac{1}{J}\sum_{j=1}^{J} ||\mathbf{t}^{(j)} - \mathbf{O}_L f_{\mathbf{W}}(\mathbf{x}^{(j)})||^2. \tag{2.11}$$

The progress achieved by DNNs has come with a counter cost regarding the computational cost needed to train the networks to achieve desired performances. To reduce the complexity requirements of training DNNs, there can be distinguished two different classes of algorithms. The first class is trying to preserve the state-of-the-art performance of famous architectures while intends to reduce the size of the network, as EfficientNet and SqueezeNet. Although the network size is reduced and therefore the training complexity is also reduced, it is still quite expensive due to the use of backpropagation algorithm and stochastic gradient descend minimization method. To deal directly with training cost reduction, a second class of algorithms is implemented. These second class algorithms are based on the use of a random matrix instance in some weight matrices of the network, meaning that do not have to be trained.

In the following subsections are described both classes of algorithms. First, the typical stages of the first-class algorithms are described together with their main disadvantages. It is followed by a well-known algorithm named ELM, in order to illustrate the second-class algorithm.

2.3.3.1 First class of DNN algorithms: stochastic gradient descent and Back-propagation

Consider a FNN as the SLFN sowhn in Figure 2.4 but with $L-1$ hidden layers and $\mathbf{O}_L$ output matrix, with all weights randomly initialized. The network have two different weight matrices, the hidden layers weights $\mathbf{W}_l \in \mathbb{R}^{N_l \times N_{l-1}}$ with $1 \leq l \leq L-1$, and the output matrix $\mathbf{O}_L \in \mathbb{R}^{Q \times N_1}$ weights. The prediction target is defined as

$$\hat{\mathbf{t}}^{(j)} = \mathbf{O}_L f_{\mathbf{W}}(\mathbf{x}^{(j)}) \qquad j = 1, ..., J. \tag{2.12}$$

Expanding to the whole dataset with $\hat{\mathbf{T}} = \left[\hat{\mathbf{t}}^{(1)}, \hat{\mathbf{t}}^{(2)}, ..., \hat{\mathbf{t}}^{(J)}\right] \in \mathbb{R}^{Q \times J}$ the equation can be compacted as

$$\hat{\mathbf{T}} = \mathbf{O}_L \mathbf{H}, \tag{2.13}$$

where

$$\mathbf{H} = \begin{bmatrix} g(f_{\mathbf{W}}(\mathbf{x}^{(1)})) \\ g(f_{\mathbf{W}}(\mathbf{x}^{(2)})) \\ \vdots \\ g(f_{\mathbf{W}}(\mathbf{x}^{(J)})) \end{bmatrix}^T . \tag{2.14}$$

$\mathbf{H} \in \mathbb{R}^{N_L \times J}$ is defined as the hidden layer output matrix of the neural network [30]. The n'th row of the matrix corresponds to the last hidden layer $L-1$ n'th node, respect to $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(J)}$input observations.

The training procedure is carried with the goal to find the optimal parameters, i.e. weights and biases $\mathbf{W}$, $\mathbf{b}$ and $\mathbf{O}_L$, such that

$$||\mathbf{T} - \mathbf{O}_L^* \mathbf{H}(\mathbf{W}^*, \mathbf{b}^*)|| = \min_{\mathbf{W},\mathbf{b},\mathbf{O}_L} ||\mathbf{T} - \mathbf{O}_L \mathbf{H}(\mathbf{W}, \mathbf{b})||, \tag{2.15}$$

which is equal to minimize the error in (2.11).

To solve the minimization problem in previous equation, stochastic gradient descent methods are used. In the minimization procedure $\boldsymbol{\beta}$ is defined as the set of weights and biases, $\mathbf{W}$, $\mathbf{b}$ and $\mathbf{O}$. It is computed iteratively as

$$\boldsymbol{\beta}_k = \boldsymbol{\beta}_{k-1} - \eta \frac{\delta \mathcal{L}(\boldsymbol{\beta})}{\delta \boldsymbol{\beta}}, \tag{2.16}$$

with $\eta > 0$ being the learning rate.

Backpropagation method, therefore, is used to learn efficiently the gradients by propagating from the output to input and it can be summarized into the following four main steps, which are repeated until the training error is acceptable small [31]:

1. Feed-forward computation.
2. Backpropagation to the output layer.
3. Backpropagation to the hidden layer.
4. Weight updates.

As it can be guessed from the previous steps, working with stochastic gradient descent together with BP is a computationally expensive method due to gradient computation and all the propagation iterations. Several inconveniences appear and have to be taken into account when working with the combination of the two algorithms:

- Necessity of hand-tuning properly the learning rate η hyperparameter (2.16). When η is too small the algorithm converges very slowly, but when is too large, the algorithm can diverge due to instability.
- The presence of local minima may make think the backpropagation algorithm that has reached the global and stop in a non-optimal point [32].
- Over-fitting may happen and therefore modifications such as validation and stop methods are needed.
- Very time-consuming due to the computational complexity of the optimization operations.

2.3.3.2 Second class of DNN algorithms: Random weights instance

To cope with some of the exposed problems above, a second class of FNN algorithms is created. By avoiding the use of backpropagation and stochastic gradient descent, the local minima problem is solved and computational cost can be reduced. Some examples of this second class are RVFL, SSFN and ELM, this last being the most famous and widely used. They are based on the use of a random definition of network weights and biases making unnecessary their training. The output matrix requires to be optimized and it can be done with some optimization algorithm, such as least-squares (LS).

Following, ELM algorithm will be briefly explained to have a basic notion on the operation basics of this class of algorithms. Consider the SLFN architecture of Figure 2.4 and its output as in (2.12), $\hat{\mathbf{t}}_j = \mathbf{O}g(\mathbf{W}_1\mathbf{x}_j + \mathbf{b})$. ELM algorithm is based on the random initialization of the weight matrix $\mathbf{W}_1$ and biases $\mathbf{b}$, keeping them as constant. The only parameters that need to be optimized are the output weight $\mathbf{O}_2$ by solving the linear system in (2.15), which now is simply solved by finding the LS solution as

$$||\mathbf{T} - \mathbf{O}^*\mathbf{H}(\mathbf{W}_1, \mathbf{b})|| = \min_{\mathbf{O}}||\mathbf{T} - \mathbf{O}\mathbf{H}(\mathbf{W}_1, \mathbf{b})|| \tag{2.17}$$

According to [33] the LS solution is

$$\hat{\mathbf{O}} = \mathbf{H}^+\mathbf{T} = g(\mathbf{W}_1\mathbf{x})^+\mathbf{T} \tag{2.18}$$

with $\mathbf{H}^+$ being the Moore-Penrose generalized pseudoinverse of matrix $\mathbf{H}$ and $\mathbf{x} = \left[\mathbf{x}^{(1)}, ..., \mathbf{x}^{(J)}\right]$.

The algorithm steps can be summarized then with the following two steps:

1. Initialize $\mathbf{W}_1$ and $\mathbf{b}$ with random values.
2. Calculate the hidden layer output matrix $\mathbf{H}$ as in (2.14).
3. Calculate the ouptut matrix hidden weight $\mathbf{O}$ as in (2.18).

In [34] ELM is extended to H-ELM as a multilayer structure.

2.3.4 Self size-estimating feed-forward network (SSFN)

In this subsection, it is introduced and reviewed the structure and training procedure of SSFN [2]. It belongs to the second class of algorithms previously presented, therefore makes use of a random instance matrix and presents low training computational cost. It is also needed of limited human intervention, estimating itself its size, and architecture shown in Figure 2.5. It is built using a layer-wise approach with convex optimization together with the use of random matrix instances to ensure a monotonically decreasing training cost. A new layer is added on top of an optimized structure and is optimized. Is possible to maintain a decreasing cost by preserving a so-called 'lossless flow property' (LFP), a property of an SLFN. If LFP is preserved it means the signal flows through a system without any loss or change.

Consider the signal flow between the l'th layer and the network input as

$$\mathbf{y}_l = \mathbf{g}(\mathbf{W}_l\mathbf{y}_{l-1}) = \mathbf{g}(\mathbf{W}_l...\mathbf{g}(\mathbf{W}_2\mathbf{g}(\mathbf{W}_1\mathbf{x}))...) \in \mathbb{R}^{N_l}, \tag{2.19}$$

where $\mathbf{W}_l \in \mathbb{R}^{N_l \times N_{l-1}}$ is the weight matrix of l'th layer with N_l hidden neurons and the activation function $g(\cdot)$ is specifically ReLu or a derived generalization. The network is built using a layer-wise approach with convex optimization. A new layer is added on top of an optimized structure and is optimized as follows:

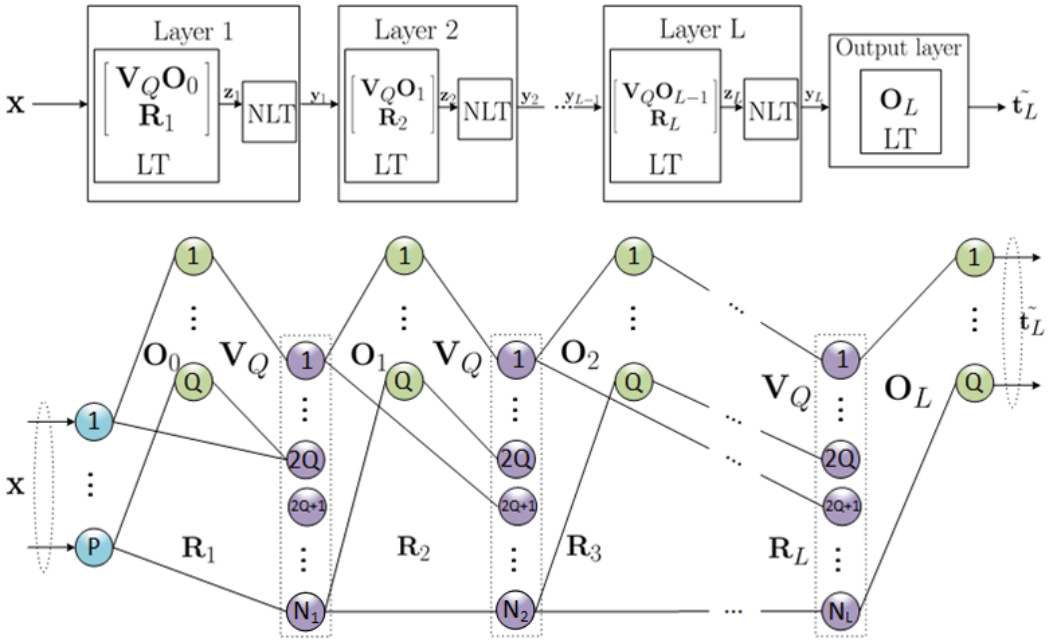


Figure 2.5: Architecture of a multi-layer SSFN with L layers and its signal flow diagram. LT stands for *linear transform*, (weight matrix) and NLT stands for *non-linear transform* (activation function). ReLU is uses as activation function (Schematic from [1]).

1. Consider the training dataset $\mathcal{D} = \{(\mathbf{x}^{(j)} \in \mathbb{R}^P, \mathbf{t}^{(j)} \in \mathbb{R}^Q)\}_{j=1}^J$. The output in the l'th layer is computed as in (2.19).

2. The prediction of the j'th sample and training cost in each l'th layer is defined as

$$\hat{\mathbf{t}}_l^{(j)} = \mathbf{O}_l \mathbf{y}_l^{(j)} \tag{2.20}$$

$$\mathcal{C}_l = \frac{1}{J} \sum_{j=1}^J ||\mathbf{t}^{(j)} - \hat{\mathbf{t}}_l^{(j)}||^2 \tag{2.21}$$

3. The output matrix $\mathbf{O}_l$ is computed by minimizing the previous training cost by solving the following optimization problem

$$\mathbf{O}_l^\star = \arg\min_{\mathbf{O}} \mathcal{C}_l \quad \text{s.t. } ||\mathbf{O}||_F^2 \leq \epsilon_l = 2\alpha Q. \tag{2.22}$$

Here ϵ_l denotes the regularization parameter with $1 \leq \alpha$, refer to [2] for more details. Two methods are used distinguishing between the first output matrix and the remaining:

- First output matrix $\mathbf{O}_0^*$: Regularized least-squares (with Tikonov regularization). Used to solve the unconstrianed Lagrangian:

$$\arg\min_{\mathbf{O}_0} \left\{ \frac{1}{J} \sum_{j=1}^J ||\mathbf{t}^(j) - \mathbf{O}_0 \mathbf{y}_l^{(j)}||^2 + \lambda_0 ||\mathbf{O}_0||_F^2 \right\} \tag{2.23}$$

where λ_0 is RLS regularization parameter.

- From second to onwards output matrix $\mathbf{O}_1^*...\mathbf{O}_L^*$: ADMM. New matrices are defined containing the whole dataset as $\mathbf{T} = \left[\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, ...\mathbf{t}^{(J)}\right]$ and $\mathbf{Y} = \left[\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, ...\mathbf{y}^{(J)}\right]$. Equivalently to (2.22),

$$\mathbf{O}_l^\star = \arg\min_{\mathbf{O}_l, \mathbf{Q}_l} ||\mathbf{T} - \mathbf{O}_l\mathbf{Y}_l||_F^2 \quad \text{s.t.} \quad ||\mathbf{Q}_l||_F^2 \leq \epsilon_\alpha, \quad \mathbf{Q}_l = \mathbf{O}_l. \tag{2.24}$$

  Refer to [2] for derivation of the process and method iterative equations. An hyperparameter $\mu > 0$ is used to control the convergence rate of ADDM.

4. The weight matrix of $(l+1)$'th layer is constructed as

$$\mathbf{W}_{l+1} = \begin{bmatrix} \mathbf{V}_Q\mathbf{O}_l^\star \\ \mathbf{R}_{l+1} \end{bmatrix}, \tag{2.25}$$

   where $\mathbf{V}_Q = [\mathbf{I}_Q;\ -\mathbf{I}_Q]^T \in \mathbb{R}^{2Q\times Q}$, with $\mathbf{I}_Q \in \mathbb{R}^{Q\times Q}$ as identity matrix, is needed to assure LFP. $\mathbf{R}_{l+1}$ corresponds to a random matrix instance.

This procedure is carried until the maximum number of layers L_{max} is reached, or the cost shows a saturation trend defined as

$$\frac{\mathcal{C}_l^* - \mathcal{C}_{l-1}^*}{\mathcal{C}_{l-1}^*} < \eta_{layer}, \tag{2.26}$$

with η_{layer} being a predefined threshold and the cost computed as in (2.21).

Additionally there is also an iterative procedure to compute each layer width. Analogously to the layer-addition procedure, nodes are added in a step-size Δ until the maximum number of nodes N_{max} is reached or it presents a saturation trend

$$\frac{\mathcal{C}_{n_l}^* - \mathcal{C}_{n_l-\Delta}^*}{\mathcal{C}_{n_l-\Delta}^*} < \eta_{node}. \tag{2.27}$$

It is important to note that this layer-wise approach avoids dealing with problems such as vanishing gradients or local minima while ensuring a reduction of the cost. In [2] is It shown that preservation of LFP training of SSFN leads to $\mathcal{C}_l \leq \mathcal{C}_{l-1}$. This procedure is summarized in Algorithm 1.

Algorithm 1 : Algorithm for construction of SSFN

Input:

Training dataset $\mathcal{D}(\mathbf{x}^{(j)}, \mathbf{t}^{(j)})\}_{j=1}^{J}$

Parameters to set:

(a) L_{max} (Maximum number of layers)

(b) $N_{max} \geq 2Q$ (Maximum number of nodes in a layer)

(c) $\alpha \geq 1$ (Parameter in (2.22))

(d) Δ (Numbers of nodes to increase in a step)

(e) μ and k_{max} (Parameters in ADMM)

(f) η_{node} and η_{layer} (Stopping thresholds)

Regularized least-squares:

$\mathbf{y}_0^{(j)} = \mathbf{x}^{(j)}$

Solve (2.23) to find $\mathbf{O}_0^\star$ (Cross-validation for λ_0)

Estimating number of nodes and layers:

Initialization: $l = 0$ (Index for l'th layer)

repeat

  $l \leftarrow l + 1$ (Increase in layers)

  $N_l = 2Q$ (Minimum number of nodes for all layers)

  repeat

    $N_l \leftarrow N_l + \Delta$ (Increase in nodes)

    Construct $\mathbf{W}_{l,N_l}$ according to (2.25)

    Find feature $\mathbf{y}_{l,N_l}^{(j)}$ (For n_l nodes)

    Solve (2.24) to find $\mathbf{O}_{l,N_l}^\star$ (using ADMM)

  until (2.27) and $N_l > N_{max}$ (Cost saturation or maximum nodes)

  $\mathbf{O}_l^\star \leftarrow \mathbf{O}_{l,N_l}^\star$, $C_l^\star \leftarrow C_{N_l}^\star$

until (2.26) and $l > L_{max}$ (Cost saturation or maximum layers)

Output:

Number of layers $L = l$ and number of nodes $\{N_l\}_{l=1}^{L}$

Weight matrices $\{\mathbf{W}_l\}_{l=1}^{L-1}$

Output matrix $\mathbf{O}_L^*$

Chapter 3

Methodology

In this chapter are presented the proposed learning scheme, based on SSFN architecture, subsection 2.3.4, together with the implementation of deterministic transforms in section 2.1. It is followed by the four different learning approaches implemented. A box filled with different linear transforms is constructed and iteratively a transform is chosen to be used in layers addition. Supervised and unsupervised decision criteria to chose the transform are implemented and compared. To generalize the second part of SSFN's linear transform is redefined as $\mathbf{W}_{part2,l}$, as in the figure below.

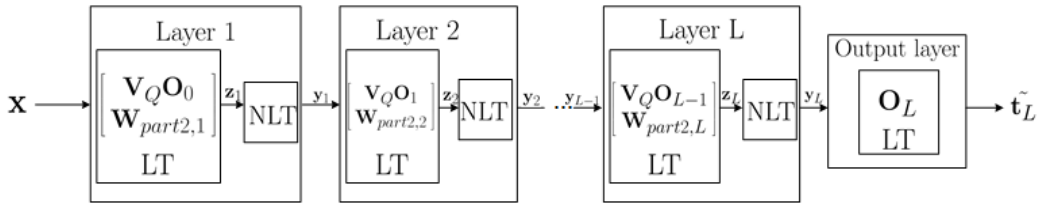


Figure 3.1: Architecture of SSFN as in Figure 2.5 generalizing the LT, replacing the random instance by a linear transform $\mathbf{W}_{part2,l}$.

3.1 Proposed Learning Scheme

This section is subdivided into two subsections. In the first subsection, the term of the linear transform of network layers is redefined with new notation. In the second, is introduced the algorithm to iterate and chose a transform among the linear transforms box.

3.1.1 Linear transform redefinition

In subsection 2.3.4 has been shown that SSFN guarantees a monotonic reduction of the training cost when a new layer is added with weights according to (2.25). This property remains when replacing $\mathbf{R}_{l+1}$ for any linear transform. To ease generalization the linear transform weight matrix is expressed as

$$\mathbf{W}_l = \begin{bmatrix} \mathbf{W}_{part1,l} \\ \mathbf{W}_{part2,l} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_Q \mathbf{O}^*_{l-1} \\ \mathbf{W}_{part2,l} \end{bmatrix}, \tag{3.1}$$

as represented in Figure 3.1.

In this thesis is proposed to place deterministic transforms in $\mathbf{W}_{part2,l+1}$. The main advantage is to incorporate low computational complexity achieved by fast transform algorithms while conserving monotonically decreasing cost achieved by keeping LFP. The previous equation from now on is represented as

$$\mathbf{LT}_l(\,\cdot\,) = \begin{bmatrix} \mathbf{V}_Q\mathbf{O}_{l-1}^* \\ \mathbf{w}_{part2,l}(\,\cdot\,) \end{bmatrix}. \tag{3.2}$$

The linear transform $\mathbf{LT}_l(\,\cdot\,)$ and $\mathbf{w}_{part2,l}(\,\cdot\,)$ are represented in function notation instead of matrix, to emphasize that the linear transform is not necessarily implemented as a matrix-vector product with $\mathcal{O}(N^2)$ complexity.

The signal at the output of the linear transform at l'th layer is therefore denoted as

$$\mathbf{z}_l = \begin{bmatrix} \mathbf{z}_{part1,l} \\ \mathbf{z}_{part2,l} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_Q\mathbf{O}_{l-1}^*\mathbf{y}_{l-1} \\ \mathbf{z}_{part2,l} \end{bmatrix}, \tag{3.3}$$

with $\mathbf{y}_{l-1} = g(\mathbf{z}_{l-1})$, where $\mathbf{g}(\,\cdot\,)$ is ReLu activation function. Note that to preserve FLP, ReLu is a necessary condition only in the upper part $\mathbf{z}_{part1,l}$. In consequence it would be possible to use any other non-linear function as activation for the bottom component $\mathbf{z}_{part2,l}$.

In the previous equation (3.3) the bottom term is not expanded. Before expand it, it is necessary to define an intermediate step as

$$\mathbf{z}'_{part2,l} = \mathbf{w}_{part2,l}(\mathbf{y}_{l-1}). \tag{3.4}$$

The reason behind this intermediate step is that pruning nodes based on variance is necessary. In Section 2.1 it has been mentioned that most of the deterministic linear transforms, e.g. DCT, FHT, among others, are squared matrices, and some as Haar transform, FWHT, etc, are also 2-power squared matrices. The concatenation of a DLT output together with the upper part, as in equation (3.3), makes the network width (number of hidden neurons) monotonically increasing as the network gets deeper. To control the width growth a pruning step based on node variance is carried.

The output of a DLT in l'th layer is defined by $\mathbf{z} = \mathbf{z}_{part2,l} = [z_1,\, z_2, ...,\, z_n, ...,\, z_{N_l}]^T$ with N_l being the number of nodes on the layer. The variance of each node n is computed across the training set as

$$\sigma_n^2 = \frac{1}{J}\sum_{j=1}^{J}(z_n^{(j)} - \frac{1}{J}\sum_{k=1}^{J} z_n^{(k)})^2, \quad n = 1, 2, ..., N_l. \tag{3.5}$$

The pruning function $p(\,\cdot\,)$ is defined as follows

$$p(\sigma_n^2) = \begin{cases} \text{Keep node} & \text{if } \sigma_n^2 \geq \eta_{var} \\ \text{Remove node} & \text{if } \sigma_n^2 < \eta_{var} \end{cases}, \tag{3.6}$$

where η_{var} is a predefined threshold to remove the nodes presenting lower variance. According to the concept explained in Subsection 2.2.1, the pruned nodes do not contain relevant information.

After low variance node pruning, a normalization step to arrest energy increase of the signal flow is needed. The norm is computed across the remaining nodes of the linear transform, and each of them is normalized. The bottom part of the l'th layer linear transform is then obtained by the following equation

$$\mathbf{z}_{part2,l} = \frac{p(\mathbf{z}'_{part2,l})}{||p(\mathbf{z}'_{part2,l})||} = \frac{p(w_{LT,l}(\mathbf{y}_{l-1}))}{||p(w_{LT,l}(\mathbf{y}_{l-1}))||}. \tag{3.7}$$

These two operations, node pruning and normalization are critical to achieve a stable behavior for the algorithm performance. Finally equation (3.3) is extended as

$$\mathbf{z}_l = \begin{bmatrix} \mathbf{z}_{part1,l} \\ \mathbf{z}_{part2,l} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_Q \mathbf{O}^*_{l-1} \mathbf{y}_{l-1} \\ \frac{p(w_{LT,l}(\mathbf{y}_{l-1}))}{||p(w_{LT,l}(\mathbf{y}_{l-1}))||} \end{bmatrix}. \tag{3.8}$$

Signal flow defined in equation (2.19) is redefined now to include the linear transform function notation as

$$\mathbf{y}_l = g(\mathbf{LT}_l(\mathbf{y}_{l-1})) = g(\mathbf{LT}_l(...g(\mathbf{LT}_2(g(\mathbf{LT}_1(\mathbf{x})))))) \in \mathbb{R}^{N_l}. \tag{3.9}$$

3.1.2 Linear transform decision algorithm

The proposed learning scheme is based on creating a box, referred onwards as linear transform box, which contains different linear transforms. This linear transform box will be iterated at each layer addition to choose the most suitable transform for the l'th layer. The box is denoted as LT and each linear transform contained as LT_i with $i = 1, 2, ..., T$ and T being the total number of linear transforms in it. The different approaches are presented in the next sections as follows:

1. Supervised approach (Algorithm 3): decision based on the transform minimizing the error between the real target and the estimated.

2. Unsupervised approach (Algorithm 4): decision approach based on statistical signal properties and two different choice methods are proposed.

The network construction is summarized below in Algorithm 2. It is similar to the SSFN original algorithm with an extra iterating procedure to chose the linear transform used in every layer and representing it as $\mathbf{W}_{part2,l}$.

3.2 Supervised Approach with and without Random Instance

An approach based on supervised linear transform selection is presented. The algorithm iterates among all the transforms, LT_i, contained in the linear transform box, estimates the training instances network output $\hat{\mathbf{T}}$ and computes the training cost $\mathcal{C}_l$ with equation (2.21). The chosen transform is the one presenting the lowest cost. The procedure is summarized in algorithm 3.

Algorithm 2 : Proposed algorithm for construction of SSFN

Input:

1: Training dataset $\mathcal{D}(\mathbf{x}^{(j)}, \mathbf{t}^{(j)})\}_{j=1}^{J}$
2: Parameters to set:
   (a) L_{max} (Maximum number of layers)
   (b) $\alpha \geq 1$ (Parameter in (2.22))
   (c) μ and k_{max} (Parameters in ADMM)
   (d) η_{layer} (Stopping thresholds)
   (e) Box of linear transforms: LT_i with $i = 1, 2, ..., T$

Regularized least-squares:

1: $\mathbf{y}_0^{(j)} = \mathbf{x}^{(j)}$
2: Solve (2.23) to find $\mathbf{O}_0^\star$ (Cross-validation for λ_0)

Estimating number of nodes and layers:

1: Initialization: $l = 0$ (Index for l'th layer)
2: **repeat**
3: $l \leftarrow l + 1$ (Increase in layers)
4: Chose the optimal $\mathbf{W}_{part2,l}$ using Algorithm 3 or 4
5: Construct $\mathbf{LT}_l(\,\cdot\,)$ according to (3.3)
6: Compute $\mathbf{y}_{l,i}$ according to (2.19)
7: Solve (2.24) to find $\mathbf{O}_l^\star$ (using ADMM)
8: **until** (2.26) or $l > L_{max}$ (Cost saturation or maximum layers)

Output:

1: Number of layers $L = l$ and number of nodes $\{n_l\}_{l=1}^{L}$
2: Linear transforms $\{\mathbf{LT}_l(\,\cdot\,)\}_{l=1}^{L}$

Algorithm 3 : Supervised approach to compute the optimal linear transform in each layer

Input:

1: $\mathbf{y}_{l-1}$ (Input of l-th layer)
2: η_{var} (Variance threshold)
3: Bag of linear transforms: LT_i with $i = 1, 2, ..., T$

Estimation of the optimal linear transform:

1: **for** $i = 1 : T$ **do**
2: $w_{part2,i} = LT_i$ (Choose i-th LT in the bag)
3: Compute $\mathbf{z}_{part2,l,i}$ according to (3.7)
4: Construct $\mathbf{LT}_l(\,\cdot\,)$ according to (3.3)
5: Compute $\mathbf{y}_{l,i}$ according to (2.19)
6: Solve (2.24) to find $\mathbf{O}_l^\star$ (using ADMM)
7: Compute $\mathcal{C}_l$ according to (2.20) and (2.21)
8: **end for**
9: $w_{part2,l} \leftarrow \min_{LT_i}(\mathcal{C}_l)$ (Choose LT with minimum training cost)

Output:

1: $w_{part2,l}$

One must note that when including the random matrix instance within the linear transform box there is an extra step in Algorithm 3 operation 2. It is to compute the optimal random matrix as done in Algorithm 1, operations 5-10 [2].

3.3 Unsupervised Approaches

In order to perform a legitimate comparison between the proposed learning scheme and the presented in [2] the linear transform decision must be performed following an unsupervised method. This section intends to propose two different methods to choose a suitable deterministic linear transform without making use of the sample targets.

Two different methods are proposed in order to chose a DLT following an unsupervised approach. Both methods analyse the linear transform output, $\mathbf{z}_{part2,l}$ in equation (3.7), and assign a score sc_1 to the transform tested. The score assignment is based on statistical signal properties, presented in Section 2.2. The scores are used to compare the different DLT candidates from the box and choose the most suitable in each layer following its decision criteria defined as

$$w_{part2,l} = \arg\min_{DLT_i}(sc_{1,i}) \qquad \text{with} \qquad i = 1, ..., T, \tag{3.10}$$

which coincide for both methods.

This procedure is summarized in 4.

Algorithm 4 : Unsupervised learning of deterministic transforms

Input:

1: $\mathbf{y}_{l-1}$ (Input of l-th layer)
2: η_{var} (Variance threshold)
3: Bag of deterministic transform: DLT_i with $i = 1, 2, ..., T$

Estimation of a suitable deterministic transform:

1: **for** $i = 1 : T$ **do**
2: $w_{DLT,i} = DLT_i$ (Choose i-th DT in the bag)
3: Compute $\mathbf{z}_{part2,l,i}$ according to (3.7)
4: Apply Method1 (3.3.1) or Method2 (3.3.2)
5: **end for**
6: $w_{part2,l} \leftarrow \arg\min_{DLT_i}(sc_1)$ (Choose DT with min sc_1)

Output:

1: $w_{part2,l}$

3.3.1 Method 1: Nodes standard deviations

Variability in layer nodes give an insight about the amount of information each node handles [35], also mentioned in Subsection 2.2.1. Nodes standard deviation σ_n is computed by square root of equation (3.5). Variability over layer l'th nodes standard deviation is related to how differ the amount of information carried by different nodes. This variability is illustrated in Figure 3.2 and computed as follows

$$\sigma_T = \frac{1}{\sqrt{N}} \sum_{n=1}^{N} (\sigma_n - \frac{1}{N} \sum_{n=1}^{N} \sigma_n). \tag{3.11}$$

The score is set to the previous standard deviation

$$sc_1 = \sigma_T. \tag{3.12}$$

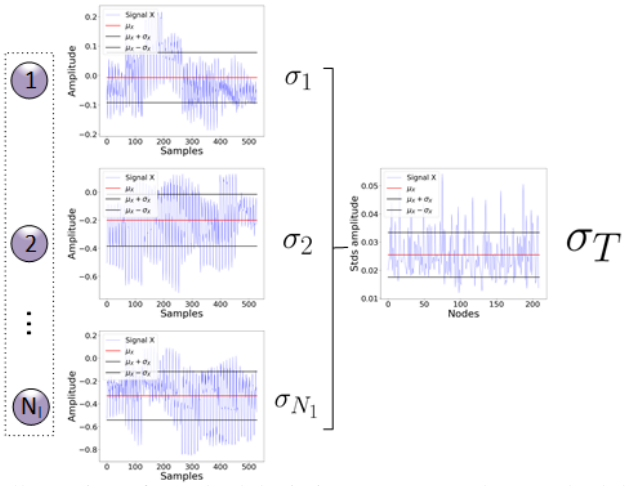


Figure 3.2: Illustration of standard deviation σ_T over nodes standard deviation σ_n.

By choosing the minimum sc_1 among all the transforms, according to (3.10), the transform chosen is the one considered to carry more equally information amount over all the nodes. One must note that nodes carrying a small amount of information do not exist since low variance nodes have been removed previously according to (3.7).

3.3.2 Method 2: Singular values of cross-correlation matrix

In feed-forward neural networks, and basically all neural networks, initial layers matter more [36]. In the network's initial layers the input signal has suffered fewer modifications and therefore the choice of layer weights must be carried more carefully. As the samples go through the network, disturbances and noise is added and also amplified.

This second method intends to choose the transform that is more related to the input signal meaning that it contains a bigger amount of original information. In Subsections 2.2.2 and 2.2.3 the concepts of cross-correlation and SVD have been introduced. Cross-correlation and SVD are computed to measure the similarity between l'th layer output and network input signal.

Consider $\mathbf{x} = \left[\mathbf{x}^{(1)}, ..., \mathbf{x}^{(J)}\right]$ as the network input and the bottom part of the linear transform for simplicity as $\mathbf{z}_{part2,l} = \mathbf{z}_l = \left[\mathbf{z}^{(1)}, ..., \mathbf{z}^{(J)}\right]$. The cross-correlation matrix $\mathbf{R}_{\mathbf{x},\mathbf{z}_l}$ is computed by equation (2.6).

The interest lays in the singular values of the cross-correlation matrix computed by SVD. From the matrix in (2.8) are extracted the singular values λ_k, with $1 \leq k \leq K$ and K the total number of singular values. Singular values represent the sparsity variance of the data in the space. When the data is more concentrated in fewer components means that it is concentrated in fewer dimensions which is convenient. An indicator of the correlation of two signals is that its singular values are more concentrated in fewer components, as illustrated in Figure 3.3.

The singular values are sorted in descending order, meaning from more shared information to less shared information. The cumulative singular value is defined as

$$C_\lambda(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{K} \lambda_i} \qquad \text{where } 0 \leq C_\lambda(k) \leq 1. \tag{3.13}$$

To choose the DLT more related with the input signal is needed the introduction of a threshold for the cumulative singular value. $0 \leq \gamma \leq 1$ is defined as cumulative singular value percentage

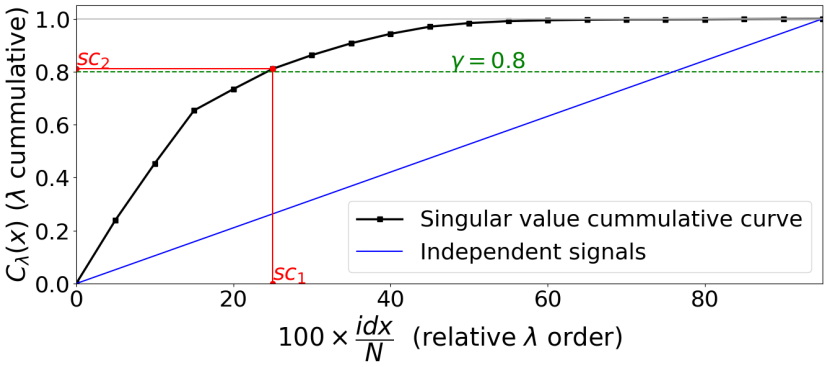


Figure 3.3: Example of correlation eigenvalue curve in black. In red, the first cumulative value higher than the threshold (in green) and its designated scores. In blue, a line corresponding to if the signals were uncorrelated.

threshold. To chose a transform, the position index idx for which the cumulative singular value is higher than the threshold is computed as

$$idx = \arg\min_{k}(C_\lambda(k) \geq \gamma). \tag{3.14}$$

Note as smaller idx is, more related are both signals. Each transform score sc_1 is defined by

$$sc_1 = 100 \times \frac{idx}{K}, \tag{3.15}$$

where the index is normalized by the number of eigenvalues K, to avoid the influence of different outputs length. It might be the case that two transforms have the same index and it is the lower producing a tie. To cope this problem, a new score sc_2 is introduced. This score is defined as the percentage value of the previously chosen idx as

$$sc_2 = C_\lambda(idx). \tag{3.16}$$

In case of a tie in sc_1 a new decision criterion is chosen for this second method as

$$w_{part2,l} = \arg\min_{DLT_i}(sc_{1,i}),\ \arg\max_{DLT_i}(sc_{2,i}) \qquad \text{with} \qquad i = 1, ..., T. \tag{3.17}$$

A visual representation of the method decision criterion is illustrated in Figure 3.3.

To summarize, the interest lays in the transform that presents a greater value of the cumulative singular values in fewer components, meaning that the linear transform output is more correlated to the network input.

Chapter 4

Experimental evaluation

In this chapter, the experimental setup and results are presented. As an introduction, the datasets and deterministic transforms used in the experiments are summarized. It is followed by the experiments carried, which consist in test the different architecture approaches introduced in the previous sections 3.2 and 3.3. The results are presented by representing the obtained accuracies and error, as well as network architectures in each case. Additionally, in the last section is presented an experiment to improve performance on CIFAR10 dataset is performed by combining SSFN with a CNN architecture.

4.1 Datasets

This section presents the datasets used to evaluate the network. All datasets are widely used for multi-class classification, being the same datasets as used in [2] to legitimately compare our results with the presented in the article. In Table 4.1 datasets dimensions are shown, i.e. number of train and test samples, and input $\mathbf{x}$ and output $\mathbf{t}$ dimensions. Additionally, a column has been added to indicate if the dataset partition is constant or randomly chosen at each implementation.

Table 4.1: Used datasets for multi-class classification.

Dataset	Number of train samples	Number of test samples	Input dimension (P)	Number of classes (Q)	Random partition
Vowel	528	462	10	11	No
Satimage	4435	2000	36	6	No
Caltech101	6000	3000	3000	102	Yes
Letter	13333	6667	16	26	Yes
NORB	24300	24300	2048	5	No
Shuttle	43500	14500	9	7	No
MNIST	60000	10000	784	10	No
CIFAR-10	50000	10000	3072	10	No

The datasets are chosen due its popularity in literature, diversity in their signals for a better network performance generalization and level of complexity for tasks. Among the datasets there are speech recognition tasks, in vowel dataset, and image object classification among all other datasets. Following all datasets are defined in more detail:

(a) Vowel: speech recognition task (vowel recognition). Consists of fifteen individual speakers where each says each vowel six times [37].

(b) Satimage: image classification task. Input dimension is 36, corresponding to 3x3 pixels square neighbourhood in 4 different spectral bands [37].

(c) Caltech101: image classification task (different objects with unequal number of samples per object). As implemented in [2], it is used a 3000-dimensional feature vectors suggested in [38].

(d) Letter: image classification task (letter classification). Identify from a black-and-white image one of the 26 capital letters in the English alphabet. Feature details are presented in [37].

(e) NORB: image classification task (3D object recognition from shape) in black-and-white [39].

(f) Shuttle: image classification task. Approximately 80% of the samples belong to the same class, so the goal is to achieve a 99-99.9% accuracy [37].

(g) MNIST: image classification task (handwritten digits classification) [40].

(h) CIFAR-10: image classification task (coloured objects) [41].

4.2 Deterministic Transforms Used

Along the report to refer linear transforms used has been done by a linear transform box. When the box includes a random matrix instance, apart from deterministic transforms is referenced as linear transform box $LT(\cdot)$, and when it does not as deterministic linear transform box $DLT(\cdot)$. Both boxes are illustrated in Figure 4.1.

In this section, the 11 deterministic transforms used in the experimental part are listed. Additional information is included, as the computational cost achieved by its fast algorithms [21] and also its typical applications:

1. Discrete Trigonometric Transforms (Discrete Cosine and Sine Transforms (DCT-II and DST-I)). Both have uses in audio, image and video processing as well as data compression among others. Due to the even symmetry of the DCT, its use is more extended and convenient than DST, which has odd symmetry. Computational complexity $\mathcal{O}(Nlog_2N)$.

2. Fast Walsh-Hadamard Transform (FWHT). Two different coefficient order are used. FWHT1 indicates that the coefficients are in normal Hadamard order while FWHT2 indicates that they are in order of increasing sequence value. FWHT1 is widely used in signal processing tasks such as spectrum analysis, processing speech and medical signals, among others, while FWHT2 is mostly used for control applications [42]. Computational complexity $\mathcal{O}(Nlog_2N)$.

3. Discrete Hartley Transform (DHT). Used in image and optical signal processing and computer vision applications. Computational complexity $\mathcal{O}(Nlog_2N)$.

4. Discrete Wavelet Transforms (DWT). Used in signal coding, image processing and digital communications applications. Computational complexity $\mathcal{O}(N)$. The DWTs used are:

    a) Discrete Haar Transform (Haar)

    b) Daubechies 4 (DB4) and 20 (DB20).

    c) Symlets 2 (sym2) and Coiflets 1 (coif1).

    d) Biorthogonal 1.3 (bior1.3) and Reverse Biorthogonal 1.1 (rbior1.1).

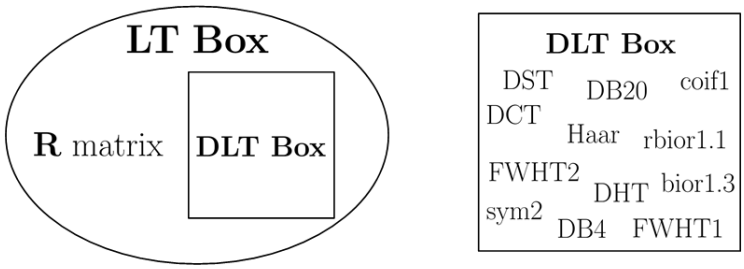


Figure 4.1: Two transforms box used in the experiments carried, LT with a random matrix and DLT with only deterministic transforms.

It is important to mention, for reproducibility, that DWTs output is treated as a concatenation of both approximation and detail coefficients and maximum decomposition level.

A table of computational complexity comparison between matrix-vector multiplication and DLTs fast algorithms is presented in Appendix A.

4.3 Experiment overview

Three different experiments are performed implementing the learning scheme defined in Section 3.1 and the approaches defined in the previous chapter:

1. **Experiment 1, supervised learning approach with random matrix instance** (Subsection 4.4.1). The random matrix instance taken into account is optimized as in Algorithm 1, steps 5 to 10. The resulting matrix is a candidate together with all other deterministic transforms contained in the box.

2. **Experiment 2: supervised learning approach without random matrix instance** (Subection 4.4.2). The transforms proposed to be chosen are only the deterministic transforms.

3. **Experiment 3, Unsupervised learning approach** (Section 4.5). It considers the same DLT transform box as experiment 2 but following an unsupervised approach. Both methods presented in Subsections 3.3.1 and 3.3.2 are tested.

With these experiments, it is intended to show that the use of deterministic transforms is suitable to SSFN architecture and can achieve similar performance to the original SSFN, shown in Appendix B, where state-of-art performances are also stated. Also, a goal is to show that when a supervised approach is carried together with the optimized random matrix, a deterministic transform can be more convenient. The second experiment is pretended to show that by removing the random matrix instance, performances are still valid. Finally, with the last experiment, two methods are proposed to chose the deterministic transform in an unsupervised manner without deteriorating the network performance.

In the following sections, the results are shown for Vowel, Caltech101, Letter and CIFAR-10, for their variability and interpretability. The four remaining dataset results are presented in Appendix D. Also, state-of-art performances

4.4 Supervised approaches

In this section two experiments are performed following the architecture approaches shown in Section 3. The experiments are divided into whether the random matrix is considered within the liner transform box or not.

In Table 4.2 are summarized the train and test performances achieved in both cases. Notice that when not using the random instance (DLT box, right column) there are only two datasets with variations in their accuracy. Both datasets correspond to the random partition datasets (Table 4.1). For the remaining deterministic dataset partitions, the accuracy is constant as the transforms are also. Both approaches' performances are quite similar between them and also to the ones obtained by the original SSFN, in table B.1.

Table 4.2: Supervised SSFN approach classification accuracies across 20 monte-carlo simulations with and without random matrix instance. Hyperparameters are defined in Appendix C.

Dataset	LT Box		DLT Box	
	Accuracy (in %) (avg. $\pm$ std. dev)		Accuracy (in %) (avg. $\pm$ std. dev)	
	Train dataset	Test dataset	Train dataset	Test dataset
Vowel	99.99 ± 0.04	59.91 ± 1.76	99.81	62.99
Satimage	92.13 ± 0.23	89.51 ± 0.36	92.97	89.4
Caltech101	99.95 ± 0.02	76.34 ± 0.81	99.95 ± 0.02	76.34 ± 0.81
Letter	100 ± 0	95.23 ± 0.25	100 ± 0	92.89 ± 0.24
NORB	100 ± 0	87.74 ± 0.4	100	87.56
Shuttle	99.84 ± 0.06	99.8 ± 0.05	99.98	99.92
MNIST	97.82 ± 0.17	96.65 ± 0.13	97.57	96.74
CIFAR-10	70.41	54.88	70.41	54.88

4.4.1 Supervised with random matrix

Network accuracy results are shown in the above table 4.2 left column (LT Box). Following are summarized the architectures for Vowel, Caltech101, Letter and CIFAR10 datasets, the remaining dataset results graphs are displayed in Appendix D.1.

Vowel, Caltech101 and Letter present an accuracy with variations across monte-carlo simulations. In table 4.3 is shown the average of network layers for each dataset, as it also varies across simulations. In figures 4.2 and 4.3 are shown the network architectures of each dataset across all the simulations. Network architectures are summarized with two figures for each dataset, a figure in the left illustrating the LT chosen at each layer and a figure in the right showing the number of nodes at each layer for all simulations. The last or two last layers of the network may not belong to all 20 MC simulations as the network may be constructed with fewer layers.

Table 4.3: Network layers average across MC simulations for supervised approach with random matrix.

Dataset	Layers avg.
Vowel	14.6
Caltech101	4
Letter	13.35
CIFAR-10	4

In the figure below, for Vowel dataset, the random instance matrix is observed to be the most chosen transform except in layers 7, 9, 11 and 12. When the chosen transform is not the random instance, it presents a wide variability among the other transforms. Regarding the node distribution, it presents a wide deviation from the mean, meaning that the network size is not consistent across different simulations.

Caltech101 dataset presents a special scenario where all the chosen transforms remain equal across all simulations and also being the four of them deterministic transforms. In consequence, the node distribution remains almost constant with small variability.

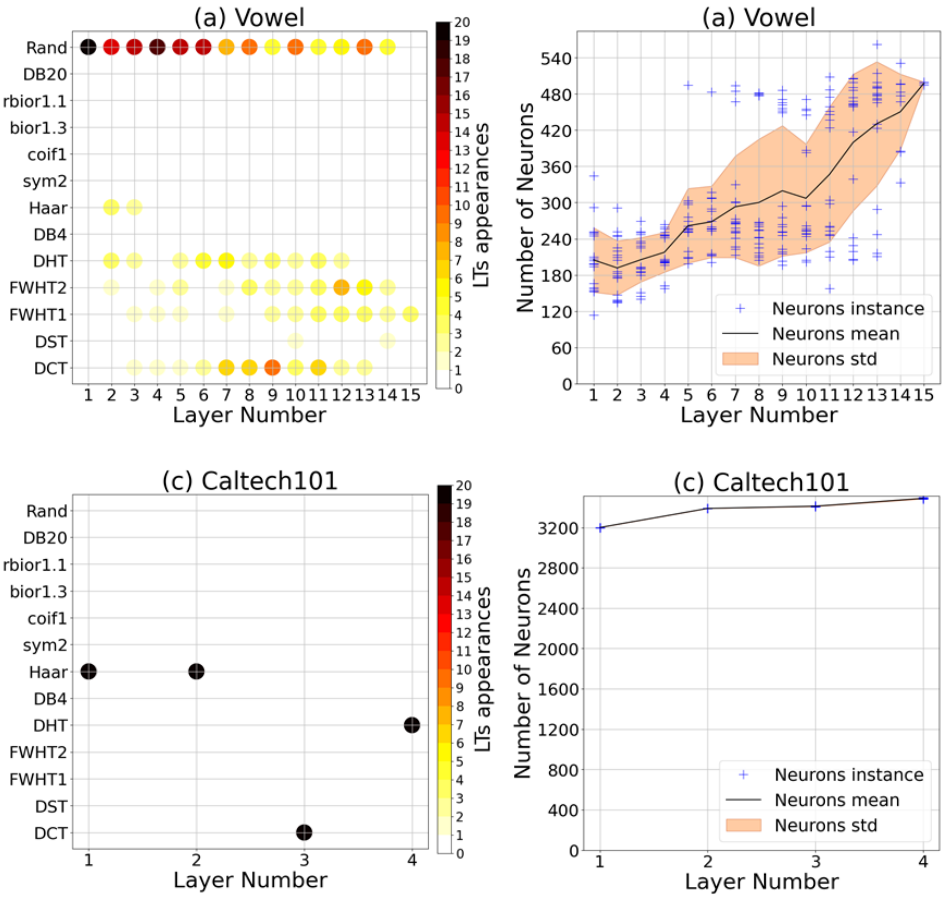


Figure 4.2: Network architecture for Vowel and Caltech101 datasets implementing supervised approach with random matrix.

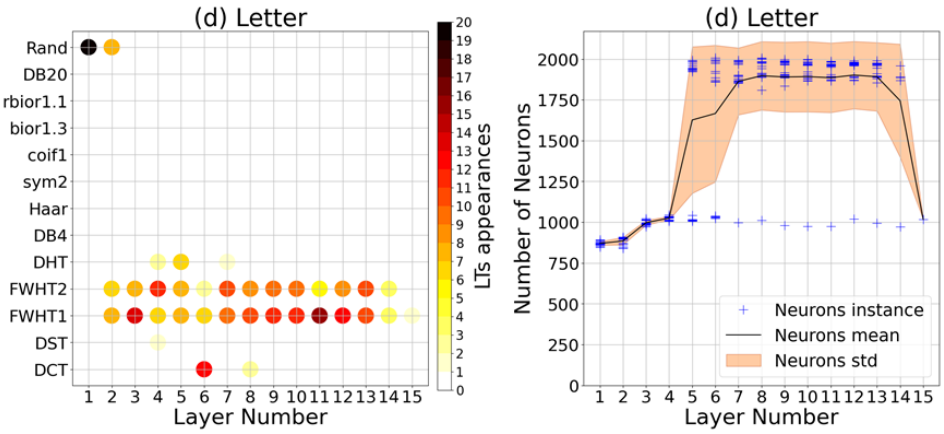


Figure 4.3: Network architecture for Letter dataset implementing supervised approach with random matrix.

Letter dataset presents an opposite scenario compared to Vowel. The random instance is always chosen in the first layer and sometimes in the second layer, but for the remaining layers, only deterministic transforms are chosen. Chosen DTs present a consistency among MC simulations where FWHT1 and FWHT2 predominate, being both closely related. Nodes are closely distributed across all executions but one, presenting a constant network architecture for this dataset.

CIFAR10 is totally a different case as in table 4.2 presents a constant accuracy. Similar to Caltech101, it always presents the same chosen transforms being all of them DLTs. CIFAR10 is also a constant dataset, with random partition set to "No", table 4.1. These both facts make all MC simulations to be equal. Table 4.4 shows the CIFAR-10 network architecture and following in figure 4.4 are presented the accuracy and NME curves across the number of layers. Although the accuracy achieved is low compared to the state-of-art, it is around 17% greater on training and 8% on testing than the original SSFN, table B.1.

Table 4.4: Network architecture applying supervised approach with random matrix for CIFAR10 dataset.

Dataset	Nodes arrangement (Deterministic Transform layout)	Accuracy
CIFAR-10	3799-3047-2502-3176 (FWHT1-FWHT2-FHT-FWHT2)	54.88

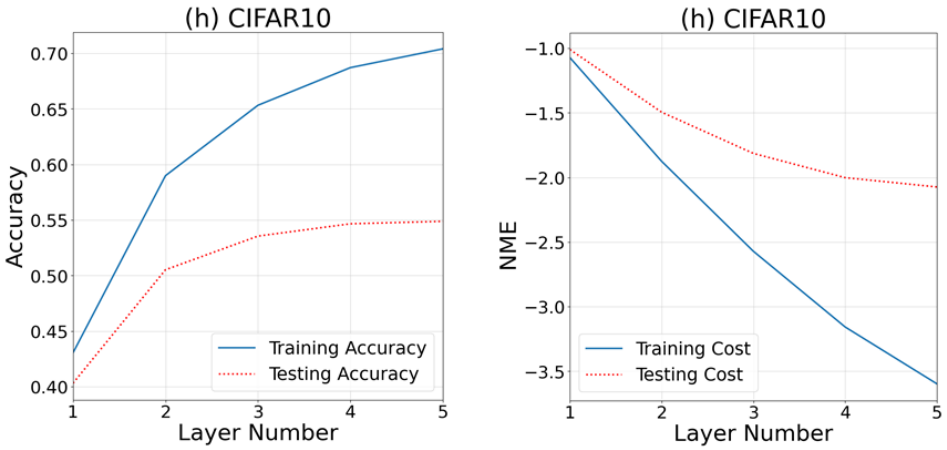


Figure 4.4: Accuracy and NME curves on every layer for CIFAR10 dataset implementing supervised approach with random matrix.

The implementation of this experiment shows that the use of a deterministic transform may be preferable in terms of cost than a random matrix, without taking into account the lower computational cost. This supervised learning approach combines the selection of random matrices with deterministic matrices. Additionally, in some cases, deterministic transforms are the only ones chosen.

4.4.2 Supervised without random matrix

Accuracy results are shown in the left column, DLT Box, in table 4.2. In this approach it is observed that all the presented accuracies, except for Caltech101 and Letter datasets, are constant, both corresponding to the datasets with a "Yes" in the random partition in table 4.1. As mentioned before, constant partition datasets together with the use of deterministic transforms result always in the same network architecture. As could be deduced from the previous case, for CIFAR10 coincides with the previous case as in any layer the random instance was chosen.

In table 4.5 are shown the different architectures for the different datasets and simulations. Compared to the previous experiment and the original SSFN in [2], the network tends to get deep and wide in most datasets, as Caltech101, Letter and NORB. The cause is the dependence of most DLTs to be squared matrix, increasing the network size in every layer. Checking table A.1 it is seen that even an increase to the size of $2^{11} = 2048$ neurons computed by a fast algorithm, requires fewer operations than $2^8 = 256$ neurons computed by the typical matrix operation.

Table 4.5: Monte Carlo trials of SSFN with supervised deterministic transforms decision.

Dataset	Nodes arrangement (Deterministic Transform layout)	Accuracy
Vowel	136-137-240-245-256-263-497-491-470-456 (DB20-DST-FWHT1-DCT-FWHT1-DHT-FWHT1-DHT-FWHT1-DCT)	62.99
Satimage	222-186-236 (DB20-DHT-FWHT1)	89.85
Caltech101	3200-3389-3415-3492 (Haar-Haar-DCT-DHT)	76.34
	3200-3389-3416-3502 (Haar-Haar-DCT-DHT)	77.03
	3200-3389-3419-3492 (Haar-Haar-DCT-DHT)	76.27
Letter	224-454-468-537-1028-1991-1925-1883-1993-1985-1963-1919-1882-1978-1960 (DB20-DB20-DCT-FWHT1-FWHT1-FWHT2-DCT-FWHT1-FWHT1-FWHT1-FWHT1-FWHT2-FWHT1-FWHT1-FWHT1)	93.11
	224-454-467-539-1029-2002-1929-1870-1917-1984-1955-1980-1938-1961-1974 (DB20-DB20-DCT-FWHT1-FWHT1-FWHT2-DCT-FWHT2-FWHT1-FWHT1-FWHT1-FWHT1-FWHT1-FWHT1-FWHT2)	93.01
	224-454-467-540-1010-1038-1923-1848-1883-1992-1896-1991-1979-1957-1975-1967 (DB20-DB20-DCT-FWHT2-FWHT1-DHT-FWHT2-DCT-FWHT1-FWHT2-FWHT1-FWHT1-FWHT1-FWHT1-FWHT1-FWHT2)	93.1
NORB	4041-3751-3310-2743-2153-3231-2541-2011 (FWHT2-DCT-DCT-DCT-DCT-FWHT1-DCT-DHT)	87.56
Shuttle	140-270-490-438 (DB20-FWHT1-FWHT1-DHT)	99.92
MNIST	979-976-1025-1882 (DB20-FHT-FWHT1-FWHT2)	96.74
CIFAR-10	3799-3047-2502-3176 (FWHT1-FWHT2-FHT-FWHT2)	54.88

In the following figure 4.5 are shown the accuracy and NME curves for Vowel dataset. Comparing the architecture shown in table 4.5 with the presented in figure 4.2 for Vowel, it presents a similar transform and nodes pattern.

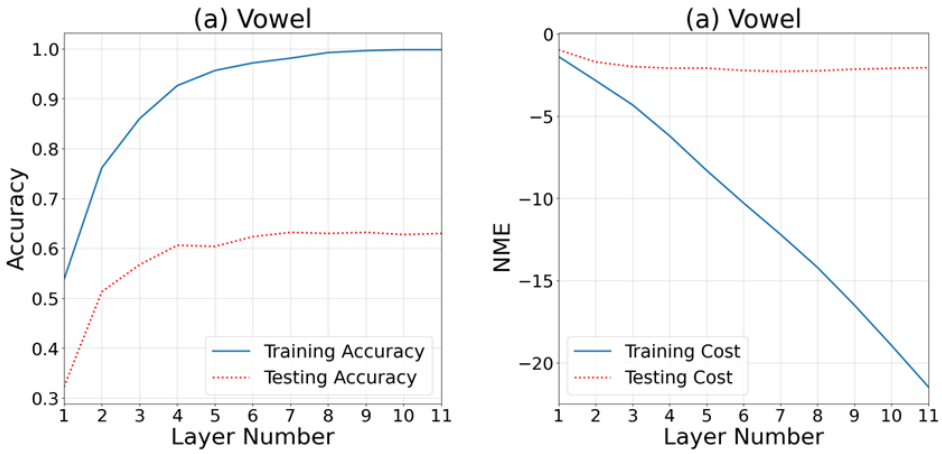


Figure 4.5: Accuracy and NME curves on every layer for Vowel dataset implementing supervised approach without random matrix.

For the random datasets, Caltech101 and Letter, the network architectures are still different across simulations. Similar to CIFAR10, in the previous case the chosen transforms for Caltech101 were only deterministic what makes equal both approaches. Caltech101 built architecture corresponds to the one presented in figure 4.2. For Letter dataset, the newly built architectures are shown below, with layers average of 15.3. The results are similar to the previous case, with a similarity of changing the random instance for DB20 transform. The deterministic transforms chosen are more constant and the number of nodes also presents a lower variance.

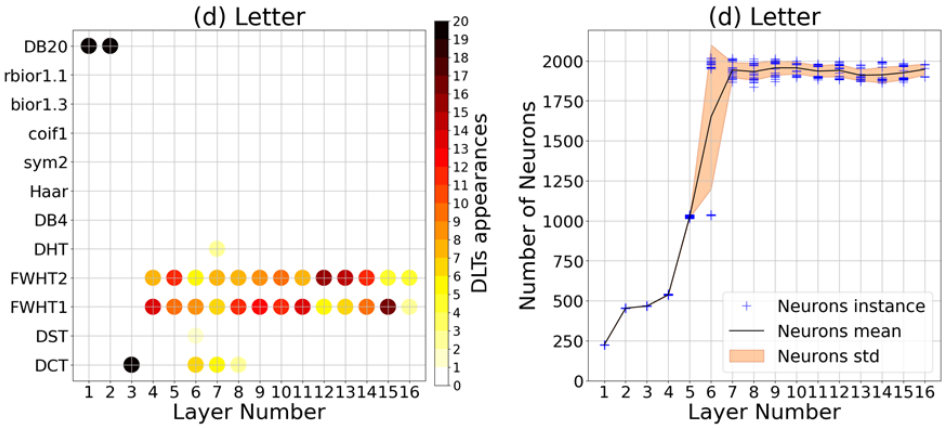


Figure 4.6: Network architecture for Letter dataset implementing supervised approach without random matrix.

The results show that in a supervised manner the use of random instances can be replaced by using deterministic transforms, as the presented results show a similar performance regarding the accuracy and similar network architectures.

4.5 Unsupervised approaches

In this experiment SSFN unsupervised approach presented in section 3.3 is implemented and both methods defined in subsections 3.3.1 and 3.3.2 are tested. The performances achieved are shown in the table below. As in the previous unsupervised without random matrix case, only Caltech101 and Letter datasets present variations on the performances for every simulation as well as network architecture. Both datasets present low standard deviation, meaning consistent classification performance and architectures. Also, the networks present a constant number of layers, in table 4.7, and moreover, both methods present the same number of layers for Caltech101. Also, similar accuracies are presented by both methods and also close to both presented implementing supervised approaches in table 4.2 and therefore to the original SSFN performance.

Table 4.6: SSFN Classification accuracy of unsupervised LT selection across 50 Monte-Carlo simulations, applying method 1 and 2. Hyperparameters set are defined in Annex C.

Dataset	Method 1		Method 2	
	Accuracy (in %) (avg. $\pm$ std. dev)		Accuracy (in %) (avg. $\pm$ std. dev)	
	Train dataset	Test dataset	Train dataset	Test dataset
Vowel	99.62	64.72	93.75	63.42
Satimage	91.41	89.15	93.62	89.15
Caltech101	99.95 ± 0.02	76.73 ± 0.82	99.93 ± 0.02	76.39 ± 0.67
Letter	100 ± 0	92.72 ± 0.3	95.43 ± 0.12	91.07 ± 0.42
NORB	98.22	84.75	100	87.46
Shuttle	99.8	99.76	99.96	99.9
MNIST	97.32	96.54	98.09	96.9
CIFAR-10	61.12	53.79	65.66	54.12

In the two following subsections are better presented the resulting network architectures for both methods and graphical representations for the same four datasets as previously. It is observed that the DLT selection between both methods is different, although the pattern of the transforms chosen is similar.

Table 4.7: Network layers average across MC simulations for variable datasets for unsupervised approaches.

Dataset	Layers avg.	
	Method1	Method2
Caltech101	4	4
Letter	12	6

4.5.1 Method 1 implementation

Accuracy results are shown in the left column, in table 4.6. In the following table are shown the network architectures, represented by the number of nodes and DLTs chosen at each layer, for each dataset together with the achieved performance.

The constructed networks for all datasets are similar to the previous supervised approach in table 4.5, both for node distribution and transforms. Both depth and width are similar to the previous case. These results show the validity of the method to chose the different layers transforms in an unsupervised manner. As in the previous case, the network gets very wide for most of the datasets, such as Caltech101, Letter and MNIST but it also gets very deep for Letter dataset. Nevertheless, it is observed for NORB dataset the size has been drastically reduced while keeping the performance.

Table 4.8: Monte Carlo trials of SSFN with deterministic transforms applying Method1.

Dataset	Nodes arrangement (Deterministic Transform layout)	Accuracy
Vowel	136-355-426-487-475-491-492 (DB20-DB20-FWHT1-FWHT1-DCT-FWHT2-FWHT1)	64.72
Satimage	222-215-263 (DB20-DB20-DB20)	89.15
Caltech101	4096-3543-3702-3854 (FWHT2-DHT-FWHT1-FWHT1)	76.83
	4096-3552-3757-3873 (FWHT2-DHT-FWHT1-FWHT1)	76.23
	4096-3552-3756-3856 (FWHT2-DHT-FWHT1-FWHT1)	76.53
Letter	224-269-521-1028-1998-2002-2011-1988-1979-1885-1987-1901 (DB20-FWHT1-FWHT1-FWHT1-FWHT1-FWHT1-FWHT1-FWHT1-FWHT2-FWHT1-FWHT2-FWHT1)	92.6
	224-268-520-1014-1024-1010-1044-1926-1989-1974-1973-1981 (DB20-FWHT2-FWHT1-FWHT1-DCT-FWHT1-FWHT2-FWHT1-FWHT1-FWHT1-FWHT1-FWHT1)	92.16
	224-268-520-1019-1032-2008-1966-2016-1980-1901-1987-1889 (DB20-FWHT1-FWHT1-FWHT1-FWHT1-FWHT2-FWHT1-FWHT1-FWHT2-FWHT1-FWHT2-FWHT1)	92.42
NORB	997-939-331 (rbio1.1-FWHT2-DB20)	84.57
Shuttle	140-317-233 (DB20-DB20-DB20)	99.76
MNIST	1044-2038-1956 (FWHT1-FWHT1-FWHT2)	96.54
CIFAR-10	2875-2700-3271-3324 (DB20-FWHT1-FWHT1-FWHT1)	53.79

The four figures below show the accuracy and NME curves versus the number of layers for Vowel and CIFAR10 datasets. Both cases show a non-saturation trend for the training curve and similar behavior to those achieved when implementing supervised approaches.

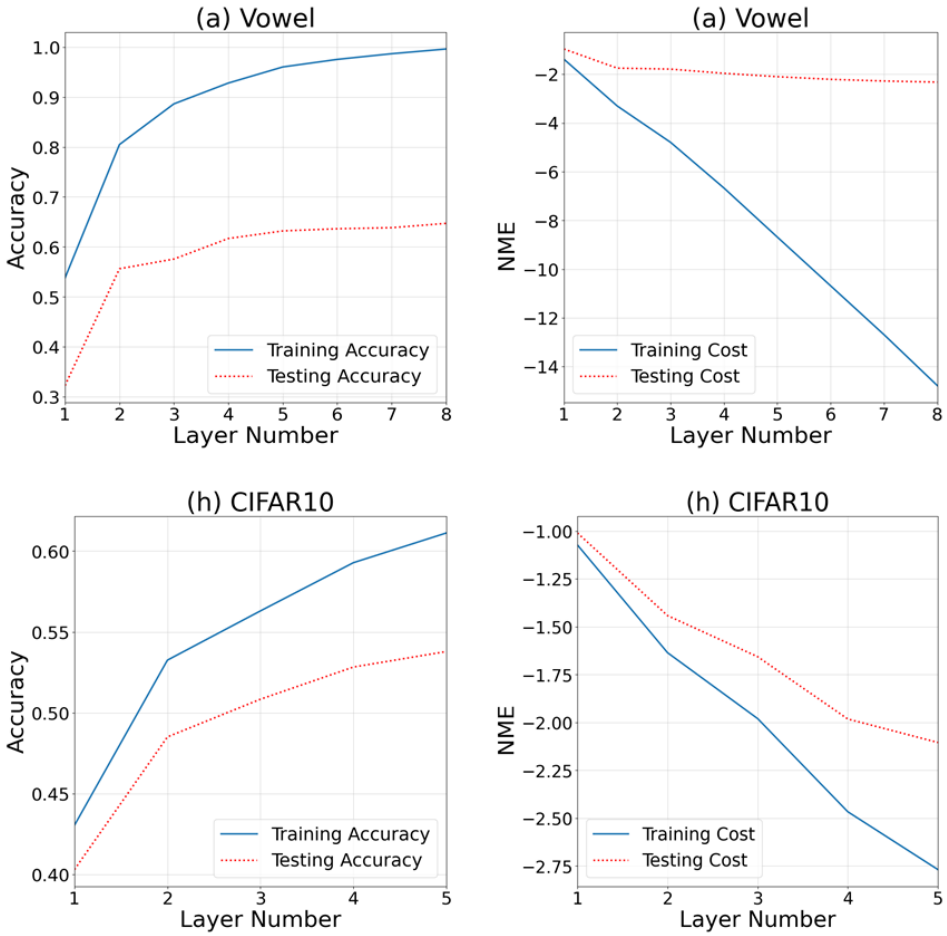


Figure 4.7: Accuracy and NME curves on every layer for Vowel and CIFAR10 datasets implementing unsupervised approach Method1.

The four figures below show the network architecture for the two variable datasets. On both of them, consistency when choosing the DTs is observed, close to the supervised without random transform approach. The node distribution deviation is small for Caltech101 while for Letter looks bigger, although this big deviation comes from some simulation outlier as the main executions lay close with a similar node distribution.

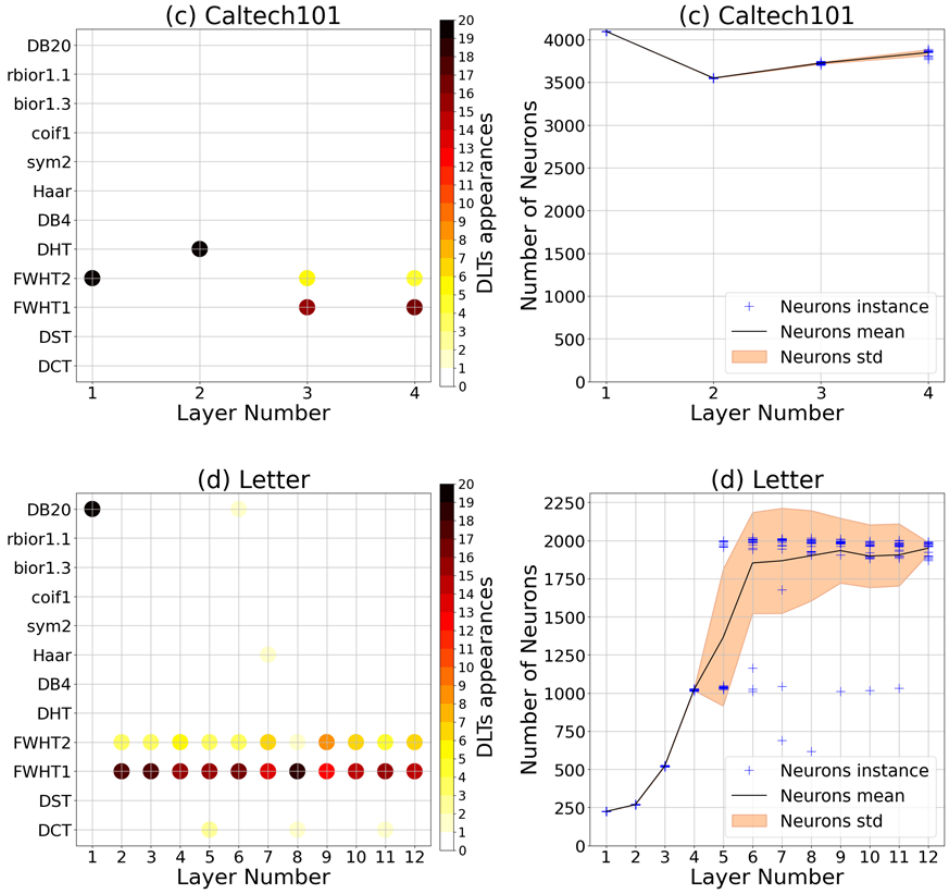


Figure 4.8: Network architecture for Caltech101 and Letter datasets implementing unsupervised approach Method1.

This first method to chose a DLT based on nodes' standard deviation has shown to achieve performance and network architectures similar to those achieved by the supervised approach when the training cost is minimized.

4.5.2 Method 2 implementation

Accuracy results are shown in the right column, in table 4.6. As in the previous method, the following table shows the network architecture for all datasets along with different executions.

Constructed networks have a similar accuracy than with the previous method and similar network architecture. It can be noticed that for some datasets the network is built smaller, as in-depth for Letter dataset and width for Caltech101 and MNIST. Nevertheless, the architecture achieved by Method1 for NORB dataset is not preserved and the size is increased, similar to the one achieved as in the unsupervised cases.

Table 4.9: Monte Carlo trials of SSFN with deterministic transforms applying Method2.

Dataset	Nodes arrangement (Deterministic Transform layout)	Accuracy
Vowel	136-355-370-332-324 (DB20-DB20-DB20-DB20-DB20)	63.42
Satimage	222-210-299-444 (DB20-FWHT2-DB20-FWHT2)	89.15
Caltech101	4096-558-1108-2170 (FWHT2-DB20-FWHT1-FWHT2)	76.4
	4096-557-1110-2162 (FWHT2-DB20-FWHT2-FWHT1)	76.5
	4096-556-1107-2157 (FWHT2-DB20-FWT2-FWHT1)	76.16
Letter	224-454-441-451-458-464 (DB20-DB20-DB20-DB20-DB20-DB20)	91.78
	224-454-441-452-457-469 (DB20-DB20-DB20-DB20-DB20-DB20)	91.11
	224-454-442-454-455-464 (DB20-DB20-DB20-DB20-DB20-DB20)	91.39
NORB	4041-3814-3710-3572-3272-3188 (FWHT2-DCT-FWHT2-FWHT1-FWHT2-FWHT2)	87.46
Shuttle	140-317-318-447 (DB20-DB20-DST-FWHT1)	99.9
MNIST	1044-2038-1998-1931-1789 (FWHT1-FWHT2-DST-DST-FWHT2)	96.9
CIFAR-10	3102-2924-3270-3181 (bior1.3-FWHT1-FWHT2-FWHT2)	54.12

The following two figures show the accuracy and NME curves for Vowel and CIFAR10 datasets as for the previous cases. All curves are similar to the supervised and unsupervised with Method1 cases.

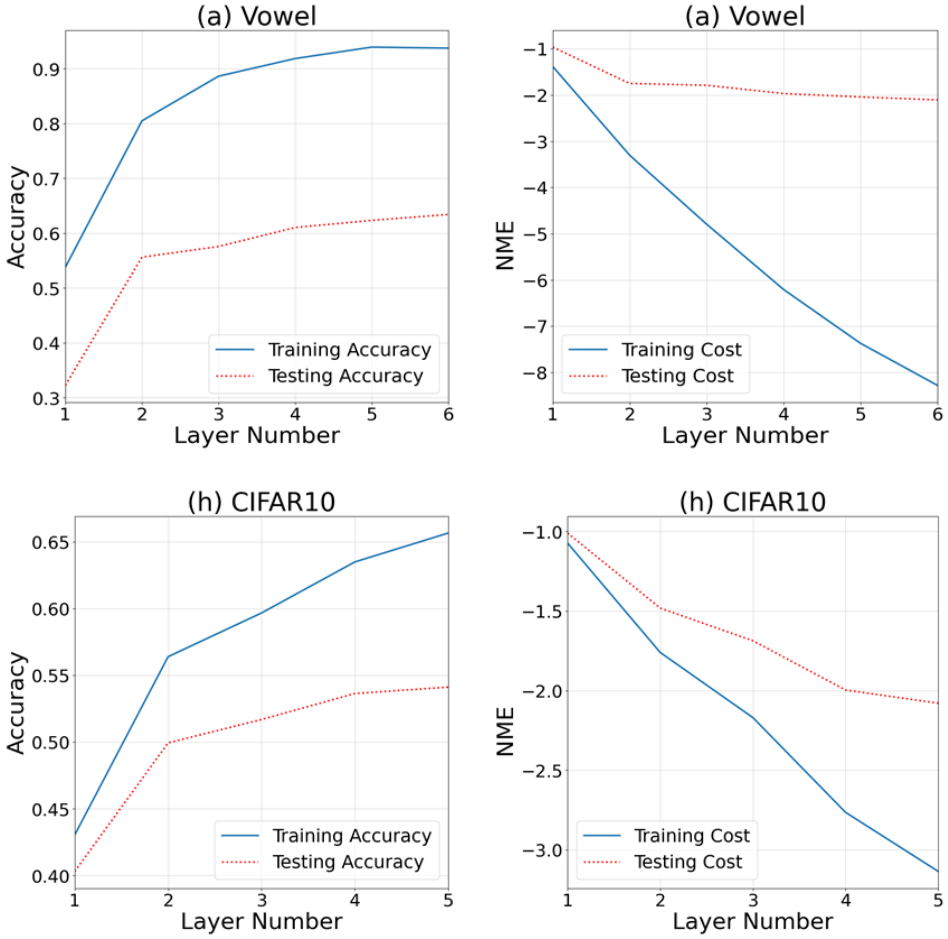


Figure 4.9: Accuracy and NME curves on every layer for Vowel and CIFAR10 datasets implementing unsupervised approach Method2.

The figures below show the network architecture for the two random datasets. The results now differ more from the previous approach. For Caltech101 the DLT distribution coincides but differing in the second layer, where instead of DHT transform, DB20 is used. This change provokes a drastic reduction in the number of neurons from the second layer onwards compared to that got applying Method1. For Letter dataset, the difference is absolute. Applying Method1, the number of layers is around 12, being the FWHT1 and FWHT2 the chosen transforms except DB20 in the first layer. In this case, the number of layers is half than before and the number of nodes at each layer has drastically decayed from around 2000 to around 450 nodes.

The results show the correct operation of both methods. Both provide similar results despite referring to performance even if the network architecture differ. For some datasets, the number of nodes decays drastically while conserving the performance. Is important to mention that the variance threshold was set to $\eta_{var} = 10^{-7}$ for all datasets and both methods. Correct tuning of this parameter for each dataset and method would probably improve the results by reducing the

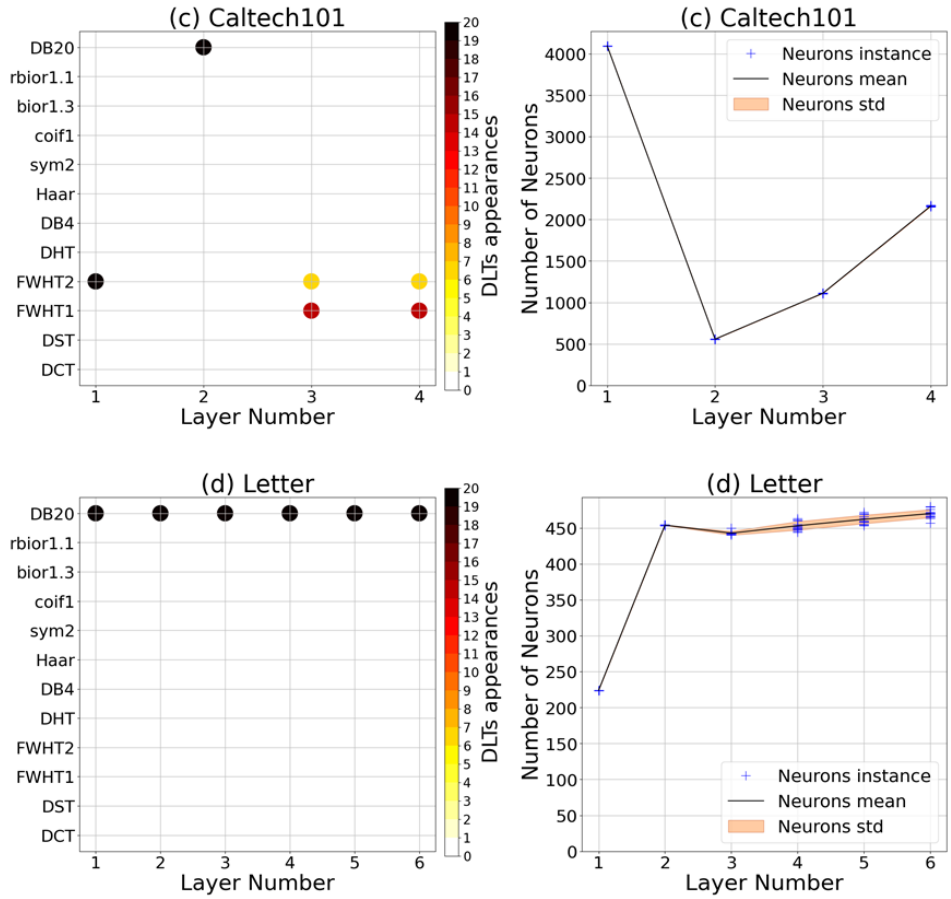


Figure 4.10: Network architecture for Caltech101 and Letter datasets implementing unsupervised approach Method2.

number of nodes on some occasions. The results for the remaining four datasets are presented in Appendix D sections D.3 and D.4.

4.6 CIFAR-10 performance improvement

SSFN has shown to be unable to compete with state-of-art performance for CIFAR10 [2] [43]. Despite the original performance has been slightly improved by the proposed approaches, it is still far from the state-of-art. A work presented in [44] exposes the non-triviality to generalize CIFAR-10 performance, although CNNs have shown to work properly for image classification tasks [45].

In [2] an ad-hoq adaptation is generated. SSFN is a flexible architecture that can use the output of a successfully previously trained network as an output matrix. The implemented adaptation consists of training a CNN on CIFAR10, achieving good performance and use it as the first layer output matrix $\mathbf{O}_0^*$. This so-called SSFN hybrid approach results in increasing SSFN performance on CIFAR-10 but without showing improvement concerning the trained CNN.

In this thesis the approaches are performed to the bottom part of the linear transform, $\mathbf{W}_{part2,l}$. In Section 3.1 it has been mentioned that this transform could be whatever. A new experiment is performed by placing a CNN instead of a DLT in the bottom part aiming to improve performance on CIFAR10 and also study if is better than applying it to LT the top part as in [2]. Additional network layers are build as an unsupervised approach applying Method2. The resulting hybrid-architecture is shown in Figure 4.11.

The implemented CNN architecture is the following:

$Conv(32\,@\,3x3) \rightarrow Conv(32\,@\,3x3) \rightarrow MaxPooling(2x2) \rightarrow Dropout(0.1) \rightarrow Conv(64\,@\,3x3) \rightarrow$
$Conv(64\,@\,3x3) \rightarrow MaxPooling(2x2) \rightarrow$
$Conv(128\,@\,3x3) \rightarrow Conv(128\,@\,3x3) \rightarrow MaxPooling(2x2) \rightarrow$
$Flatten \rightarrow Dense(128) \rightarrow Output(10, Softmax)$,

with the convolutional layers expressed as $Conv(nFilters\,@\,FilterSize)$.

The CNN has been trained for 50 epochs with a batch size of 64, achieving an accuracy of 74.45%, close to the 75.34% used in [2]. In figure 4.12 is shown the accuracy curves for the trained CNN.

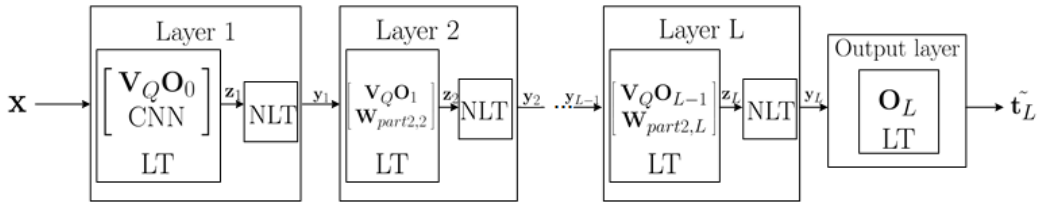


Figure 4.11: Hybrid SSFN architecture where the bottom part of the first linear transform is a CNN.

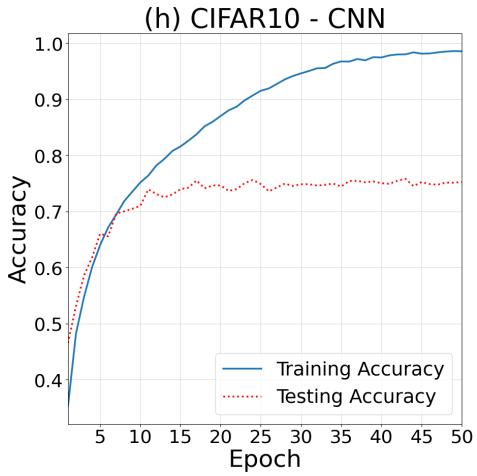


Figure 4.12: Train and test accuracy curves for CNN and CIFAR10 dataset.

The resulting performance for the hybrid SSFN is 75.53%, 1.08% better than CNN itself. This small accuracy improvement is similar to the achieved in [2]. Nevertheless, it shows that an alternative implementation in CNN is possible, although computational expensive. Below, are shown both accuracy and NME curves for the hybrid SSFN. Layer 1 corresponds to the output matrix $\mathbf{O}_0^*$ computed by RLS. CNN effect is reflected from the second layer to onwards represented by a jump in both. The network transforms are: CNN-DB20-FWHT1-DB20-FWHT1-DB20-FWHT1-DB20-FWHT1-DB20.

As mentioned, despite the failure on improving the accuracy with respect achieved by CNN itself, this experiment offers opportunities to improve the performance with SSFN that could be explored.

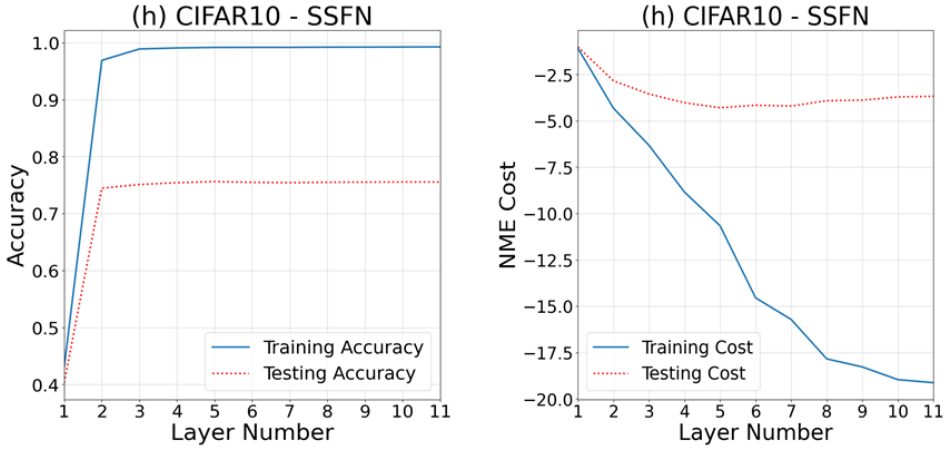


Figure 4.13: Accuracy and NME curves on every layer for CIFAR10 dataset implementing hybrid SSFN with a CNN at the first layer bottom LT.

Chapter 5

Discussion and Conclusions

In this chapter implemented methods and experiments limitations are discussed, further research and future works are proposed and conclusions are drawn.

5.1 Discussion

In the upraising AI technologies, the low complexity of developed algorithms is becoming a must for different real-time and online applications and low-cost hardware. Reduction of algorithms complexity has two main advantages, low latency operation, and energy consumption reduction.

In the project has been evaluated the validity of using deterministic transforms in the linear part of a neural network. Deterministic transforms can be computed with fast algorithms which by definition are more efficient and less complex than matrix multiplication forms. Replacing matrix operations within a neural network by deterministic transforms produces a reduction of neural network computational expenses.

The neural networks that easily allow performing the experimentation belong to a type of DNNs which use a random matrix instance. This type of neural networks already presents low complexity. SSFN is a neural network architecture belonging to this group and has been the chosen architecture to experiment on. In the previous chapters have been proposed two different methods to implement deterministic transforms on SSFN and how to decide a suitable transform among a list of transforms. The two methods are based on signal properties, like standard deviation and cross-correlation matrix respectively, which provide sufficient information about the signal. After the experimental evaluation, the results are presented in order of the four approaches proposed.

The obtained results show the evolution from the original SSFN, passing through supervised approaches and combining DLTs with random instances, to unsupervised approaches using only DLTs. The presented results are satisfactory showing consistency and stability behavior both on performance and network architecture. It has been mentioned that some of the hyperparameters could have been tuned more adequately for each case improving the presented results. The thesis, therefore, shows the suitability of deterministic transforms within neural networks and can be used as an initial basis for its implementation in the field.

5.2 Future work

The work proposed in the thesis provides proof and a first step of the validity of implementing deterministic transforms within neural networks. It is a recent and non explored topic and therefore many doors may be opened for further research. Following are proposed some future works to continue the presented work:

- *Methods and approaches implementation into other NN architectures*. Along with the project SSFN has been the only neural network used to test the proposed methods. Its generalization to other neural network architectures belonging to the same group, such as ELM, HELM and RVFL, would be of great general interest and generate further interest on the thesis topic.
- *Implementation and experimental study of the complexity of a neural network implementing deterministic transforms instead of matrix weights*: As mentioned before, this project has been presented theoretically a comparison between the matrix-vector multiplication and fast algorithms complexity. As future steps, it would be important to show experimentally the complexity comparison with a suitable and optimized software.
- *Suitability of different DTs*: Along with the project the same DTs have been implemented. Its selection was not based on any criteria other than deterministic transforms' popularity. Many DTs are used for different purposes and on different occasions. Further investigation on transforms to be used and depending on the dataset may be essential.
- *Interpretability of NNs flow*: Since its beginning has been seen as a black box without transparency due to its non-linearity [46]. Deterministic transforms are usually implemented depending on a purpose, and based on signal properties. The decision to choose a deterministic transform preferred to another, may give an insight about the signal flow through the network and therefore a bit of interpretability to the network flow.

5.3 Conclusion

In this thesis a non-explored topic has been set under research, the use of deterministic transforms in the weight matrices of a multilayer neural network. Along with the research the focus has been set under SSFN, a specific neural network architecture. The achieved results prove the possibility to implement deterministic transforms within neural networks and theoretically presenting a computational cost reduction. Although many open issues and future works remain, the results are satisfactory and introduce a topic to consider to research within neural networks. An article based on this thesis has been written [47]. Its acceptance in EUSIPCO 2021 (European Signal Processing Conference, held in Dublin - Ireland) proved the relevance and interest to research of the presented project. In Appendix E there are the acceptance and scheduling emails of the proposed article in the conference.

Bibliography

[1] S. Chatterjee, A. M. Javid, M. Sadeghi, P. P. Mitra, and M. Skoglund, "Progressive learning for systematic design of large neural networks," *arXiv preprint arXiv:1710.08177*, 2017.

[2] S. Chatterjee, A. M. Javid, M. Sadeghi, S. Kikuta, D. Liu, P. P. Mitra, and M. Skoglund, "SSFN – self size-estimating feed-forward network with low complexity, limited need for human intervention, and consistent behaviour across trials," *ArXiv e-prints*, 2020.

[3] F. Rosenblatt, *The Perceptron, a Perceiving and Recognizing Automaton Project Para*, ser. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957.

[4] A. Ivakhnenko, A. Ivakhnenko, V. Lapa, V. Lapa, and R. McDonough, *Cybernetics and Forecasting Techniques*, ser. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1967.

[5] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," pp. 2553–2561, 2013.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *Intl. J. Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015.

[8] S. Dodge and L. Karam, "A study and comparison of human and deep learning recognition performance under visual distortions," pp. 1–7, 2017.

[9] R. Wason, "Deep learning: Evolution and expansion," *Cognitive Systems Research*, vol. 52, pp. 701–708, 2018.

[10] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

[11] A. Kouris, S. I. Venieris, M. Rizakis, and C.-S. Bouganis, "Approximate lstms for time-constrained inference: Enabling fast reaction in self-driving cars," 2019.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," pp. 770–778, 2016.

[13] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *ArXiv e-prints*, 2020.

[14] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," *ArXiv e-prints*, 2016.

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv e-prints*, 2017.

[16] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv e-prints*, 2017.

[17] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *J. Trans. Sys. Man Cyber. Part B*, vol. 42, no. 2, pp. 513–529, Apr. 2012.

[18] Y. Pao and Y. Takefuji, "Functional-link net computing: theory, system architecture, and functionalities," *Computer*, vol. 25, no. 5, pp. 76–79, 1992.

[19] R. Katuwal, P. N. Suganthan, and M. Tanveer, "Random vector functional link neural network based ensemble deep learning," *ArXiv e-prints*, 2019.

[20] H. Ramchoun, M. Amine, M. A. Janati Idrissi, Y. Ghanou, and M. Ettaouil, "Multilayer perceptron: Architecture optimization and training," *International Journal of Interactive Multimedia and Artificial Inteligence*, vol. 4, pp. 26–30, 01 2016.

[21] R. Wang, *Introduction to Orthogonal Transforms: With Applications in Data Processing and Analysis*. Cambridge University Press, 2012.

[22] M. Stéphane, *CHAPTER 7 - Wavelet Bases*, third edition ed., M. Stéphane, Ed. Boston: Academic Press, 2009.

[23] F. Moreau de Saint-Martin, P. Siohan, and A. Cohen, "Biorthogonal filterbanks and energy preservation property in image compression," *IEEE Transactions on Image Processing*, vol. 8, no. 2, pp. 168–178, 1999.

[24] N. U. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Signal Processing*. Berlin, Heidelberg: Springer-Verlag, 1975.

[25] A. N. Akansu and R. A. Haddad, *Chapter 6 - Wavelet Transform*, A. N. Akansu and R. A. Haddad, Eds. San Diego: Academic Press, 2001.

[26] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd ed. USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[27] Z. Boger and H. Guterman, *Knowledge extraction from artificial neural network models*, 1997, vol. 4.

[28] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[29] M. Valueva, N. Nagornov, P. Lyakhov, G. Valuev, and N. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation," *Mathematics and Computers in Simulation*, vol. 177, pp. 232–243, 2020.

[30] Guang-Bin Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *IEEE Transactions on Neural Networks*, vol. 9, no. 1, pp. 224–229, 1998.

[31] R. Rojas, *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer-Verlag, 1996.

[32] M. Gori and A. Tesi, "On the problem of local minima in backpropagation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 1, pp. 76–86, 1992.

[33] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006, neural Networks.

[34] J. Tang, C. Deng, and G. Huang, "Extreme learning machine for multilayer perceptron," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 4, pp. 809–821, 2016.

[35] A. P. Engelbrecht, "A new pruning heuristic based on variance analysis of sensitivity information," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1386–1399, 2001.

[36] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," 2017.

[37] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[38] Z. Jiang, Z. Lin, and L. S. Davis, "Label consistent K-SVD: Learning a discriminative dictionary for recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2651–2664, Nov 2013.

[39] Y. LeCun, Fu Jie Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," vol. 2, pp. II–104 Vol.2, 2004.

[40] Yann LeCun, Courant Institute, NYU, Corinna Cortes, Google Labs, New York, and Chris Burges, Microsoft Research, Redmond, "MNIST handwritten digit database." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[41] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 08 2009.

[42] C. Jayathilake, A. Perera, and M. Chamikara, "Discrete walsh-hadamard transform in signal processing," *International Journal of Research in Information Technology*, vol. 1, pp. 80–89, 01 2013.

[43] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation policies from data," *arXiv preprint*, 2018.

[44] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do cifar-10 classifiers generalize to cifar-10?" *arXiv preprint*, 2018.

[45] W. Rawat and Z. Wang, "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review," *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, 09 2017.

[46] V. Buhrmester, D. Münch, and M. Arens, "Analysis of explainers of black box deep neural networks for computer vision: A survey," 2019.

[47] P. Grau Jurado, X. Liang, A. M. Javid, and S. Chatterjee, "Use of deterministic transforms to design weight matrices of a neural network," in *2021 29th European Signal Processing Conference (EUSIPCO)*, Accepted for publication in August 2021.

[48] C. Kanan and G. Cottrell, "Robust classification of objects, faces, and flowers using natural image statistics," pp. 2472–2479, 2010.

[49] R. Salakhutdinov and G. Hinton, *Deep Boltzmann Machines*, ser. Proceedings of Machine Learning Research, D. van Dyk and M. Welling, Eds. PMLR, 16–18 Apr 2009, vol. 5.

[50] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, *Regularization of Neural Networks using DropConnect*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds. PMLR, 2013, vol. 28, no. 3.

Appendix A

Theoretical computational complexity

In the following table is presented a comparison of the computational complexity and therefore operations needed between three different cases: matrix-vector multiplication, fast algorithms with $Nlog_2N$ and fast algorithms with N operations. For easier visual comparison, the input dimensions N are defined as powers of 2.

In the table, it is important to notice that from an input dimension $N > 2^8$, the usual matrix-vector operation is more computationally expensive than both other options carried by fast algorithms with an input size $N \leq 2^{12}$.

Table A.1: Computational complexity comparison between normal matrix-vector product and different complexities achieved by DLT fast algorithms.

N (input dimension)	Operations		
	N^2 (Matrix-vector)	$\mathcal{O}(Nlog_2N)$	$\mathcal{O}(N)$
2^3	2^6	$2^3 \cdot 3$	2^3
$2^4 = 16$	$2^8 = 256$	$2^4 \cdot 4 = 64$	$2^4 = 16$
2^5	2^{10}	$2^5 \cdot 5$	2^5
$2^6 = 64$	$2^{12} = 4.098$	$2^6 \cdot 6 = 384$	$2^6 = 64$
2^7	2^{14}	$2^7 \cdot 7$	2^7
$2^8 = 256$	$2^{16} = 65.546$	$2^8 \cdot 8 = 2.048$	$2^8 = 256$
2^9	2^{18}	$2^9 \cdot 9$	2^9
$2^{10} = 1.024$	$2^{20} = 1.048.576$	$2^{10} \cdot 10 = 10.240$	$2^{10} = 1.024$
2^{11}	2^{22}	$2^{11} \cdot 11$	2^{11}
$2^{12} = 4.098$	$2^{24} = 16.777.216$	$2^{12} \cdot 12 = 49.152$	$2^{12} = 4.098$

Appendix B

Original SSFN performance

Original performance achieved by SSFN in [2] for an easier comparison with the results acquired in Chapter 4.

Table B.1: Classification accuracy of original SSFN architecture across 50 Monte-Carlo simulations. Extracted from [2].

Dataset	Original SSFN Accuracy (in %) (avg. $\pm$ std. dev)		State-of-art (reference)
	Train dataset	Test dataset	Test accuracy (in%)
Vowel	100 ± 0	60.2 ± 2.4	65.95 [34]
Satimage	95.55 ± 0.15	89.90 ± 0.5	90.90 [38]
Caltech101	99.51 ± 0.06	76.10 ± 0.8	78.50 [48]
Letter	99.02 ± 0.07	95.70 ± 0.2	95.82 [34]
NORB	99.11 ± 0.04	86.10 ± 0.2	89.20 [49]
Shuttle	99.73 ± 0.08	99.80 ± 0.1	99.91 [17]
MNIST	97.21 ± 0.03	95.70 ± 0.1	99.79 [50]
CIFAR-10	53.19 ± 0.15	47.30 ± 0.2	98.52 [43]

Appendix C

Hyperparameter values

Here are presented the tuned hyperparameters chosen to build the proposed architectures presented in Chapter 3 and implemented and tested in Chapter 4. In Table C.1 are summarized the following hyperparameters and its chosen values:

- λ_0 (equation (2.23)): RLS regularization parameter to compute SSFN first layer output matrix.
- μ (equation (2.24) and [2]): control the ADMM convergence rate.
- η_{var} (equation (3.6)): variance threshold to remove nodes.
- Random matrix parameters (algorithm 1): parameters used to control the random matrix shape when including random instance in the linear transforms box.
- γ (equation (3.14)): cumulative singular values threshold when using method 2 in unsupervised approach.

Table C.1: Overall tuned parameters indicating to the experiments "Exp." they belong.

λ_0 (RLS) and μ (ADMM) (Exp. 1-2-3)	η_{var} (Exp. 1-2-3)	Random matrix parameters (Exp. 1)	γ (Exp. 3, Method 2)
Table C.3	10^{-7}	Table C.2	0.8

Table C.2: Parameter tuning for random matrix instance in first experiment.

Dataset	k_{max}	α	$n_{max} - 2Q$	η_{node}	η_{layer}	L_{max}	Δ
All	100	2	4000	0.005	0.15	20	500

Table C.3: Tuned RLS (λ_0) and ADMM (μ) parameters for the three different experiments "Exp.".

Dataset	λ_0 (RLS)	μ (ADMM)			
	Exp.1, Exp.2, Exp.3	Exp.1	Exp.2	Exp.3	
				Method1	Method2
Vowel	10^{1}	10^{3}	1	10^{3}	10^{3}
Satimage	10^{6}	10^{5}	10^{8}	10^{8}	10^{8}
Caltech101	3	10^{-2}	10^{-2}	10^{-2}	10^{-2}
Letter	10^{-5}	10^{8}	10^{6}	10^{5}	10^{9}
NORB	10^{2}	10^{4}	10^{4}	10^{4}	10^{4}
Shuttle	10^{5}	10^{4}	10^{6}	10^{4}	10^{4}
MNIST	10^{8}	10^{5}	10^{5}	10^{7}	10^{7}
CIFAR-10	10^{8}	10^{4}	10^{4}	10^{4}	10^{4}

Appendix D

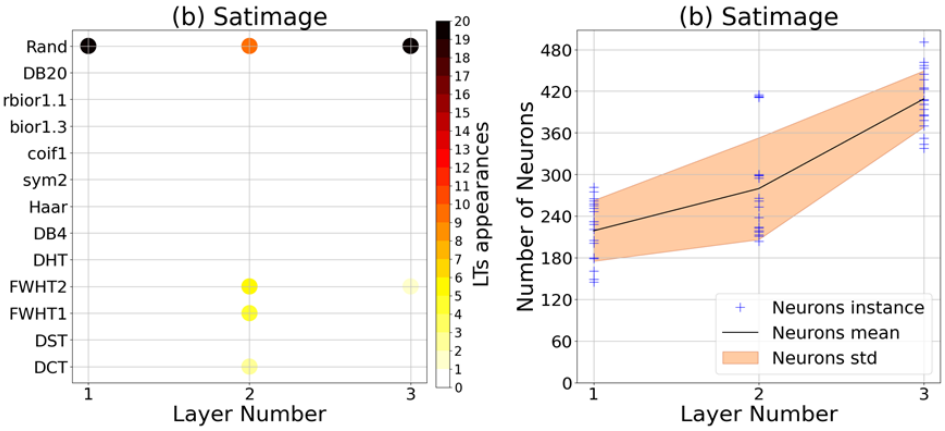
Remaining experimental results

This appendix presents the graphical results for the datasets not presented in chapter 4, Satimage, Norb, Shuttle and Mnist datasets.

D.1 Supervised approach with random matrix

Table D.1: Network layers average across MC simulations for supervised approach with random matrix for the remaining datasets.

Dataset	Layers avg.
Satimage	3
NORB	6.68
Shuttle	3.55
MNIST	4.9



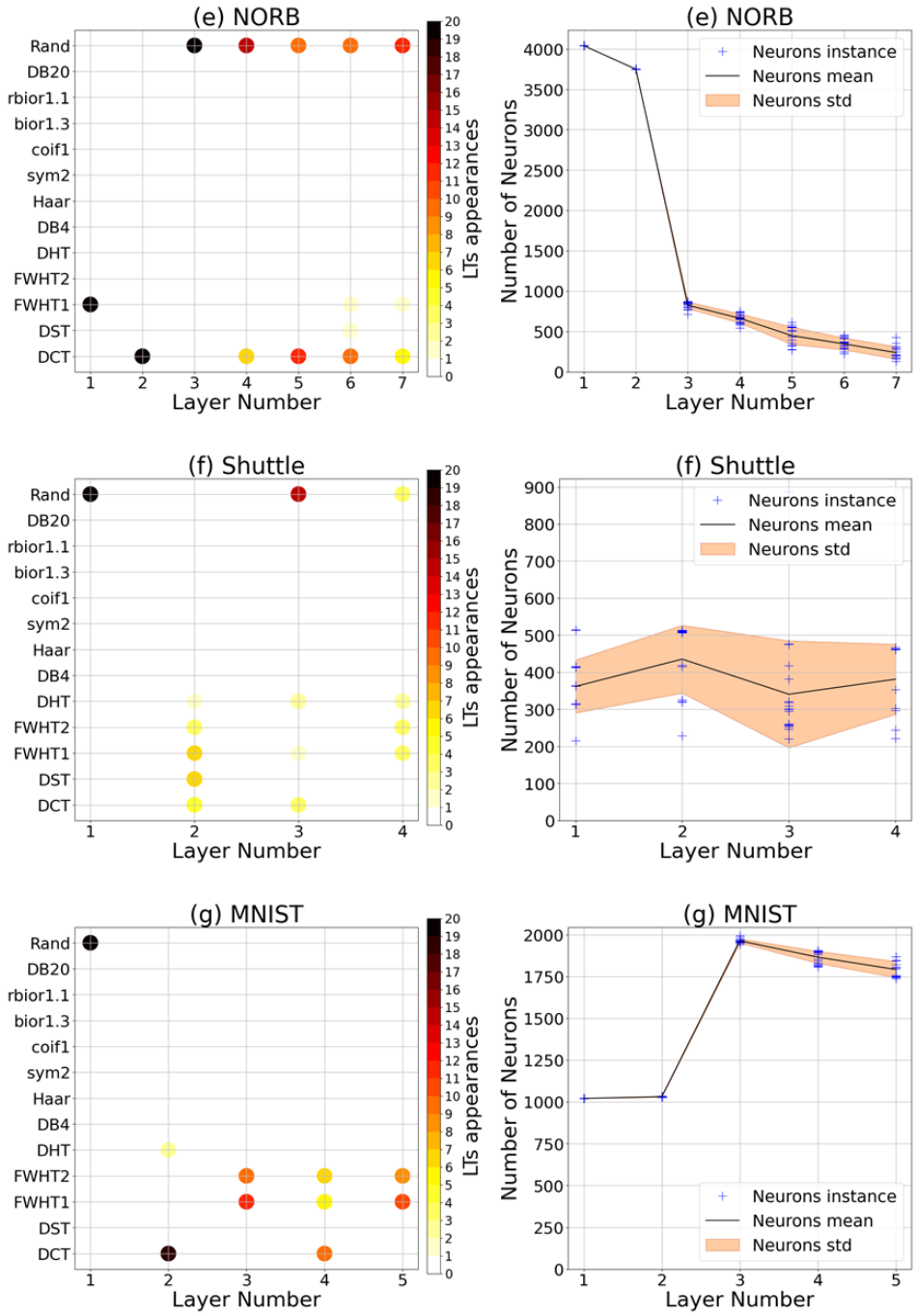
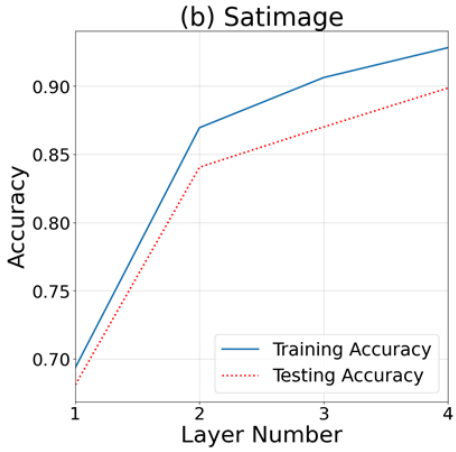
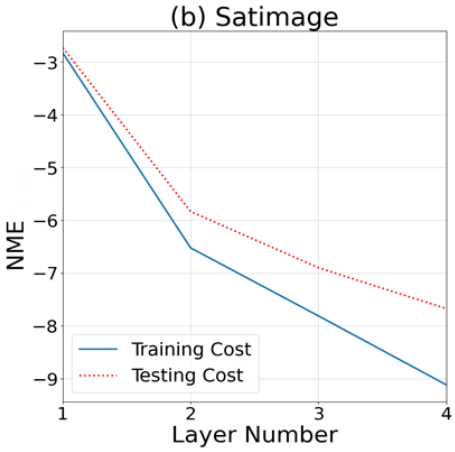
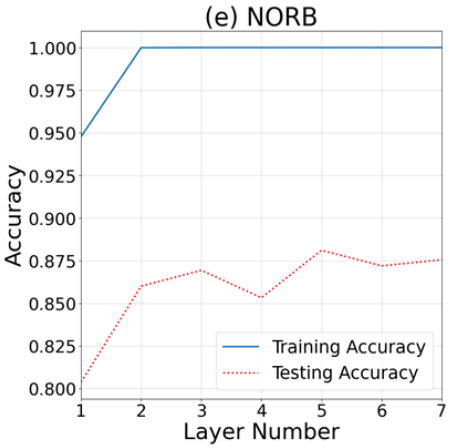


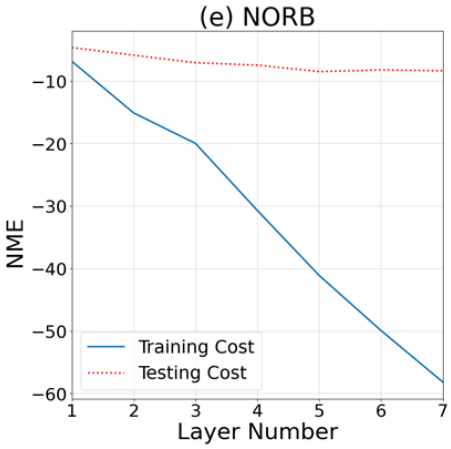
Figure D.1: Network architecture for Satimage, NORB, Shuttle and MNIST datasets implementing supervised approach with random matrix.

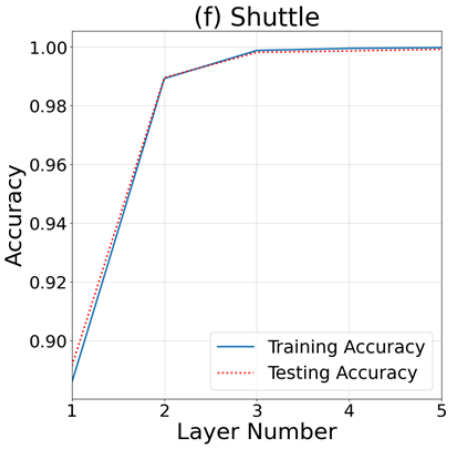
D.2 Supervised approach without random matrix

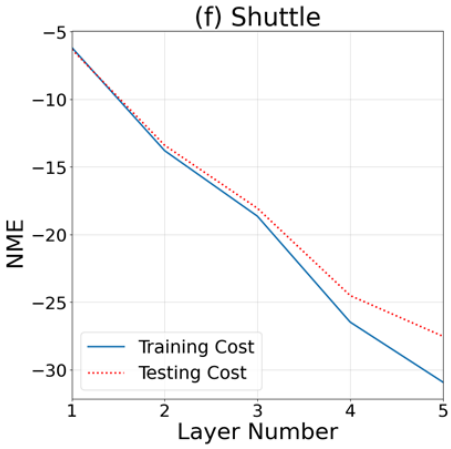












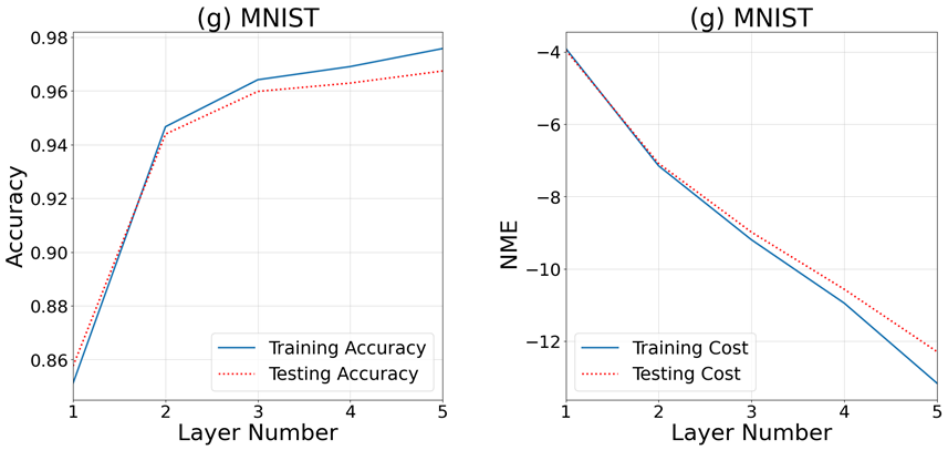
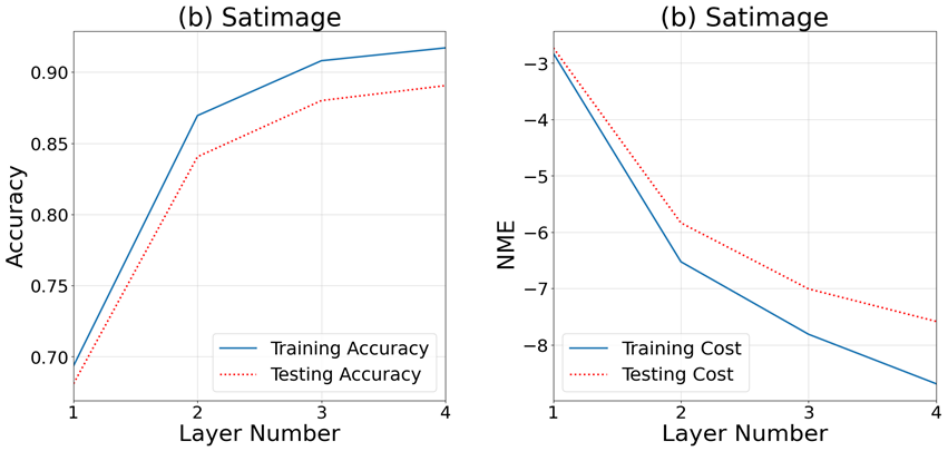


Figure D.2: Accuracy and NME curves on every layer for Satimage, NORB, Shuttle and Mnist datasets implementing supervised without random matrix.

D.3 Unsupervised approach Method 1 implementation



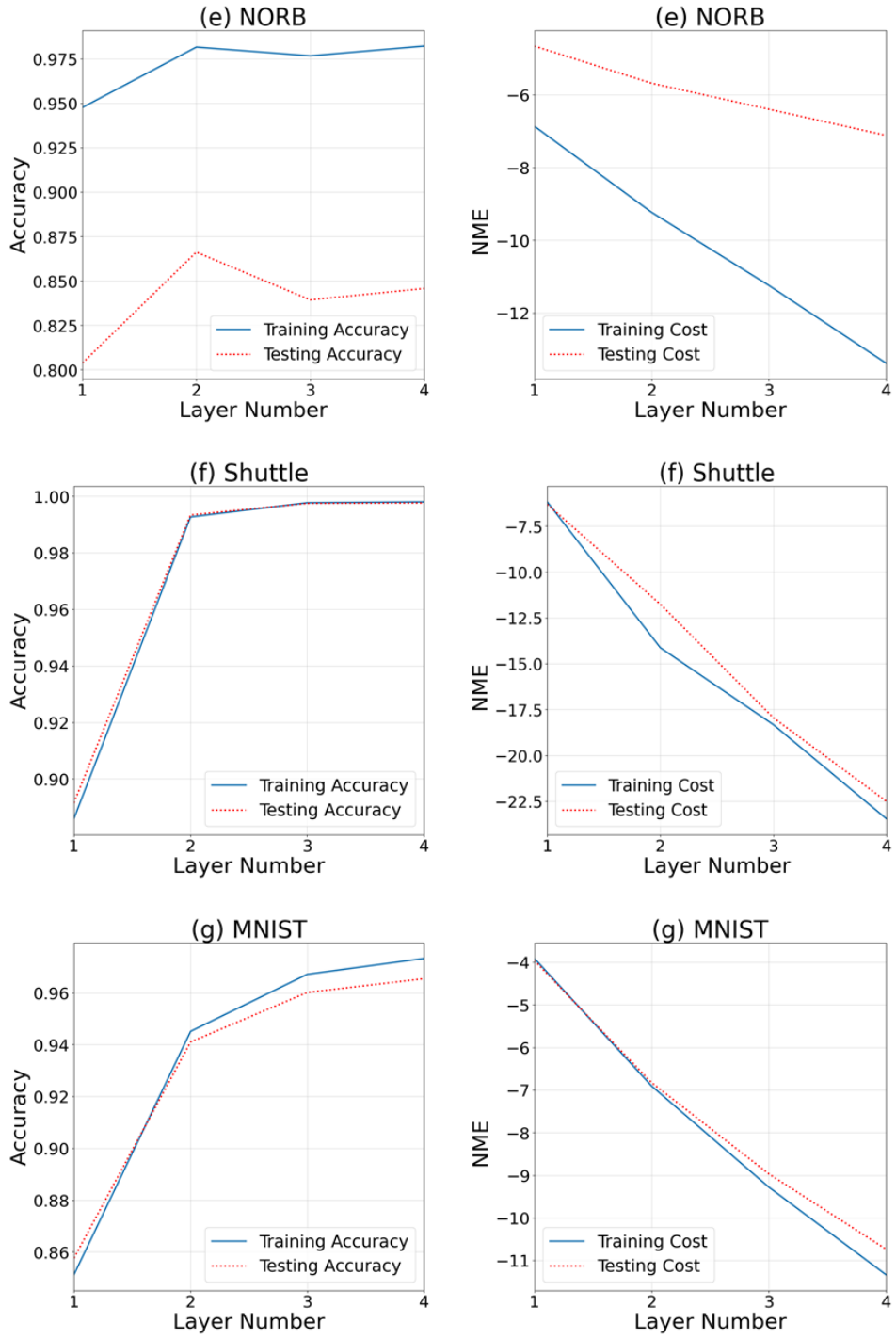
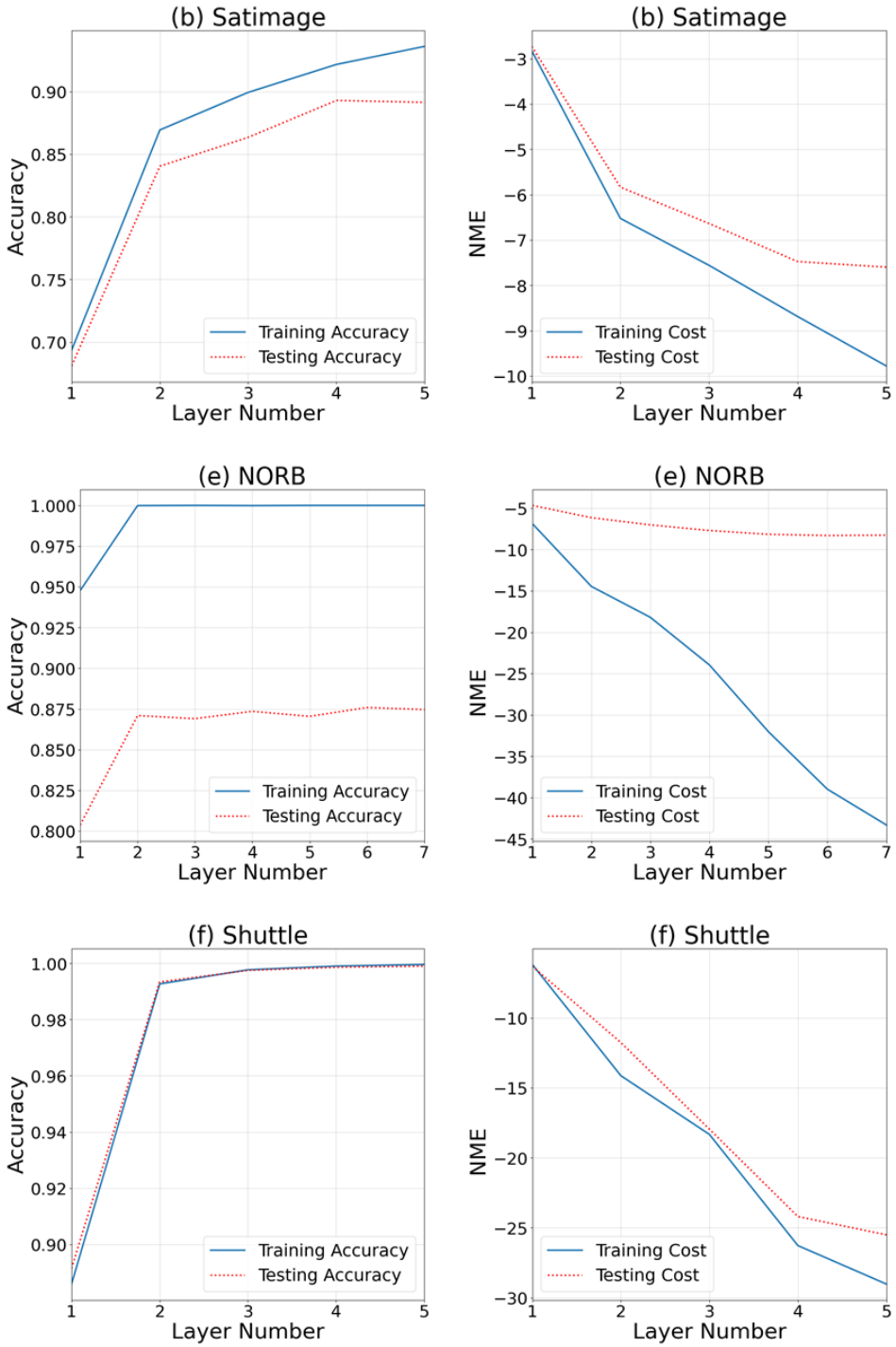


Figure D.3: Accuracy and NME curves on every layer for Satimage, NORB, Shuttle and Mnist datasets implementing unsupervised approach Method1.

D.4 Unsupervised approach Method 2 implementation



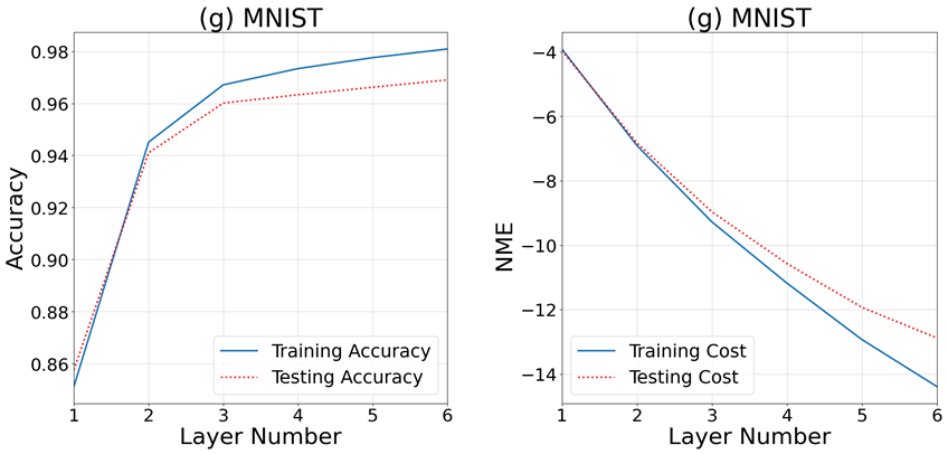


Figure D.4: Accuracy and NME curves on every layer for Satimage, NORB, Shuttle and Mnist datasets implementing unsupervised approach Method2.

Appendix E

EUSIPCO 2021 Article Publication

E.1 EUSIPCO confirmation

EUSIPCO 2021 Review Results [Paper #6716]

EUSIPCO 2021<eusipco2021@cmsworkshops.com>

mar 04/05/2021 21:17

Para:Pol Grau Jurado <polgj@kth.se>; Xinyue Liang <xinyuel@kth.se>; Alireza Mahdavi Javid <almj@kth.se>; Saikat Chatterjee <sach@kth.se>;

Dear Pol Grau Jurado, Xinyue Liang, Alireza M. Javid, Saikat Chatterjee,

Paper ID: 6716
Title: Use of Deterministic Transforms to Design Weight Matrices of a Neural Network

The review process for the 29th European Signal Processing Conference (EUSIPCO 2021) has now been finalized, and it is our pleasure to inform you that your paper has been accepted for publication. Congratulations! The review comments that have led to this decision can be found below.

Please note that due to the COVID-19 outbreak, the event will be FULLY VIRTUAL.

Carefully consider the following instructions for the publication of your paper and your participation in the conference:

- A camera-ready PDF version of your paper needs to be submitted by May 31, 2021 in the online submission system: https://cmsworkshops.com/EUSIPCO2021/Papers.asp. Please make sure to take into account the below review comments when preparing your camera-ready paper. The layout of the camera-ready paper is identical to the layout of the submitted paper. Details can be found in the Author's Information page on the EUSIPCO 2021 website. Please do not include page numbers or headers/footers in your camera-ready paper.

- Your paper needs to be covered by a regular (non-student) conference registration by June 1, 2021. This requirement also holds for papers authored or presented by students. Registration will soon open on the conference website: https://eusipco2021.org. When registering, you will be asked to enter the Paper ID of the papers you wish to cover with your registration. Papers that are not covered by a regular (non-student) conference registration by June 1, 2021 will be removed from the conference proceedings. Detailed instructions will be soon available on the conference website.

- In the following days, with another email, you will be informed about the format of your paper (lecture/poster) and on its location in the conference program. You will be asked to record a video for presenting your work and we will provide you with a link for the instructions.

- The conference proceedings will be published by September 2021 in the open-access proceedings archive of the European Association for Signal Processing (EURASIP) and on IEEE Xplore.

We look forward to meeting you virtually in Dublin in August 2021!

Sincerely,

Maria Sabrina Greco, Stephen McLaughlin and Josiane Zerubia
EUSIPCO 2021 Technical Program Co-Chairs

---- Comments from the Reviewers ----
Review #0071

Paper Format: Minor improvements needed

Topic relevance: High

Originality of the content: Very High

Methodology / Research design: High

Evaluation of results and derived conclusions: High

References to previous work: Very High

Correct English usage: High

General Comments to Authors
This paper explored the use of deterministic transforms（discrete cosine transform, discrete wavelet transform） instead of random matrix instances, for the Self size-estimating feedforward network (SSFN) weight matrices. beside with two criterion such as features standards deviations and singular values of cross-correlation matrix.
The use of deterministic transforms provides a reduction in computational complexity and achieve competitive classification performance on different datasets.
and may provide understanding of interpretability/explainability, and bring new insights about the information flow within layers of a neural network.
Hennce achieve a new learning approach as a hybrid combination of supervised and unsupervised learning in each layer of the network.
It will better if the introdution part give a overall structure description of the paper.

Review #01B8

Paper Format: Correctly formatted

Topic relevance: High

Originality of the content: High

Methodology / Research design: High

Evaluation of results and derived conclusions: Average

References to previous work: High

Correct English usage: Average

General Comments to Authors
The paper presents a comparative performance analysis study on the use of deterministic transforms instead of random matrices in the neural network weights (specifically the SSFN architecture) to reach final models that are better suited for limited resource scenarios.
The problem addressed is interesting, theoretical explanations are found appropriate, experimental design used is concrete with sufficient detail, literature review is considered adequate. With that being said,
1) sections II and III of the paper contains typo errors here-and-there, that should be corrected.
2) R_{l+1} in equation 4 should be explained.
3) While comparative accuracies of the state-of-the-art and the proposed methods are valuable; presentation, analyses, and detailed comparisons in terms of usages of computational resources (processing power, RAM, etc.) for both training and testing would make the conclusions drawn more convincing towards applicability of the proposed approach for scenarios with more strict real time and online requirements.

4) The result corresponding to the new trained network structure '4096-2977-3477-
3300 (FWHT2-DHT-DST-FWHT1)' stated in the end of the results section could have been added to Table II, so that the paper would convey its message in a more coherent way.

Review #02A8

Paper Format: Correctly formatted

Topic relevance: High

Originality of the content: Average

Methodology / Research design: High

Evaluation of results and derived conclusions: High

References to previous work: High

Correct English usage: High

General Comments to Authors
The authors propose the use of deterministic transformations (e.g., DCT) to design the weight matrices in a feed-forward neural network. The complexity of the proposed system varies between O(N) and 0(Nlog_base_N). The proposed system is evaluated in speech recognition and image classification datasets. In general, it is a well-written paper with a solid research design and experimental evaluation.

Some minor comments for the authors:
- Try to avoid "etc." in scientific writing
- It is better to use passive voice instead of saying "We investigate...", "We use..."
- Section should always start with a capital "S"
- "e.g.," --> a comma should follow
- '4096-2977-3477-3300 (FWHT2-DHT-DST-FWHT1)' --> Replace the beginning quote mark with ` (formatting)

-end-

E.2 EUSIPCO Schedule

EUSIPCO 2021: Presentation Schedule [Paper #6716]

EUSIPCO 2021<eusipco2021@cmsworkshops.com>

lun 24/05/2021 16:21

Para:Pol Grau Jurado <polgj@kth.se>; Xinyue Liang <xinyuel@kth.se>; Alireza Mahdavi Javid <almj@kth.se>; Saikat Chatterjee <sach@kth.se>;

Dear Pol Grau Jurado, Xinyue Liang, Alireza M. Javid, Saikat Chatterjee,

Paper ID: 6716
Paper Title: Use of Deterministic Transforms to Design Weight Matrices of a Neural Network

The session and presentation times for your paper are:

Session: SiG-DML-L2: Neural network learning Part 2

Session Time:
Wednesday, 25 August, 10:30 - 12:30 IST (UTC +01:00)

Session Format: Lecture

In the acronym of the session label, P# stands for poster and L# for lecture.

You can find the instructions on pre-recorded videos at https://eusipco2021.org/virtual-participation/

The deadline for author registration is July 1st. Registrations will be open soon at https://eusipco2021.org/registration/

Best regards,
Maria Sabrina Greco, Stephen McLaughlin and Josiane Zerubia
Technical co-Chairs, EUSIPCO 2021