Performance Enhancement of Time Delay and Convolutional Neural Networks Employing Sparse Representation in the Transform Domains

Masoumeh Kalantari Khandani
University of Central Florida

PERFORMANCE ENHANCEMENT OF TIME DELAY AND CONVOLUTIONAL NEURAL NETWORKS EMPLOYING SPARSE REPRESENTATION IN THE TRANSFORM DOMAINS

By

MASOUMEH KALANTARI KHANDANI
M.Sc. Simon Fraser University, BC, Canada 2011

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2021

Major Professor: Wasfy B. Mikhael

ABSTRACT

Deep neural networks are quickly advancing and increasingly used in many applications; however, these networks are often extremely large and require computing and storage power beyond what is available in most embedded and sensor devices. For example, IoT (Internet of Things) devices lack powerful processors or graphical processing units (GPUs) that are commonly used in deep networks. Given the very large-scale deployment of such low power devices, it is desirable to design methods for efficient reduction of computational needs of neural networks. This can be done by reducing input data size or network sizes. Expectedly, such reduction comes at the cost of degraded performance.

In this work, we examine how sparsifying the input to a neural network can significantly improve the performance of artificial neural networks (ANN) such as time delay neural networks (TDNN) and convolutional neural networks (CNNs). We show how TDNNs can be enhanced using a sparsifying transform layer that significantly improves learning time and forecasting performance for time series. We mathematically prove that the improvement is the result of sparsification of the input of a fully connected layer of a TDNN. Experiments with several datasets and transforms such as discrete cosine transform (DCT), discrete wavelet transform (DWT) and PCA (Principal Component Analysis) are used to show the improvement and the reason behind it. We also show that this improved performance can be traded off for network size reduction of a TDNN.

Similarly, we show that the performance of reduced size CNNs can be improved for image classification when domain transforms are employed in the input. The improvement in CNN performance is found to be related to the better preservation of information when sparsifying transforms are used. We evaluate the proposed concept with low complexity CNNs and common datasets of Fashion MNIST and CIFAR. We constrain the size of CNNs in our tests to under 200K learnable parameters, as opposed to millions in deeper networks. We emphasize that finding optimal hyper parameters or network configurations is not the objective of this study; rather, we focus on studying the impact of projecting data to new domains on the performance of reduced size inputs and networks. It is shown that input size reduction of up to 75% is possible, without loss of classification accuracy in some cases.

This dissertation is dedicated to my parents, my husband and my children for all their support and sacrifice. I would also like to dedicate this work to all who sacrificed their lives while serving others during the 2020-2021 pandemic, especially my brother-in-law (Dr. Seyyed Reza Shojaosadati)

ACKNOWLEDGMENTS

I am sincerely grateful to my Ph.D. advisor, Dr. Wasfy Mikhael; this research would not have been possible without his great insights, guidance, encouragements, and support.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Networks
ARIMA	Autoregressive Integrated Moving Average
CDF	Cumulative distribution Function
CDF_9/7	Cohen–Daubechies–Feauveau wavelets, a family of biorthogonal wavelets, CDF 9/7 wavelet for lossy compression
CIFAR-10	Canadian Institute for Advanced Research image dataset with 10 classes of data
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
DCT	Discrete Cosine Transforms
DFT	Discrete Fourier transform
DL	Dictionary Learning
DWT	Discrete Wavelet Transform
DRNNs	Dynamic Recurrent Neural Networks
ERSST	Extended Reconstructed Sea Surface Temperature
FFNN	feed-forward neural network
FMNSIT	Fashion MNSIT dataset
GPU	Graphics Processing Unit
HWT	Haar Wavelet Transform
HL, HH	High-Low, and High-High, two bands of DWT
IoT	Internet of Things
ICA	Independent Component Analysis
JPEG	Joint Photographic Experts Group
LL, LH	Low-Low, Low-High, two bands of DWT
LSTM	Long-Short Term Memory

mDCT	Magnified Discrete Cosine Transforms
Mixed-T	Mixture of Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT)
ML	Machine Learning
MSE	Mean Squared Error
NOAA	National Oceanic and Atmospheric Administration
NDDS	Naturalistic Driving Dataset Study
PCA	Principal Component Analysis
PSNR	Peak Signal-to-Noise Ratio
Res-Net	Residual Neural Network
RGB	Red Green Blue, three channels of colored images
SARIMA	Seasonal Autoregressive Integrated Moving Average
SST	Sea Surface Temperature
TDNN	Time-Delay Neural Networks
US101	Dataset collected at highway US101
CVP	Constant Velocity Prediction
VGG	Visual Geometry Group (a group of researchers at Oxford who developed this architecture)
WNN	Wavelet Neural Network

CHAPTER 1. INTRODUCTION

1.1 Motivation and Problem Statement

Nowadays, small embedded computing and sensor devices are finding widespread use. However, their limited computing and storage capacity limits the possibility of running state of the art learning systems, for example methods based on deep neural networks, on these devices. Recent advancements in Artificial Neural Networks (ANN) have paved the way for significant improvements in prediction and classification applications over traditional methods. In particular, deep convolutional neural networks are now considered a major tool for this purpose. However, these networks are often large and require computing and storage resources that are not available in many smaller computing devices. For example, most IoT (Internet of Things) devices are usually designed with minimal computing and storage capacity with the aim of reducing cost. Using deep neural networks in large scale deployment of IoTs will require either considerable cost increase, or considerable reduction of performance. Moreover, the task of training these networks, which is often done offline, is sometimes needed to be done online for retraining of networks. This task is generally very computing intensive, compared to the forward operation of an ANN. Therefore, it becomes necessary to look for methods of reducing the computational cost of ANNs for low power computing devices. This is more challenging when the task of online or in-device learning is at hand. Simpler neural network-based schemes, such as shallower neural networks, are considered as feasible solutions in these scenarios. In this work, we investigate how introducing an input transform layer that sparsifies the input can considerably accelerate the learning rate or accuracy of such neural network-based solutions. We examine this for feed-forward based networks, such as time-delay neural networks (TDNN) and convolutional neural networks (CNN).

TDNNs are used in prediction and forecasting application with time series; whereas, CNNs are one of the primary tools in image classification. We note that the observations here are expected to be applicable to general neural networks; however, our investigation and tests have been focused on TDNN and CNN networks.

In this thesis, we first investigate the impact of using domain transforms on TDNNs. We consider transforming input data to a domain where it could be represented in a sparse fashion. It is shown that this method reduces the training time or equivalently improves the accuracy for a fixed training time. While such a hypothesis is not intuitively obvious, it turns out that mathematical properties of a neural network and its training algorithm lead to this phenomenon. To show this concept and demonstrate its effect, we consider time series data used with a TDNN. For sparsifying transforms, we consider transforming data to other domains using DCT (Discrete Cosine Transforms), PCA (Principal Component Analysis), and Mixed transform of Haar and DCT, as in our earlier work in [1]. It should be noted that the sparsity of the transformed data will depend both on the transform and the characteristics of data. As a result, we need to check the sparsity of the transformed data and its effect on TDNN learning. In fact, it can be shown that the improvement is the result of sparsity, and not the result of a specific transform being used. We also validate the idea by feeding simple synthetic data to the network. We investigate the use of domain transform for possible network size reduction, examining the impact of the level of sparsity on performance improvement of TDNN.

Following an analysis on TDNNs, we expand our study and investigate the impact of using domain transform with shallow convolutional neural networks (extending to 2-D and images). We note that reducing the computational cost of CNNs is done through reducing the number of layers or the input data (image) size. To put this in perspective, most deep networks that are used for classification have millions (or tens of millions) of learnable parameters. Such networks are too complex for embedded devices. We limit our choices of networks to those with an order of magnitude less parameters (i.e., under 200K learnable parameters and down to 50K). The reduction in network and input size, as expected, usually comes at the cost of reduced classification performance. In this dissertation, we examine how domain transforms can be used for mitigating the negative effect of input and network size reduction.

We also show that using transforms, such as Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT) as an input transform layer, it is possible to efficiently improve the performance of size-reduced networks. We note that these transforms project the original image data to a domain where data is represented in a sparser form, allowing for selectively removing part of the input data and reducing the input image size in a more efficient way than simple resizing of an image. We observe that such transforms also have the positive side effect of improving the learning rate. Similar benefits were seen in our earlier works on time series and shallow networks [40]. The improved learning rate and network size reduction allows for lower computational cost, in particular during the training phase. Training of deep networks is usually the most computationally expensive aspect of CNNs; while retraining a network requires less computations, it is still considered a heavy load on smaller devices. The reduced cost of training, achieved using domain transforms, is an important factor that can enable retraining of CNNs in small devices.

The basic hypothesis that is examined here is that transforming image data to a sparser form will allow for more efficient network and input size reduction than simply resizing the input.

While in data compression the use of such transforms is common, in classification applications it is not intuitive that sparser representation will be useful. In fact, it is seen that the usual application of CNNs with convolutional filters in the input layers is not useful when data is represented in domains such as DCT. On the other hand, it is seen that more efficient configurations becomes possible if DCT is used. Transfer to DWT domain shows different properties, since the spatial relationship between initial data pixels are maintained (as opposed to DCT that the spatial relationship is not explicitly kept). DWT proves to be very effective when significant size reduction is needed (improving the result by up to 5%). To evaluate and examine these hypotheses and observations, we use two standard datasets of small images (representing IoT processable images), including Fashion MNSIT (FMNSIT) and CIFAR-10. Evaluating our proposed methods, it is shown that input size reduction of up to 75% is possible, without loss of classification accuracy. We demonstrate that in most cases the improvement can be traced to higher entropy of resized input using transforms.

1.2 Contributions and Dissertation Arrangement

The main contributions of this thesis can be described as follows:

1- We introduce the use of sparsifying domain transforms with TDNNs for significant reduction of training time or increase in forecasting accuracy of TDNNs

2- We provide a mathematical analysis on the impact of sparsifying input to a feed forward ANN such as TDNN and prove positive effect on learning performance

3- We present a method for reduction of TDNN network size using transforms such as PCA and DCT while maintaining forecasting performance.

4- We improve the performance of reduced-size CNN using DCT and DWT transforms.

The above contributions are described in chapters 2-4. First in Chapter 2 we demonstrate the effect of sparse representation of time series data on learning rate of time delay neural network. This is done through adding a transform layer at the beginning of a feed forward neural network. In chapter 2, we also mathematically prove that applying a sparsifying transform to input layer will reduce the training time considerably. In Chapter 3 we investigate the effect of domain transform on network size reduction. The core concept of the proposed idea is the possibility of reducing network size in TDNNs with the help of sparse input representation. With a given level of performance (in this case, prediction error), it is possible to considerably reduce the hidden layer size of a network, simply by applying sparsifying transform to the input layer. Furthermore, the considerable improved performance allows for additional network size reduction through removing some of the coefficients of the sparse representation of the input vector.

Overall, it is consistently seen that applying sparsifying transforms to the input of a TDNN allows for better performance which can be traded off for reducing network size in both the input and the hidden layers as discussed in chapter 3.

In Chapter 4 the same idea from chapter 2 is generalized and applied to image data processing using CNN’s. Fashion Mnist and CIFAR-10 (RGB images) datasets are used in this study. We consider networks with of up to 200k parameters in this study. This is an order of magnitude smaller than larger deep networks that are usually studied in the literature. It is observed that reducing input data size can be efficiently achieved using transforms such as DCT or DWT. Compared to averaging-based image resizing, methods based on DCT and DWT are shown to always provide improved classification accuracy (generally 1-4%). These findings are validated using image datasets of Fashion MNIST and CIFAR-10. In some cases, input size could be reduced by 75% while still maintaining the accuracy.

The thesis is concluded in chapter 5, where we provide some insight into future direction and the lessons learned in this research.

CHAPTER 2. EFFECT OF SPARSE REPRESENTATION OF TIME SERIES DATA ON LEARNING RATE OF TIME DELAY NEURAL NETWORKS

2.1 Introduction

[1]In this chapter, we investigate how introducing a sparsifying input transform layer can considerably accelerate the learning rate or accuracy of neural network-based solutions. We examine this for feed-forward based networks, such as time-delay neural networks (TDNN). We note that the observations here are expected to be applicable to general feed forward neural networks; however, our investigation and tests have been focused on TDNN and time series. The proposed method enables designing many new AI applications for limited capacity devices. In addition, the faster learning will facilitate online learning when changes in data trends may require frequent retraining inside an IoT device.

[1] The work presented in this chapter is based on the following publications:

[1] Khandani M. K. and Mikhael W. B.: Effect of Sparse Representation of Time Series Data on Learning Rate of Time Delay Neural Networks, Circuits, Systems, and Signal Processing, 2021. DOI 10.1007/s00034-020-01610-8

[2] Khandani, M. K., Mikhael, Wasfy B.: Using Mixed DCT and Haar Transforms for Efficient Compression of Car Trajectory Data. IEEE 61 international Midwest Symp.On Circuits and sys., 2018

[3] Khandani M. K., Mikhael, Wasfy B.: Efficient Time Series Forecasting Using Time Delay Neural Networks with Domain Pre-Transforms. IEEE MWSCAS, Dallas, Texas, August 2019

The basic concept investigated here is that transforming input data to a domain where it could be represented in a sparse fashion will reduce the training time or equivalently improves the accuracy for a fixed training time. While such a hypothesis is not intuitively obvious, it turns out that mathematical properties of a neural network and its training algorithm lead to this phenomenon. To show this concept and demonstrate its effect, we consider time series data used with a TDNN. For sparsifying transforms, we consider transforming data to other domains using DCT (Discrete Cosine Transforms) or PCA (principal component analysis), and Mixed transform of Haar and DCT [1]. It should be noted that the sparsity of the transformed data will depend both on the transform and the characteristics of data. As a result, we need to check the sparsity of the transformed data and its effect on TDNN learning. In fact, it can be shown that the improvement is the result of sparsity, and not the result of a specific transform being used. We also validate the idea by feeding simple synthetic data to the network. The datasets used in this work are ERSST from National Weather Forecast-NOAA [17], vehicle speed time series from two datasets available from public sources (NDDS[5] and US101[6]) Figure 2.1, and synthetic data.

ERSST dataset is a global monthly sea surface temperature (in Celsius) dataset derived from the International Comprehensive Ocean–Atmosphere Dataset (ICOADS) and reported by NOAA (National Oceanic and Atmospheric Administration). For our tests, we have used data from 1991 to 2018, and from longitudes 1 to 180 and latitudes 20 to 25. Around 330,000 input samples are derived and user to train a TDNN.

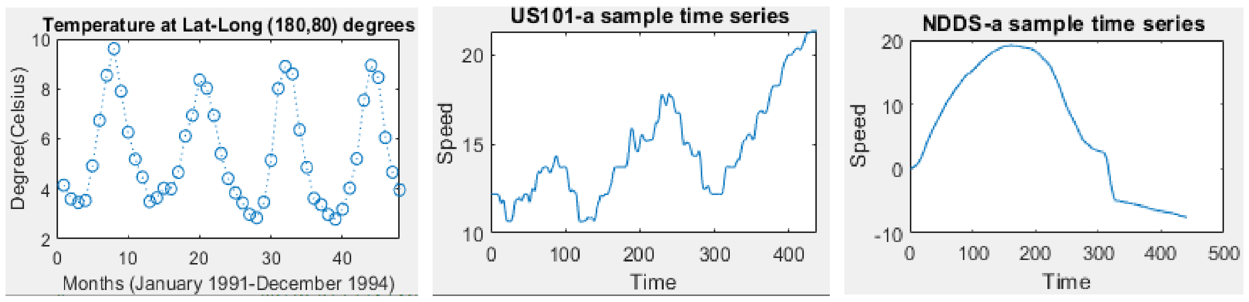


Figure 2.1: Visualizing sample series from ERSST(left) , US101 (middle), and NDDS (right) datasets

NDDS dataset includes movement data from 100 cars sensed using on board sensors (e.g., accelerometer, odometer). The US-101 datasets include vehicle movement that is extracted from camera/video frames. The datasets are collected using different methods and at different time scales. The NDDS dataset has information for almost 800 near crash scenarios (approx. 30-40 second trips) in the Washington DC area with approximately 400-time samples each, resulting in over 300,000 samples for each attribute such as speed or acceleration. The speed values from NDDS are used here. The US-101dataset include data from certain times of day on US-101 highway in California containing around 2000 vehicle trajectories. Each trajectory contains between 400 to 1000 time series datapoints; we use the speed values from this set as well. US-101 dataset is to some degree different from NDDS dataset (since NDDS is related to near crash scenarios). The NDDS dataset contains high resolution (10Hz) acceleration data, and lower resolution (3Hz) velocity; as a result, we used the acceleration data to reproduce speed and position at 10Hz. The US101 datasets already provide acceleration, speed and position information at 10Hz. Data in both datasets are in imperial units' system; we converted them to the metric system for the results reported here. While there are various metrics collected in these datasets, we only focused on measurements that were related to our studies such as speed.

For the Synthetic datasets we have produced two datasets of 500 time series, each containing 400 samples. One dataset contains simple lines in which the time series value increased linearly with random slope and with some noise. For the other dataset, we have used a combination of three randomized sinusoidal and Gaussian noise, in order to be able to control its sparsity properties.

Some insight from the mathematical analysis shows that faster or better learning may be happening due to the reduction in the search space of the learning algorithm (which can be seen as an optimization problem); equivalently, it could be seen as a dimension reduction of the search space, which allows faster optimization. From an input data perspective, the sparse representation means that most of the variation in data is concentrated in a smaller number of dimensions, as it can be seen in Figure 2.3 and Figure 2.5 for some sample datasets and transformations. As a result, the neural network needs to learn a smaller number of patterns. With non-sparse data, a larger region of patterns needs to be learnt.

In the rest of this chapter, we first review some related literature and establish how evaluation of the proposed method should be done. Then the system architecture and the proposed input transform operation for a TDNN are explained. Mathematical reasoning for the improvement due to sparsifying transforms are then explained, followed by experiments results with the abovementioned datasets and transforms.

2.2 Related Works

There are numerous methods and tools applied for time series prediction in different applications. Neural networks (or Artificial Neural Networks, ANN) have shown great ability in modeling and forecasting nonlinear and nonstationary time series [25]. TDNN is one of the main neural network based tools for this purpose that has recently received attention [8] [12] [14] [15] [16] [30] [31][13] [32]. In general, TDNNs have the ability to catch diverse characteristics of the data [33]. While models based on deep recurring neural networks (such as Long-Short Term Memory or LSTM [38]) have shown great promise in learning patterns in time series, simpler TDNNs are also very effective and much simpler at the same time. In this work, we focus on feed forward and shallow neural networks such as TDNNs, given our target applications in small devices with limited computing capabilities.

One of the recent works, Girish et al. [8], compared the performance of TDNN based methods with linear models such as Auto Regressive Integrated Moving Average -ARIMA for forecasting commodity pricing, concluding TDNNs outperform linear fitted models, one main reason the ANN methods outperforms the methods such as ARMA and ARIMA is that the data should be at least weakly stationary for these linear methods. ANNs are also widely used in sea surface temperature (SST) prediction studies and they are shown to outperform the traditional linear methods, one main reason could be the non-linearity that is inherent in neural networks. It has been shown that ANN is an alternative to complex physics-based coupled models that additionally require a large amount of tuning effort [30] [28][29].

The use of transforms and neural networks has also received some attention. Although none of these works apply domain transform directly as a sparsifying input layer for accelerating learning, the general concept of using transforms in some form with neural networks has been around. Generally, the transforms are applied to reduce data size, or the size of input or filtering out certain parts of data. For example, wavelets have been used with shallow ANNs and TDNNs in several works. preprocessing using Fourier transform and then frequency selection is reported in [16] used for classification and steps taken are specifically for that data set. Aggarwal et al. [34] have applied a hybrid approach using wavelet transform on time series fed into TDNN for forecast day-ahead electricity prices in the New England market for 2014. Authors of [10] utilize a dynamic time-delay Wavelet Neural Network -WNN model with a recurrent feedback topology for forecasting traffic flow. The model is a combination of a WNN and a conventional neural network with a sigmoid activation function and they have their own nonlinear estimator in the WNN part. Nonetheless, they method is customized for traffic flow data only not for general use and had access to limited traffic flow data. Nonetheless, they do not consider the individual time series vehicle movement and had access to limited traffic flow data. Alex et al. at [9], consider parallel DRNNs (Dynamic Recurrent Neural Network), each treating a part of the output of a Wavelet transform; like LSTM, the recurrent methods are complex. However, the existing works do not study the impact on learning time or prediction.

The abovementioned methods are generally tailored for specific applications and data types. While they do provide improvements, they are fundamentally different from how we apply a domain transform as a layer to the neural network with the aim of sparsifying input and accelerating learning. The concept we propose and study here, based on sparsifying the input, is generic and can be applied to general ANNs. Moreover, domain transforms for sparsifying, such as PCA, DCT or mixed transforms are also not considered for time series data. Mathematical reasoning on the effect of these transforms are also not presented. Additionally, these works have not considered training efficiency, which can become critical if online training is needed.

The use of transforms such as DCT with 2-D data (images) has also been reported. While this is not directly related to our work since the same concept cannot be used in TDNNs, it is worth mentioning works such as that by Pan et al [24] which has shown that reducing image information redundancy using DCT with Convolutional Neural Network (CNN) can be beneficial. They have demonstrated that when DCT coefficients are fed into a backpropagation neural network for classification, a good recognition rate can be achieved by using a very small proportion of transform coefficients. This makes DCT-based face recognition faster than other approaches. Additionally, recent research [35][39] have highlighted that most of weights are useless in CNNs and can be set to zero without obvious deterioration in performance.

In this work, we study the impact of adding a sparsifying layer to a TDNN. The layer will transform the input vector to a domain where it is represented in a sparser form. We show that it is the sparser form that leads to better learning. Without the added layer, the TDNN should still be able to learn the model and be used for forecasting. It is the learning accuracy or speed that is improved. Given that the existing methods are designed for specific applications and data types, it is not possible to directly compare the impact of adding a sparsifying layer with existing methods that use transforms. Therefore, in this work we take a TDNN with no transform as the baseline and study the impact on this structure.

2.3 Method and System Description

The concept considered here is to add an input transform layer immediately following the first layer of a feed-forward neural network (FFNN). Considering a TDNN as a FFNN, the first layer is the layer that converts the serial input to a parallel vector of length *D* (see Figure 2.2 for

TDNN seen as a FFNN with an input buffer). *D* is basically the network delay size. The network with the additional transform layer is shown in Figure 2.2. In time series forecasting application, the size of the network output is determined by the number of future samples of the series that the network will forecast. This is called prediction length in this work and is denoted as *P*. A number of hidden layers are also used with different sizes (denoted as *H*). Since our study is focused on the effect of the transform layer, and not particularly what the best configuration and network structure is for each data type, we use a single hidden layer in this work. It turns out that this configuration is capable enough for the several datasets that we experimented with.

The first layer of the TDNN parallelizes the past *D* samples and feeds them to the feed forward neural network. If a general FFNN is considered here, and not a time series in particular, the first level is basically the input layer of size *D*. For the augmented network in Figure 2.2, this input layer is followed by a transform layer which will project the *D* samples to a new domain (based on what the transform is). The transformed data is then the new input to the rest of the FFNN. The specific transforms that we have studied here include PCA, DCT and a mixed-transform of DCT and Haar.

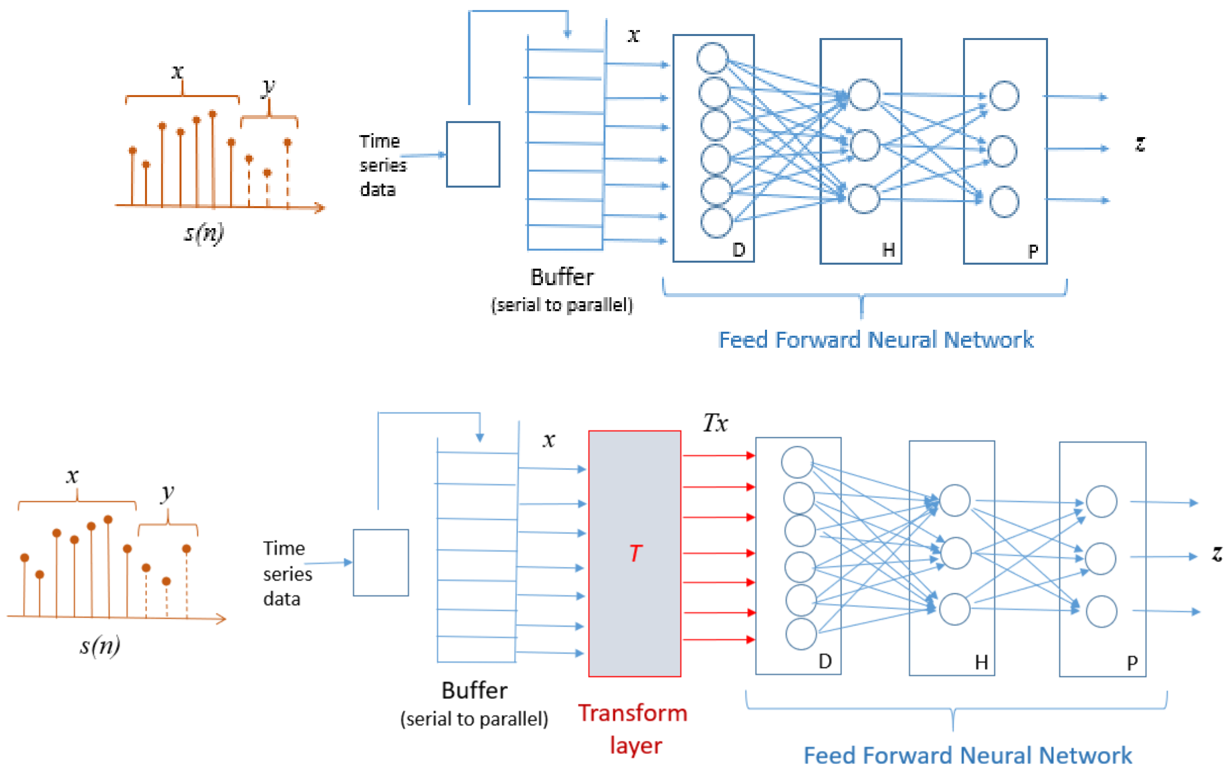


Figure 2.2: Architecture of the TDNN seen as a buffer and multi input FFNN of size *DxHxP* (Top figure). *H* is the size of hidden layer, *D* is the history length and *P* the length of prediction. As shown in the bottom a Transform layer can be inserted between the buffer and the FFNN.

2.3.1 Options for input transforms

There are several options for sparsifying data. Generally, each type of data may require a different transform for the sparser representation. Transforms based on DCT and PCA generally show the best results for the datasets that we have experimented with. In the following sections we briefly describe these transforms and how our data is represented in each domain. This list is by no means exhaustive and there may exist other options. However, for the purpose of this study on the effect of sparseness on learning rate, we have selected several transforms that produce results with different levels of sparsity. This allows a better understanding and validation of the hypothesis considered in this work. As an illustration, the result of applying these transforms to sample datasets of sea surface temperature (noted as ERSST) and car movement (noted as US101) are shown in Figure 2.3 and Figure 2.5.

2.3.1.1 Principal component analysis

The main idea of principal component analysis (PCA) is to project multi-dimensional data to a new coordinate system, with the same or different number of dimensions, with the aim of concentrating data along a fewer dimensions than the original data. This is usually achieved by deriving the bases from the data itself (as principal components). The aim of reducing dimensionality is achieved by first finding a dimension vector (component or base) along which the variance of data is maximized. The next dimensions, each orthogonal to all previously found dimensions, are then iteratively derived to maximize the variation of the residual of data orthogonal to the first dimensions. For time series data, as we consider TDNNs, each sample of the dataset is a vector of length $p=D$. This means that we can consider our data as p dimensional. By applying PCA, we can convert these vectors to vectors of a different (or same) length in a different domain. Given the nature of PCA, the new representation will have the largest variations in the first components, while the last components will have less energy. This will naturally form a sparse representation of data.

The PCA transformation can be written as $T = X.Q$, where transform Q maps data X (a matrix of all data vectors) from an original space of p variables to a new space of p variables which are uncorrelated over the dataset. Q will be a pxp matrix. With PCA, it is not needed to keep all the principal components in Q (since the representation, T, is usually sparse). In this work, however, we use all components to ensure a fair analysis and produce transformed data size equal to the original data.

PCA can be done through eigenvalue decomposition of a data covariance (or correlation) matrix or singular value decomposition (SVD) of a data matrix [18][19]. The principal components are the eigenvectors of a covariance matrix of the original data. These vectors are an uncorrelated orthogonal basis set, each being a linear combination of the variables and containing n observations.

To use SVD to calculate the score matrix (or the weights resulting from transformation), consider that the nxp data matrix of X can be written as $X = U\Sigma Q^T$,

where Σ is a nxp matrix of scalars that are the singular values of X and U is a nxn matrix of n left singular vectors of X (columns of U are the orthogonal unit vectors of length n called the left singular vectors of X). Q is a pxp matrix. Columns of Q are orthogonal unit vectors of length p, also known as principle components (PCs) are our eigen vectors (bases), which are the right singular vectors of X. Using the SVD based method the transform result, called score matrix T, can be written as: $T = XQ = U\Sigma Q^T Q = U\Sigma$.

Therefore, each of the columns of T, which are the transforms of each data samples, is given by one of the left singular vectors of X multiplied by the corresponding singular value.

In our process, PCA is applied to the dataset (the training portion of the whole dataset) to derive the principal components in Q. The resulting transform, Q, is then used as an input transform layer. Q contains the bases; each time it is multiplied to our data (buffer-delay data with size $16x1$ here) and takes the data into the new domain with the same size as the original.

Figure 2.3 and Figure 2.5 show the raw data and the data after passing through the transform. As it is observed, the scores (transform results) from PCA are very sparse, with the first component containing most of the energy. A more general way of finding bases that allow sparse representation of a signal is through dictionary learning (DL) [7]. There are several different dictionary learning algorithms which are essentially optimization schemes that try to find bases that sparsify data to a specified level. Dimensionality reduction could also be achieved and is one of the main objectives of most DL algorithms. However, in this work we are interested in keeping the dimension. It was observed that the available DL algorithms could not achieve higher sparsity than DCT for the datasets used here, and under the condition of maintaining the same dimensionality. Therefore, we did not include the results from DL here.

2.3.1.2 *Discrete Cosine Transforms*

Discrete cosine transforms (DCTs) is one of the sinusoidal transforms. Similar to other transforms in this family, it is an invertible linear transform. Sinusoidal transforms use kernels that are defined by a set of complete, orthogonal discrete cosine and/or sine basis functions. Generalized Discrete Fourier transform (DFT) and various types of the DCT and DST (Discrete Sine Transform) are members of this class of unitary transforms. The complete set of DCTs and DSTs consists of eight versions of DCT and corresponding eight versions of DST. Being a discrete transform, DCT is a finite sequence of data points, as a sum of cosine functions at different frequencies. DCT has been used for lossy compression (for example for images) due to its strong property in compacting energy. For example, with DCT-II transform, the coefficients (transform results) for a signal *x* of length *N* are calculated as follows:

$$X_c(\mathrm{k}) = \sum_{n=0}^{N-1} x[n].\cos[\frac{\pi k}{N}(n + 1/2)] \qquad (2.1)$$

Where *k=0,1, 2, …, N-1*. For the datasets studied here, DCT has proven to be a very strong sparsifying transform. We can further multiply the X_0 term by $1/\sqrt{2}$ and multiply the resulting matrix by an overall scale factor of $\sqrt{2/N}$ which makes the DCT matrix orthogonal. Note that the DCT basis are independent of dataset used. We have used DCT-II from MATLAB. The DCT matrix coefficients will look like as in (2.2).

$$C = \sqrt{\frac{2}{n}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\ \cos\frac{\pi}{2n} & \cos\frac{3\pi}{2n} & \cdots & \cos\frac{(2n-1)\pi}{2n} \\ \vdots & \vdots & \cdots & \vdots \\ \cos\frac{(n-1)\pi}{2n} & \cos\frac{(n-1)3\pi}{2n} & \cdots & \cos\frac{(n-1)(2n-1)\pi}{2n} \end{bmatrix} \qquad (2.2)$$

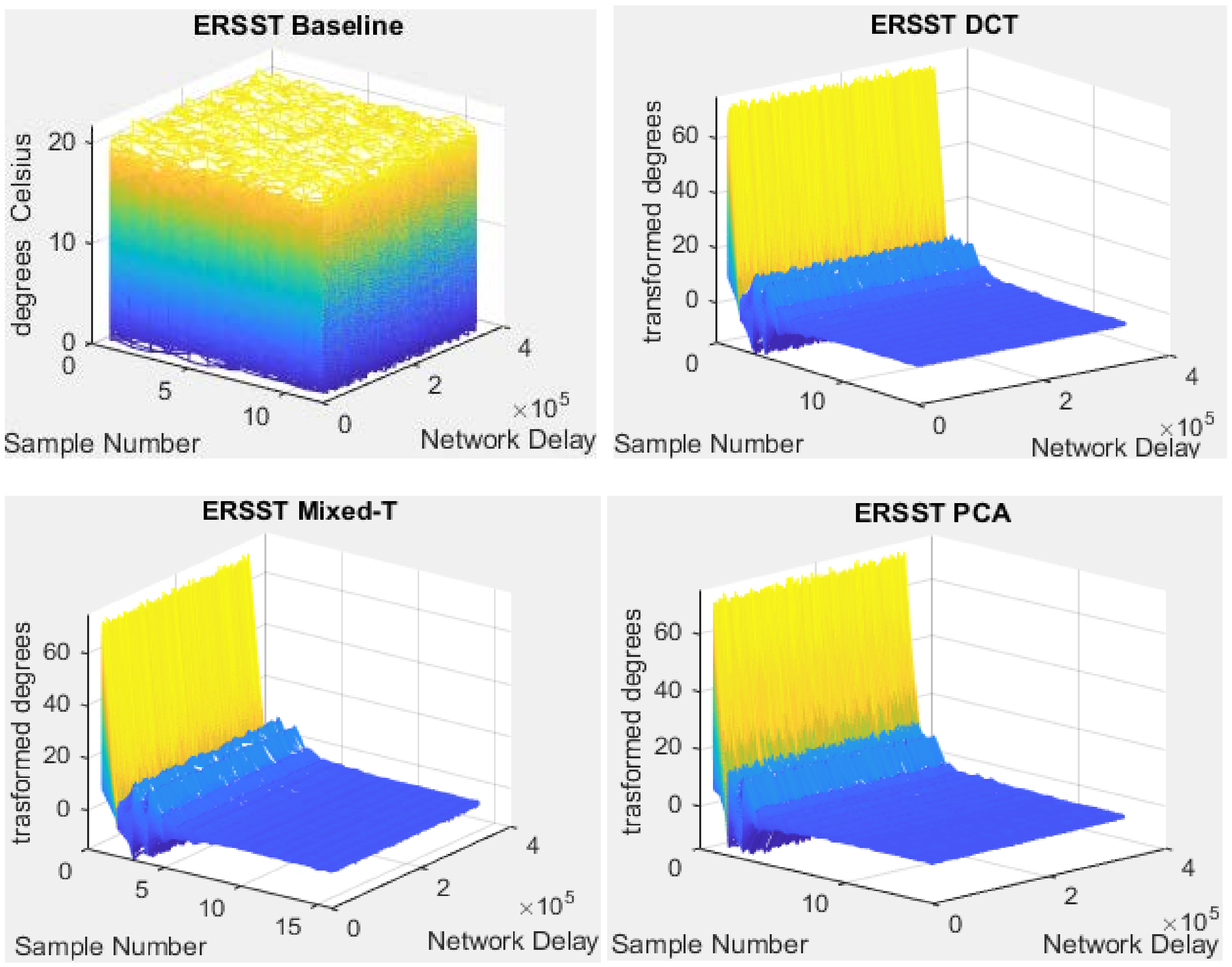


Figure 2.3: Visualizing effect of sparsifying transforms; showing original data of ERSST dataset arranged in a matrix of input vectors (top left) and its transformations: DCT (top right), mixed-transform (bottom left), and PCA (bottom right).

2.3.1.3 *HAAR Transform*

The Haar Transform, or the Haar Wavelet Transform (HWT) is one member of a group of transforms known as the Discrete Wavelet Transforms (DWT). Haar uses non-sinusoidal basic wave functions, providing a simple and computationally efficient approach for analyzing the local aspects of a signal [36][37]. The Haar Wavelet Transform (HWT) is from Discrete Wavelet Transforms (DWT) and is not symmetric. Wavelets are especially useful for compressing image data. Fourier analysis breaks up a signal into sine waves of various frequencies. Similarly, wavelet analysis is the breaking up of a signal into shifted and scaled versions of the original (or mother) wavelet, equation (2.3).

$$\psi^{a,b}(x) = |a|^{-1/2}\,\psi\left(\frac{x-b}{a}\right). \tag{2.3}$$

DWT has a key advantage over Fourier transforms, it captures both location (time) and frequency information and also capture discontinuity in the signal. The Haar transform matrix is real and orthogonal. Before normalization the Haar matrix for N=8 will look like as in (2.4).

$$H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{2.4}$$

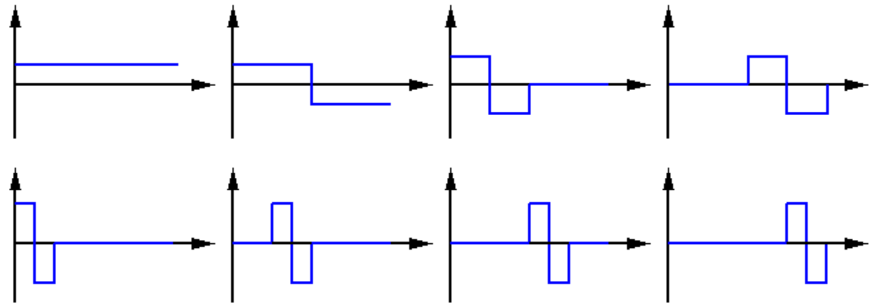


Figure 2.4: Illustrating Haar basis functions for N=8, source: fourier.eng.hmc.edu /e161 /lectures /Haar/index.html

2.3.1.4 *Mixed Transform*

Mixed transform is a method of applying multiple domain transforms to data to provide a better representation. Usually multiple sparse representations are applied; however, depending on data characteristics, the final mixed transform result may or may not be sparser than either of the single transforms (it may generally have similar sparsity levels). Nevertheless, mixed transforms are shown to be more efficient in achieving higher compression levels [1]. In this work, we use mixed transform of DCT and Haar as an alternative to DCT. An overview of how mixed transform works is given here.

For mixed transform the reason to use DCT and Wavelet (Haar) is that DCT is known to be very efficient in representing narrowband signals. On the other hand, Haar and Walsh transforms are considered appropriate for wideband signals with sudden changes. In general, combining DCT and Haar in a mixed-transform setting is expected to be a more efficient way of representing a signal [1]. The mixed-transform method uses subsets of basis functions of two or more mutually non-orthogonal transforms to represent data. Our earlier work in [1] finds the optimal balance of bases from DCT and Haar for data representation by minimizing an energy function; however, here we take a simpler approach here to avoid complexity in applying the mixed transform.

For example, consider a signal X. If we denote the DCT transform bases as K, and Haar as H, we first derive coefficients from DCT transform as $C_1=KX$. Then, the larger coefficients in C_1 (e.g., first half) are kept and the rest set to zero to get C_2. We then apply the Inverse DCT to C_2. i.e., $X_2 = K'C_2$ and find the error $E = X-X_2$. The error is then fed to the second transform to find its representation in the Haar domain $V_1 = HE$. The largest coefficients from V_1 are then kept and truncated in a final result vector that contains the largest coefficients of C_1 and V_1. More details about mixed transform can be found in [3][1]. Depending on how many coefficients from each transform is kept, the size of the resulting vector may be smaller or larger than original data. In this work, and for a fair comparison with the original (non-transformed) data, the mixed-transform is configured to produce results with the same size of original data, therefore keeping the size of the TDNN unchanged.

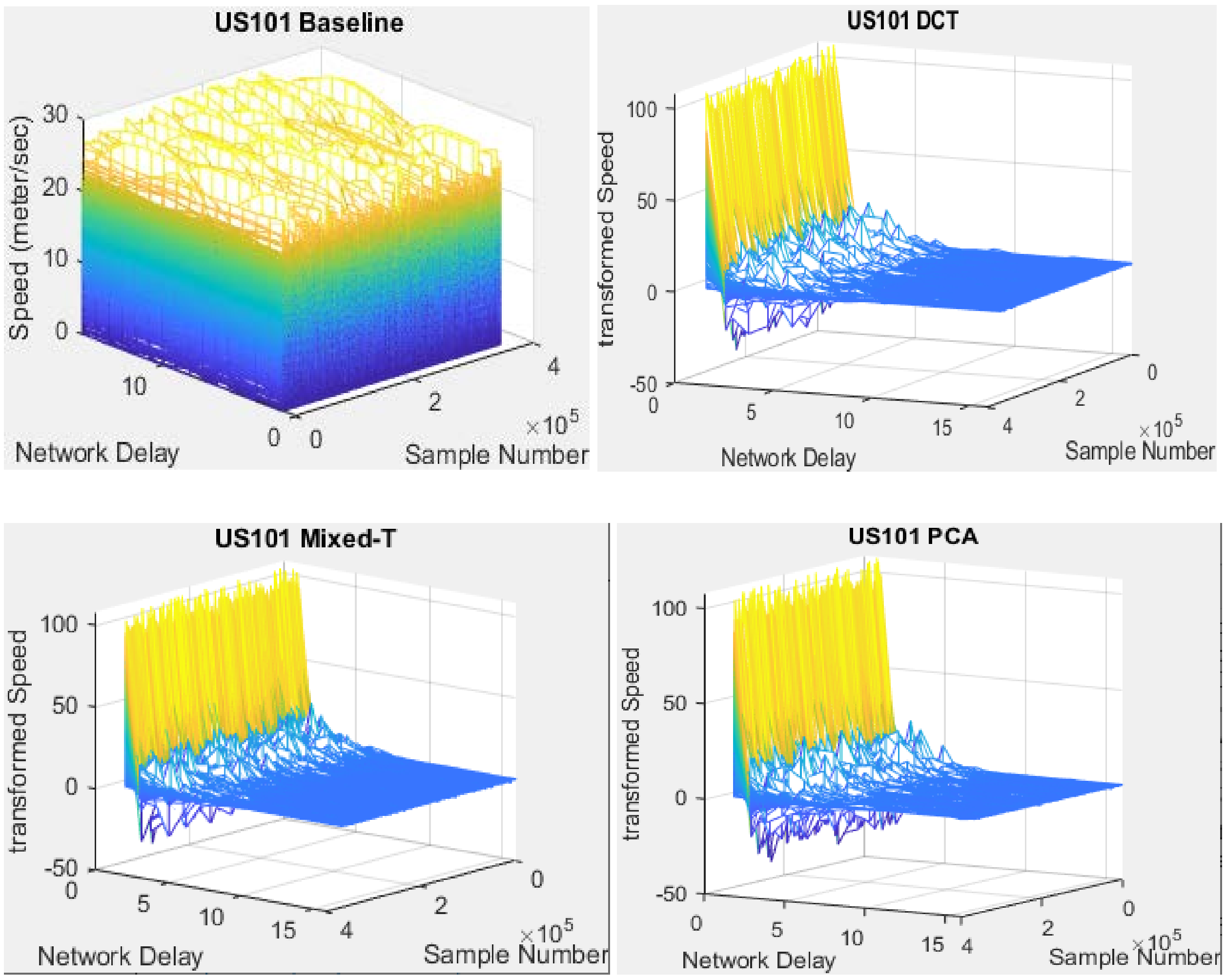


Figure 2.5: Visualizing effect of sparsifying transforms; showing original data of US101 dataset arranged in a matrix of input vectors of size 16 (top left) and its transformations: DCT(top right), mixed-transform (bottom left), and PCA (bottom right).

We note that it is also possible to use a smaller set of transform coefficients (e.g., similar to compression using DCT); however, we do not consider such a case as we observed that it reduces the accuracy of the system (as some data is lost during compression).

The results of applying a DCT-Haar mixed transform that keeps 50% of the largest coefficients of each transform is shown in Figure 2.3 and Figure 2.5.

2.4 Mathematical Analysis

In this section we analyze the mathematical properties of TDNN learning algorithms that are affected by sparsity of data. We show that if input data is made sparser, the learning process is accelerated. Here we examine how the gradient descent-based backpropagation methods are impacted. It is seen that the sparseness of data leads to a fewer number of weights in the network to be reactive to the backpropagation process. This will result in an optimization process (another way to see the learning process) that has a smaller search space. The gradient descent-based methods are used in several algorithms, including the commonly used Levenberg-Marquardt algorithm, which is used in out tests.

Consider a feedforward neural network, such as a TDNN, as in Figure 2.6. As shown later in this section, we know that each weight in the gradient descent is updated by the derivative of a loss function (error) *J* with respect to that weight [69]. Here, ${W_{ij}}^{t}$ is the value of weight from neuron *i* to *j* at time step *t*:

$$ {W_{ij}}^{t+1} = {W_{ij}}^{t} - \frac{\partial J}{\partial w_{ij}} \tag{2.5} $$

Error *J* can be calculated as the difference of network output value and the desired target value (*T*). If output is a vector, then this could be the Mean Squared Error (MSE) of the values. It is possible to show (elaborated below) that the derivative of loss function with respect to each weight is proportional to the input to that weight; i.e., $\frac{\partial J}{\partial w_{ij}} \propto x_i$. Here, x_i is the i^{th} input. Therefore, when data is sparse and some x_is are close to zero, their corresponding weights do not receive much updates and the weight update will be concentrated on non-zero inputs. With sparse data, the non-zero inputs are few. This means that the optimization that happens during training is done over a smaller number of parameters (weights). This is the reason for faster training, as the search space for the optimization is much smaller.

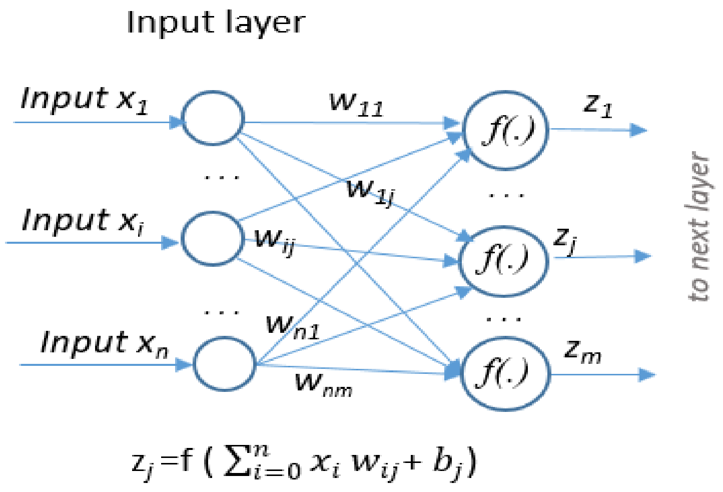


Figure 2.6: The input layer and the first layer of a feedforward network

Now, to see why the relationship of $\frac{\partial J}{\partial w_{ij}} \propto x_i$ holds, consider the example network in Figure 2.6. This is the first layer of a FFNN. Suppose $W \in R^{nxm}$ represents all the weights. The training algorithm computes the gradient of loss function J with respect to a matrix $W \in R^{nxm}$. We can think of J as a function of W taking nxm inputs (the entries of W) to a single output (J). This means the Jacobian $\partial J/\partial W$ can be represented as:

$$\frac{\partial J}{\partial W} = \begin{bmatrix} \frac{\partial J}{\partial W_{11}} & \cdots & \frac{\partial J}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial J}{\partial W_{n1}} & \cdots & \frac{\partial J}{\partial W_{nm}} \end{bmatrix} \tag{2.6}$$

Since this matrix has the same shape as *W*, in a gradient descent method we can simply subtract it (times the learning rate) from *W*:

$$W^{t+1} = W^{t} - \frac{\partial J}{\partial W} \tag{2.7}$$

Where *t* is the iteration number. For individual weights we can write:

$$W_{ij}{}^{t+1} = W_{ij}{}^{t} - \frac{\partial J}{\partial w_{ij}} \tag{2.8}$$

Now, if we assume that the network functions are simple weighted sums (we look at more complex case later in the work), we have the output z_k as:

$$z_k = \sum_{l=1}^{n} W_{lk} x_l \tag{2.9}$$

Consequently, we have

$$\frac{\partial z_k}{\partial W_{ij}} = \sum_{l=1}^{n} x_l \frac{\partial W_{lk}}{\partial W_{ij}} \tag{2.10}$$

Note that for $i=l$ and $j=k$, we have $\partial W_{lk}/\partial W_{ij} = 1$; otherwise, $\partial W_{lk}/\partial W_{ij} = 0$. So, if $l \neq i$ all the related terms in the sum are zero. The only non-zero element of the sum is when $i=l$ and if $k = j$, so we just get x_i as the gradient value. Thus, we find $\partial z_k/\partial W_{ij} = x_i$ if $k = j$ and 0 if otherwise.

$$\frac{\partial z_k}{\partial W_{ij}} = \begin{cases} x_i & k = j \\ 0 & o.w. \end{cases} \quad \text{for k=1…m} \tag{2.11}$$

Another way of writing this for vector z (all outputs k)

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W_{ij}}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_i \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{j'th element} \tag{2.12}$$

Therefore, we can compute $\partial J/\partial W_{ij}$ as

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial z} \cdot \frac{\partial z}{\partial W_{ij}} = \boldsymbol{\delta} \cdot \frac{\partial z}{\partial W_{ij}} = \sum_{l=1}^{m} \delta_l \frac{\partial z_l}{\partial W_{ij}} = \delta_j x_i \tag{2.13}$$

For some vector $\boldsymbol{\delta}$. As it is seen in equation (2.13), the gradient for each weight is dependent on the corresponding input. In cases where the data is sparse, some of the inputs are close to *0*, leading to small values of gradient and very slow updates. This allows the larger inputs to dominate the training and as a result a smaller search space forms, leading to faster learning.

Now, if we consider a more general neural network with non-linear activation functions, each output in Figure 2.6 network can be basically seen as a function *f* of the weighted sum of inputs and bias (*f* being the activation function such as sigmoid or tanh):

$$z_k = f(\sum_{l=0}^{n} x_l\, W_{lk} + \; b_k) \tag{2.14}$$

The gradient descent-based update algorithm tries to minimize the loss *J* through updating the weights as in equation (2.7). Here, with the activation function being considered, we can rewrite (2.13) as, denoting z_k of equation (2.14) as f(.) for brevity:

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial z} \cdot \frac{\partial z}{\partial W_{ij}} = \boldsymbol{\delta} \cdot \frac{\partial z}{\partial W_{ij}} = \sum_{k=1}^{m} \delta_k \frac{\partial z_k}{\partial W_{ij}} = \sum_{k=1}^{m} \delta_k \frac{\partial f(.)}{\partial W_{ij}} \tag{2.15}$$

Where $\boldsymbol{\delta}$ is a vector describing how *J* changes with respect to changes in each output of this layer (in output vector *z*). To find $\frac{\partial z_k}{\partial W_{ij}}$, or $\frac{\partial f(.)}{\partial W_{ij}}$, we note that the output of each neuron is basically the activation function applied to the weighted sum of inputs and the bias. If we consider the weighted sum as another function *h*, we have:

$$z_k = f\,(\sum_{l=0}^{n} x_l\, W_{lk} + b_k) = f(h_k(.)) \tag{2.16}$$

To simplify the notations and since we are only interested in the derivative of z_k with respect to W_{ij}, we will denote $h_k(.)$ as h_k (W_{ij}). Therefore, we can rewrite this as $z_k = f(h_k(W_{ij}))$. Using the chain rule, we can then derive the derivative as follows:

$$\frac{\partial z_k}{\partial W_{ij}} = \frac{\partial f\big(h_k(Wij)\big)}{\partial W_{ij}} = \frac{\partial f}{\partial h_k} . \frac{\partial h_k(Wij)}{\partial W_{ij}} \tag{2.17}$$

Where the derivative of *f* with regard to h_k is the slope of *f* at some value of u= $\sum_{l=1}^{n} x_l W_{lk} + b_k$. Given the shape of *f* (for example a sigmoid), this is a bounded scaler c_k:

$$\frac{\partial f}{\partial h_k}|_u = \{slope\ of\ f(h_k)\ at\ h_k = u\} \quad = c_k \tag{2.18}$$

On the other hand, we had shown that for the weighted sum as in equations (2.9) to (2.12), the only non-zero value for the derivative with respect to W_{ij} was found when $k=j$. That is, we found $\partial h(k) / \partial W_{ij} = x_i$ if $k = j$ and *0* otherwise (note that h_k is in fact z_k of equation (2.10)); therefore, equation (2.17) can be rewritten as:

$$\frac{\partial z_k}{\partial W_{ij}} = \begin{cases} \frac{\partial f(.)}{\partial W_{ij}} = \frac{\partial f}{\partial h_k} . \frac{\partial h_k}{\partial W_{ij}} = c_k x_i & k = j \\ 0 & k \neq j \end{cases} \tag{2.19}$$

Now, by substituting (2.19) in (2.15), and noting that $\frac{\partial f(.)}{\partial W_{ij}} = c_j x_i$ only for $k=j$, and zero otherwise, we find:

$$\frac{\partial J}{\partial W_{ij}} = \sum_{k=1}^{m} \delta_k \frac{\partial f(.)}{\partial W_{ij}} = \delta_j c_j x_i \tag{2.20}$$

As it can be seen, only one finite term (c_j) has been added to equation (2.13). With sparse input, many x_i's have a very small value close to zero. Therefore, given the finite values for δ_j and c_j, for those *j*'s the derivative $\partial J/\partial W_{ij}$ becomes close to zero, meaning that the update to weights will be very small and insignificant. On the other hand, for the large x_i's, the derivative may not be small and weight changes do happen; as a result, the optimization and gradient descent will be more concentrated around the weights that correspond to large values of x_i, which are few for sparse data. The optimization steps are therefore taken in a smaller space. That is, instead of reaching the global minimum equally with small steps from everywhere, the minimum can be reached within fewer steps and epochs through only optimizing a few weights. It turns out that the results are also equal or better when data is sparsified, which points to the fact that the same minimum for the loss function is found in both sparsified or original data case.

This finding can be observed from the evolution of weights in the training process. We have plotted the *Hinton weight bias plot* of the network weights and biases in Figure 2.7. The Hinton plot shows the value of all parameters (weights and biases) of a specific network at any time that the parameters are sampled for observation. We have logged the values of these parameters after training epochs 3, 7 and 14. Each dot (small square) in the plot corresponds to a specific weight or bias. The size of the squares shows the magnitude of the parameter and the color shows positive (green) or negative (red) values. It is seen that with the baseline (original nonsparse) data, most weights change value during the training process. Whereas, with the sparsified input (Figure 2.8), only a small number of weights have significant updates during the process and quickly converge as seen in Figure 2.8. In particular, the weights connecting input to layer 1 for inputs number 1 to 6 show more changes, compared to other weights.

For more insight into how the weight values evolve when a sparsifying layer is introduced to a TDNN, we also plotted some sample weights in Figure 2.9. Here, we see that for a fixed number of epochs, the weights for TDNN with DCT transform quickly converge to their desired values, whereas for a TDNN with no input transform, some weights do not converge even at epoch 20. The later convergence means that at fixed number of epochs, the TDNN with no transform will produce higher forecasting errors than the one with DCT input transform layer. It is also observed that for the TDNN with DCT, the weights associated with close to zero values in DCT domain (e.g., from input 10 and 16, see Figure 2.5) tend to stop changing after first few iterations, leaving the training to weights for larger inputs (e.g., inputs 1 and 5). The next section studies the impact that several different transform options may have on learning rate or accuracy.

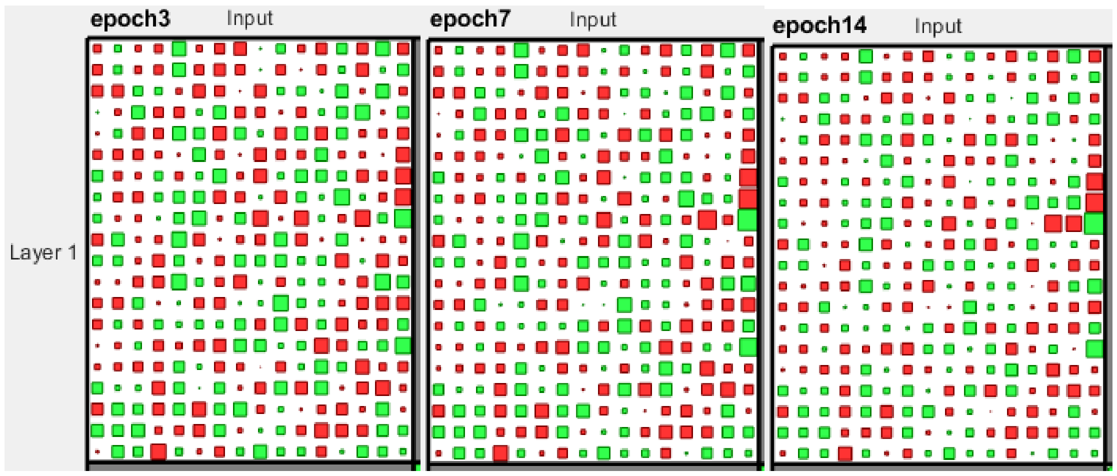


Figure 2.7: Hinton weight bias plot at epochs 3, 7 and 14 (left to right) for TDNN with no input transform layer

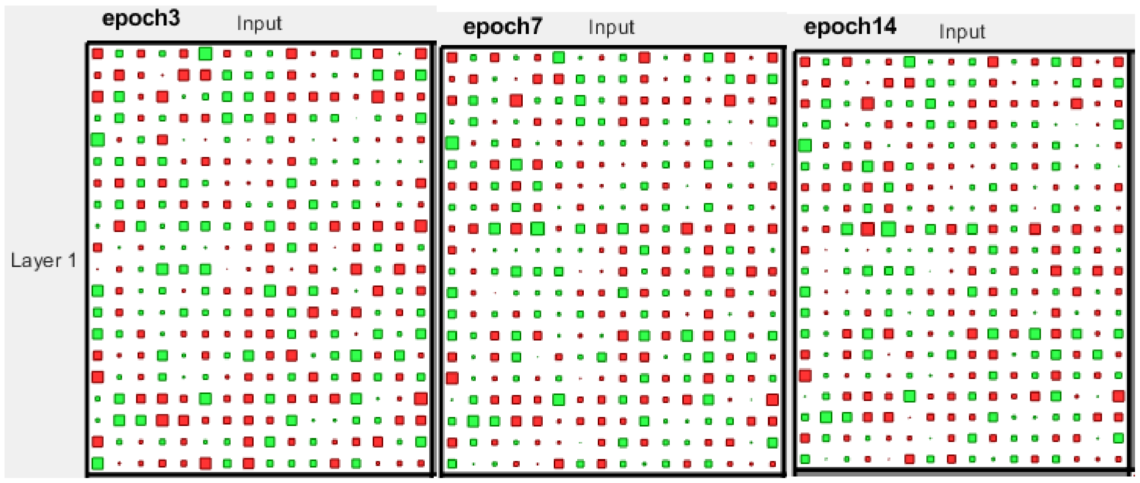


Figure 2.8: Hinton weight bias plot at epochs 3, 7 and 14 (left to right) for TDNN with DCT input transform layer

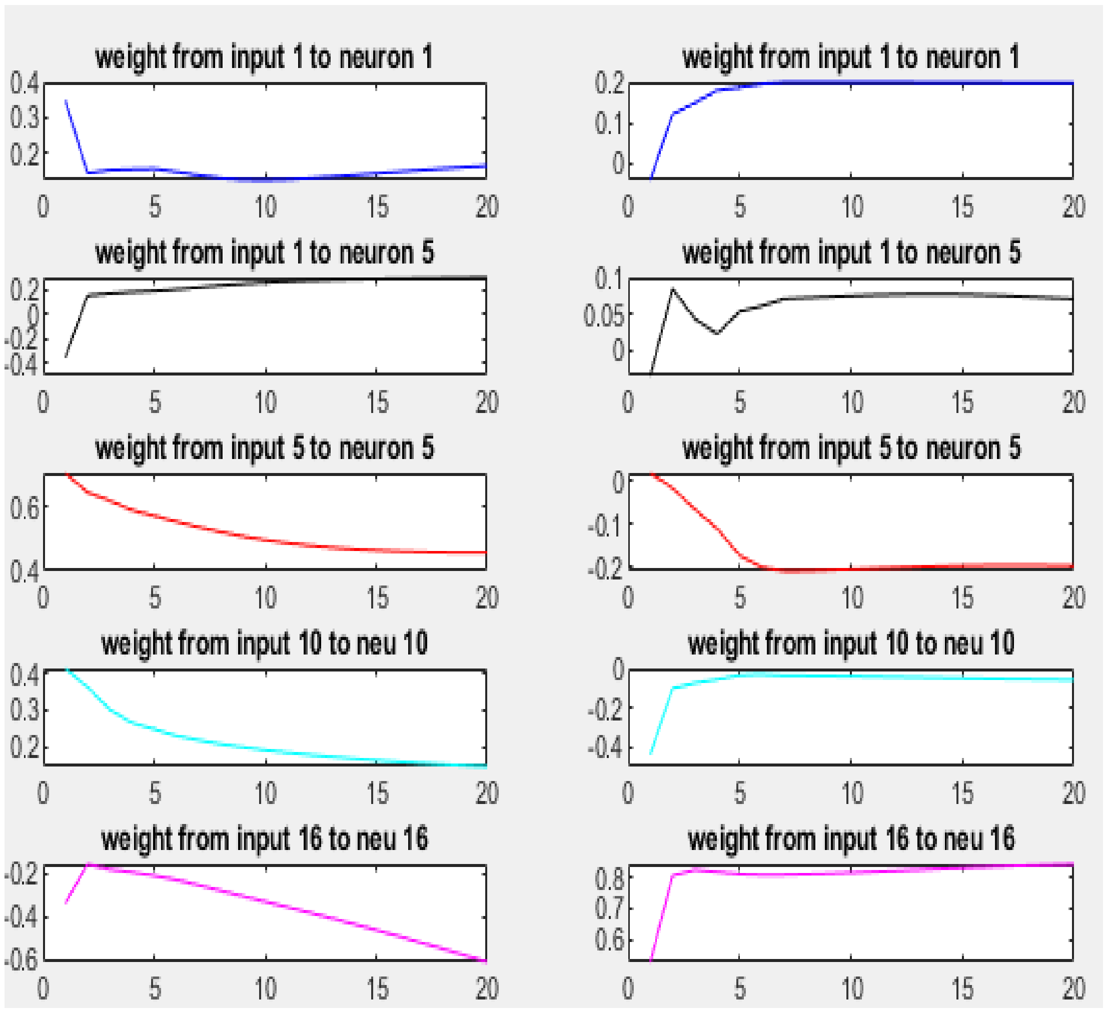


Figure 2.9: Comparing how weights evolve in time in the first 20 epochs for TDNN with DCT input transform (right) and TDNN with no transform (left). Only some sample weight values are shown due to space limitation.

2.5 Evaluation with different transforms

In this section, we use several datasets to show how different transforms, or in other words, different levels of sparsity, affect the learning rate or performance (accuracy of forecasting). We look at two metrics of prediction error and learning time. For prediction error we look at the error of forecasting; time series data is passed through the network and the error of prediction at different future times are measured. For each sample at time t in the dataset, there will be a forecast and an error instant for each future time (e.g., t+1 or t+10). For a given future prediction, we then take the 95-percentile of the error instances and report that as the error metric (e.g., 95-percentile of prediction at time t+10). Results for other percentiles or future times are observed to be similar, so we do not repeat those results here. We use the 95-percentile (e.g., instead of median) as it is commonly used for some of the datasets that we use [21]. The trends are similar for median, mean or 95%.

The second metric that we consider is the training time (number of epochs or iterations of training). This metric is an indicator of how fast the training algorithm works. While there are several options, we work with the most popular algorithm, which is the Levenberg-Marquardt backpropagation [22]. It has been shown [23] that this algorithm provides the fastest convergence for moderately sized feedforward neural networks. We note that training time and accuracy (or error) of results are related. Training parameters could be set such that error becomes lower, at the cost of longer learning time, and vice versa.

To clearly see the effect of sparsifying data, we experiment with two approaches of fixing learning parameters and then measuring time, or fixing the learning time (number of epochs) and then measuring error. For the first approach, we have observed that due to several parameters affecting training time (like learning performance/error threshold or threshold of failed learning steps) the final error is not the same for different methods. This results in both different learning time and error for each choice of domain transform. This was reported in our earlier work in [2]. Additionally, it is observed that methods that are faster in learning, are also somewhat better in prediction (have slightly lower error). This point is further validated when we fix the number of epochs (learning time) and then compare the errors (approach 2). The two approaches consistently point to the faster learning for sparser data.

The experiment settings and parameters are explained in this section. We used the default setting of initial μ=0.001, μ decrease factor=0.1, applied after 10 epochs. We also used the MATLAB implementation of the backpropagation and the neural network toolbox.

For the purpose of comparison with the existing state of the art methods, we note that each dataset has a different state-of-the art for time series forecasting (since methods are highly customized to data). For the NDDS and US101 datasets, the state of the art in industry is the prediction method based on constant velocity cruising prediction (CVP) [66] which we also compared to conventional neural network baseline in our earlier work [2] . Figure 2.10 shows how the CVP compares to neural network-based methods. One of the ANN-based methods is the conventional TDNN that we have optimized for these data sets and can be considered as the state of the art in ANN-based methods and used as a baseline. Comparison is done on prediction accuracy for all methods (Figure 2.10) and on training time for ANN-based methods (Figure 2.11).

For the ERSST dataset, the state of the art in methods that are not based on neural networks is the seasonal ARIMA (SARIMA)[67]. While we considered this method, we have also optimized a conventional neural network for temperature prediction and use that as a baseline of ANN-based methods for comparison as well. The results are shown in Figure 2.10 for prediction accuracy of all methods, and in Figure 2.11 for training time and accuracy of neural network specific methods. The superiority of ANN based methods can be seen in Figure 2.10.

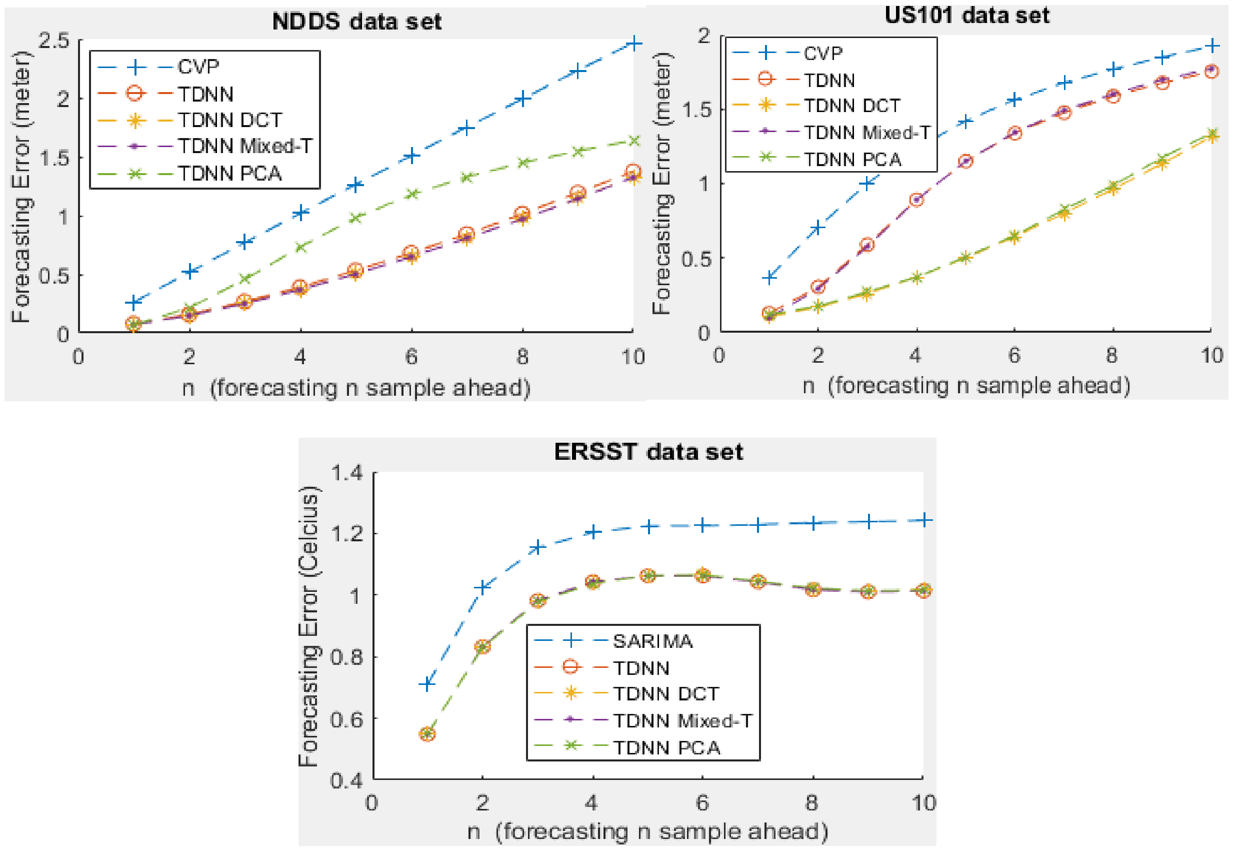


Figure 2.10: Comparing forecasting error for *n* samples ahead (n=10) using different estimators and for different datasets. The US101 and NDDS data set have been compared with CVP[66]; all TDNN based methods outperform CVP, with methods using DCT and PCA input transforms showing best results. For ERSST data set, seasonal ARIMA method is compared to TDNN based methods with different input transforms or none. All results for TDNNs are very similar and overlap and show considerable improvement over seasonal ARIMA (SARIMA)[67].

For NN based methods, we fix the network configuration for all cases to avoid unfairness in comparison. It is important to note that our concern is not finding the best forecasting method, instead our concern is to study the possible improvements to neural network training and performance using input transforms and TDNNs in general. Therefore, the baseline method may not include all possible improvements that one might possibly find for a given dataset. For example, one might create a very large neural network that produces better results, but with significant computational cost. Such cases will not satisfy the requirements for small computing devices which are the target platform of this work. Our goal, here, is to show the difference between sparsifying and not sparsifying data in FFNN or TDNN. For each dataset, we have experimented with many different network configurations and are reporting results from the network setups that produce the best results under baseline.

In all these datasets, we use 70% of samples for training and the remaining 30% for testing.

The best network size for the TDNN for ERSST data was found to be *16x20x10* (for an *IxHxO* network, where *I* is the input size, *H* is the hidden layer size and O is the output layer size). For the synthetic data, the network size of *16x15x10* and *16x20x10* were the best option respectively for synthetic line and synthetic curve. The TDNN size used for US101 and NDDS datasets in this section was *16x20x10*. We have found these network sizes to produce the best prediction results after many trial and error investigations. However, the network configuration is not the primary concern here as we are interested in relative performance and the effect of input transforms and sparsity. In an earlier work reported in [2], it was shown that the trend reported on the improvement due to transforms remains similar at different network sizes. For the sparsifying transforms, we have selected three options of DCT, mixed-DCT/Haar, and PCA. We have run the algorithms ten times and reported the average for the metrics described earlier.

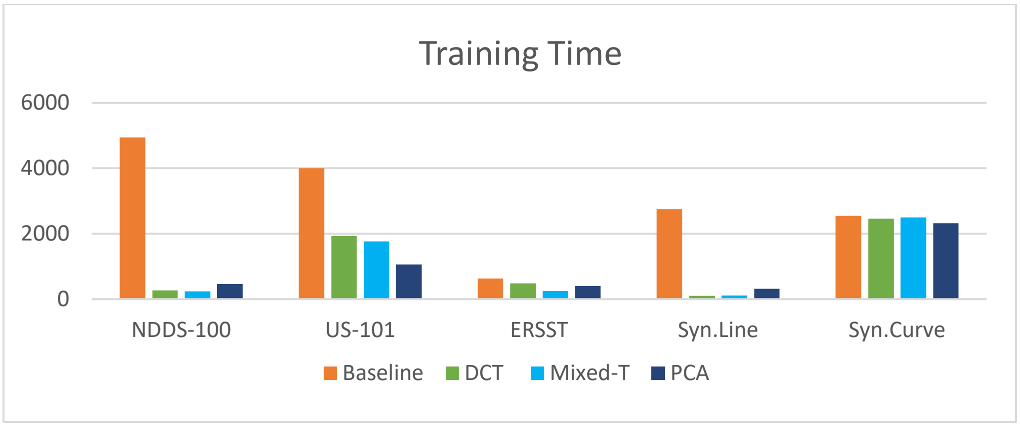
In the first set of comparisons the system parameters are fixed and we measure the learning time and prediction error. Here the stopping criteria of ML algorithm is unchanged and it is left to the algorithm to stop training. The maximum allowed number of epochs was set to 1000. Results for this experiment are reported in Table 2.1 and summarized in Figure 2.11. The result indicates the time series prediction error for future sample time of t+10. The time that it took the MATLAB implementation to complete the training is also reported. A core i-9 3.5GHz CPU was used for these tests. It is observed that in all datasets, the methods employing the sparsifying transform produce lower error, while taking considerably shorter amount of time to end the training. For example, the training time for baseline method produces around 3.3% higher error and takes over 20 times (2000%) longer to complete for NDDS dataset, compared to results from DCT transformed inputs. This is a significant improvement in learning time and can be observed at different levels in Table 2.1.

Table 2.1: Training time and accuracy for different transforms (max number of epochs set to 1000). Error is for prediction error at time t+10; prediction error improvement compared to baseline is shown in parenthesis under error.

Dataset	NDDS-100		US101		ERSST		Syn. line		Syn. curve	
method	Error	time	Error	time	Error	time	Error	time	Error	time
Baseline	0.52	4945	1.140	4009	1.062	5723	0.60	2748	0.038	2550
DCT	0.49 (%3.4)	263	1.129 (0.1%)	1927	1.062	1425	0.60	97	0.019 (50%)	2463
Mixed-T	0.49 (3.3%)	237	1.099 (3.5%)	1763	1.066	1702	0.60	108	0.017 (55%)	2495
PCA	0.50 (3%)	464	0.987 (13.5%)	1059	1.062	1753	0.605	316	0.017 (%55)	2322

For some datasets, such as US101, we see from Table 1 that the PCA method produces results that have 13.5% lower error than baseline. For the same scenario, DCT produces 3.5% improvement. The better performance of PCA corresponds to the sparser input that it produces, as is observable in Figure 2.5. In this figure, we see that PCA transformed input vectors have around 5 non-zero values (out of 16 data points in each input). This number is around 9 for DCT; thus, the sparser data out of PCA seems to produce better results than the slightly less sparse output of DCT.

Note that the much longer training time is observed to be due to the learning algorithm operating in a relatively flat area of the error surface. We suspect that even with the baseline, i.e., original non-sparse data, the performance would probably reach an acceptable level after a short while. Nonetheless, it is expected that sparse methods still perform better even at small number of epochs. To investigate this point, in the second experiment approach, we fix the training time to a small number of epochs and observe how different methods perform.



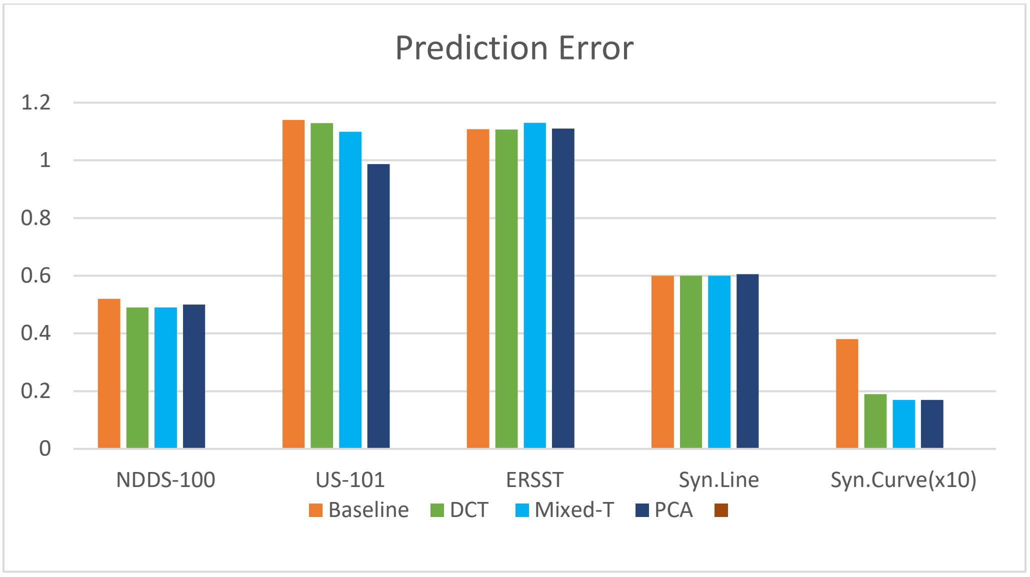


Figure 2.11: Training time in seconds (left) and prediction error for different transforms and datasets. The maximum epoch was set to 1000. Note that the values in Table 2.1 for synthetic curve have been multiplied by 10 so that they can be visible in this figure.

Table 2.2, Table 2.3 and Figure 2.12 report the results for 25 and 10 epochs respectively. It is observed that the prediction error for sparse methods show improvement in most cases. Also, the amount of improvement seems to be directly related to how sparse the result of input transform is. For example, PCA in Table 2.3 produces the sparsest results for NDDS and US-101, and consequently the improvement in prediction error is higher (up to 7.5%). DCT and mixed-transform also produce very sparse results and therefore better accuracy at epoch 25. Synthetic curve data shows the difference in a more marked way, as expected. The synthetic curve data was designed to have 2 large sinusoids, which are easily found by DCT, mixed transform, and PCA. As a result, there is a 7.5% improvement.

It must be noted although PCA provides better results in several cases, PCA bases are data dependent. PCA requires analysis of the dataset prior to learning to derive the appropriate transform (bases). on the other hand, DCT does not require preprocessing and is not data dependent. As a result, the DCT maybe the preferred method.

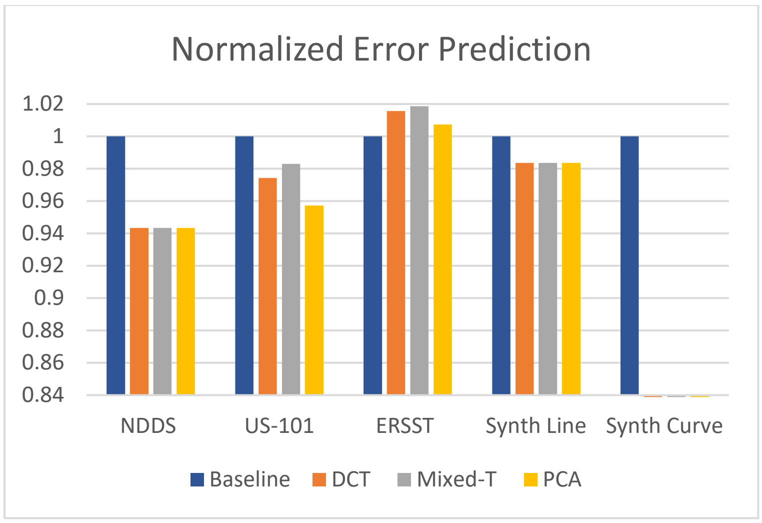
ERSST dataset does not show a marked improvement at 25 epochs, mainly due to the fact that data had a very obvious periodic pattern (annual pattern of temperature) that could be learnt by all methods quickly; nevertheless, from Table 2.1 it is seen that if training was let to continue, it would take almost 40% less time to achieve an accuracy that is also 1% better. The improvements in more randomized data, such as trajectories in NDDS and US101, are more obvious.

Table 2.2: Forecasting Error and improvement for 10 timesteps ahead (number of epochs set to 10)

Transform \ Data set	NDDS-100 (m)	US-101 (m)	ERSST (degrees Celsius)	Synth	Synth Curve
Baseline	0.53	1.18	1.079	0.61	0.072
DCT	0.50 (5.6%)	1.14 (3.4%)	1.096	0.60	0.037(49%)
Mixed-T	0.50 (5.6%)	1.15 (2.5%)	1.099	0.60	0.037(49%)
PCA	0.50 (5.6%)	1.13 (4.2%)	1.087	0.60	0.042 (41%)

Table 2.3: Forecasting Error and improvement for 10 timesteps ahead (number of epochs set to 25)

Transform \ Data set	NDDS-100 (m)	US-101 (m)	ERSST (degrees Celsius)	Synth line	Synth Curve
Baseline	0.53	1.18	1.133	0.61	0.040
DCT	0.50 (5.6%)	1.13 (4.2%)	1.12 (1.1%)	0.60(1%)	0.036 (10%)
Mixed-T	0.50 (5.6%)	1.12 (5%)	1.14	0.60(1%)	0.037(7.5%)
PCA	0.49 (7.5%)	1.13 (4.2%)	1.12 (1.1%)	0.60(1%)	0.037 (7.5%)



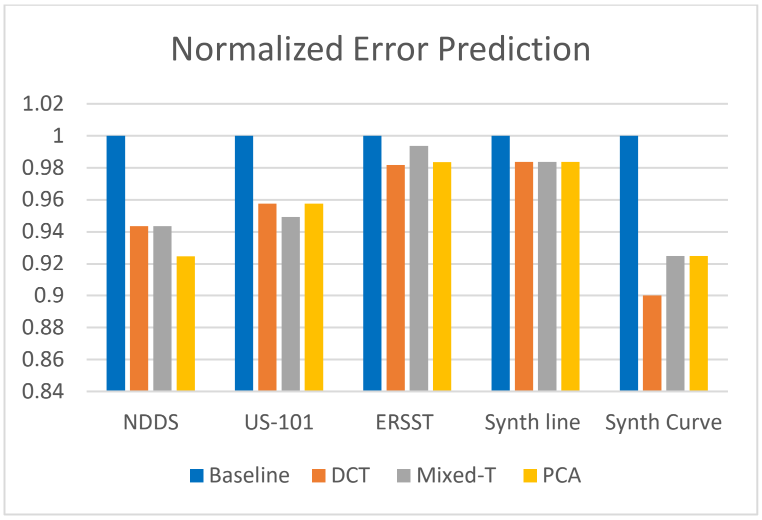


Figure 2.12: Normalized prediction error at epoch 10 (left) and epoch 25 (right) for different datasets and transforms. Prediction errors are normalized to baseline error in each scenario. Note that for synthetic curve scenario in the left plot the values of error for DCT, Mixed-T and PCA are below the range shown and are not visible here.

As discussed in the mathematical analysis section, the improvement in accuracy or training time for sparsified data can be attributed to the reduction in size of the search space for the training algorithm. From another perspective, when DCT is used as the input transform, each input of the TDNN is mapped to a specific frequency. Therefore, the TDNN will be learning the frequency components, rather than time domain values. With the input transform layer, the weights corresponding to each frequency component will have to adapt to a smaller range of values compared to the time domain TDNN, in which all possible values in time may be observed by each weight. The concentration of changes in the near DC frequencies, and smaller range of values for most other components leads to a smaller search space, as was also mathematically shown in section 2.4.

A different perspective on the very long training time of the baseline method is that the optimization error function inside Neural network will be stuck on a flat area. The error function surface for sparsified data is perhaps steeper, as error performance quickly converges and the ML algorithm ends when it oscillates around a minimum.

2.6 Summary and Concluding Remarks

In this work, the impact of sparsifying input data of a TDNN has been studied. A sparsifying input transform layer is applied to the input of the network, transforming data using to domains such as DCT, PCA or a mixed-transform domain. Through mathematical analysis of the backpropagation and gradient descent-based methods, it is shown that with sparse data, most of the weight updates are concentrated on components that are associated with non-zero or large input values. Seeing the training algorithm as a kind of optimization, it can be deduced that the optimization process has a much smaller search space, when the input is sparsified. As a result, the training process happens much faster (much lower number of iterations), at the cost of a single extra multiplication to the neural network. Given that the transforms are lossless, we expect no loss in accuracy.

Through training and testing with three realistic datasets and two synthetic datasets, we observe that training time can be reduced significantly, while also improving the accuracy. The improvement in both training time and accuracy suggests that TDNNs for time series forecasting can greatly benefit from inclusion of an input transform layer. We have observed that for datasets such as NDDS, the training time can be reduced up to 20 times, while improving accuracy up to 3.3%. At a fixed and very short training time of 25 epochs, the use of sparsifying transforms offer up to 7.5% improvement in accuracy. For datasets with more predictable patterns, such as US101 or ERRST, the accuracy improvements were smaller; nevertheless, the training time was reduced to 40% and 25% of baseline for ERRST and US101 datasets respectively (equivalent to 2- and 4-times improvement in training time).

It is also observed that the amount of improvement is correlated with the level of sparsity; i.e., sparser input results in faster and better learning. It is therefore reasonable to assume that methods that achieve further sparsity, perhaps with non-orthogonal bases, could also be beneficial. This is a direction of or future studies.

From an implementation perspective, the addition of an input transform layer adds a single matrix multiplication operation (outside the training loop); in return and in addition to improved accuracy, the number of iterations in the training process (backpropagation) can be greatly reduced. This reduction leads to significantly lower overall training time (thus computational cost), compared to networks without the sparsifying input layer, as also seen through mathematical analysis in [40] section 2.4. The improved accuracy can also be traded off for reduction of network size (thus computational cost). We will investigate the possibility of reducing network size in TDNNs with the help of sparse input representation in the next chapter (chapter 3). The improved performance for some sparsified-input-TDNN networks can be traded off for a smaller network size that achieves the same performance as the original TDNN. In general, smaller networks are expected to perform worse; by applying the sparsifying layer, we expect to improve the performance back to the original network size level.

CHAPTER 3. NETWORK SIZE REDUCTION FOR TDNN USING DOMAIN TRANSFORM

3.1 Introduction

[2] In the previous chapter we have shown how sparsifying input data of TDNN, by adding an input transform layer to a TDNN, can result in performance improvement. In this chapter, we show the improved performance can be traded off for network size reduction. Therefore, we can make the network smaller while maintaining the same performance levels. We used the same set of datasets as in the chapter 2. ERSST, or the Extended Reconstructed Sea Surface Temperature, NDDS or naturalistic driving data set, and US101. The focus of here is on the impact of network size reduction using sparse input representation by applying input transforms to a TDNN (and in general to a FFNN).

In most of the works, transforms (in particular Wavelet) are used for improved performance, and the specific impact of sparsifying transforms or the impact on network size is not discussed. Transforms such as Discrete Cosine Transform (DCT) or Principal Component Analysis (PCA) are generally not considered or studied for their impact on sparsity.

[2] The work presented in this chapter is based on the following publications:

Khandani, M. K., Mikhael W. B. : A Study on Network Size Reduction Using Sparse Input Representation in Time Delay Neural Networks. 2020 IEEE 63rd International Midwest Symposium

It must be noted that our concern is not on finding the best forecasting method or neural network structure; instead, our concern is to study the possible improvements to neural network training and performance using sparsifying input transforms in neural networks with low computational needs (such as TDNNs or generally FFNN).

We show that for some data types it is possible to make the networks smaller using sparse representation of input. In the next section we will first describe how the transform layer is integrated in a network and in later sections the impact is studied.

3.2 TDNN with Sparsified Input

The network under study is a TDNN, as depicted in Figure 2.2. It can be seen as a FFNN with an input delay buffer added. The delay buffer accepts the serial input X (time series data) and parallelizes it to form a D-element vector of the recent history of the time series at that given time. D is the input size to the rest of the network, which looks like a general FFNN. The input vector can be transformed to another domain of the same or different size. For example, we could apply the DCT transform to the vector. This task can be done by inserting a transform layer between the delay buffer and the first input layer of the FFNN [2], as seen in Figure 2.2.

The FFNN may have one or more hidden layers. In this chapter, we consider the simpler case of only one hidden layer (of size H), as it is quite sufficient for the time series prediction for the datasets studied here. The output layer is designed to have the same size as the desired number of samples in the future that the network will forecast. We call this the prediction length, denoted as P. With this configuration, a *DxHxP* neural network results. The size of this network, as considered in this chapter, is evaluated as the number of weights needed to be trained and used in the feedforward path. Considering the bias input to neurons, the total number of weights can simply be calculated as $DxH + HxP + B_H + B_P$, where B_H is the number of biases in hidden layer and B_p is the number of biases in the output layer.

In this chapter, we investigate how network size could be reduced using sparsifying transforms that are applied to the input vector. Two different transforms of Discrete Cosine Transform-DCT- and Principle Component Analysis-PCA- are considered. Both transforms can be easily applied to the input vector using multiplication with a D*x*D matrix of bases. DCT bases are known and can be simply derived from the two-dimensional DCT formulation. For PCA, the bases are derived after analyzing the dataset and deriving the data-dependent principle components. This means that for each dataset we have a different set of bases. We will observe that this makes PCA more efficient, but less practical than DCT if enough prior data is not available for a particular application (though that is not expected for the cases studied here).

We also note that the training and backpropagation only apply to the FFNN part of the network and the input transform layer is not involved and the transform operation is not repeated during training. Therefore, in our study of the network size, we focus on the size of the FFNN which is the common part in training and testing. It is the weights of the FFNN part that are adjusted in each backpropagation run during the learning process.

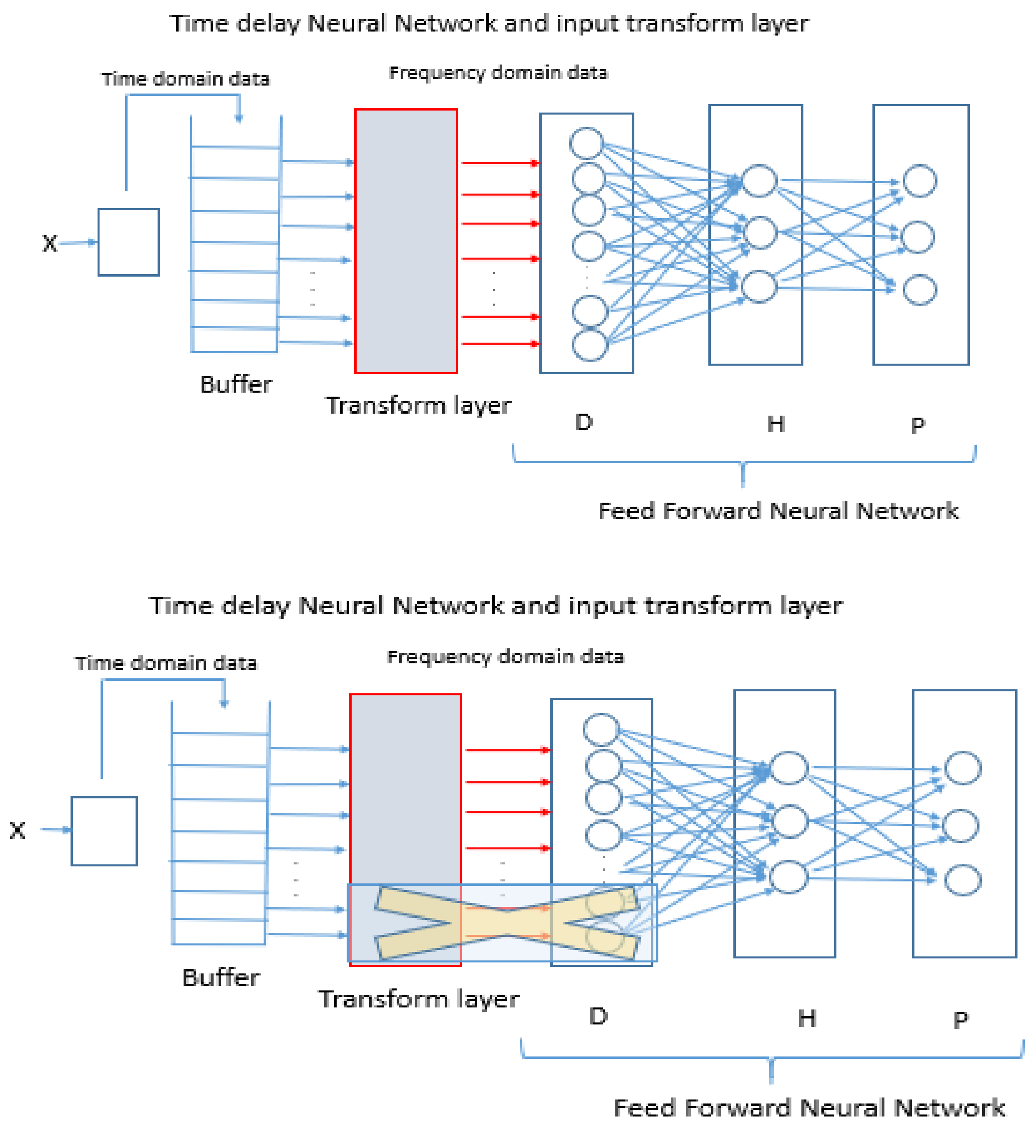

Time delay Neural Network and input transform layer
Time domain data
Frequency domain data
X
Buffer
Transform layer
D
H
P
Feed Forward Neural Network
Time delay Neural Network and input transform layer
Time domain data
Frequency domain data
X
Buffer
Transform layer
D
H
P
Feed Forward Neural Network

Figure 3.1: Network size reduction by compressing input: the architecture of the TDNN seen as a buffer and multi input FFNN of size DxHxP. D is the input (history) length; here a fraction (e.g. 75%) of coefficients are used.

Using sparsifying transforms, it is expected that the main energy of the signal will be concentrated in fewer elements of the input vector (transfer coefficients), possibly reducing the need to pass the whole vector the FFNN part. This is similar to compression using DCT, in which small coefficients are set to zero (thus not needed to be passed in our network). This allows the use of a smaller network, while possibly maintaining the same performance. Figure 3.1 shows how the network can be made smaller by only using a fraction of input coefficients. In this chapter we have studied 25% and 40% input size reduction. Additionally, the hidden layer may be made smaller with sparser input as well. We examine these hypotheses in the next section, after an analysis of the network size impact without any transforms.

3.3 Baseline Network Size Analysis

Prior to studying the impact of sparsifying transforms, we analyze the impact of network size on the performance of a baseline TDNN, i.e., a TDNN with no input transform. For this purpose, we have considered three input sizes of 32, 16 and 8 (history lengths that were meaningful for our datasets), as well as four hidden network sizes of 5, 10, 20 and 30. The prediction performance was analyzed through measuring the 95 percentile of the prediction (forecasting) error at some time samples ahead. For each sample at time t in the dataset, we considered a forecast and an error instant for each future time from t+1 to t+5. Since it is usually the longer-term prediction values that are critical, we take the t+5 prediction for error measurement. Here we considered the 95-percentile of the error instances for prediction at t+5. We note that the network was designed to predict 5 points in the future, so the output size was P = 5. We use the 95-percentile, instead of other statistics as it is commonly used for some of the datasets that we use [2]. The trends are similar for median, mean or 95%.

The training for all the scenarios was done for a fixed number of epochs, 15, to remove the effects of other parameters such as validation error threshold. We have also used the default training setting of initial μ=0.001 and μ decrease factor=0.1, applied after 10 epochs. We also used the MATLAB implementation of the backpropagation and the neural network toolbox. We used 70% of the samples in the dataset for training and 30% for testing. Results from these tests are reported in Figure 3.2.

It is observed that for all datasets the larger network sizes produce better performance, as expected. Another important observation is that hidden network size of 5 produces noticeably higher error, while H=10, 20 and 30 are not considerably different. Similarly, larger input sizes produce better results. Given the consistency in improvement with input and hidden layer size, we will take the middle option of input size 16 for our study of input compression (through sparse representation) in the next sections.

3.4 Evaluation

To study how sparsifying the input may allow using smaller networks, we vary the hidden layer size while keeping the input to be a vector of size 16 (D=16) under different transforms. In some instances, when a transform is applied to the input, we can also remove some transformed input elements (coefficients of the transform results) as shown in Figure 3.1. This will allow reducing the input size as well as the reduction that is possible by adjusting the hidden layer size. This is examined in detail in this section. The number of training epochs is kept at 15.

The transforms that are applied are DCT and PCA; the PCA bases are separately calculated for each data set. The results are different for each data set. Starting from the ERSST dataset, the prediction error results in Figure 3.3 show that all options of DCT, PCA and their compressed version (where 25% of the smallest coefficients were removed from the input, resulting in vectors of length 12) perform better than the baseline. In particular, it is seen that for smaller network sizes (smaller hidden layer size, for example 5 and 10), the gain in performance is more notable. The gain is also more visible at hidden layer size 30.

Similarly, for the NDDS dataset, with results shown in Figure 3.4, we observe that the gain in performance is more notable for smaller hidden layer size of 5. The performance gap is also notable at H=30, similar to the ERSST dataset.

The results for US101, demonstrated in Figure 3.5, show similar trend but with an across the board performance gap for sparsified inputs (through either DCT or PCA). The improvement is visible for all hidden layer sizes. Here, we could obviously use this performance gain and remove some of the coefficients (25% in this case, yielding input vectors of length 12), while still maintaining the performance at the baseline level.

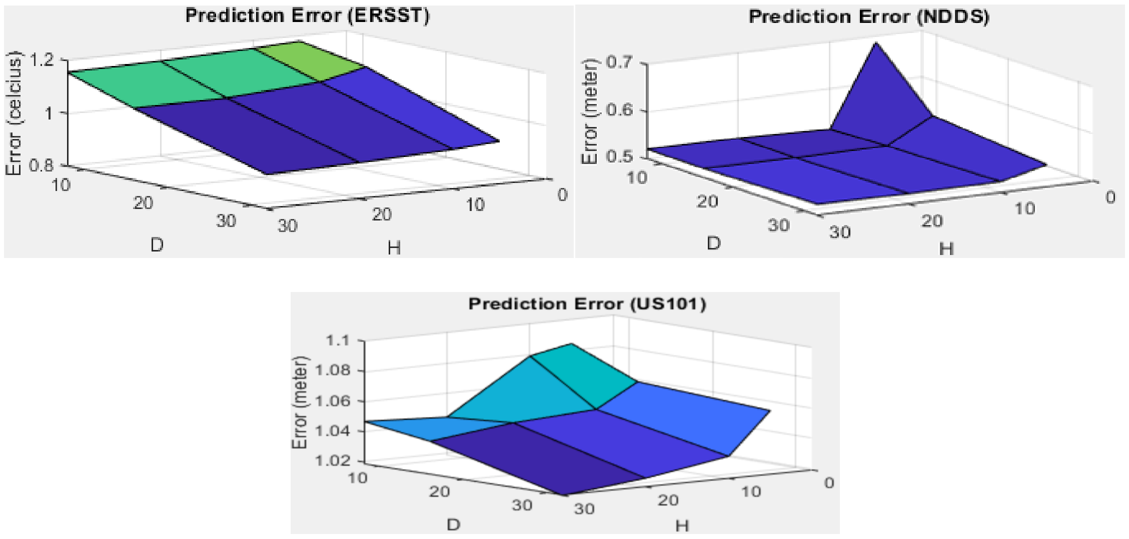


Figure 3.2: Forecasting (prediction) error for twelve network size configurations (D=8, 16, 32, and P = 5, 10, 20, 30) for three datasets of ERSST (top), NDDS (middle), and US101 (bottom)

Overall, we observe that applying DCT or PCA and removing 25% of the coefficients will always allow performance of at least at the same level as baseline.

We examined higher levels of compression (removing more than 25% of coefficients from input) and it was observed that the prediction results become worse than baseline, except for the NDDS dataset. This dataset includes driving maneuvers with sudden movement, which are perhaps more suitable to be captured by sparse representations, rather than in time domain. For this dataset, we could remove up to 40% of the coefficients from PCA or DCT transform results (yielding input vectors of size 10), and still maintain the error below baseline levels. The results are reported in Table 1, along with the network sizes and the number of weights that are used in each network.

From the results in this section, it is seen that the network size choices are first dictated by the performance requirements (acceptable error levels). Given a particular performance target, the use of sparse representations, such as DCT or PCA will allow selection of a smaller hidden layer size. Furthermore, compression can also be applied to the input layer. For example, for the ERSST dataset, if an error of 1.11 degrees Celsius is acceptable, the smallest baseline choice will be H=10. However, with the use of PCA or DCT, a considerably smaller layer of H=5 can be used to achieve an error of 1.10. Furthermore, with compressing either PCA or DCT coefficients, the network could further be made smaller while still maintaining error at 1.11 degrees.

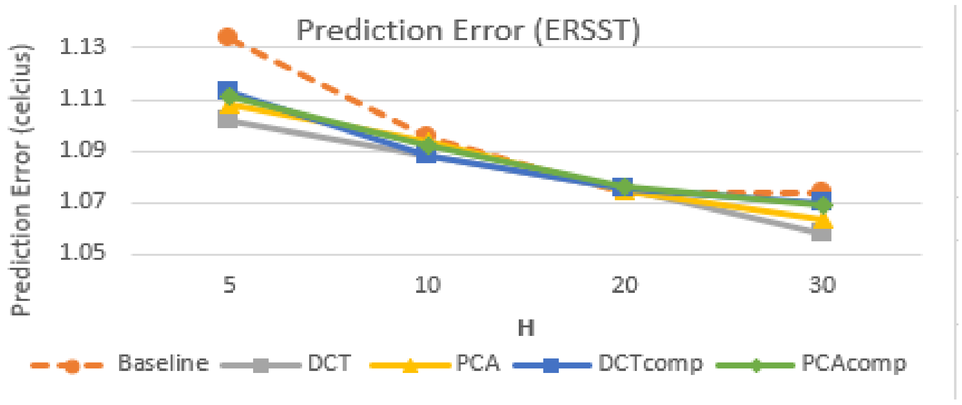


Figure 3.3: Prediction error results for ERSST dataset; different choices of sparsified and compressed input compared to baseline. The horizontal axis shows the hidden layer size H

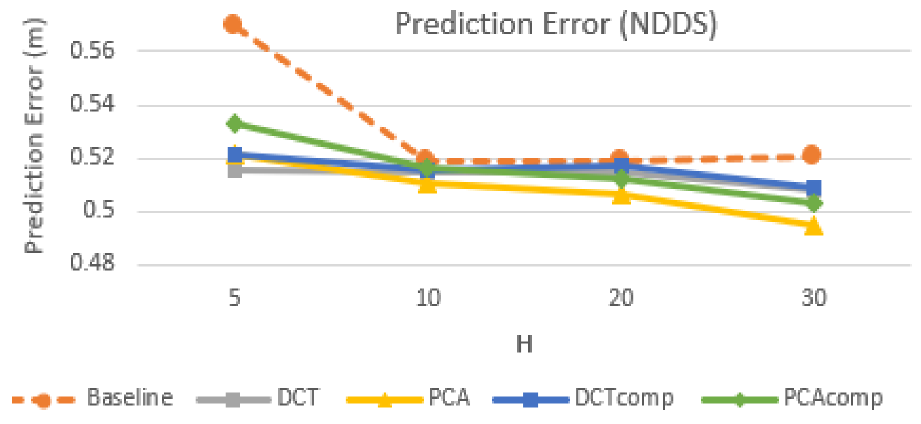


Figure 3.4: Prediction error results for NDDS dataset; different choices of sparsified and compressed input are compared to baseline. The horizontal axis shows the hidden layer size.

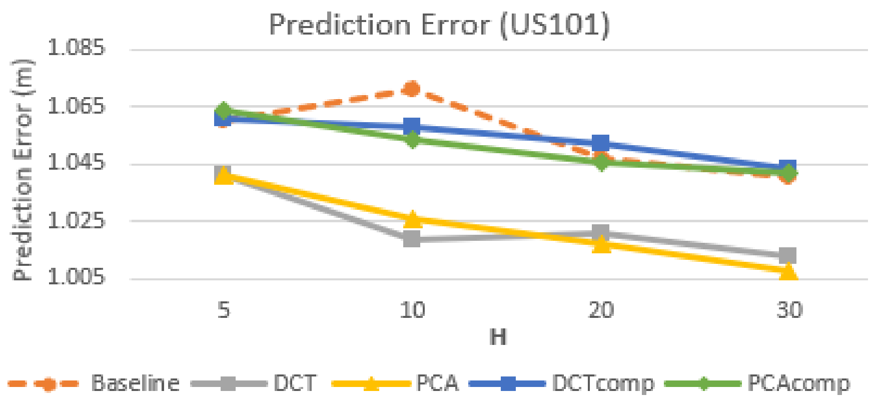


Figure 3.5: Prediction error results for US101 dataset; different choices of sparsified and compressed input are compared to baseline. The horizontal axis shows the hidden layer size.

3.5 Summary and Concluding Remarks

We have studied the possibility of reducing network size in TDNNs with the help of sparse input representation. With a given level of performance (in this case, prediction error), it is possible to considerably reduce the hidden layer size of a network, simply by applying sparsifying transform to the input layer. Furthermore, the considerable improved performance allows for additional network size reduction through removing some of the coefficients of the sparse representation of the input vector. It is observed that this trend is generally seen for all datasets, with a 25% reduction of the input size possible for all of them. However, for some datasets such as NDDS, the input can be further compressed with up to 40% reduction in size.

Overall, it is consistently seen that applying sparsifying transforms to the input of a TDNN allows for better performance which can be traded off for reducing network size in both the input and the hidden layers. While we have analyzed the DCT and the PCA based transforms, other types of transforms may exist that could provide further improvements. The concept described here could also be employed with larger and deeper networks and is being currently investigated.

Table 3.1: NDDS data set: different number of weights (network sizes) for TDNN based methods, each row represents the prediction error for each method.

#weights / transform	665 16x30x5	445 16x20x5	225 16x10x5	115 16x5x5
None	0.520552	0.517456	0.518695	0.569658
DCT	0.508220	0.498591	0.518596	0.515907
PCA	0.494928	0.506184	0.510347	0.521136

#weights / transform	545 12x30x5	365 12x20x5	185 12x10x5	95 12x5x5
DCT-comp 25%	0.509024	0.517149	0.510361	0.521448
PCA-comp 25%	0.502860	0.512252	0.516338	0.513015

#weights / transform	485 10x30x5	325 10x20x5	165 10x10x5	85 10x5x5
DCT-comp 40%	0.518945	0.520815	0.525214	0.562984
PCA-comp 40%	0.521770	0.527519	0.523211	0.534384

CHAPTER 4. ENHANCING CONVOLUTIONAL NEURAL NETWORK PERFORMANCE USING DOMAIN TRANSFORMS IN CONSTRAIEND NETWORKS

4.1 Introduction

[3]The task of image classification has many applications in computer vision. Recent advancements in Convolutional Neural Networks (CNN) have paved the way for significant improvements in image classification over traditional image processing methods. In particular, deep convolutional neural networks are now considered the main tool for this purpose. However, these networks are often large and require computing and storage resources that are not available in many smaller computing devices. For example, very small IoT (Internet of Things) devices are usually designed with minimal computing and storage capacity with the aim of reducing cost. Using image classification applications in large scale deployment of IoTs will require either considerable cost increase, or considerable reduction of performance. Moreover, the task of

[3] The work presented in this chapter is based on the following under-review paper:

Masoumeh Kalantari Khandani, Wasfy B. Mikhael, "Efficient Size Reduction of Convolutional Neural Networks Using Domain Transforms", Under review training CNNs, which is often done offline, is sometimes needed to be done online for retraining of networks. This task is generally very computing intensive, compared to the forward operation of CNN. Therefore, it becomes necessary to look for methods of reducing the computational cost of CNNs for low power computing devices. This is generally achieved in some applications by reducing the size of input data (images) and the processing neural network. To put this in perspective, most deep networks that are used for classification have millions of learnable parameters. We limit our choices of networks to those with an order of magnitude less parameters (i.e., under 200K learnable parameters and down to 50K). The reduction in network and input size, as expected, usually comes at the cost of reduced classification performance. In this chapter, we examine how domain transforms can be used for mitigating the negative effect of input and network size reduction.

We show that using transforms, such as Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT) as an input transform layer, it is possible to efficiently improve the performance of size-reduced networks. We note that these transforms project the original image data to a domain where data is represented in a sparser form, allowing for selectively removing part of the input data and reducing the input image size in a more efficient way than simple resizing of an image. We observe that such transforms also have the positive side effect of improving the learning rate. Similar benefits were seen in our earlier works on time series and shallow networks [40]. The improved learning rate and network size reduction allows for lower computational cost, in particular during the training phase. Training of deep networks is usually the most computationally expensive aspect of CNNs; while retraining a network requires less computations, it is still considered a heavy load on smaller devices. The reduced cost of training, achieved using domain transforms, is an important factor that can enable retraining of CNNs in small devices.

The basic hypothesis that is examined in this chapter is that transforming image data to a sparser form will allow for more efficient network and input size reduction than simply resizing the input. We demonstrate that in most cases the improvement can be traced to higher entropy of resized input using transforms. While in data compression the use of such transforms is common, in classification applications it is not intuitive that sparser representation will be useful. In fact, it is seen that the usual application of CNNs with convolutional filters in the input layers is not useful when data is represented in domains such as DCT. On the other hand, it is seen that more efficient configurations becomes possible if DCT is used. Transfer to DWT domain shows different properties, since the spatial relationship between initial data pixels are maintained (as opposed to DCT that the spatial relationship is not explicitly kept). To evaluate and examine these hypotheses and observations, we use two standard datasets of small images (representing IoT processable images), including Fashion MNSIT (FMNSIT) and CIFAR-10. Evaluating our proposed methods, it is shown that input size reduction of up to 75% is possible, without loss of classification accuracy. While transforms such as DCT allow variable input and network sizes to be utilized, DWT proves to be very effective when significant size reduction is needed (improving the result by up to 5%).

In the rest of this chapter, we first review some related literature. Then the system architecture and the proposed method for addition of an input transform to a CNN is explained. Evaluations and discussion of different configurations and network size reduction options are presented last.

4.2 Related Work

The focus of this work is on the effect of domain transforms in improving the performance of shallow or small convolutional neural network in image classification. In particular we are interested in devices with limited computing capacity or CPUs instead of GPUs. There are many methods and tools employed for image classification in different applications with larger computing facilities, such as the works on Res-Net, VGG, Alex net[41][42][43]. We do not review these works here as they are out of the scope of this chapter. It is worth noting that if large and complex deep networks were possible with our target devices, better results for image classification could be achieved for the datasets of interest. For example, utilizing the work in[44], a classification accuracy of above %91 is achievable; similarly, the methods in [45] achieves %96.5 with the help of expanded training with data augmentation. Other recent large networks also perform similarly[44]. However, the above methods require orders of magnitude larger networks, in terms of number of learnable parameters, than what we are considering in this chapter. Our interest in this chapter is to study networks with under 200,000 parameters, while methods in [44] and [45], respectively have 1.3 and 50 million parameters in their proposed networks. As a result, we do not consider such large designs in this chapter and use smaller generic networks as explained later. Our focus in this chapter is on the impact of domain transform on smaller networks, and the specific network architectures are not the emphasis of this study.

The use of transforms and neural networks has received some attention for time series and image classification applications, we review the use of domain transforms in image classification in this section.

Transforms such as DCT and DWT are popular in image processing applications and consequently have been considered in some specific image classification designs. For example, Pan et al [24] have shown that reducing image information redundancy using DCT can be beneficial to face recognition applications. They have demonstrated that when DCT coefficients are fed into a backpropagation neural network for classification, a good recognition rate can be achieved by using a very small proportion of transform coefficients. This work is specific to face recognition and is not directly applicable in our case. Wang et al [35] have introduced a method called CNN-pack, in which a convolutional neural network itself is transformed into frequency domain and packed by linearly combining the convolutional responses of DCT bases. This method is shown to reduce the computational burden since the network becomes sparser. This work is in principle different from our work in its method of transforming convolution operations in a CNN.

In another work, Ghosh et al [46], propose to apply DCT on the feature maps generated by the first convolutional layer in the network. They observe that the training phase convergence is faster when feature extraction takes place in the DCT domain. Performance remains comparable in this case. This is similar to our observation for time series [40]; although our proposed method in this chapter primarily achieves improved performance without sacrificing training time.

A different approach in using domain transforms is taken in [47]. Here, authors propose to feed and train a CNN by modifying the input representation of the JPEG compressed data. This means that the image does not have to be in the RGB or similar original domain and the proposed CNN can be trained with JPEG compressed DCT coefficients. This work utilizes deep and complex networks and is shown to produce good accuracy. However, their method is more suitable for training on already JPEG compressed images and is not directly applicable in our case. We do not expect images to have gone through JPEG compression prior to becoming available to the classification application running on the CPU.

A completely different approach to using domain transforms and CNNs is proposed in [48]. This work proposes Hybrid Cosine Based CNNs which use a cosine basis to represent the weights of the convolutional filters. This amount to a different method of applying the filters. Authors show that better performances can be obtained than VGG and Res-Net architectures using less parameters in the convolutional layers. The complex networks designed in this approach are not useable for the small CPUs that we consider our case.

In this chapter, we study the impact of adding a domain transform layer to the input of a small or shallow CNN. The layer will transform the input to a domain where it is represented in a sparser form. We show that it is possible to efficiently reduce the input image size using transforms, compared to the averaging methods for resizing an input. Our work is different from existing works in that we do not utilize transforms in the operation of a CNN (how filtering is done), or directly in compression. Rather, our aim is to maintain enough information in the data that makes classification work better. We note that since the existing methods are designed for specific datasets and use large and deep CNNs, it is not possible to directly compare the impact of adding a transform layer to existing methods. Therefore, in this chapter we customize CNNs for specific datasets and compare them with the results of the same networks in the visual domain (baselines).

Figure 4.1: Sample images from Fashion MNIST[64] and Cifar-10[57] datasets.

4.3 Method and System Description

The concept studied here is to transfer the input image to a different domain before passing it through the CNN either with the same size or at a reduced size. This is achieved by adding a transfer and size reduction layer immediately as the first layer of a convolutional neural network. In this chapter we consider two examples of domain transform using DCT and DWT (Wavelet) transforms, as well as their variations. For size reduction, we consider reducing the width and length to either half or a quarter of their originals. Respectively, this will result in images with total pixels that are ¼ and 1/16 of the original. For simplicity, in this chapter we refer to the former as -half and the latter as -quarter sizes, indicating the size reduction of each dimension of the image. Therefore, the outputs of the domain transform and size reduction operations will have at least 9 options as indicated in Table 4.1. We note that the size reduction using transforms can happen in multiple ways. For example, with DWT, halving the width and length can happen by only taking the LL component (or any other wavelet output component, as illustrated in Figure 4.4). For DCT, the lower frequency components may be kept. Table 4.1 describes all the choices that are used in this study.

Table 4.1. the outputs of the domain transform and size reduction will have at least 9 options

Transform and Resize	Utilized Transform	Resize method	Description
Baseline	-	-	Original image
Baseline-Half	-	Resize to N/2 x N/2	Resize using averaging
Baseline-Quarter	-	Resize to N/4 x N/4	Resize using averaging
DCT	DCT2	-	DCT transformed
DCT -Half	DCT2	Resize to N/2 x N/2	Apply DCT, take lower 1/4 of coefficients
DCT -Quarter	DCT2	Resize to N/4 x N/4	Apply DCT, take lower 1/16th of coefficients
DWT	CDF_9/7	-	DWT transformed
DWT-Half	CDF_9/7	Resize to N/2 x N/2	Apply DWT once, take LL band
DWT-Quarter	CDF_9/7	Resize to N/4 x N/4	Apply DWT twice, take LL2 band

The network with the additional transform and resizing layer is shown in Figure 4.4. The configuration of the CNN can vary greatly based on different solutions that are found in the literature. Nevertheless, there are common components such as convolutional layers, Relu, maxpool, batch normalization layer, fully connected layer and soft-max layer that are used in a chain

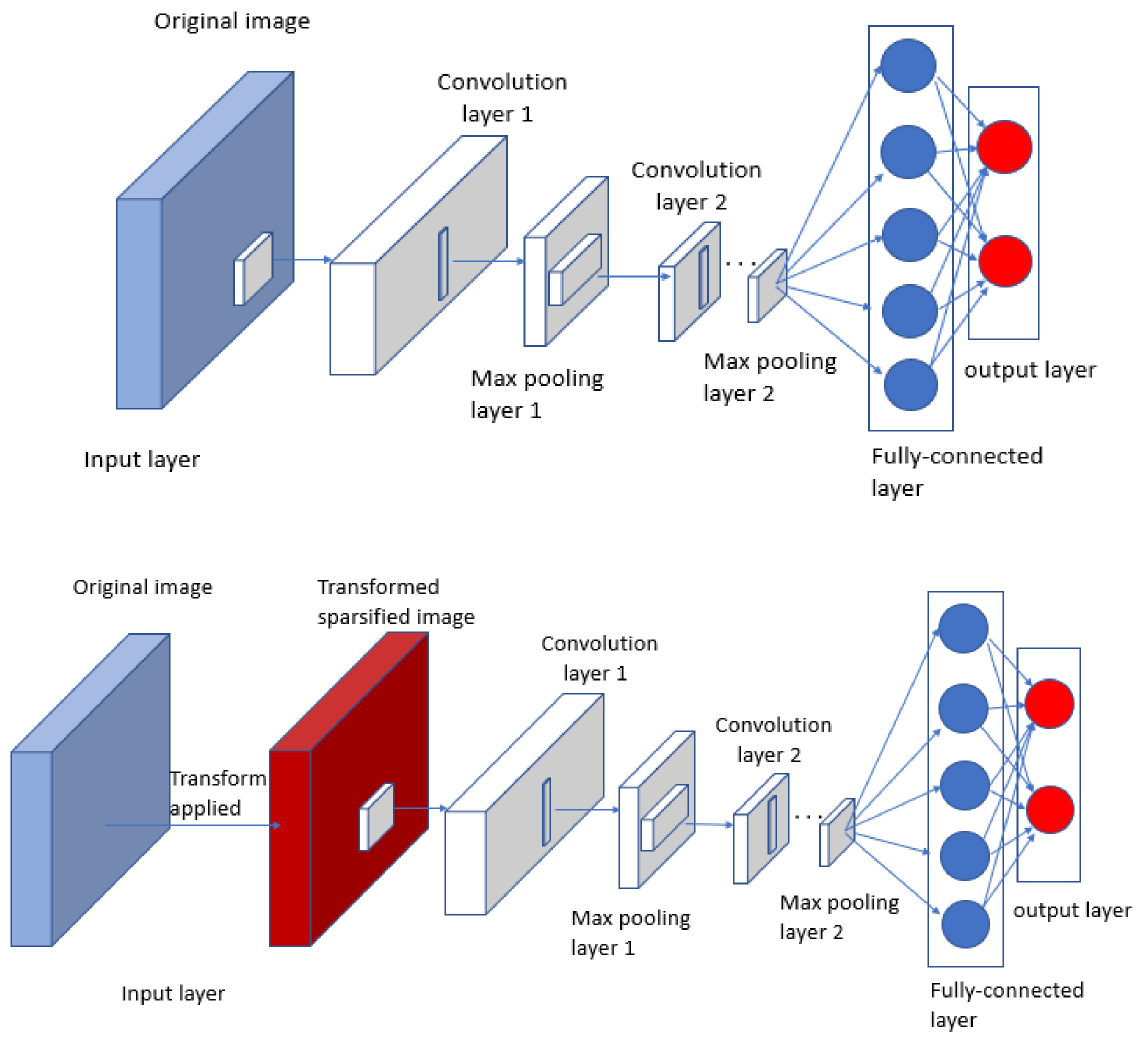


Figure 4.2: Architecture of the CNN in general and adding a transform layer. A Transform layer can be inserted between the input and the CNN.

In this chapter, our objective is not to find the best possible CNN architecture for a given input type (or image dataset), instead our focus is to study the impact of domain transform on the performance of a general shallow CNN and how network size could be made smaller. Therefore, we have used architectures that are used as basic CNNs for image classification with some small changes to improve their performance. We also emphasize that the objective of this chapter is to devise solutions that require low computational power; therefore, complex and very deep CNNs are not acceptable solutions for this purpose. As a result, we use networks with less than 3 convolutional layers (2 for FMNIST and 3 for CIFAR-10), 32 filter with size of at most 5, and at most two fully connected layers (see Table 4.2 and Table 4.3). We note that there are very complex deep networks that have been proposed for the datasets used in this chapter. For example [49][50] for Fashion MNIST, and [51] [52] for CIFAR-10. These architectures are designed after extensive hyper parameter fine tuning and require many layers and large filters, resulting in very lengthy training and computational cost [53] . Changes in input form (from transform) and size will render those solutions non-optimal; their computational requirements are also well beyond the power of the small processor devices that we are considering in this chapter. Therefore, the state of the art that we compare our methods against are smaller forms of the networks derived from the [56][57] for CIFAR-10 and [54][55] for FMNSIT, which utilize the number of filters and layers acceptable in this work. We have fine-tuned these state-of-the-art methods to optimize them for the datasets and input sizes used here. These are referred to as the “baseline” solutions in Table 4.2 and Table 4.3.

4.3.1 *Data Sets*

We utilize two very well-known datasets of Fashion MNIST (FMNIST) and CIFAR-10 in this chapter. We have particularly selected these datasets as they contain small images and are good representatives for applications with low computational requirements. The FMNIST dataset was introduced in [58]; it includes fashion product images in grayscale. It is intended to be a replacement of the original MNIST dataset and provides a more challenging alternative for benchmarking machine learning algorithm. The images are of size 28x28 and their small size allows fast and computationally inexpensive methods to be used. On the other hand, the small size of the images creates a challenging task in classification. In total, FMNIST contains 70,000 grayscale images of fashion articles in 10 categories. For the machine learning application, the training set of FMNSIT contains 60,000 images (6000 from each category) and the test set includes 10,000 images (1000 from each category).

While FMNIST images are in grayscale, the CIFAR-10 dataset contains small color (RGB) images of size $32x32$ in 10 classes. There are in total of 60000 images, with 6000 images per class. 50000 of the images are training images and 10000 are test images [59][56].

4.3.2 *Network Architecture*

As described above, a different network setup is used for each dataset. The reason is that the FMNIST dataset includes grayscale images of $28x28$ pixels, whereas the CIFAR dataset contains RGB color images of 32 by 32. For the purpose of our study, we have zero-padded the FMNIST images to create $32x32$ images to allow better applications of transform. We expect no degradation of quality due to zero padding as confirmed in [60]. We found the following networks layers to produce the best results under the size constraints set out earlier for small processor devices.

For the FMNSIT dataset the network design is detailed in Table 4.2. Basically, we use only two convolution layers. To demonstrate that the network choices were robust and not very sensitive to the hyper parameter changes, we have examined few different choices of the number and sizes of the filters and present the results. The number of filters used in these layers were from four different configurations; we named them network A, B and C. For network A both first and second convolution layers have 8 filters with size one. Network B uses 8 filters with size one for the first convolution layer and size 5 for the second convolution layer. For network C, we have used 8 filters for each convolution layer, with filters of size 3 for the first layer and size 5 for the second one.

For the CIFAR dataset we use the network in Table 4.3. We only use one network size with three convolutional layers. The number of filters and their sizes are fixed for all situations as shown in the table. The combinations of pooling layers and operations are found after extensive study to optimize the performance with the constraint of 3 convolutional layers. In all of the studies with domain transform, we take the best network choices for each network (the three networks for FMNIST and the selected network for CIFAR). These networks are then used for the study of the input domain transform and size reductions. The process of domain transformation and size reduction is explained below, for each choice of the transform.

Table 4.2: Convolutional Neural Network layers used for Fashion MNIST images (networks A, B, C); input images are *n*x*n*x*1*, *n*=*N*, *N/2*, or *N/4*

Network A	Network B	Network C
*image Input Layer (nxn) *convolution2dLayer **(8 filters of 1x1)** * Batch-Normalization Layer *Relu Layer *maxPooling2dLayer (size=2, stride=2) *convolution2dLayer **(8 filters of 1x1)** *Batch Normalization Layer *Relu Layer *fully Connected Layer (10) *softmax Layer *classification Layer	*image Input Layer (nxn) *convolution2dLayer **(8 filters of 1x1)** *Batch Normalization Layer *Relu Layer *maxPooling2dLayer (size=2, stride=2) *convolution 2d Layer **(8 filters of 5x5)** *Batch Normalization Layer *Relu Layer *fully Connected Layer (10) *softmax Layer *classification Layer	*image Input Layer (nxn) *convolution2dLayer **(8 filters of 3x3)** *Batch Normalization Layer *Relu Layer *maxPooling2dLayer (size=2, stride=2) *convolution2dLayer **(8 filters of 5x5)** *Batch Normalization Layer *Relu Layer *fully Connected Layer (10) *softmax Layer *classification Layer

Number of learnable parameters:

Input Image size network	NxNx1	N/2xN/2x1	N/4xN/4x1
Network A	28978	9778	4018
Network B	19634	5554	2354
Network C	17378	4578	2018

Table 4.3: Convolutional Neural Network layer used for CIFAR-10

Layers for input image of nxnx3; (3 RGB channels) n = N, N/2, or N/4
*Image Input Layer([nxnx3]) *convolution2dLayer (32 filters of size 5) *Relu Layer *convolution2dLayer (32 filters of size 5) *Relu Layer *convolution2dLayer (32 filters of size 5) *Relu Layer *averagePooling2dLayer (size 3, Stride 2) *fully Connected Layer (20) *Relu Layer; *Fully Connected Layer (10) *Softmax Layer *Classification Layer

Image sizes	Number of learnable parameters:
NxNx3	197926
N/2xN/2 x3	85286
N/4xN/4 x3	59686

4.3.3 *Input Domain Transforms and Size Reduction*

There are numerous options for domain transform. We consider two specific transforms that have different characteristics: DCT and DWT. The DCT transform projects image data to some frequency domain, representing it in a form that is sparse and spatially distributed in a very different form from the original data. On the other hand, DWT utilizes wavelets and produces 4 sub-bands of LL, LH, HL, and HH, representing data in forms that can be visually related to the features of the original data. LL is the low frequency sub-band, resembling a low pass filtered version of the original image. The other bands, LH and HL, include higher frequency components in the vertical and horizonal dimensions of the image. HH sub-band gives diagonal features. The visual similarity points to the fact that the spatial relationships in the original image domain is more explicitly maintained with DWT. However, the four bands are disjoint and each only has 1/4th the size of the original. Therefore, the methods of size reduction or constructing full size transformed images will be different for each scheme. These processes are explained in detail below.

4.3.3.1 *Discrete Cosine Transforms*

The first transform that we consider is Discrete Cosine Transform (DCT), which is one of the sinusoidal transforms. It was explained in detail in Chapter 2. A common variant of DCT is DCT-II and can be applied to signals with multiple dimensions. For example, a 2-dimensional DCT transform of an image or 2-D signal *x* (of size N*x*M) can be calculated as follows:

$$X_{cd}(\mathrm{k}) = a_c a_d \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[n] . \cos[\frac{\pi c}{M}(\mathrm{m} + 1/2)] . \cos[\frac{\pi d}{N}(n + 1/2)] \qquad (4.1)$$

$$a_c = \begin{cases} \frac{1}{\sqrt{M}} & c = 0 \\ \frac{\sqrt{2}}{\sqrt{M}} & 1 \leq c \leq M - 1 \end{cases} , \quad a_d = \begin{cases} \frac{1}{\sqrt{N}} & d = 0 \\ \frac{\sqrt{2}}{\sqrt{N}} & 1 \leq d \leq N - 1 \end{cases}$$

Where X is the matrix of coefficients (results) of the transform, and the indexes c and d are as: *c=0,1, 2, …, N-1,* and *d=0,1, 2, …, M-1.*

For data, such as natural images, most of the information is usually concentrated near lower frequencies. This leads to very sparse coefficients being produced by sinusoidal transforms such as DCT. While the sparsity is useful in compression applications, it has the side effect that the range of values for different coefficients is usually much larger than the difference of values for original image pixels (intensity levels). We consider reducing such differences in later sections of this chapter and show that it has a positive impact on results.

To visually observe the impact of applying a 2-D DCT on an image, Figure 2.3 shows the DCT coefficients of an image from FMNIST dataset. We note the sparsity of coefficients and the large range of coefficient values.

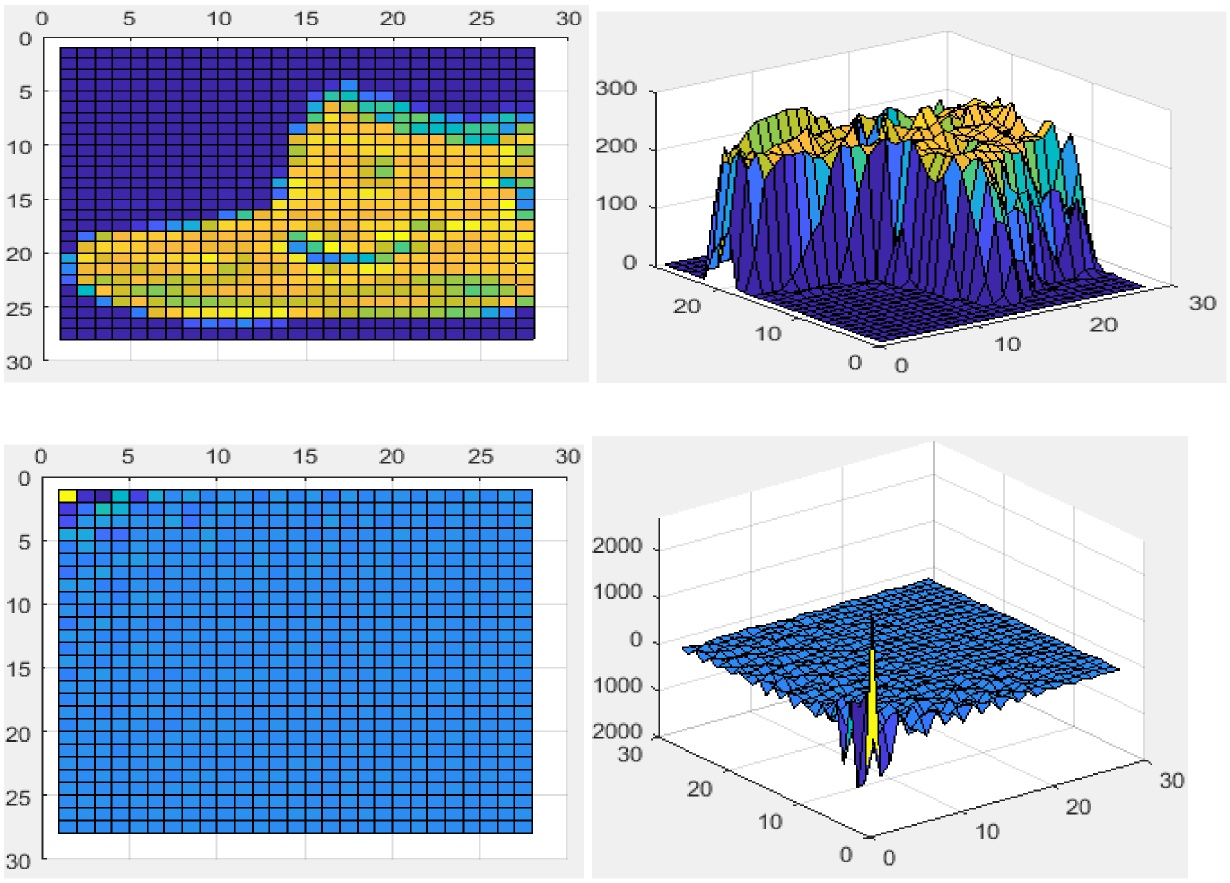


Figure 4.3: Visualizing effect of DCT. Top: a sample image from FMNIST shown in 2-D and 3-D (x, y, and intensity) forms; bottom: DCT coefficients of the same image in 2-D and 3-D (x, y, and coefficient value) forms.

Resizing DCT transformed data

Resizing images in the DCT domain is straightforward and can be done by selecting a fraction of the lowest frequencies of the coefficients. This is in particular useful for natural images that usually have most of their energy concentrated in the lower frequencies as seen in Figure 4.3. Original image can be approximately reconstructed by setting the value of the removed higher frequency coefficients to zero, and performing an inverse DCT transform. The advantage of DCT in size reduction is that it allows any number of the coefficients to be removed.

A disadvantage of DCT is that the spatial relationship between neighboring pixels are not directly capturable by the convolutional filters. The convolutional filters are designed such that the spatial relationship of neighboring pixels, which is usually visible, is easily captured through the convolution operation. DCT coefficients are structured in such a way that the local relationships are spread over different frequency coefficients. This means that the design of convolutional filters may have a harder time capturing the spatial features in an image.

4.3.3.2 *Discrete Wavelet Transforms*

The second transform considered in this work is the Discrete Wavelet Transforms (DWT). DWT produces four sub-bands low-low (LL), low-high (LH), high-low (HL) and high-high (HH). By using these four sub-bands we can regenerate original image through inverse DWT. Wavelets are especially useful for compressing or denoising two dimensional signals, such as images. Wavelet analysis is basically the process of decomposing a signal into shifted and scaled versions of an original wavelet function (for more information refer to Chapter 2).

To construct 2-D transform, the transform is applied in both directions. In contrast to frequency-based transforms such as DCT or DFT, DWT has a key advantage it can capture both location (time) and frequency information, in addition to the discontinuity in the signal. The non-sinusoidal nature of the basic wave functions provide a simple and computationally efficient approach for analyzing the local aspects of a signal[36][37].

The specific version of the DWT used in this chapter is based on Cohen–Daubechies–Feauveau wavelets. These are a family of biorthogonal wavelets that was made popular by Ingrid Daubechies [61][62]. We use the CDF 9/7 wavelet, which is also used in JPEG 2000 standard for lossy compression.

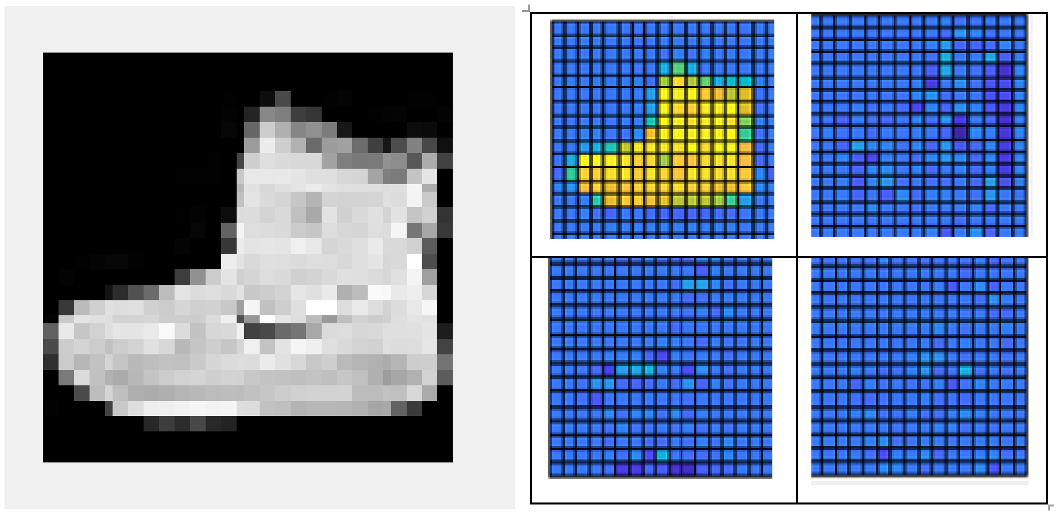


Figure 4.4 Visualizing the effect of DWT transforms; left: sample image from FMNIST data. Right: DWT transformations, the four bands: top left (LL), top right (LH), bottom left (HL) and bottom right (HH).

An example of the DWT applied to an image is seen in Figure 4.4. Applying DWT to an N*x*N image produces four signals of LL, LH, HL, and HH, which are all of size N/2*x*N/2. LL includes lower band and is visually similar to the original image. Putting the four bands together will create a 2-D signal which has the same size as the original image; however, this composition may not be directly fed to a CNN, since the transition from boundaries of the four different bands in a N*x*N composition as in Figure 4.4 has no spatial meaning. That is, the bottom edge of LH and the top edge of HH are not spatially related, so they cannot be interpreted by the CNN as being directly relevant. Therefore, composing the full signal in DWT form has to happen in a different way than what is shown in Figure 4.4. One method is to stack the four bands and create a signal of size 4*x*N/2*x*N/2, which basically has the same size as the original image.

Resizing DWT transformed data

Reducing input size using DWT is much more limited than when DCT is used. With DWT, we can use one or more of the four bands. If one band, e.g., LL, is used, a compressed image of size N/2*x*N/2 is obtained. This will generate a transformed data with the half length and width as noted in Table 1. To further reduce the image size, we can apply DWT to the LL band for a second time, producing four bands out of the first LL. If the LL band of the second transform (which will be the obtained LL of the previous LL band) is kept as the signal, an image of size N/4*x*N/4 is obtained. Of course, there are numerous possibilities for stacking several bands and achieving data sizes in between half and quarter dimension images. In this chapter we study the full (stacked), half and quarter dimension options of the DWT transformed data.

4.4 Entropy Analysis

This section provides some insight into the reason for improved performance when size reduction is accomplished using domain transforms. When a transform like DWT or DCT is applied to an image, the resulting coefficients contain the same information as the original image, but organized in a different way. These transforms are not lossy. Therefore, theoretically we do not expect that the transformed data provide advantages or disadvantages in terms of the amount of information for classification. However, it is possible that the new organization of image data (coefficients of transformed data) may change the behavior of specific CNN configurations either positively or negatively. This is studied in the evaluation section.

In this section we focus on the case of resized input data, in which some loss of information happens. It is generally expected that the loss of information will result in degraded classification performance. The more information is lost, the worst the performance would be. While this logic may generally hold, the organization of the resized image (how coefficients represent the data) is also important in the performance of a CNN. Nevertheless, preserving more information during the lossy process of resizing input data is expected to be beneficial. Measuring the amount of information in an image is a challenging task. For the purpose of our study, and to qualitatively compare different methods, we use the image entropy measure.

Entropy of a signal was introduced in 1948 by Claude Shannon to study the amount of information in a message. A higher value of entropy would mean that there is more information in the message and more bits were needed for its transfer. It was used to derive the lowest bound of the channel capacity (bandwidth) needed for a message (which would be achieved if an ideal coding was used). The concept of entropy has been used for images (2-D signal) as well [63]. We can use entropy of an image to quantify how much information is available in the image, which can be used by the CNN for classification. So, a method that preserves more information, is expected to produce 2-D signals with higher entropy and possibly higher classification accuracy in CNN. However, we also note that the different organization of transformed information may also impact the results. Considering these facts, in this section we look at the entropy of resulting signals for different methods of input size reduction.

Entropy calculation is done according to the following formula:

$$H(X) = -\sum_{i=1}^{n} p(x_i) log p(x_i) \tag{4.2}$$

H(X) is the entropy of the random variable X. Here $p(x_i)$ is the probability that outcome x_i happens. In the case of gray image, the probability density p(.) is calculated using the gray level histogram, and the sum runs from n=1 to 256. The bins represent possible states. For RGB images, we treat each channel separately and then the average of the three channel is used. Here the assumption is that each data point (pixel intensity in the image) is represented in a quantized discrete form. This poses a difficulty in comparing entropy of different methods, since our data is in a real number format (double).

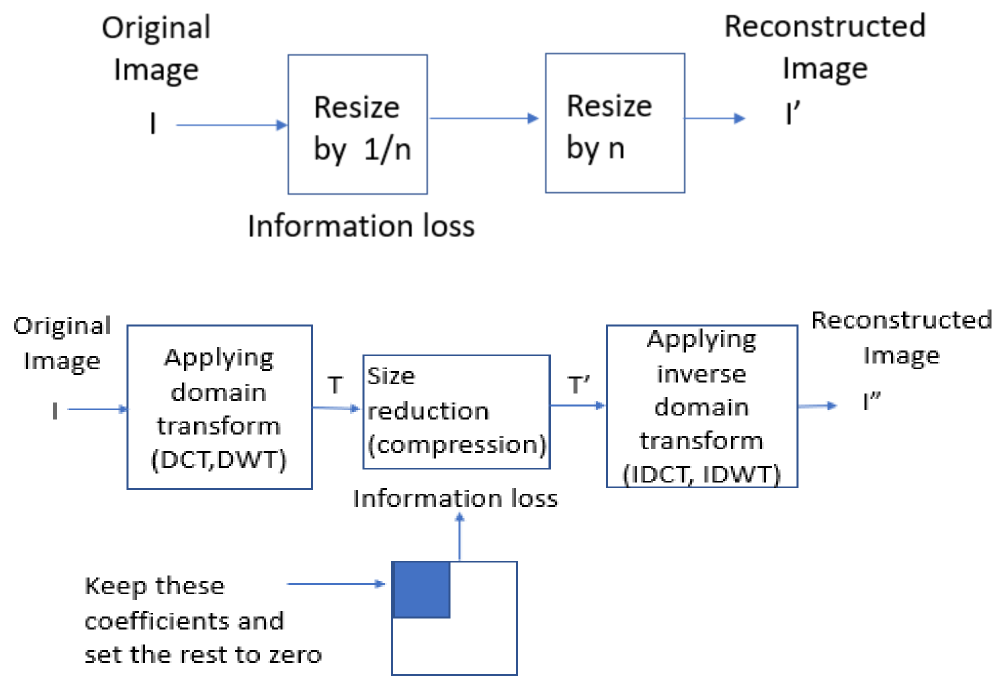

Original Image
I
Resize by 1/n
Resize by n
Reconstructed Image
I'
Information loss
Original Image
I
Applying domain transform (DCT,DWT)
T
Size reduction (compression)
T'
Applying inverse domain transform (IDCT, IDWT)
Reconstructed Image
I''
Information loss
Keep these coefficients and set the rest to zero

Figure 4.5: process that images go through for CNN with: baseline (top), DCT or DWT input transform layer (bottom), the entropy is calculated for reconstructed image I' and I". Resizing by 1/n can be done for n=2 or 4

Quantizing transformed data in different domains is not possible in a fair way since the range of values in DCT, DWT and original images are different and quantization with a small number of levels (e.g., 256) will hide a lot of details in methods such as DCT due to large range of coefficient values. To overcome this issue, we can transform the reduced size data back to the original domain original size through a lossless inverse transform of the resized data.

We note that the transforms and their inverses do not change the amount of data (thus they do not change the entropy). So, the end-to-end change in the entropy is due to the resizing (or compression) in the middle. It is straightforward to apply an entropy calculation method using 256 quantization levels in the original image domain. This process is shown in Figure 4.5.

Table 4.4: Cifar-10 normalized mean Entropy value for different transforms.

method	Mean Entropy half	Mean Entropy Quarter
Baseline	1.0000	1.0000
DCT	1.0144	1.0215
DWT	1.0081	1.0125

Table 4.5: Fashion Mnist normalized mean Entropy value for different transforms.

method	Mean Entropy half	Mean Entropy Quarter
Baseline	1.000	1.000
DCT	1.1417	1.5780
DWT	1.0572	1.5576

We have applied the size reduction methods of using DCT and DWT and then keeping half and quarter of the length and width data. The entropy resulting from these operations are measured for all images in the datasets of FMNIST and CIFAR. The results are shown in Table 4.4 and Table 4.5.

It is observed that reduced size data using DCT and DWT transforms produce higher entropies than the averaging based resizing (baseline). In Figure 4.6 we show the CDF (Cumulative distribution function) of the entropy values for the entire datasets and the different methods. It is observed that DCT produces reduced-size data with highest entropy. The DWT based method is also seen to produce results better than baseline (Table 4.4 and Table 4.5). Since higher entropy does not directly mean higher classifiable features (more noise also creates higher entropy), we also look at the quality of reconstructed images using the PSNR (peak signal-to-noise ratio) measure. Higher PSNR means that the reconstructed image is closer to the original image. We measured the PSNR for all of the reconstructed images for the two datasets and plotted their CDF in Figure 4.7. Comparing the PSNR values, it is obvious that all percentiles of the PSNR for DCT and DWT based resized images are higher than baseline, indicating a better preservation of original image quality using DCT and DWT based methods. Therefore, it is expected that the CNN performs better in classification of the higher entropy data. These methods are evaluated in the next section.

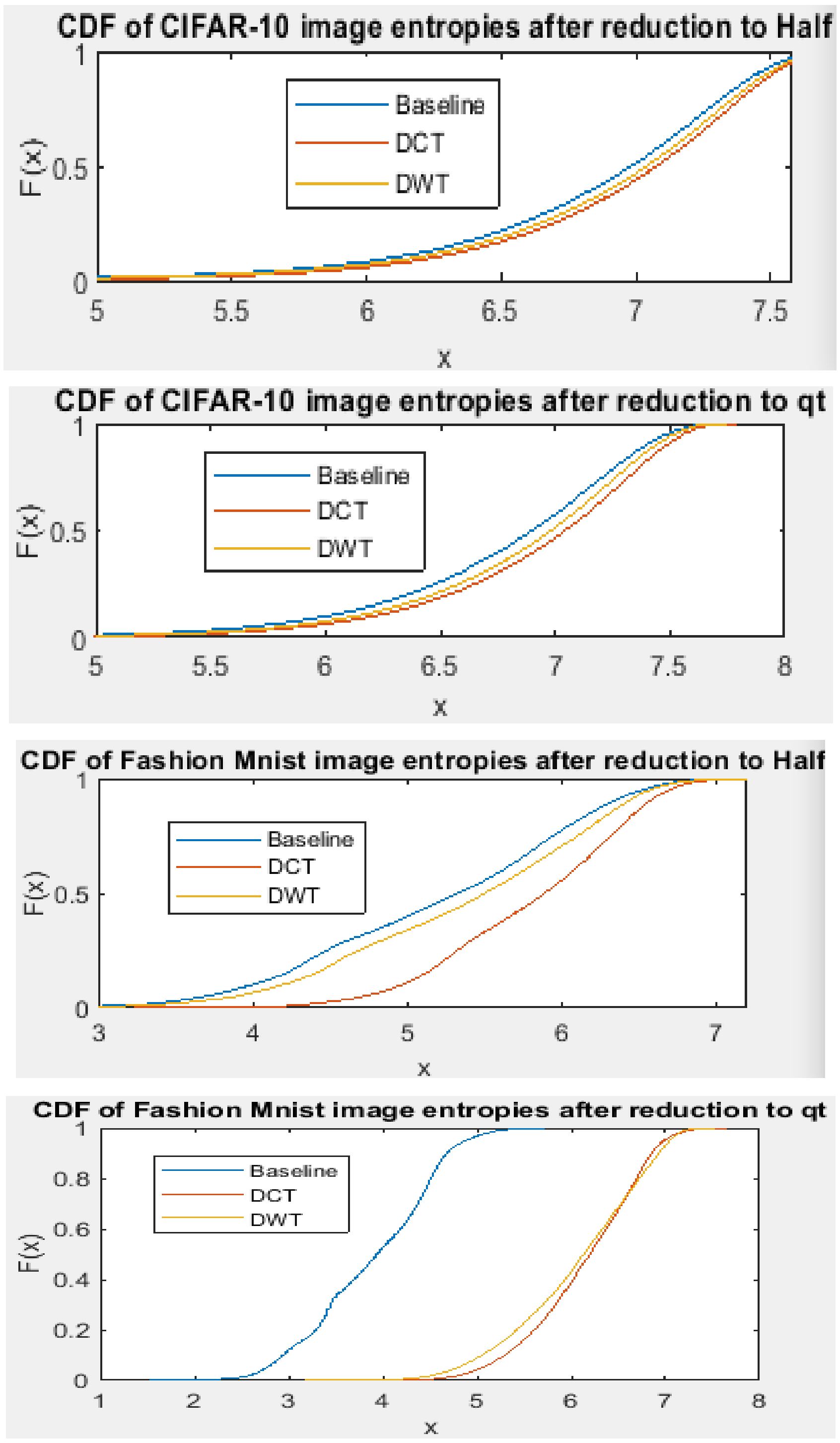


Figure 4.6: CDF of the CIFAR-10 and Fashion Mnist datasets image entropies

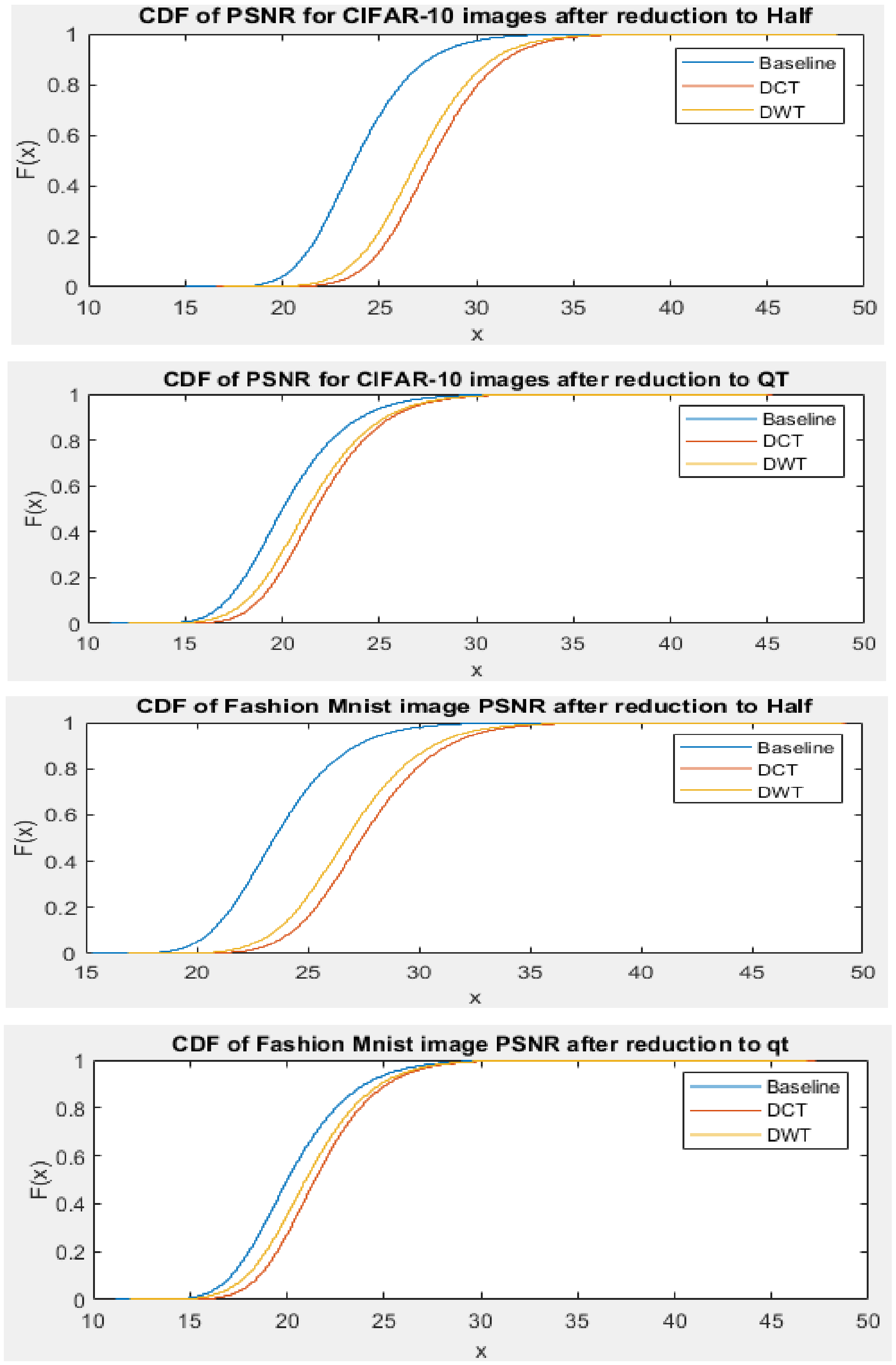


Figure 4.7: PSNR of the CIFAR-10 and Fashion Mnist datasets image entropies

4.5 Evaluation with Different Transforms

In this section, we evaluate the performance of each of the methods described in table 1. We quantify how domain transforms can improve the accuracy performance of the classification application for FMNIST and CIFAR datasets. The datasets are divided into two parts; 70% of images are used for training and 30% for testing. To ensure that a fair comparison is made between different methods, training is done for a fixed number of epochs for all methods. We use 15 epochs which is enough to reach a plateaued performance for all methods. Moreover, our objective is to compare the performance of different domain transform methods under processing constraint for small processor devices; therefore, we limit the training time. The metrics that were used here were the top-1, top-2, and top-3 accuracy performance measures. However, the comparative results for these metrics turned out to be very similar; therefore, we report top-1 for all tests, and top-3 results for one set to demonstrate the similarity of results. We used MATLAB implementation of the convolutional neural networks. Training was done using SGDM (stochastic gradient descent with momentum) method, with max epochs set to 15. Other default parameters were kept.

We first analyze the results from FMNIST dataset. In addition to evaluating performance of different domain transform and size reductions, we also look at different network configurations (sizes) to ensure that the results are not sensitive to network parameter choice. We also propose some modifications to the transform-based methods. Following confirming the robustness of results to network size variation, the larger CIFAR-10 dataset is studied.

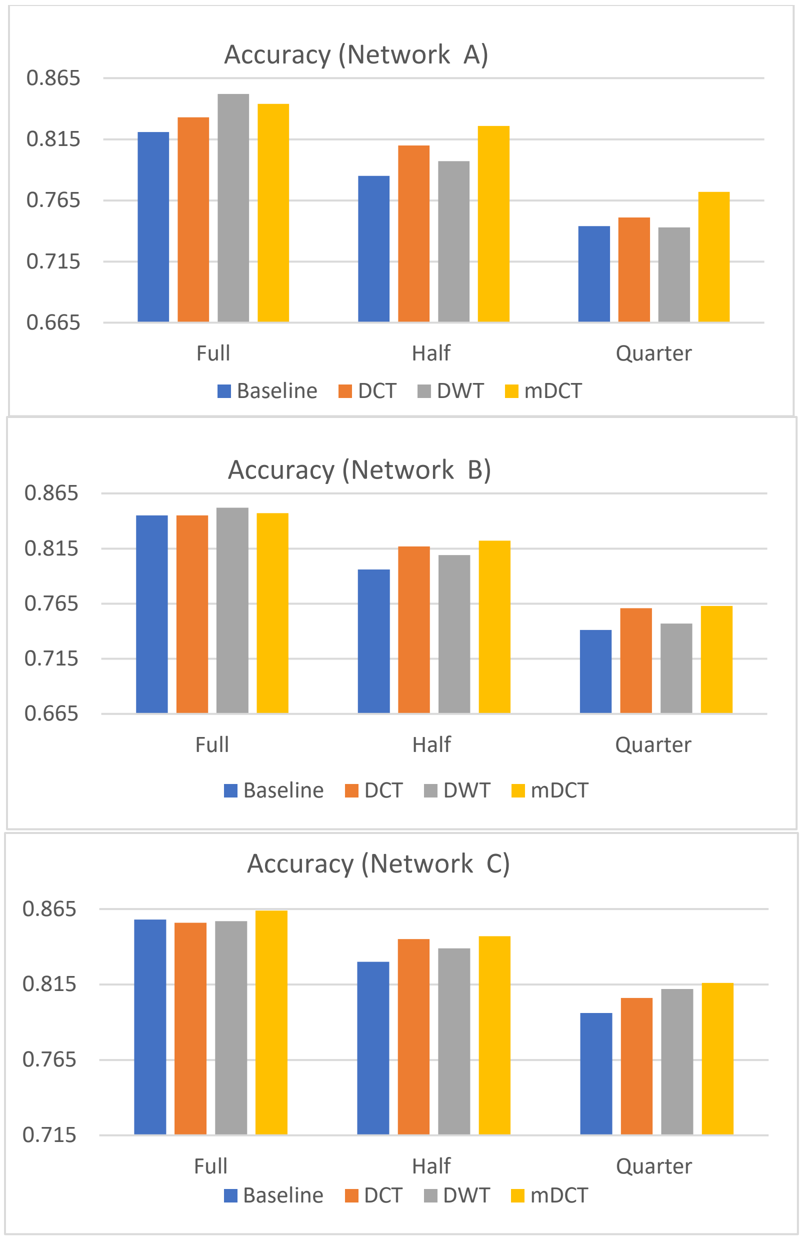


Figure 4.8: classification accuracy for three different network sizes with fashion MNIST dataset. Networks use two convolutional networks with filter size and numbers of A: (1,8;1,8), B: (1,8; 5,8), and C:(3,8; 5,8). Three inputs sizes of "Full", "Half", and "Quarter" were tried, corresponding to 32*x*32, 16*x*16 and 8*x*8 images

Results for the FMNSIT dataset are shown in Figure 4.8. We use three network configurations where the two convolutional layers have the size and number of filters as follows:

A: (1,8;1,8), B: (1,8; 5,8), and C:(3,5; 5,8). We have tried with larger number of filters (such as 256, but the run time was considerably higher without much performance gain). From Figure 4.8, we observe that with Full image size (no input size reduction), the method based on DWT produces the best results, followed by DCT in Network A and B. In network C, it is a variant of DCT based method (mDCT) that produces better results. It is also seen that the highest improvements from domain transforms happen when the network A is used.

When Half and Quarter image sizes are considered, we see that in most cases DCT-based size reduction produces the best results, up to 2.5% better than the baseline. DWT is also better than baseline in all scenarios. Results from these experiments are mostly in line with the entropy results from Table 4.5, which show DCT based compressed input has the highest median entropy, followed by DWT, for all the images in the dataset. However, it is noteworthy that a direct relationship between entropy value and CNN performance cannot be established. The main reason for the complexity of this relationship is that convolutional filters put more emphasis on localized features in images, which are better preserved in DWT (despite possibly lower entropy of a resized image).

In addition to trying DCT-based compression, we also considered enhancing the DCT coefficient, by magnifying some of the higher frequency coefficients. The rationale for this approach is that with DCT the range of values for different coefficients is very large and may not be well understood by a neural network. While for image compression, the large difference in range of coefficient values is useful in removing visually unimportant data, it may be a detriment for CNN based classification. For this purpose, we created a new method of representing DCT coefficient by multiplying each coefficient with a weight that increases linearly with frequency.

For example, if the first coefficient (1,1) is multiplied by 1, the last coefficient (32,32) is multiplied by 32; coefficient (*i*,*j*) is multiplied by *(i+j)/2*. This method magnifies the higher frequencies and is shown to slightly improve the results for DCT based method. We call this magnified DCT or mDCT in Table 4.6 for all different networks and input size options.

We also observed that while magnifying the DCT coefficients, each network type performed differently. With network C, the improvement for Half data size could be boosted with further magnification of higher frequencies (using a weight of ixj for the (i,j) weight). However, the linear method explained above was the one that performed well for all networks and is reported in Table 4.6.

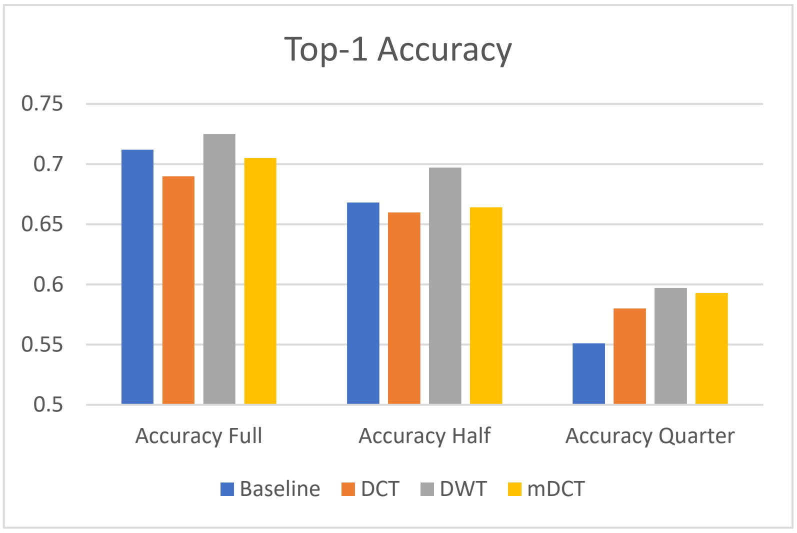
Overall, by using DCT or mDCT we could reduce the input size by 75% (using half dimensions) and still maintain the same classification accuracy (of baseline) for some cases such as network A. For network A, classification accuracy of transformed data in all sizes is 3% better than the baseline. For Networks B and C, the half-size mDCT is only 2% below what the full-size baseline method produces; also, an improvement of around 2-3% is achieved when mDCT based method is compared with the same size baseline.

In addition to the three methods reported in Table 4.6 we have also experimented with several other methods for composing input using domain transformed data. For example, we considered creating a mixed transform data by stacking coefficients from DCT-half and DWT-half in two channels. The resulting data improved the classification results by around 0.003, but at the cost of increasing the processing time by 20%. Overall, considering the small improvements, we found such higher size data to not offer enough improvement and did not further study them.

Table 4.6: Results for Fashion MNIST: classification Accuracy and training time (TT)

method	Accuracy Full / TT	Accuracy Half / TT	Accuracy Quarter /TT
Network A (1,8;1,8)			
Baseline	0.821/22	0.785/13	0.744 /7
DCT	0.833	0.810	0.751
DWT	0.852	0.797	0.743
mDCT	0.844	0.826	0.772
Network B (1,8;5,8)			
Baseline	0.845/22	0.796/10	0.741/8
DCT	0.845	0.817	0.761
DWT	0.852	0.809	0.747
mDCT	0.847	0.822	0.763
Network C (3,8;5,8)			
Baseline	0.858 /21	0.830 /15	0.796 /8
DCT	0.856	0.845	0.806
DWT	0.857	0.839	0.812
mDCT	0.864	0.847	0.816

We note that using mixed-transform methods where coefficients from multiple domains are combined to produce new representations has been studied in our recent works for compression applications [1] and regression using time delay neural networks (TDNN) [40]. Although it was shown to be useful for TDNNs, we do not see much improvement when used with CNNs. Therefore, we do not report the results here.



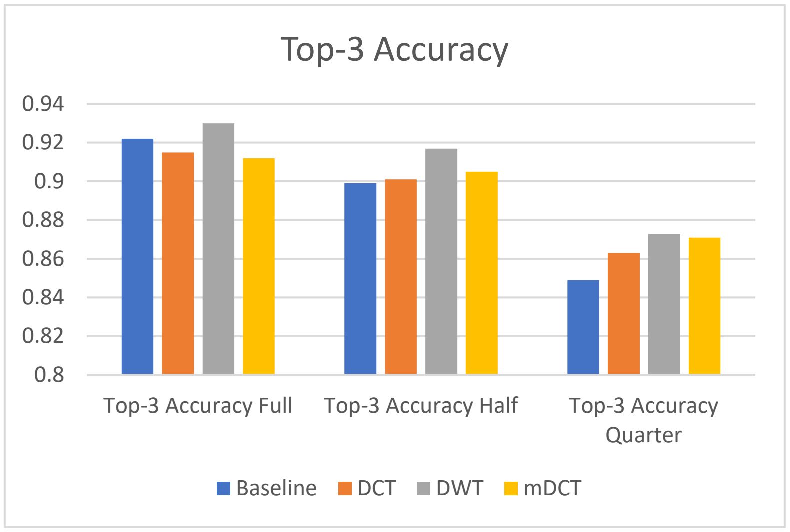


Figure 4.9: Top-1 and Top-3 classification accuracy for different transforms for Cifar-10 dataset. The maximum epoch was set to 15

Table 4.7: Cifar-10 top-1 and top-3 classification accuracy for different transforms (The maximum epoch was set to 15)

Method	Top-1 Accuracy (Full)	Top-1 Accuracy (Half)	Top-1 Accuracy (Quarter)
Baseline	0.712/100	0.668/90	0.550/90
DCT	0.690	0.660	0.580
DWT	0.725	0.697	0.597
mDCT	0.705	0.664	0.593

Method	Top-3 Accuracy (Full)	Top-3 Accuracy (Half)	Top-3 Accuracy (Quarter)
Baseline	0.922	0.899	0.849
DCT	0.915	0.901	0.863
DWT	0.930	0.917	0.873
mDCT	0.912	0.905	0.871

Following the study with FMNIST, we conclude that while the network size (e.g., number and size of filters) has moderate impact on performance, the trend for using different domain transforms is seen for all network configurations. The improvements are more prominent for CNNs with smaller and lower number of filters. We emphasize that the goal here is not to optimize the network size and configuration, but to study the impact of domain transforms.

To further confirm the findings with FMNIST, we also investigated the performance of the methods described above, with the CIFAR-10 dataset. For this dataset we have created a customized CNN (Table 4.3) for all data sizes (original, half and quarter) and considered different transforms and input size reductions using DCT and DWT. The results of experiments with the CIFAR-10 dataset, for 15 epoch training.

It is observed that the performance for the full data degrades when domain transforms are used. This is in contrast to the result that we see for the simple domain transform with FMNIST. However, the trend reverses and is similar to FMNIST results when input data size reduction is applied using DCT and DWT. Most notably, when the quarter length-width is considered, the DCT based method shows an improvement of over %4.3, and the DWT-based method outperforms baseline by around 5%. These results are in line with entropy analysis presented in Table 4.4. For the halved input length-width data, DWT improves over baseline by around 3%, while DCT based methods show no improvement in top-1 (some slight improvement in top-3). The significant improvement with quarter inputs could be due to the fact that the amount of information in quarter input size is severely limited and the more concentrated representation of DCT and DWT becomes more apparent. This means that the use of the domain transforms would be most useful when significant size reduction is needed. This trend was also seen in experiments with FMNIST dataset as well.

Another interesting point is the degrading effect of using DCT-based domain transforms for full size data in CIFAR-10, which was not seen in FMNIST. For FMNIST we observed that the improvements were lower for full-size data and higher for quarter size inputs. This might point to the fact that the improvements with using domain transforms are more visible for smaller networks.

4.5.1 Discussion on Domain Transform and The Impact of Spatial Correlation

There are several differences between CNNs and fully connected neural networks like TDNN that result in the effect of domain transforms to be different for these two network types. In our earlier work [40] we have shown that sparsifying domain transforms such as DCT would considerably speed up training of the TDNN, while also moderately improving performance. The positive impact was the result of concentration of data in a few inputs (due to sparsifying the input using transforms like DCT) and simplifying the learning task by reducing the search space for optimal weights. However, this feature is not useful for CNNs, since the convolution operation and the use of filters is meant to find features with local correlation. That is, filters in CNNs are applied to input image data assuming that features and patterns in smaller regions of the image exist and are formed through spatial correlation of data points.

When transforms such as DCT are applied, data is projected to a domain that the local spatial relationships are no longer kept in small regions of the 2-D input, but are spread over different coefficients that may not be neighbors in the new 2-D data. This means that the filters will have to learn new patterns that can help in classification. This might be the reason why application of DCT or DWT does not improve the performance when full size 2-D input is used. However, when smaller input sizes are considered, the network is able to learn the new patterns more efficiently and utilize the preserved information better than simple average based resizing of images.

4.6 Summary and Concluding Remarks

In this paper, we investigated the impact of applying domain transform to 2-D input of Convolutional Neural Networks. We considered networks with of up to 200k parameters in this study. This is an order of magnitude smaller than larger deep networks that are usually studied in literature. It is observed that reducing input data size can be efficiently achieved using transforms such as DCT and DWT. Compared to averaging-based image resizing, methods based on DCT and DWT are shown to always provide improved classification accuracy (generally 1-5%). These findings are validated using image datasets of Fashion MNIST and CIFAR-10. In some cases, input size could be reduced by 75% while still maintaining the accuracy.

The improved performance using the proposed methods can be attributed to the better preservation of the amount of information in reduced size images using transforms such as DCT and DWT. We have verified this fact through checking the entropy of resulting reduced size and transformed inputs. Reduced inputs generated using domain transform had higher entropies, and consequently produced better classification results. However, we note that higher entropy does not automatically mean better classification, as network configuration and design play also an important role.

Overall, using domain transforms with CNNs should be done carefully, as CNN filters are designed to capture localized patterns and features. If domain transforms spread the patterns such that they cannot be captured by filters, the performance of CNNs will degrade. However, it must be noted that new patterns are usually formed as a result of domain transform which visual inspection may not reveal. This is one of the questions that our future research will consider.

CHAPTER 5. CONCLUDIG REMARKS AND FUTURE DIRECTIONS

We introduced the use of sparsifying domain transforms with TDNNs for significant reduction of training time or increase in forecasting accuracy of TDNNs, we provided a mathematical analysis on the impact of sparsifying input to a feed forward ANN such as TDNN and prove positive effect on learning performance. We showed that TDNN network size could be reduced using transforms such as DCT and PCA, while maintaining forecasting performance. For sparsifying transforms, we consider transforming data to other domains using DCT (Discrete Cosine Transforms) or PCA (Principal Component Analysis), and Mixed transform of Haar and DCT. We also validated the idea by feeding simple synthetic data to the network. We investigated the use of domain transform for possible network size reduction, examining the impact of the level of sparsity on performance improvement of TDNN.

Through training and testing with three realistic datasets and two synthetic datasets, we observed that training time can be reduced significantly, while also improving the accuracy. The improvement in both training time and accuracy suggests that TDNNs for time series forecasting can greatly benefit from inclusion of an input transform layer. We have observed that for datasets such as NDDS, the training time can be reduced up to 20 times, while improving accuracy up to 3.3%. At a fixed and very short training time of 25 epochs, the use of sparsifying transforms offer up to 7.5% improvement in accuracy. For datasets with more predictable patterns, such as US101 or ERRST, the accuracy improvements were smaller; nevertheless, the training time was reduced to 40% and 25% of baseline for ERRST and US101 datasets respectively.

It is also observed that the amount of improvement is correlated with the level of sparsity; i.e., sparser input results in faster and better learning. It is therefore reasonable to assume that methods that achieve further sparsity, perhaps with non-orthogonal bases, could also be beneficial. This is a direction of or future studies.

The considerable improved performance made the possibility for additional network size reduction through removing some of the coefficients of the sparse representation of the input vector. As a result, a smaller and more focused network could deal with the major components of a sparse data, avoiding training weights that are not expected to change considerably.

As observed that this trend was generally seen for all datasets, with a 25% reduction of the input size possible for all of them. However, for some datasets such as NDDS, the input can be further compressed with up to 40% reduction in size.

Overall, it was consistently seen that applying sparsifying transforms to the input of a TDNN allows for better performance which can be traded off for reducing network size in both the input and the hidden layers. While we have analyzed the DCT and the PCA based transforms, other types of transforms may exist that could provide further improvements. The concept described here could also be employed with larger and deeper networks.

The above contributions were described in Chapters 2-4. First in Chapter 2 we demonstrated the effect of sparse representation of time series data on learning rate of time delay neural network. This was done through adding a transform layer at the beginning of a feed forward neural network. In Chapter 2, we also mathematically proved that applying a sparsifying transform to input layer will reduce the training time considerably. In Chapter 3 we investigated the effect of domain transform on network size reduction. The core concept of the proposed idea is the possibility of reducing network size in TDNNs with the help of sparse input representation. With a given level of performance (in this case, prediction error), it is possible to considerably reduce the hidden layer size of a network, simply by applying sparsifying transform to the input layer. Furthermore, the considerable improved performance allows for additional network size reduction through removing some of the coefficients of the sparse representation of the input vector.

Overall, it was consistently seen that applying sparsifying transforms to the input of a TDNN allows for better performance which can be traded off for reducing network size in both the input and the hidden layers.

In Chapter 4 the same idea from Chapter 2 was generalized and applied to image data processing using CNN’s. Fashion Mnist and CIFAR-10 (RGB images) datasets were used in this study. We considered the networks with of up to 200k parameters in this study. This was an order of magnitude smaller than larger deep networks that are usually studied in literature. It was observed that reducing input data size can be efficiently achieved using transforms such as DCT and DWT. Compared to averaging-based image resizing, methods based on DCT and DWT were shown to always provide improved classification accuracy (generally 1-4%). These findings were validated using image datasets of Fashion MNIST and CIFAR-10. In some cases, input size could be reduced by 75% while still maintaining the accuracy. The improved performance using the proposed methods can be attributed to the better preservation of the amount of information in reduced size images using transforms such as DCT and DWT. We have verified this fact through checking the entropy of resulting reduced size and transformed inputs. Reduced inputs generated using domain transform had higher entropies, and consequently produced better classification results. However, we noted that higher entropy does not automatically mean better classification, as network configuration and design play an important role too. This can be considered as an important factor for further studies.

Overall, using domain transforms with CNNs should be done carefully, as CNN filters are designed to capture localized patterns and features. If domain transforms spread the patterns such that they cannot be captured by localized filters, the performance of CNNs will degrade. However, it must be noted that new patterns are usually formed as a result of domain transform which visual inspection may not reveal and could be captured with larger filters.

While we have analyzed the DCT, DWT, mixed transform and PCA based transforms, other types of transforms may exist that could provide further improvements. For example, independent component analysis (ICA) and dictionary learning (DL) methods can be studied. With methods such as DL, we may be able to control the level of sparsity and further investigate the impact of sparsity on the neural network performance.

REFERENCES

[1] Masoumeh Kalantari Khandani. and Wasfy B. Mikhael.: *Using Mixed DCT and Haar Transforms for Efficient Compression of Car Trajectory Data. IEEE 61 international Midwest Symp.On Circuits and sys.*, 2018

[2] Masoumeh Kalantari Khandani. and Wasfy B. Mikhael: *Efficient Time Series Forecasting Using Time Delay Neural Networks with Domain Pre-Transforms.* IEEE MWSCAS, Dallas, Texas, August 2019

[3] Mikhael, W. B., Ramaswamy, A,:*Efficient Representation of Nonstationary Signals Using Mixed-Transforms with Applications to Speech IEEE Transanction on Circuits and Systems-11:Analog and Digital Signal Processing*, Vol. 42, NO. 6. June 1995

[4] Courbariaux, M., Bengio, Y., David, P. B. : *BinaryConnect: training deep neural networks with binary weights during propagations, Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, pp. 3123-3131, 2015.

[5] Dingus, T. A., Klauer, S. G., Neale, V. L., Peterson, A., Lee, S. E., Sudweeks, J., Perez, M. A., Hankey, J., Ramsey, D., Gupta, S., Bucher, C., Doerzaph, Z. R., Jarmeland, J., & Knipling, R. R. 2006.: *The 100-Car naturalistic driving study, Phase II – Results of the 100-Car field experiment. DC, National Highway Traffic Safety Administration.* 2006

[6] *NGSIM Homepage. FHWA. http://ngsim.fhwa.dot.gov.*

[7] Aharon, M., Elad, M., and A. Bruckstein: *An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. IEEE Transactions on Signal Processing, K-SVD*, 2006

[8] Girish, K. Jha, Sinha, K.: *Time-delay neural networks for time series prediction:an application to the monthly wholesale price of oilseeds in India. Neural Computing and Applications.* Springer Link March 2014,

[9] Aussem, Al., Murtagh F.: *Combining Neural Network Forecasts on Wavelet-transformed Time Series. Connect. Sci.* 9 (1997): 113-122.

[10] X. Jiang, H. Adeli: *Dynamic Wavelet Neural Network Model for Traffic Flow Forecasting. Journal of Transportation Eng*.2005

[11] D. Huang Xing-rong Bai.: *A Wavelet Neural Network Optimal Control Model for Traffic-Flow Prediction in Intel. Transport Systems. Int. Conf. on Intelligent Computing* ICIC 2007

[12] Benmahdjoub,K., Ameur,Z., Boulifa, M.: *Forecasting of Rainfall Using Time Delay NN in Tizi-Ouzou. Energy Procedia,* Vol 36, 2013

[13] Wanga, J., Tsapakisb, I., Zhonga, C.: *A space–time delay neural network model for travel time prediction. Eng. Appls of AI* 2016

[14] Cecotti, H., Gräser, A.: *Time Delay Neural Network with Fourier transform for multiple channel detection of Steady-State Visual Evoked Potentials for Brain-Computer Interfaces,* 2008 European Signal Processing Conf

[15] Shi, D., Zhang, H., Yang, L.: *Time-Delay Neural Network for the Prediction of Carbonation Tower's Temperature. IEEE Transactions on Instrumentation and Measurement*, Vol. 52, NO. 4, 2003

[16] Meng, H., Bianchi-Berthouze, N., Deng, Y., Cheng, J., Cosmas, J. P.: *TDNN for Continuous Emotional Dim Prediction From Facial Expression Sequences", IEEE Trans on Cybernetics*, Vol. 46, NO. 4, 2016

[17] *https://www.ncdc.noaa.gov/data-access/marineocean-data/extended-reconstructed-sea-surface-temperature-ersst-v5*

[18] Abdi., H., Williams, L.J.:*Principal component analysis. Wiley Interdisciplinary Reviews: Computational Statistics*. 2 (4): 433–459. arXiv:1108.4372. doi:10.1002/wics.101. (2010).

[19] Shaw P.J.A.: *Multivariate statistics for the Environmental Sciences, Hodder-Arnold.* ISBN0-340-80763-6(2003)

[20] Pearson, K.: *On Lines and Planes of Closest Fit to Systems of Points in Space. Philosophical Magazine*. 2 (11): 559–572. doi:10.1080/14786440109462720. (1901).

[21] Rezaei S, Sengupta R, Krishnan H, Guan X.: *Tracking the position of neighboring vehicles using wireless communications* 2010 - Elsevier

[22] Hagan M.T, Menhaj M Training.: *feed-forward networks with the Marquardt algorithm. IEEE Trans Neural Netw* 5:989–993(1994)

[23] Demuth H, Beale M.: *Neural network toolbox user's guide.* Mathworks, Natic(2002)

[24] Pan, Z., Bolouri,H.: *High Speed Face Recognition Based on Discrete Cosine Transforms and Neural Networks. http://citeseer.ist.psu.edu/270448.html. submitted to IEEE trans.* On PAMI 1999.

[25] Kim,Tae-W., Valdes, J, B.: *Nonlinear Model for Drought Forecasting Based on a Conjunction of Wavelet Transforms and Neural Networks. ASCE, Journal of Hydrologic Engineering*, 2003

[26] Patil, K., Deo,M. C.: *Basin-Scale Prediction of Sea Surface Temperature with Artificial Neural Networks.* 2018

[27] Patil, K.,Deo M. C., 2017: *Prediction of daily sea surface temperature using efficient neural networks. Ocean Dyn., 67, 357–368,* https://doi.org/10.1007/s10236-017-1032-9.

[28] Mahongo, S. B., M. C. Deo, 2013: *Using artificial neural networks to forecast monthly and seasonal sea surface temperature anomalies in the western Indian Ocean. Int. J. Ocean Climate Syst.*, 4, 133–150, https://doi.org/10.1260/1759-3131.4.2.133.

[29] Pisoni, E.,. Pastor,F., Volta, M.: *Artificial Neural Networks to reconstruct incomplete satellite data: application to the Mediterranean Sea Surface Temperature. Nonlinear Processes in Geophysics*, vol. 15, no. 1, pp. 61-70, Feb. 2008

[30] Vijayaditya,P., Daniel P.,Sanjeev, K.: *A time delay neural network architecture for efficient modeling of long temporal contexts Interspeech*. 3214-18, 2015

[31] Huang, X., Zhang W., Xu, X., Yin, R., Chen, D.: *Deeper Time Delay Neural Networks for Effective Acoustic Modelling. Published under licence by IOP Publishing Ltd, Journal of Physics: Conference Series,* Volume 12229- 012076'

[32] Buono, A., Agmalaro, M. A., Almira, A. F.: *Forecasting the length of the rainy season using time delay neural network. 2014 International Conference on Advanced Computer Science and Information System, Jakarta*, 2014, pp. 315-320.

[33] Benmahdjoub, K., Ameur, Z., Boulifa, M.: *Forecasting of rainfall using time delay neural network in Tizi-Ouzou (Algeria). Energy Procedia,* vol 36, page 1138-1146, 2013

[34] Aggarwal A., Tripathi, M. M.: *A novel hybrid approach using wavelet transform, time series time delay neural network, and error predicting algorithm for day-ahead electricity price forecasting. IEEE 6th International Conference on Computer Applications In Elec. Eng-Recent Advances* (CERA), 2017

[35] Wang, Y., Xu, C., Xu C., Tao, D.: *Packing Convolutional Neural Networks in the Frequency Domain. in IEEE Trans. on Pattern Analysis and Machine Intel.,* vol. 41, no. 10, pp. 2495-2510, 2019.

[36] "haar". *Fourier.eng.hmc.edu. 30 October 2013. Retrieved 23 November 2013. http://fourier.eng.hmc.edu/e161/lectures/Haar/node1.html*

[37] Lee, B.; Tarng, Y. S.: *Application of the discrete wavelet transform to the monitoring of tool failure in end milling using the spindle motor current. International Journal of Advanced Manufacturing Technology.* 15 (4): 238–243. doi:10.1007/s001700050062(1999)

[38] Hochreiter, S., Schmidhuber J.: *Long short-term memory. Neural Comput. 1997; 9(8):1735–1780. doi:10.1162/neco.1997.9.8.1735*

[39] Song Han, Huizi Mao, and William J Dally. *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding.* 2016.

[40] Masoumeh Kalantari Khandani. and Wasfy B. Mikhael: *Effect of Sparse Representation of Time Series Data on Learning Rate of Time Delay Neural Networks, Circuits, Systems, and Signal Processing*, 2021. DOI: 10.1007/s00034-020-01610-8

[41] arXiv:1512.03385: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: *Deep Residual Learning for Image Recognition:* 2015

[42] arXiv:1409.1556v6: Karen Simonyan, Andrew Zisserman :*Very Deep Convolutional Networks for Large-Scale Image Recognition,*2015

[43] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. (2017-05-24). "*ImageNet classification with deep convolutional neural networks". Communications of the ACM.* 60 (6): 84–90. doi:10.1145/3065386. ISSN 0001-0782. S2CID 195908774.

[44] arXiv:1412.6806v3, Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, *STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET, workshop contribution at ICLR* 2015

[45] Graham, Benjamin. *Fractional max-pooling. In arxiv:cs/arXiv:1412.6071, 2015.*

[46] Ghosh A., Chellappa R. : *Deep Feature Extraction in the DCT domain. 23rd International Conference on Pattern Recognition (ICPR)* , 3525-3530(2016)

[47] Rajesh B. , Javed, M., Ratnesh , Srivastava, S. :DCT-CompCNN: *A Novel Image Classification Network Using JPEG Compressed DCT Coefficients.*

[48] Ciurana, A., Mosella-Montoro, A., Ruiz-Hidalgo, J.: *Hybrid Cosine Based Convolutional Neural Networks,* https://arxiv.org/pdf/1904.01987.pdf

[49] *Classification of Garments from Fashion MNIST Dataset Using CNN LeNet-5 Architecture*

[50] Kayed, Mohammed, Anter, Ahmed, Mohamed, Hadeer: *Int. Conf. on Innovative Trends in Communication and Computer Engineering (ITCE) Innovative Trends in Communication and Computer Engineering (ITCE), 2020 International Conference on.* :238-243 Feb, 2020

[51] Lars Hertel, Erhardt Barth, Thomas Käster, Thomas Martinetz: *Deep Convolutional Neural Networks as Generic Feature Extractors, IJCNN* 2015

[52] A. Coates, H. Lee, and A. Ng. *An analysis of single-layer networks in unsupervised feature learning. In NIPS*2010 Workshop on Deep Learning,* 2010.

[53] S Shubathra, PCD Kalaivaani; *S Santhoshkumar Clothing Image Recognition Based on Multiple Features Using Deep Neural Networks , International Conference on Electronics and Sustainable Communication Systems (ICESC),* 2020 IEEE

[54] Johannes Langelaar (2020). *MNIST neural network training and testing (https://www.mathworks.com/matlabcentral/fileexchange/73010-mnist-neural-network-training-and-testing), MATLAB Central File Exchange. Retrieved* Dec.11, 2020.

[55] *https://www.mathworks.com/solutions/deep-learning/examples/training-a-model-from-scratch.html*

[56] Alex Krizhevsky*: Learning Multiple Layers of Features from Tiny Images:* 2009

[57] *https://www.cs.toronto.edu/~kriz/cifar.html*

[58] Han Xiao, Kashif Rasul, Roland Vollgraf, *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*:2017: https://arxiv.org/pdf/1708.07747.pdf

[59] A. Torralba, R. Fergus, and W. Freeman. *80 million tiny images: A large data set for nonparametric object and scene recognition. IEEE PAMI,* 30(11):1958–1970, 2008.

[60] Mahdi Hashemi, *Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation: Journal of Big Data, Springer* (2019) 6:98 https://doi.org/10.1186/s40537-019-0263-7

[61] Cohen, A.; Daubechies, I.; Feauveau, J.-C. (1992). *"Biorthogonal bases of compactly supported wavelets". Communications on Pure and Applied Mathematics*. 45 (5): 485–560. doi:10.1002/cpa.3160450502

[62] *Daubechies, Ingrid (1992). Ten Lectures on wavelets. SIAM*. doi:10.1137/1.9781611970104. ISBN 978-0-89871-274-2.

[63] Ch. Thum,: *Measurement of the Entropy of an Image with Application to Image Focusing:* Pages 203-211 (2010) https://doi.org/10.1080/713821475

[64] https://github.com/zalandoresearch/fashion-mnist

[65] Masoumeh Kalantari Khandani, Wasfy B. Mikhael. : *A Study on Network Size Reduction Using Sparse Input Representation in Time Delay Neural Networks*. 2020 IEEE 63rd Int. Midwest Symposium

[66] Huang, Ch-L; Fallah, Y. P., Sengupta, R.; Krishnan, H., *"Adaptive Intervehicle Communication Control for Cooperative Safety Systems"*, IEEE Network, ISSN 0890-8044.vol 23. Issue 1.6 – 13(2010)

[67] Chen, P., Niu, A., Liu, D., Jiang, W., Ma, B. : *Time Series Forecasting of Temperatures using SARIMA: An Example from Nanjing*. 2018 IOP Conf. Ser.: Mater. Sci. Eng. 394 052024 (2018)

[68] Masoumeh Kalantari Khandani, Wasfy B. Mikhael: *Efficient Size Reduction of Convolutional Neural Networks Using Domain Transforms*: Submitted to Circuits, Systems, and Signal Processing, 2021.

[69] KevinClark:*ComputingNeuralNetworkGradients*: https://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf