DETERMINISTIC TRANSFORM BASED WEIGHT MATRICES FOR NEURAL NETWORKS

Pol Grau Jurado[1] *Xinyue Liang*[2] *Saikat Chatterjee*[1]

[1]*School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Sweden*
[2] *College of Information Science and Engineering, Ocean University of China, China*
polgrauj@gmail.com, xinyuel92@163.com, sach@kth.se

ABSTRACT

We propose to use deterministic transforms as weight matrices for several feedforward neural networks. The use of deterministic transforms helps to reduce the computational complexity in two ways: (1) matrix-vector product complexity in forward pass, helping real time complexity, and (2) fully avoiding backpropagation in the training stage. For each layer of a feedforward network, we propose two unsupervised methods to choose the most appropriate deterministic transform from a set of transforms (a bag of well-known transforms). Experimental results show that the use of deterministic transforms is as good as traditional random matrices in the sense of providing similar classification performance.

Index Terms— Randomized neural networks, multilayer neural network, deterministic transforms, weight matrices.

1. INTRODUCTION

Over the last decade, machine learning has lived a revolution born from the exponential technology evolution. Nowadays deep neural networks (DNNs) [1] and convolutional neural networks (CNNs) [2] are leading the machine learning field due to their capability to outperform classical methods on the classification and regression tasks [3, 4]. However, these typical neural network structures rely on extensive computational resources.

Two classes of neural networks have been developed to deal with the high computational complexity. The first class focuses on maintaining state-of-art performance while reducing the network depth and width as much as possible. EfficientNet [5], SqueezeNet [6], and MobileNet [7] are examples of this kind. The second class is based on gradient-free training. One popular technique implemented is to set some weight matrices as random instances and fix them during the training stage. Extreme learning machine (ELM) [8], random vector functional link (RVFL) and its variants [9, 10], progressive learning network (PLN) [11], and self size-estimating feedforward network (SSFN) [12] are examples of this class. Even the training process is reduced, mainly removing the backpropagation algorithm, this class of networks has shown to achieve competitive performances in various applications.

Article contribution: Our recent work [13] shows the use of deterministic transform instead of random matrices for SSFN architecture [12]. The main reason for the use of deterministic transform is a further reduction in computational complexity than the random matrix based SSFN. This helps for real time implementation of neural networks, including deep versions.

In this article, we go beyond the SSFN architecture. We investigate the use of deterministic transforms for other randomization based neural networks, mainly ELM, RVFL and deep RVFL (dRVFL) [8–10]. Therefore this article shows a comprehensive study on the use of deterministic transforms for a large class of randomization matrix based neural network architectures, replacing the random matrices. We show that the use of deterministic transforms lead to a significant reduction in computational complexity by analytical arguments, without loss in performance by experimental verifications.
Relevant literature: The prominent related method where deterministic transforms are used as weight matrices is the scattering network [14]. In the scattering networks, Wavelets are used as the weights matrices and the neural network becomes wider with an increase in layers. The major difference with our proposed approach is that we choose appropriate transform from a set of transforms, if required in each layer of a network architecture. That means we do not only restrict to the use of Wavelets. In addition, we also prune the neural networks to control the width of a network.

We also mention that, due to low-complexity advantage, random matrix based neural networks are scalable for large-scale application scenarios. For example, they can be efficiently used for decentralized machine learning scenarios with large size training data across nodes. Such decentralized learning scenarios were demonstrated in recent results [15–17]. While we do not consider decentralized learning scenarios in this article, we mention that the proposed deterministic transform based methods can be easily deployed for decentralized learning, like the random matrix based counterparts.
Article organization: We provide preliminaries of randomization-based neural networks and deterministic transforms in Section 2. Section 3 describes the use of deterministic transforms as weight matrices. Finally, experimental results for classification using nine datasets are discussed in Section 4.

2. PRELIMINARIES

2.1. Randomization-based Feedforward Networks

Randomization-based feedforward networks are designed to initialize some of their weights as random matrices and keep them fixed in the training stage. This technique was initially designed to reduce the computational training complexity. In this Subsection are presented the four network architectures considered in this article and divided between two groups, single-layer networks (SLFNs) and multi-layer or deep neural networks (DNNs).

For the networks description, consider the training dataset $\mathcal{D} = \{(\mathbf{x}^{(j)} \in \mathbb{R}^P, \mathbf{t}^{(j)} \in \mathbb{R}^Q)\}_{j=1}^J$, with $\mathbf{x} = \left[\mathbf{x}^{(1)} ... \mathbf{x}^{(J)}\right] \in \mathbb{R}^{P \times J}$ and analogously $\mathbf{t} \in \mathbb{R}^{Q \times J}$. Consider the predicted target vector as

$$\tilde{\mathbf{t}} = \mathbf{O}\mathbf{y}_l = \mathbf{g}(\mathbf{W}_l \mathbf{y}_{l-1}) = \mathbf{g}(\mathbf{W}_l ... \mathbf{g}(\mathbf{W}_2 \mathbf{g}(\mathbf{W}_1 \mathbf{x})) ...), \quad (1)$$

where $\mathbf{y}_0 = \mathbf{x}$, $\mathbf{y}_l \in \mathbb{R}^{N_l}$ is the output of the l'th layer, $\mathbf{W}_l \in \mathbb{R}^{N_l \times N_{l-1}}$ is the weight matrix of l'th layer with N_l hidden neurons, $\mathbf{g}(\cdot)$ is the activation function, and $\mathbf{O} \in \mathbb{R}^{Q \times N_l}$ being the output

matrix. To optimize the output matrix, a minimization optimization problem has to be solved. This optimization is expressed as

$$\mathbf{O}^{\star} = \arg\min_{\mathbf{O}} \mathcal{C} = \frac{1}{J}\sum_{j=1}^{J} ||\mathbf{t}^{(j)} - \tilde{\mathbf{t}}^{(j)}||^2 \quad \text{s.t. } ||\mathbf{O}||_F^2 \leq \epsilon, \quad (2)$$

where $\mathcal{C}$ as a predefined ℓ_2-norm cost function and ϵ acting as a regularization parameter [12]. Output weights are learned implementing alternating direction method of multipliers (ADMM).

2.1.1. Single-layer feedforward networks (SLFNs)

ELM and RVFL architectures are both a SLFN, based on the randomization of the hidden layer and optimization of the output layer. Their main and unique difference, is that RVFL present a direct link between the network input and output. RVFL feature vector is therefore obtained as follows

$$\mathbf{y}_{RVFL} = \begin{bmatrix} \mathbf{y}_{ELM} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{g}(\mathbf{R}\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \in \mathbb{R}^{P+N}, \quad (3)$$

where $\mathbf{R} \in \mathbb{R}^{N\times P}$ is a random instance weight matrix, fixed once initialized [8] and N the number of hidden neurons.

2.1.2. Deep RVFL (dRVFL)

dRVFL is created by extending RVFL, placing randomized weight layers on top of each other creating a multi-layer architecture with L layers. Each layer output is taken as the next layer input and also stacked to the output matrix input vector, as

$$\mathbf{y}_{L,dRVFL} = \begin{bmatrix} \mathbf{g}(\mathbf{R}_1\mathbf{x}) \\ \vdots \\ \mathbf{g}(\mathbf{R}_L\mathbf{g}\ldots(\mathbf{R}_1\mathbf{x})) \\ \mathbf{x} \end{bmatrix} \in \mathbb{R}^{P+\sum_{l=1}^{L} N_l} \quad (4)$$

2.1.3. Self Size-estimating Feedforward Network (SSFN)

SSFN is a multi-layer randomization-based feedforward network architecture built following a layer-wise approach that estimates its own size. It presents a more complex structure than the previous architectures but still relies on the use of randomized matrices. A new layer is added on top of the previously built network until the training cost presents a saturation trend on the l'th layer, $\frac{\mathcal{C}_l^{\star} - \mathcal{C}_{l-1}^{\star}}{\mathcal{C}_{l-1}^{\star}} < \eta_{layer}$, or the maximum number of layers L_{max} is reached.

Network weights are formed concatenating the previously optimized output matrix, together with a random instance, defining the feature vector on l'th layer as

$$\mathbf{y}_{l,SSFN} = \mathbf{W}_l\mathbf{y}_{l-1,SSFN} = \begin{bmatrix} \mathbf{V}_Q\mathbf{O}_{l-1}^{\star} \\ \mathbf{R}_l \end{bmatrix} \mathbf{y}_{l-1,SSFN}, \quad (5)$$

where $\mathbf{V}_Q = [\mathbf{I}_Q \ -\mathbf{I}_Q]^T \in \mathbb{R}^{2Q\times Q}$ and $\mathbf{O}_{l-1}^{\star}$ is optimized by (2).

2.2. Deterministic Transforms

In this article, the term *deterministic transforms* (DT) is used to refer to discrete linear transforms used signal processing tasks, for example, orthogonal transforms. They can be defined either as a matrix product or as linear functions, thus the feature vector is defined by

$$\mathbf{y} = \mathbf{W}_{DT}\mathbf{x} \triangleq \boldsymbol{w}_{DT}(\mathbf{x}), \quad (6)$$

Algorithm 1 : Unsupervised learning of deterministic transforms

1: **for** $i = 1 : I$ **do**
2: $\quad \boldsymbol{w}_{DT,i}(\cdot) = DT_i$ (Choose m'th DT in the bag)
3: $\quad \mathbf{z}_{l,i} = \boldsymbol{w}_{DT,i}(\mathbf{y}_{l-1})$ (Compute linear transform output)
4: $\quad sc(i) = m(p(\mathbf{z}_{l,i}))$ ("m" corresponds to apply score-based methods in 3.1 and "p" to the pruning function in 3.2)
5: **end for**
6: $\boldsymbol{w}_{DT}(\cdot) \leftarrow \arg\max_{DT_i}(sc(i))$ (DT to be used)

where $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^N$, $\mathbf{W}_{DT} \in \mathbb{R}^{N\times N}$ and, $\boldsymbol{w}_{DT}(\cdot) \in \mathbb{R}^N \rightarrow \mathbb{R}^N$.

In this article, the following deterministic transforms are used: Discrete Cosine Transform (DCT), Discrete Sine Transform (DST), Walsh-Hadamard transform with two different output orderings (FWHT1 and FWHT2), discrete Hartley transform (DHT), discrete Haar transform (Haar) and, wavelet transforms (Daubechies 4 (DB4) and 20 (DB20), symlets 2 (Sym2), coiflets 1 (Coif1), biorthogonal 1.3 (bior1.3) and, reverse biorthogonal 1.1 (rbior1.1)). The different transforms are chosen according to popularity and task diversity. For example, DCT and DST decompose signals in cosine and sine components, providing interpretability and computational advantage.

Deterministic transforms present a structure that allows being computed by fast algorithms which typically make use of matrix factorization, cascade filterbanks and, recursive implementations. Implementing these fast algorithms the cost can be reduced from $\mathcal{O}(N^2)$ in matrix-vector product, to $\mathcal{O}(Nlog_2N)$ [18, 19], or even $\mathcal{O}(N)$ [20,21]. For the neural networks discussed in Section 2.1. We will explore the use of deterministic transforms instead of random matrices in the next section.

3. USE OF DETERMINISTIC TRANSFORMS

In this section, we present a method to choose a suitable deterministic transform to replace random matrices in randomized neural network architectures. A fact is that a random matrix can be any matrix, therefore any matrix can replace a random matrix, and by using deterministic transforms the computational complexity can be reduced.

Let us consider a randomization-based neural network without loss of generality, for example, dRVFL. We now replace random matrices of dRVFL with suitable deterministic transforms as weight matrices. Note that the same deterministic transform may not be used throughout all layers in the deep structures of the dRVFL. We first define a bag that contains I number of deterministic transforms $DT = \{DT_1\ DT_2 \ldots DT_I\}$, where DT_i denotes i'th transform. We can choose DT_1 as DCT, DT_2 as DST, and further.

Lets define the signal flow across layers as

$$\mathbf{y}_l = \mathbf{g}(\mathbf{z}_l') = \mathbf{g}(p(\mathbf{z}_l)) = \mathbf{g}(p(\boldsymbol{w}_{DT,l}\mathbf{y_{l-1}})), \quad (7)$$

with $\mathbf{z}_l$ being the deterministic transform output on layer l, $p(\cdot)$ representing a node pruning function, further defined in Section 3.2, and $\mathbf{z}_l'$ the corresponding pruned vector.

To choose a transform in l'th layer, an iterative procedure is carried to $\mathbf{z}_l'$ along all the I transform candidates. Two different methods based on statistical signal properties have been developed. Both of them are score-based methods, meaning that a score sc is assigned to each transform. The transform presenting a greater score is the one chosen to be implemented in that layer. The procedure is summarized in Algorithm 1 and the score-based methods are explained in the subsection.

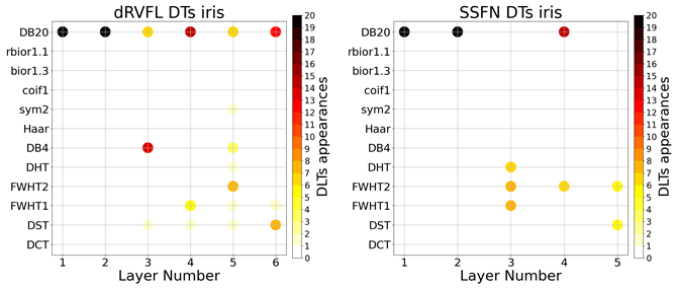

Fig. 1: Network deterministic transforms architecture for Iris dataset implementing Method 2. "Y" axis presents the transforms within the bag. Legend indicates the number of times a transform is chosen in 20 Monte-Carlo simulations, as darker, more times has been chosen.

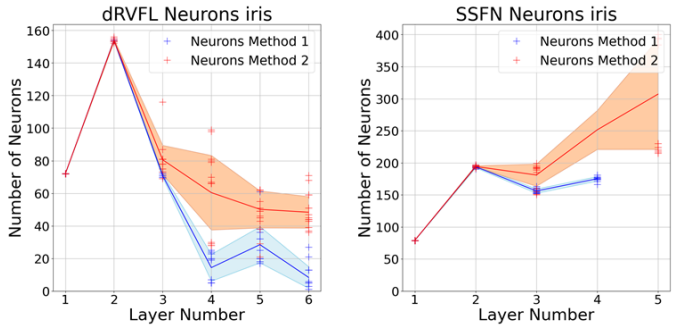

Fig. 2: Network nodes architecture for Iris dataset. Results for Method 1 are represented in blue and for Method 2 in orange. Each point represents neurons for a simulation, straight line is the simulations mean and the shading represents nodes standard deviation.

3.1. Score-based methods

3.1.1. Method 1: Standard deviations

Variability in layer nodes gives an insight about the amount of information each node handles [22]. The score sc corresponds to the standard deviation among all nodes standard deviation, i.e., the variation among all nodes variation. The score is assigned then as

$$sc_1 = \frac{1}{\sigma_T} = \frac{1}{\sqrt{\frac{1}{N_l}\sum_{n''=1}^{N_l}(\sigma_n^{(n'')} - \frac{1}{N_l}\sum_{n'=1}^{N_l}\sigma_n^{(n')})^2}}, \quad (8)$$

where σ_n is a single node standard deviation computed across all samples as $\sigma_n = (\frac{1}{J}\sum_{j'=1}^{J}(z_n^{(j')} - \frac{1}{J}\sum_{j=1}^{J} z_n^{(j)})^2)^{\frac{1}{2}}$.

3.1.2. Method 2: Singular values of cross-correlation matrix

In [23] was suggested to preserve the maximum amount of information after each layer. The insight of this second method is to measure the information shared between the network input $\mathbf{x}$ and linear transform output $\mathbf{z}_l$, to choose the transform with both signals presenting more similarities.

To measure the information shared between "network input-layer output", their correlation matrix $\mathbf{R}_{\mathbf{x},\mathbf{z}}$ is computed. Principal component analysis (PCA) is carried to the correlation matrix to obtain the corresponding K singular values $\{\lambda_k\}$. Cumulative singular value curve $C_\lambda(k)$ is defined by sorting the singular values in descending order and normalizing, resulting

$$C_\lambda(k) = \frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{K}\lambda_i}, \qquad \text{where } 0 \leq C_\lambda(k) \leq 1. \quad (9)$$

A constant slope of $C_\lambda(k)$ means that all singular values are similar and consequently the signals are independent. On the contrary as greater the gradient, more percentage contained in fewer singular values and therefore more information shared between both signals.

An hyperparameter $0 \leq \gamma \leq 1$ is introduced to act as a percentage threshold. γ is set as the percentage considered adequate to keep the information, in order to choose the transform that presents a greater energy in less singular values. $k^\star$ is defined as the first singular value from where the cumulative is greater or equal than γ,

$$k^\star = \arg\min_k (C_\lambda(k) \geq \gamma). \quad (10)$$

It may happen that the output of the different transforms varies in length, therefore K being different. To avoid the influence of different signal lengths, the final score sc is defined as the previously computed index $k^\star$ normalized by the amount of singular values as

$$sc_2 = \frac{1}{k^\star/K} = \frac{K}{k^\star}. \quad (11)$$

3.2. Pruning of nodes in neural networks

Most deterministic transforms are squared matrices, e.g., DCS, Hartley transform, among others. Furthermore, some transforms size are integers of power of 2. In DNNs, this results in a non-decreasing network width as the network gets deeper, and therefore a node pruning technique needs to be applied.

The pruning function $p(\cdot)$, in equation (7) has been defined to remove nodes presenting less variance than a threshold η_{var}, an hyperparameter previously set.

4. EXPERIMENTAL EVALUATIONS

The experiments are carried using nine different datasets, all of them widely used multi-class classification. Datasets are chosen due to its popularity in literature, diversity, and level of complexity for tasks. It is important to mention that the train and test partitions for Iris, Glass, Wine and Letter datasets are chosen randomly, while the other dataset partitions remain constant.

The bag of deterministic transforms is created with the $T = 11$ different deterministic transforms mentioned in Subsection 2.2. It is important to mention that for discrete wavelet transform, the output is constructed by concatenating approximation and detail coefficients at its maximum decomposition level. The bag transform can be modified by adding or replacing other deterministic transforms. As a activation function $g(\cdot)$ ReLu function has been chosen.

For reproducibility of the experiments, refer to the software codes in Python posted in `https://github.com/Polgrauj/DTs_in_RandomizedNN.git`.

In Tables 1 and 2 are shown the accuracies obtained for all four different network architectures. The top table exhibits SLFNs (ELM and RVFL) accuracies, while on the bottom table the accuracies correspond to DNNs (dRVFL and SSFN). Each network architecture presents three columns, from left to right: achievable accuracy using random matrices and, accuracies by implementing deterministic transforms by applying Method 1 and Method 2 respectively. The accuracies achieved do not present overall degradation regarding the original networks. Only in some isolated cases, as in Letter

Table 1: Classification accuracy of SLFNs across 20 Monte-Carlo simulations, by using the randomized method and the two methods proposed in the article. Method2 hyperparameter $\gamma = 0.8$ has been used.

Dataset	Test Accuracy (in %) (avg. $\pm$ std. dev)					
	ELM			RVFL		
	Rand	Method 1	Method 2	Rand	Method 1	Method 2
Iris	98.1 ± 2.1	97.8 ± 2.4	97.8 ± 2.4	98.1 ± 2.2	97.2 ± 2.6	97.2 ± 2.6
Glass	68.5 ± 6.5	67.9 ± 6.5	67.9 ± 6.5	69.7 ± 6.7	70.9 ± 7.5	70.9 ± 7.5
Wine	95.6 ± 3.1	95.8 ± 2.9	95.7 ± 2.9	98.7 ± 2.1	98.6 ± 1.3	98.6 ± 1.3
Vowel	53.9 ± 1.7	51.7	51.7	54.1 ± 1.3	54.8	54.8
Satimage	84.6 ± 0.5	85.4	85.4	85.3 ± 0.3	86.7	86.7
Letter	95.7 ± 0.2	74.2 ± 5.0	74.2 ± 5.0	95.0 ± 0.3	74.2 ± 5.0	74.2 ± 5.0
NORB	89.8 ± 0.5	85.3	84.7	90.1 ± 0.3	88.1	87.5
Shuttle	99.2 ± 0.1	96.9	96.9	99.4 ± 0.0	97.9	97.9
MNIST	96.9 ± 0.1	93.6	93.6	97.3 ± 0.1	94.3	94.3

Table 2: Classification accuracy of DNNs across 20 Monte-Carlo simulations, by using the randomized method and the two methods proposed in the article. Parameters set with manual effort: $L = 6$ for RVFL and $L_{max} = 20$ for SSFN, $\eta_{var} = 10^{-7}$ (except $\eta_{var} = 10^{-8}$ for MNIST on dRVFL) for both networks and Method2 hyperparameter $\gamma = 0.8$.

Dataset	Test Accuracy (in %) (avg. $\pm$ std. dev)					
	dRVFL			SSFN		
	Rand	Method 1	Method 2	Rand	Method 1	Method 2
Iris	99.0 ± 1.9	99.0 ± 1.4	99.0 ± 1.9	97.2 ± 2.7	97.8 ± 1.9	97.7 ± 2.1
Glass	68.4 ± 6.2	73.0 ± 6.9	71.0 ± 6.9	69.6 ± 6.7	72.0 ± 7.5	71.3 ± 6.4
Wine	98.7 ± 1.7	99.7 ± 0.8	99.3 ± 1.1	97.5 ± 2.0	94.5 ± 3.5	94.3 ± 2.7
Vowel	54.7 ± 1.4	51.9	52.6	60.2 ± 2.4	64.7	63.4
Satimage	86.3 ± 0.5	86.0	86.2	89.9 ± 0.5	89.2	89.2
Letter	93.5 ± 0.1	88.5 ± 0.3	85.4 ± 0.4	95.7 ± 0.2	92.7 ± 0.3	91.1 ± 0.4
NORB	90.0 ± 0.5	85.4	90.6	86.1 ± 0.2	84.8	87.5
Shuttle	99.2 ± 0.1	98.1	98.1	99.8 ± 0.1	99.8	99.9
MNIST	96.9 ± 0.1	95.6	96.9	95.7 ± 0.1	96.5	96.9

dataset for SLFNs, an accuracy degradation is considerable. It is also worth to note that for Vowel, Satimage, NORB, Shuttle and, MNIST datasets, of which dataset partitions are not random, the performances are constant. This is a result of implementing deterministic transforms to deterministic datasets, building the same network for a dataset, independent of the number of simulations.

For the SLFNs case, in Table 1, it is observed that in most of the cases the networks built by the two methods present the same accuracy. This observation illustrates that both methods lead for the same transforms. This similarity is also present in DNNs initial layers.

In DNNs, when using random partition datasets, the network architectures differ among executions as the datasets vary. Even with these variations, in most datasets the network built presents a consistency of the chosen transforms across different trials. Figure 1 illustrates this stability for Iris dataset by implementing Method 2.

Two different methods have been proposed, both producing almost identical results for SLFN case and similar results for DNNs case, although there are differences between both. The first difference is found on the number of neurons per layer, which affects to the computational complexity, being lower on most of the networks created by Method 1. dRVFL with MNIST dataset provides a considerable example where the number of nodes, from left as network input to right as the output, is "1023-1020-202-133-99-57" and "1023-1310-1835-1520-1673-4", presenting a significant reduction. An example for Iris dataset is illustrated in Figure 2. The second difference is regarding the computational training cost. Method 2 needs more computational resources to be trained as makes use of the autocorrelation function and PCA. While dRVFL took 6.05s to be trained by Method 1, Method 2 needed 33.95s.

5. CONCLUSIONS

This paper proposes to replace weights from randomization-based neural network architectures with deterministic transforms. To chose a transform in an unsupervised manner, two methods based on statistical signal properties are created. Both methods present consistent results when building the network across simulations, although Method 1 stands out as more convenient. The achieved results, in terms of accuracy, remain similar to the original randomized networks. This results prove the compatibility of deterministic transforms to work as neural network weights. An important future work is to study interpretability of signal flow through the layers of neural networks due to the use of deterministic transforms. Additionally, the bag of deterministic transforms could be extended and a study on the use of many more transforms may be of interest.

6. REFERENCES

[1] Christian Szegedy, Alexander Toshev, and Dumitru Erhan, "Deep neural networks for object detection," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, Red Hook, NY, USA, 2013, NIPS'13, pp. 2553–2561, Curran Associates Inc.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1097–1105. Curran Associates, Inc., 2012.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, "Imagenet large scale visual recognition challenge," *Intl. J. Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015.

[4] Samuel F. Dodge and Lina J. Karam, "A study and comparison of human and deep learning recognition performance under visual distortions," *ArXiv e-prints*, 2017.

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

[6] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size," *ArXiv e-prints*, 2016.

[7] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv e-prints*, 2017.

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

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

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

[11] Saikat Chatterjee, Alireza M. Javid, Mostafa Sadeghi, Partha P. Mitra, and Mikael Skoglund, "Progressive learning for systematic design of large neural networks," *arXiv preprint arXiv:1710.08177*, 2017.

[12] Saikat Chatterjee, Alireza M. Javid, Mostafa Sadeghi, Shumpei Kikuta, Dong Liu, Partha P. Mitra, and Mikael Skoglund, "SSFN – self size-estimating feed-forward network with low complexity, limited need for human intervention, and consistent behaviour across trials," *ArXiv e-prints*, 2020.

[13] Pol Grau Jurado, Xinyue Liang, Alireza M. Javid, and Saikat Chatterjee, "Use of deterministic transforms to design weight matrices of a neural network," 2021.

[14] Joan Bruna and Stephane Mallat, "Invariant scattering convolution networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1872–1886, 2013.

[15] Simone Scardapane, Dianhui Wang, Massimo Panella, and Aurelio Uncini, "Distributed learning for random vector functional-link networks," *Information Sciences*, vol. 301, pp. 271 – 284, 2015.

[16] Xinyue Liang, Alireza M. Javid, Mikael Skoglund, and Saikat Chatterjee, "A low complexity decentralized neural net with centralized equivalence using layer-wise learning," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[17] X. Liang, A. M. Javid, M. Skoglund, and S. Chatterjee, "Asynchrounous decentralized learning of a neural network," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 3947–3951.

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

[19] Nasir U. Ahmed and K. Ramamohan Rao, *Orthogonal Transforms for Digital Signal Processing*, Springer-Verlag, Berlin, Heidelberg, 1975.

[20] Ali N. Akansu and Richard A. Haddad, "Chapter 6 - wavelet transform," in *Multiresolution Signal Decomposition 2nd Edition*, Ali N. Akansu and Richard A. Haddad, Eds., pp. 391–442. Academic Press, San Diego, 2001.

[21] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Addison-Wesley Longman Publishing Co., Inc., USA, 2nd edition, 2001.

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

[23] Ralph Linsker, "Self-organization in a perceptual network.," *IEEE Computer*, vol. 21, pp. 105–117, 03 1988.