I. Troch, F. Breitenecker, eds. ISBN 978-3-901608-35-3

LEARNING OVER SETS WITH HYBRID RECURRENT NEURAL NETWORKS: AN EMPIRICAL CATEGORIZATION OF AGGREGATION FUNCTIONS

W. Heidl[1], C. Eitzinger[1], M. Gyimesi[2], F. Breitenecker[2]
[1]Profactor GmbH, Steyr, Austria; [2]Vienna University of Technology, Vienna, Austria

Corresponding Author: W. Heidl, Profactor GmbH
Im Stadtgut A2, 4407 Steyr-Gleink, Austria; wolfgang.heidl@profactor.at

Abstract. Numerous applications benefit from parts-based representations resulting in sets of feature vectors. To apply standard machine learning methods, these sets of varying cardinality need to be aggregated into a single fixed-length vector. We have evaluated three common recurrent neural network (RNN) architectures, Elman, Williams & Zipser and Long Short Term Memory networks, on approximating eight aggregation functions of varying complexity. The goal is to establish baseline results showing whether existing RNNs can be applied to learn order invariant aggregation functions. The results indicate that the aggregation functions can be categorized according to whether they entail (a) *selection* of a subset of elements and/or (b) *non-linear* operations on the elements. We have found that RNNs can very well learn to approximate aggregation functions requiring either (a) or (b) and those requiring only linear sub functions with very high accuracy. However, the combination of (a) and (b) cannot be learned adequately by these RNN architectures, regardless of size and architecture.

1 Introduction

Numerous applications benefit from parts-based representations resulting in sets of feature vectors. As an example, the good/bad decision in surface inspection tasks often depends on the whole set of extracted fault feature vectors, such as their spatial distribution, their total area and similar quantities. This information cannot be utilized if faults are processed one-by-one. Learning the decision process in this context thus requires methods for classification of sets of feature vectors extracted from the faults [11].

To apply standard machine learning methods, these sets of varying cardinality need to be aggregated into a single fixed-length vector. We define the classification task over sets $X_i \coloneqq \{x_{i,1}, x_{i,2}, \ldots, x_{i,n_i}\}$ of feature vectors $x_{i,j} \in \mathbb{R}^m$

$$f \circ g \colon \mathbb{R}^{m \cdot n_i} \mapsto \{-1,1\} \tag{1}$$

$$X_i \mapsto c_i \coloneqq (f \circ g)(X_i) \tag{2}$$

as the composition of the aggregation function $g \colon \mathbb{R}^{m \cdot n_i} \mapsto \mathbb{R}^k, X_i \mapsto a_i$ computing the fixed length k-aggregate a_i and the classification function $f \colon \mathbb{R}^k \mapsto \{-1,1\}, a_i \mapsto c_i$. Since, per-se, no order is imposed on the set elements, the aggregation function has to be order invariant.

Traditionally, in applications like machine vision aggregation has been solved by experts choosing the appropriate application-specific functions [11]. In practice this is often a time-consuming and expensive iterative process. Furthermore, excluding an important pre-processing from machine learning of complex models can severely limit the resulting system performance. Hence, from a machine learning perspective, trainable aggregation for sets of varying cardinality is an important ingredient to flexible and accurate modelling which has received little attention yet.

Numerous methods have been developed in the field of image processing for the classification of single feature vectors, for the generation/invention of features and for feature selection. All of these algorithms cover a wide range of applications and can be trained on samples. However, to date no machine vision classification methods have been reported, that fulfil all the requirements needed for surface inspection:

- The number of potential faults extracted varies for each instance
- The set of potential faults has no inherent ordering of its elements
- Typically, there are no correspondences between the faults extracted from different parts

Related Work. Some work has been done on classification of structures containing an unknown number of elements [3,6], where the authors propose to use *generalized recursive neurons* to represent the structure of a graph. In order to achieve this, they make use of the linked structured of the graph and directly represent the graph topology in the network. They put their method in context with standard neural networks (for the classification of single patterns) and RNNs (for the classification of sequences).

With respect to training of aggregation functions the work of Uwents and Blockeel [12] is very similar to our classification problem. The authors propose a method to solve a combined selection and aggregation problem in a structured data set. They investigate data in a relational database, from which they select a relevant set of objects and classify the set using a trainable aggregation function. Because the size of the set is not known (it depends on the results of the selection process) they used RNNs for classification. However, the work was focused primarily on the selection process; furthermore, the trainable aggregation has only been tested on fairly simple functions. Still, they identify a few key questions such as which architecture to choose for the network or how to achieve order-invariance.

Recently classification of *bags-of-features* has received attention in the field of object recognition. Using local features extracted from patches around salient image locations, recognition is performed by matching features of a new image with those of known objects (e.g. [8]). Grauman and Darrell [4] proposed a kernel function operating on sets of features, which computes the partial match similarity between sets. Utilizing this kernel, a variety of machine learning algorithms such as Support Vector Machines (e.g. [10]) can be applied to sets of features. The method draws on the assumption that direct correspondences exist between the elements of sets to be compared. Another approach to define kernels over sets is based on rewriting the sets of features as matrices and calculating the principal angles between the two linear subspaces spanned by these matrices [14]. While not requiring direct correspondences between the set elements, this method is in practice limited to sets of equal cardinality [4].

We investigate the capability of common RNN architectures to learn order invariant aggregation functions relevant the context of surface inspection. Based on the results, we identify two categories of aggregation components. We show that either category can be learned accurately; however, none of networks can manage to learn a combination of both. Section 2 describes the architectures and mode of operation of RNNs used for learning feature aggregation functions. Section 3 presents the experimental setup and results on eight aggregation problems. Concluding remarks are given in Section 4.

2 Recurrent Neural Networks for Feature Aggregation

Due to their recurrent structure, RNNs can deal with a variable number of input vectors without growing with the number of objects that we want to process. Furthermore aggregation and succeeding processing steps such as classification can be integrated into a homogenous structure, which allows for training of both components in a uniform manner. The principle structure is shown in Figure 1.

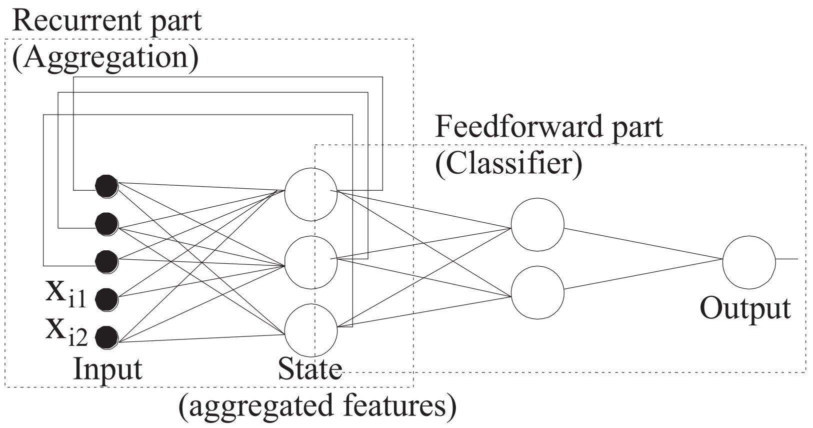

Figure 1: Recurrent Neural Network architecture for classification of sets of feature vectors.

The n input vectors $x_1, x_2, \ldots, x_n$ will be processed by the network as if they were a time sequence. The classification result will be available at the output after n "time steps". However, this sequence of vectors lacks an important property of time series, as it has no inherent ordering. Therefore, special methods for achieving invariance of the RNN with respect to the presentation order of the set elements have to be applied.

For this paper, we have not investigated the aspect of classification, since once the varying number of potential faults have been aggregated to a fixed-length representation, standard classification methods can be applied. If the classifier provides for an error signal at its input, the combined aggregation/classification system can be simultaneously trained by gradient based learning algorithms [1].

2.1 Network Architectures

We consider three common RNN architectures for learning aggregation functions $g\colon \mathbb{R}^{m \cdot n_i} \mapsto \mathbb{R}^k$ producing fixed length aggregates a_i out of sets of varying cardinality X_i. The Elman network [2] and the architecture proposed by Williams & Zipser (WZ) [13] date back to the late 1980s and introduce feedback loops to the then well known Multilayer Perceptron (MLP) feed forward neural networks. While Elman limited the feedback to within the hidden layer of the network, Williams & Zipser added feedback loops within the output layer, from the output to input layer and feedforward connections bypassing the hidden layer to their so-called fully connected network (Figure 2).

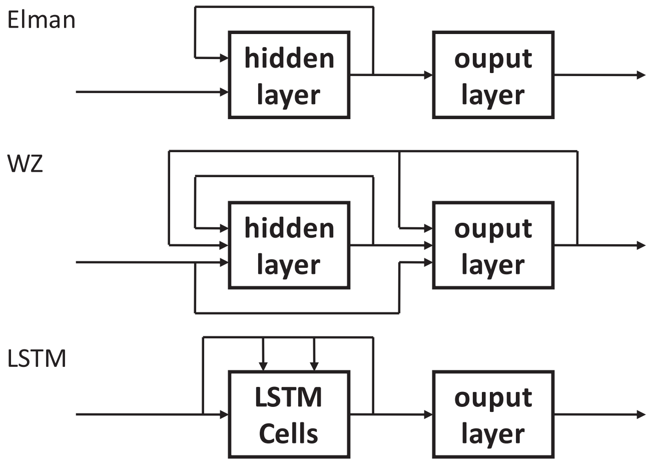

Figure 2: The network architectures used for evaluation are the Elman network, the fully connected WZ network proposed by Williams & Zipser and the Long Short Term Memory (LSTM) network. Arrows indicate data flow direction and lines represent vector quantities, except for the output, which is a scalar. The networks differ in the connection architecture as well as in the special cells used in the LSTM hidden layer (Figure 3).

Long short-term memory (LSTM) networks have been introduced by Hochreiter [7] as a solution to the *vanishing gradient* problem. Standard RNNs can hardly learn long-term dependencies between the data, since the error gradient with respect to the weights vanishes for larger time-lags between output and input. The error signal used to adapt the weights during training thus decreases exponentially when going back through time. Long short-term memory (LSTM) networks solve this problem, by incorporating *constant error carrousels* (CEC), neurons that propagate the error unchanged, thus avoiding the decaying gradient. Access to the CEC is controlled by multipliers which are driven by dedicated neurons called input and output gates. The inputs of the gates are connected the network inputs and the outputs of all cells in the network.

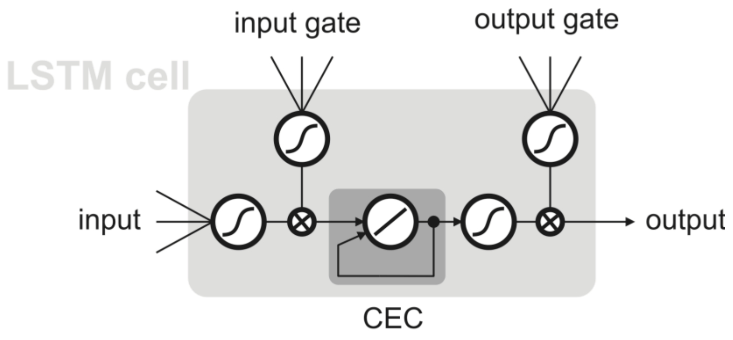

Figure 3: LSTM cell. The constant error carrousel (CEC) prevents the vanishing gradient problem of standard RNNs by propagating the error signal unchanged. Access to the CEC is controlled by the input and output gates.

2.2 Order Invariance

Order invariance of RNNs can be realized by reshuffling of the data during training: The sequence of the feature vectors in the set will be randomly permuted after each training epoch of the network. The neural network should thus learn to be order-invariant. It is clear that it is impossible to cover all permutations of the input data, but experiments [12] showed that the network will be able to generalize from a comparable small number of permutations.

As a second option we use sorting as a pre-processing step that establishes order invariance among the set elements. Sorting, however, requires an ordering, which is non-trivial for data with more than one dimension. For the general, multi-dimensional case lexicographical precedence [5] or vector norms can be used as sorting criterion in a first approach.

3 Experiments

We have evaluated Elman, WZ and LSTM RNN architectures, on approximating eight aggregation functions g of varying complexity (see Figure 2 and Table 1). The data sets have been generated by applying aggregation functions to sets of scalar entries $x_{i,j} \in \mathbb{R}$. We have trained each network on both sorted and shuffled representations of the data sets. The goal is to establish baseline results showing whether these existing RNNs can in principle learn order invariant aggregation functions.

3.1 Data Sets

We have evaluated the network's performance on approximating functions g aggregating sets of varying cardinality to a fixed-length aggregate. These functions have been designed with visual quality inspection in mind. For example the `Sum` function could describe the total area of detected faults. Another example is the minimum geometric distance between faults, which is an important measure to decide if local build-ups of otherwise uncritical faults can be tolerated. Table 1 shows the definition of the 8 aggregation functions used for the evaluation.

Name	Definition
`Sum`	$g \coloneqq \sum_j x_j$
`Norm`	$g \coloneqq \sqrt{\sum_j {x_j}^2}$
`MeanNorm`	$g \coloneqq \frac{1}{n_i}\sqrt{\sum_j {x_j}^2}$
`SumMax2`	$g \coloneqq x_u + x_v$ $x_u \geq x_v \geq x_j\,, j \in \{1,\dots,n\}\backslash\{u,v\}$
`Max`	$g \coloneqq \max_j x_j$
`SumMax2Abs`	$g \coloneqq \lvert x_u\rvert + \lvert x_v\rvert$ $\lvert x_u\rvert \geq \lvert x_v\rvert \geq \lvert x_j\rvert\,, j \in \{1,\dots,n\}\backslash\{u,v\}$
`MaxAbs`	$g \coloneqq \max_j \lvert x_j\rvert$
`MinDist`	$g \coloneqq \min_{j,k,j\neq k} \lvert x_j - x_k\rvert$

Table 1: Definition of the aggregation functions underlying the 8 data sets under evaluation.

Data sets have been produced by applying aggregation functions to sets of scalar entries $x_{i,j} \in \mathbb{R}$. The number of elements in each set is uniformly distributed

$$|X_i| \sim \mathrm{U}(15,30) \tag{3}$$

with a minimum of 15 and a maximum of 30 elements. The range has been chosen such that the networks cannot memorize all set elements. The entries itself are also generated randomly

$$x_{i,j} \sim \mathrm{U}(-0.5,0.5). \tag{4}$$

Target values for the data sets are generated by scaling

$$a_i = \frac{g(X_i) - \mu}{6\sigma} + 0.5 \tag{5}$$

with $\mu = \frac{1}{n}\sum_{i=1}^{n} g(X_i)$ and $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(g(X_i) - \mu)^2}$, such that almost all values are in the range [0,1], compatible with the logistic sigmoid activation function at the network's output.

3.2 Network Size

We have run the experiments with three different network sizes to assess their influence on learning result. The network sizes are given in Table 2. For a given size, each of the network architectures should have equal complexity. To account for the additional neurons in LSTM cells over standard networks, the number of cells in the LSTM networks is chosen approximately halve the number hidden neurons for the other architectures.

Size	Elman [hidden neurons]	WZ [hidden neurons]	LSTM [cells]
small	5	5	3
medium	10	10	5
large	15	15	8

Table 2: Network sizes chosen for the experiments.

3.3 Training

During training the set elements are fed into the networks as a sequence. For the given artificial data sets training could be closely guided by providing target values at every step of the sequence. However, such input is typically not available in practical settings. We have therefore decided that during training target values are presented to the network with the last set element only.

Adaptation of the network weights has been performed using Backpropagation through time (BPTT) algorithm [2] for the Elman and Williams & Zipser network. The central idea is to unfold the recurrent neural network into a multilayer feedforward neural network. The feedforward network thus has a layer for each time step in the sequence. Then the standard Backpropagation algorithm is applied to the unfolded networks. The LSTM networks are trained by a combination of truncated BPTT and Real-time Recurrent Learning (RTRL) [13], an algorithm which is specifically tailored to the LSTM architecture [7].

The weights and biases have been randomly initialized in the range of $[-0.5, 0.5]$ for the Elman and WZ networks and $[-0.2, 0.2]$ for the LSTM networks, except for the biases to the input and output gates. These biases are initialized starting with -0.5 for the first, -1.0 for the second, -1.5 for the third and so forth. This way, cell activations are initially close to zero, and during training the cells will be sequentially activated to contribute to the network function [7].

The networks are trained for a fixed amount of 100 epochs, which we have found enough for training of all network / data set combinations to converge. During one epoch each of the 500 training sets X_i is fed into the network once. For the shuffled version of the data, the set elements are randomly permuted after each epoch. The network weights are updated after each set. In the given setting no signs of overfitting have been observed, i.e. the performance of the networks on separate test data is very close the performance on the data used during training.

3.4 Results

We have evaluated Elman, WZ and LSTM networks, on approximating the eight aggregation functions presented in Section 3.1. We have trained each network using both sorted and shuffled representations of these data sets. For each combination 10 independent networks have been trained to reduce the probability of reporting results where training converged to a bad local minimum of the cost function. Results are reported only for the best network of each combination, since we are interested in whether existing RNNs can in principle learn the given aggregation functions. The accuracy of the approximation is assessed by the Pearson correlation coefficient *R* of the network‘s output with the target value on fresh test data not used during training.

Aggregation Functions. Results for the small networks (Section 3.2) are presented in Figure 4, where they have been grouped by aggregation function and arranged in decreasing order of the median group accuracy. From the chart it is apparent that the 5 best aggregation functions are distinguished from the remaining, in that they can be learned with high accuracy. Categorization of the aggregation functions according to whether they entail (a) *selection* of a subset of elements and/or (b) *non-linear* operations on the elements (see Table 3) reveals an interesting structure in the results: We have found that RNNs can very well learn to approximate aggregation functions requiring either (a) or (b) and those requiring only linear sub functions. For aggregation functions not requiring selection, *R* is greater than 95%. For those requiring selection but only linear operations on the elements, *R* is greater than 85%. However, if both selection and non-linear operations are required, the accuracy achieved is less than 60%.

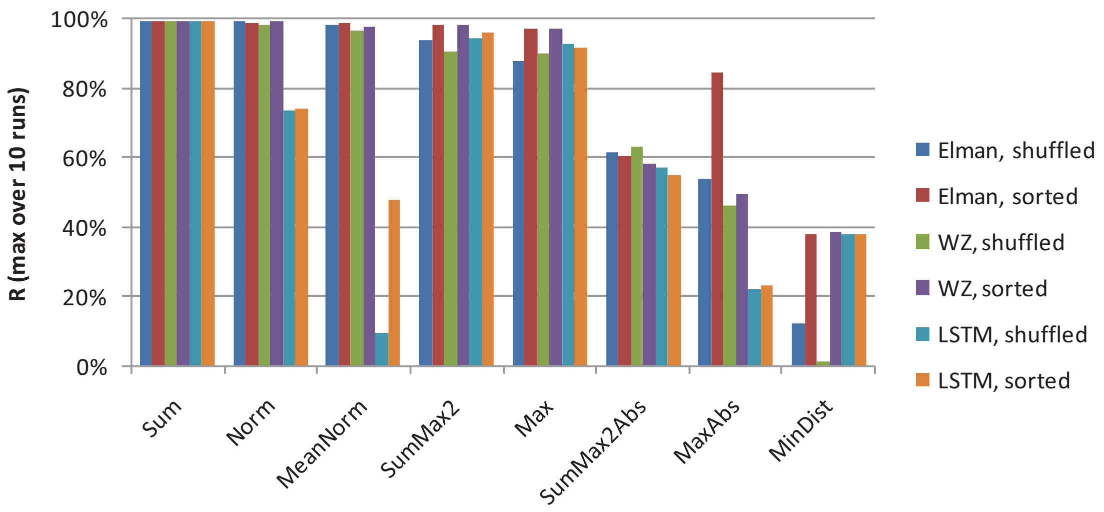

Figure 4: Approximation performance of RNNs on feature aggregation functions of varying complexity. The approximation is assessed by the correlation coefficient *R* of network's outputs with the target values. For both shuffled and sorted representations of the data the best result over 10 independently training runs are illustrated.

Name	Selection	Non-linear
`Sum`	no	no
`Norm`	no	yes
`MeanNorm`	no	yes
`SumMax2`	yes	no
`Max`	yes	no
`SumMax2Abs`	yes	yes
`MaxAbs`	yes	yes
`MinDist`	yes	yes

Table 3: Categorization of the aggregation functions underlying the 8 data sets under evaluation. The two rightmost columns indicate whether *selection* of subsets and/or *non-linear* operations are needed to compute the function.

Network size. For the given data sets and networks, the larger network sizes provide no significant improvement in general. Taking the most difficult-to-estimate `MinDist` data set as an example, changing from small to medium networks results in a correlation increase below 0.3% for either network type. Only the border case `Sum-Max2Abs` data set can be approximated significantly better by the large Elman network, yielding 78% correlation compared to around 60% reached by all small networks.

Network structure. Similar to the network size, the additional feedback loops and the direct input-output connection available in the WZ network provide no advantage over the simpler Elman network. The LSTM network is inferior to the other network types for most data sets except for the most difficult-to-approximate `MinDist` aggregation function. This remains true even if larger networks with up to 8 LSTM cells are compared to the 5-Hidden-neuron Elman and WZ networks. On `MinDist` the Elman and WZ network's correlations deteriorate to below 10% on shuffled data, unlike the LSTM network's 40%, which is equal to the easier-to-handle sorted case. Although correlation values below 40% are too low for practical applications, the performance of the LSTM on the shuffled `MinDist` data is significantly better than random.

Sorting. For the scalar, real-valued set elements presented here, sorting within the instance sets is very straight forward. It simplifies approximation of the aggregation functions by either moving the relevant elements to the end of the sequence or by arranging elements such that those relevant to the result are adjacent. Sorting helps for the function involving taking the plain max, since then all relevant elements are presented at the end of the sequence.

4 Conclusions

We have investigated the capability of three common RNN architectures in three different sizes to learn order invariant aggregation functions. For 5 aggregation functions the reached accuracy is already close to 100% with the small sized network. Increase size and complexity in the network structure does not result in increased approximation accuracy of the remaining aggregation functions. For these functions the trained networks produce accuracies below 60%.

Empirically, we have found that the presented aggregation function can be categorized according to two building blocks: (a) *selection* of a subset of elements and/or (b) *non-linear* operations on the elements. We have shown that Elman, Williams & Zipser and LSTM networks learn to approximate aggregation functions entailing either (a) or (b) with very high accuracy. However, the combination of (a) and (b) cannot be learned adequately by these RNNs, regardless of size and architecture. We thus assume that approximation of these aggregation functions requires changes in the network architecture on a higher level:

One research direction should be the extension of existing architectures, which could be done by adding more feedforward layers at either the input or output. Implementation of such networks is straight-forward, since existing learning algorithms can be used with minor adaptations.

Our main direction of research will aim at changing network architecture on a more fundamental level. The major challenge in using recurrent neural networks for approximating aggregation functions is to achieve order invariance. The two approaches investigated here are based on shuffling of the set elements during training or pre-sorting of the sets. While the latter requires application-specific knowledge for multi-dimensional data, training on shuffled data creates an unnecessarily large input space which complicates learning. We will therefore consider networks where order-invariance is an inherent property of the structure. For the basic building blocks of such networks we will consider symmetric functions [9], which are defined as being invariant under permutations of their arguments. We assume that by using these basic building blocks, an order-invariant neural network could be created.

5 Acknowledgements

This work has been supported by Austrian FWF grant P19376-N13.

6 References

[1] L. Bottou, and P. Gallinari, "A Framework for the Cooperation of Learning Algorithms, " in: Advances in Neural Information Processing Systems, pp. 781–788, Morgan Kaufmann, 1991.

[2] J. L. Elman, "Finding structure in time," Cognitive Science, vol. 14, iss. 2, pp. 179–211, 1990.

[3] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," IEEE Transactions on Neural Networks, vol. 9, iss. 5, pp. 768–786, 1998.

[4] K. Grauman and T. Darrell, "The Pyramid Match Kernel: Efficient Learning with Sets of Features," J. Mach. Learn. Res. vol. 8, pp. 725–760, 2007.

[5] A. Graves, S. Fernandez, and J. Schmidhuber, "Multi-Dimensional Recurrent Neural Networks," in Proc. Artificial Neural Networks ICANN, 2007, pp. 549–558.Casti, J. L.: *Reality Rules – Picturing the World in Mathematics: I, II*. Wiley, New York, 1992.

[6] B. Hammer, "Recurrent networks for structured data - A unifying approach and its properties," Cognitive Systems Research, vol. 3, iss. 2, pp. 145–165, 2002.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput. vol. 9, iss. 9, pp. 1735–1780, 1997.

[8] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," International Journal of Computer Vision, vol. 60, iss. 2, pp. 91–110, 2004.

[9] I. G. Macdonald, "Symmetric Functions and Hall Polynomials (Oxford Mathematical Monographs)," 2nd ed., Oxford University Press, USA, 1999.

[10] B. Schölkopf and A. J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning), 1st ed., The MIT Press, 2001.

[11] M. Tahir, J. Smith, and P. Caleb-Solly, "A Novel Feature Selection Based Semi-supervised Method for Image Classification," in Proc. Computer Vision Systems - ICVS, 2008, pp. 484–493.

[12] W. Uwents and H. Blockeel, "Classifying Relational Data with Neural Networks," in Proc. In Proceedings of 15th International Conference on Inductive Logic Programming, 2005, pp. 384–396.

[13] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," Neural Computation, vol. 1, iss. 2, pp. 270–280, 1989.

[14] L. Wolf and A. Shashua, "Learning over sets using kernel principal angles," J. Mach. Learn. Res. vol. 4, pp. 913–931, 2003.