SORN-based Cascade Support Vector Machine

Nils Hülsmeier*, Moritz Bärthel*, Jochen Rust‡, Steffen Paul*
* Institute of Electrodynamics and Microelectronics (ITEM.me)
University of Bremen, Bremen, Germany, +49(0)421/218-62549
‡ DSI Aerospace Technologie GmbH, Bremen, Germany
Email: {huelsmeier, baerthel, steffen.paul}@me.uni-bremen.de, jochen.rust@dsi-as.de

***Abstract*—This paper presents a SORN-based cascade support vector machine (SVM). For the first time a SVM is implemented by the usage of SORNs (Set Of Real Numbers). The SORN representation is a dual number representation that uses a fixed set of exact values and open intervals. Arithmetic operations in SORN representation can be realized by lookup tables which allows fast and low-complexity computing. This arithmetic is used for the non-support vector filtering in the early stages of a cascade SVM. Once the training subsets passed the bottom layer of the cascade, the optimization on the remaining support vectors can be done in classic representations like floating point or fixed point.**

***Index Terms*—support vector machine, SVM, unum, SORN, machine learning**

I. Introduction

The increasing digitization of society and stronger networking computers results in a rapidly increasing amount of data [1]. This age of big data promises great opportunities for machine learning. Therefore, machine learning algorithms that can handle large datasets and provide fast learning strategies are necessary. Since the support vector machine (SVM) was presented as binary classifier in [2], it has proven to be a powerful machine learning approach for classification and regression [3]. Unfortunately, the computational cost of SVM training increases with the number of training inputs. Therefore a parallel SVM implementation called cascade SVM was developed for acceleration of the training process [3]. Cascade SVMs divide the input data into subsets to reduce the computational costs of the training process [4]. Since the cascade principle is successful for SVMs for further reduction of computational complexity and acceleration of SVM training, different methods have to be taken into account. Every digital computer is based upon the underlying number format. The most common binary number representations are fixed-point and floating-point format. While computers mostly use the floating point standard IEEE-751, customized hardware architectures often use fixed point format [5]. Additionally, numerous alternative binary number formats exist.

A new approach for a digital number representation was presented with the universal number (unum) format [6]. The three different versions of the format offer various advantages and disadvantages compared to the classic formats. While the type-I and type-III unums realize different floating-point based approaches, type-II unums and the corresponding SORN arithmetic present an approach for a low resolution, interval arithmetic based format, which offers fast and low complex computing at the expense of precision [7]. This format can be used to constrain complex problems like linear or nonlinear systems of equations by excluding wrong solutions in advance. In [7] SORNs are used to reduce the number of possible solutions for a 12-dimensional nonlinear robotics problem; in [8] a SORN preprocessor is introduced in order to simplify the symbol detection in a wireless MIMO communication system.

In this paper we present a new approach of a SORN-based parallel SVM implementation. This promises speedup and lower complexity for SVM training. In detail, our work comprises the following contribution:

- An implementation of a cascade SVM to achieve parallel training, where the optimization of each subset is done using SORN arithmetic.
- A conversion of the optimization problem, utilizing the Karush-Kuhn-Tucker conditions, so it can be solved easier in SORN arithmetic.
- An evaluation of the SORN-based SVM, including a comparison with floating point implementations of an equivalent cascade SVM as well as a flat SVM for digit classification.

The mathematical fundamentals of SVMs are introduced in Section II. Section III describes the SORN arithmetic, before the SORN-based SVM implementation is presented in Section IV. Finally, the classification results are discussed compared to floating point solutions in Section V.

II. Support Vector Machine

In general, SVMs are binary classifiers separating two classes by the integration of an optimal hyperplane that maximizes the margin between the classes [2]. The determination of the optimal hyperplane is based on N training vectors $\mathbf{x}$ and labels $\mathbf{y} \in \{-1, 1\}^{N\times 1}$. The decision function of the SVM is defined as

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{1}$$

with $\mathbf{x}$ as input vector and $f(\mathbf{x}) = 0$ describing the optimal hyperplane. To separate the data linearly, $\mathbf{w}$ and b have to fulfill $y_i\left(\mathbf{w} \cdot \mathbf{x}_i + b\right) \geq 1$. Figure 1 shows the separation of the classes by the hyperplane and the distance between them. It can be shown that the optimal margin between the classes is $2/\|\mathbf{w}\|$. Since the margin has to be maximized, it follows

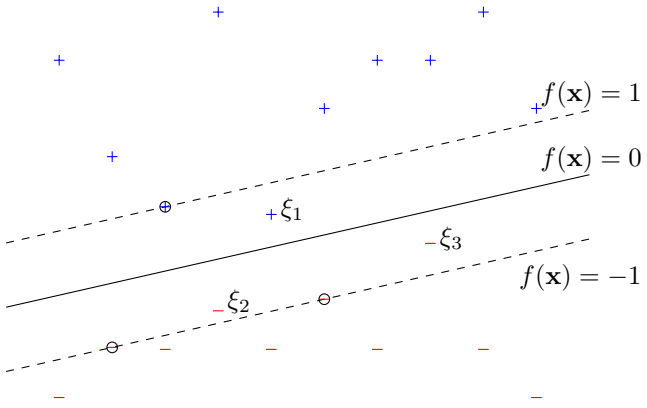

Fig. 1. SVM classification with optimal hyperplane at $f(\mathbf{x}) = 0$ and SVs marked by circles.

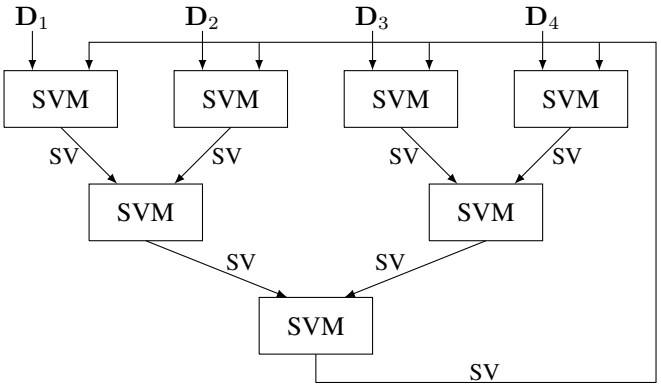

Fig. 2. Hierarchical structure of a three layer cascade SVM.

that the optimization problem becomes

$$\min_{w,b} \frac{\|\mathbf{w}\|^2}{2} \tag{2}$$

subject to the constraints

$$y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) \geq 1 \tag{3}$$

with $\mathbf{x}_i$ as a support vector whenever $y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) = 1$ is satisfied.

Assuming that the training data cannot be separated without errors, a soft-margin SVM is used to separate the data with a minimum number of errors. The optimization problem is modified to

$$\min_{w,b,\xi} \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^{N} \xi_i, \tag{4}$$

with

$$y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) \geq 1 - \xi_i, \;\; \xi_i \geq 0 \tag{5}$$

where $\sum_{i=1}^{N} \xi_i$ gives the number of training errors (see Fig. 1).
In [2] it is shown that solving the above optimization problem by constructing a Lagrangian results in the quadratic optimization problem

$$\max_{\mathbf{\Lambda}} -\frac{1}{2}\mathbf{\Lambda}\mathbf{D}\mathbf{\Lambda} + \mathbf{\Lambda}^T\mathbf{1} \tag{6}$$

subject to the constraints

$$0 \leq \alpha_i \leq C, \;\; \mathbf{y}^T\mathbf{\Lambda} = 0 \tag{7}$$

where $\mathbf{\Lambda} = (\alpha_1, ..., \alpha_N)^T$ are the Lagrange multipliers and $D_{i,j} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$. Therefore, to determine the optimal hyperplane, one has to solve eq. (6). The solution of eq. (2) by the use of Lagrange multipliers gives $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$.
To avoid problems with non-linearity, the input data can be mapped into some high-dimensional feature space $\phi\left(\mathbf{x}_i\right)$. Then, all functions including $\mathbf{x}$ are transformed to the feature space and $D_{i,j} = y_i y_j K\left(\mathbf{x}_i, \mathbf{x}_j\right)$, where the kernel represents the inner product $K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \phi\left(\mathbf{x}_i\right) \cdot \phi\left(\mathbf{x}_j\right)$.

A. Cascade SVM

The increasing complexity of eq. (6) and (7) with the number of training vectors results in time cost of SVM training. Parallel SVM structures that detect non-support vectors early in the optimization process are preferable [3]. The cascade SVM structure accelerates the training by eliminating the non-support vectors from the training dataset. The structure of a cascade SVM is shown in Figure 2. To ensure parallel computation of the optimization problem, the training data and labels are split into subsets. For each subset the optimization problem (6) under constraints (7) is solved. Since in most real world problems only a small subset of the training data are support vectors (SV), the cascade SVM reduces the subset of training data in each layer. The resulting SVs of each subset are combined two-by-two to the input of the next layer until the final training is done in the bottom layer. Thus, the cascade SVM can be understood as a filtering process of non-SVs.
One iteration of the cascade SVM often gives satisfying results, but if the global optimum is desired, the remaining SVs from the bottom are fed back to the first layer. To test if the SVs are valid for the whole dataset, each SVM of the input layer gets the SVs from the bottom layer, additionally to its original input. If there are no violations in all the SVMs in the first layer the cascade has reached the global optimum.

III. SORN Arithmetic

The unum type-II number representation and corresponding SORN arithmetic derives from the originally proposed type-I unums [6], which realize a variable length floating-point format, utilizing an interval arithmetic based concept, whenever maximum precision is exceeded. Instead of rounding, which is applied for classic floats [5], an extra bit indicates an open interval between the current and the next representable value. A similar concept is applied for type-II unums, although the focus and representation of the format is slightly different. With type-II unums, the real numbers are represented using a few exact values called *lattice values*, their negative versions, and the open intervals in between. With the lattice values

$\{0, 1, \infty\}$ a basic representation with $w_s = 8$ entries is given in [7], which read as

$$\mathcal{L}_8 = \{\pm\infty, (-\infty, -1), -1, (-1, 0), 0, (0, 1), 1, (1, \infty)\}. \tag{8}$$

When extending the representation, further lattice values are introduced together with their reciprocals in order to maintain a symmetric structure of the data type. Adding the additional lattice value 2 to the basic set results in a representation consisting of $w_s = 16$ entries, which is depicted in Fig. 3.

From this type-II unums, the Set-Of-Real-Numbers (SORN) data type can be derived, which is used for the computation of arithmetic operations. A SORN value consists of w_s bits, each bit encoding the absence (0) or presence (1) of an exact value or open interval from the above described representation. Every arithmetic operation is encoded in a pre-computed lookup-table (LUT), which can be implemented either as a (re-)programmable gate array or as fixed AND/OR logic block. This enables very fast and low complex computing, compared arithmetic components for classical number formats. Additionally, the LUT based arithmetic allows practically the same computing time and complexity for different arithmetic operations.

An exemplary LUT for the multiplication of SORN values using a simplified 5 bit data type is given in Tab. I. Union intervals can also appear during SORN computations, e.g. when two open intervals are added. An example for the simplified 5 bit data type from Tab. I is given in the following equation:

$$01000_{(-1,0)} + 00010_{(0,1)} = 01110_{(-1,1)} \tag{9}$$

In this work, the implemented data type configuration slightly differs from the one introduced in [7] and presented above. Since the SORN arithmetic is used to exclude wrong solutions of a system of equations, half-open intervals are sufficient for meeting the exclusion criteria. By replacing the exact values and open intervals with half-open intervals the information-per-bit ratio of the SORN data type is also increased. The SORN intervals are chosen as the lattice values $\{0, 0.25, 0.5, 1, 2, 5, 10, 100, \infty\}$ with half-open intervals in between and their negative opponents (comp. eq. (8)):

$$\mathcal{L}_{17} = \{[-\infty, -100), \ldots, [-0.25, 0), 0, (0, 0.25], (0.25, 0.5], \tag{10}$$

$$(0.5, 1], (1, 2], (2, 5], (5, 10], (10, 100], (100, \infty]\}. \tag{11}$$

IV. The SORN-based SVM

As shown in Sec. III, the SORN representation provides fast and low complex arithmetic operations. These advantages will be used for additional acceleration of the SVM training process (see Sec. II). The restricted options of correct solutions in SORN representation allow a fast solution of optimization problems. To reach the optimal hyperplane, the filtering of non-SVs is done in SORN representation for each subset and the final optimization is done on the remaining SV-subset in floating point representation when the dataset passed the bottom layer of the cascade (see Fig. 2).

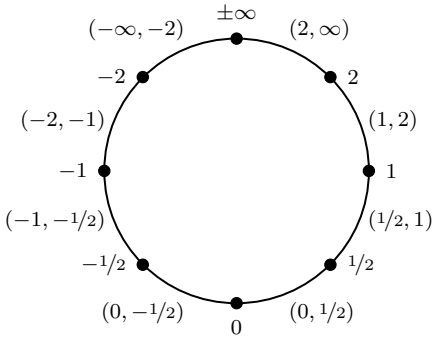

Fig. 3. Representation of the reals in the type-II unum format with lattice values $\{0, 1, 2, \infty\}$

TABLE I
LUT for the multiplication of a simple 5 bit SORN data type

×		**−1** 10000	**(−1, 0)** 01000	**0** 00100	**(0, 1)** 00010	**1** 00001
−1	10000	00001	00010	00100	01000	10000
(−1, 0)	01000	00010	00010	00100	01000	01000
0	00100	00100	00100	00100	00100	00100
(0, 1)	00010	01000	01000	00100	00010	00010
1	00001	10000	01000	00100	00010	00001

The hierarchical structure of the SORN SVM is the same as the cascade SVM in Fig. 2. The structure of each SVM unit is shown in Fig. 4. It consists of two arithmetic SORN units, which perform the calculations of eq. (7) and (13). Afterwards a decision is made if the current subset belongs to the SVs or not. Finally, subject to the length n of the SV subset, it is added to the final set of resulting SVs or it is given to the next stage of the cascade.

The number of valid combinations of Lagrange multipliers $\mathbf{\Lambda}_k$ is defined by the length of the training subset and the wordlength of the SORN data type. This set of possible solutions of eq. (6) is the input for each SVM in the first layer (see Fig. 2 and 4).

First, the constraint (see eq. (7)) is calculated and all $\mathbf{\Lambda}_k$ which do not fulfil the constraint are deleted from the set of possible solutions, resulting in a new set $\mathbf{\Lambda}_\ell$. The second step is to find the maximum of equation (6). Subject to the fact that SORNs use open intervals, additions in SORN representation can result in fast growing intervals (see eq. (9)). Therefore it is more difficult to determine an arbitrary optimum in SORN representation than one that converges to a fixed value. Hence it is reasonable to rearrange the optimization problem to a form that equals zero.

The zero of the gradient of eq. (6) is not guaranteed to be the global optimum, because it is not guaranteed that it fulfils the constraints. Therefore, we use the Karush-Kuhn-Tucker conditions (KKT) for quadratic optimization under constraints

equals zero

$$\mathbf{x}^T \cdot \frac{\partial L(\mathbf{x}, \mathbf{p})}{\partial \mathbf{x}} = 0 \tag{12}$$

from [9]. For optimization problem (7) with $\mathbf{x} = \mathbf{\Lambda}$ and $\mathbf{p} = \mathbf{1}$ results

$$\mathbf{\Lambda}^T (\mathbf{1} - \mathbf{D}\mathbf{\Lambda}) = 0 \tag{13}$$

as KKT condition. Hence, the remaining solutions $\mathbf{\Lambda}_\ell$ are valid if they fulfil the KKT condition eq. (13). Subject to the number of arithmetic operations, the SORN result will probably not equal exactly zero, but it will represent an interval that includes zero. It is possible to restrict the Λ_m, taken into account for the final solution. For example smaller intervals can be rated higher or only positive values contribute to the result. These restrictions have to be set individually according to the training data.

The final remaining set of $\mathbf{\Lambda}_m$ is analysed for zeros. Depending on the character of $\mathbf{\Lambda}_m$, the decision about the non-SVs in the actual training data subset is made.

To restrict the number of possible combinations for $\mathbf{\Lambda}_k$, the length n of one subset has to be limited to some η, depending on the SORN wordlength. The subsets which exceed this limit are written into the final set of SVs, while the remaining subsets run through the following cascade stages.

Furthermore, the constraint $0 \leq \alpha_i \leq C$ (see eq. (7)) has to be fulfilled. Since this constraint restricts $\mathbf{\Lambda}$, no additional arithmetic operations are needed. It reduces the number of valid $\mathbf{\Lambda}_k$. Also in the later stages it allows stronger restriction of results in SORN representation. It defines the upper bound for the resulting intervals of constraint $\mathbf{y}^T\mathbf{\Lambda}$ and KKT condition (see eq. (13)).

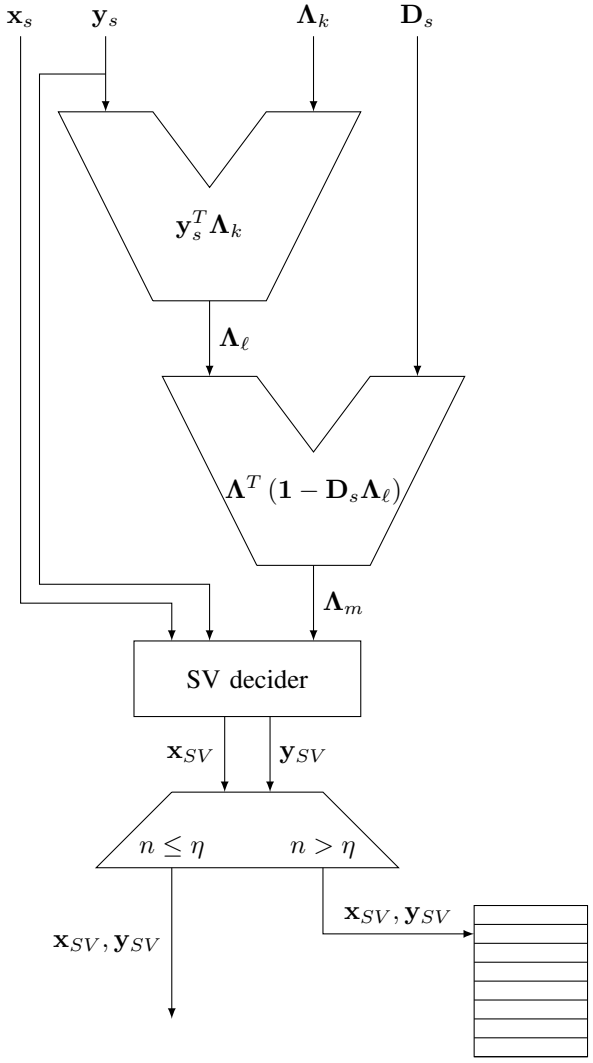

Fig. 4. The structure of one SORN-based SVM unit from the cascade SVM.

V. Evaluation

In this section we will present the classification results of the SORN-based SVM, discuss them and give an outlook to future work. For validation, the SORN SVM results are compared to the results of an equivalent cascade implementation in floating point representation and a flat SVM. Both are implemented in MATLAB and use the powerful interior-points algorithm from [10] for optimization.

Digit recognition is a well known field of application for SVMs [11]. Therefore, all implementations perform digit classification. The digit dataset from MATLAB is used. It contains 10,000 synthetic eight-bit images of digits with size 28×28 pixels [12]. The cascades are trained for one iteration and the initial subsets in the first stage (see Fig. 2) have length four. Note that both, the size of the subsets and the SORN-type, define the number of possibilities for $\mathbf{\Lambda}_k$. Therefore it is reasonable to use a small number of subset entries and a short wordlength for SORNs. The used SORN representation with a wordlength of 17 bits is given in eq. (10). For this SORN data type the maximum length of the subset is set to $\eta = 6$.

A. Experimental Results

The SVM implementations are trained by data sets of length N and the validation set includes 1000 images. The upper bound for Lagrange multipliers α_i is set to $C = 100$. Both cascade SVM implementations are executed for a single iteration.

Figure 5 shows the validation errors of flat SVM and cascade SVM implementations in floating point and SORN representation for different numbers of input training vectors, $N = \{32, 64, 128, 256\}$. Obviously, by increasing the amount of training data, the validation error decreases. All implementations reach approximately the same validation error, for every length N. Therefore, the SORN filtering process maintains the necessary SVs for the optimization.

Table II shows the number of remaining SVs in the final stage of the cascade SVM for floating point and SORN implementation after the first iteration. For a small number of training vectors, the number of remaining SVs is nearly

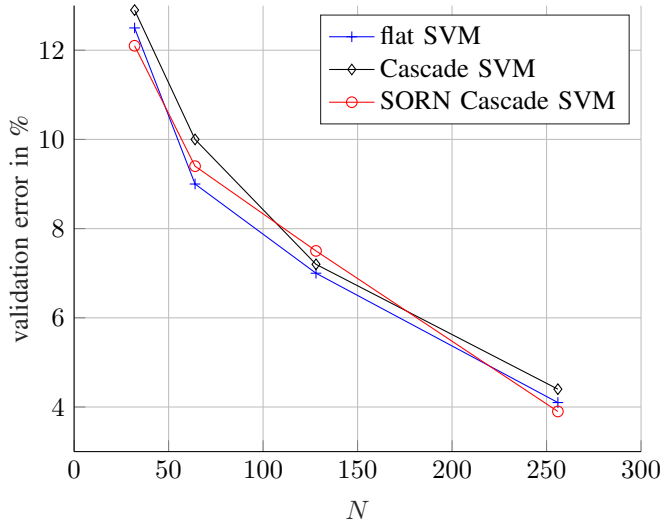

Fig. 5. Validation errors in % for flat, floating point cascade, and SORN cascade SVM implementations for different numbers of training vectors N.

TABLE II
THE REMAINING SVS FOR THE FINAL TRAINING IN THE BOTTOM LAYER OF FLOATING POINT AND SORN CASCADE SVM AFTER ONE ITERATION.

	$N = 32$	$N = 64$	$N = 128$	$N = 256$
Cascade SVM	22	36	58	81
SORN Cascade SVM	24	47	116	223

the same. But for larger input sizes the number of SVs that remains from the SORN-based non-SV filtering increases stronger than the floating point cascade SVM.

B. Discussion

As shown in the previous section, the SORN-based SVM gives promising results for digit classification. The presented results show that the filtering of non-SVs in SORN representation is successful in a way that the validation error is approximately the same as for floating point implementations. For a small number N of training vectors, it supplies a filtering performance close to the floating point representation. The almost proportional increase of remaining SVs after the SORN-based filtering for larger N can be explained by the strong limited size of the subsets $\eta = 6$. This subset size is often reached after the first one or two stages.
The large wordlength of 17 bits for this SORN representation causes a large number of valid combinations for $\mathbf{\Lambda}$, although it is restricted by the constraints eq. (7). Furthermore, the sets of $\mathbf{\Lambda}_k$ (see Fig. 4) increase exponentially with the subset size. Although the presented SORN-based SVM structure and SORN arithmetic itself allow strong parallelisation, large SORN wordlength lead on to computational costs.

C. Future Work

The presented results, in particular the good performance of SV filtering for small amounts of training vectors and the promising characteristics of SORNs that lead on to an acceleration method of SVM training, show that further research on SORN SVM implementations seems necessary.
Firstly, it is preferable to find a SORN representation with a reduced wordlength that provides also good classification results. Hence, the performance of a parallel SORN SVM can be verified for a larger number of training vectors.
Additionally, a hardware architecture of the SORN-based SVM would be interesting, to analyse its timing and area performance as well as its complexity.

VI. CONCLUSION

In this paper we presented the first SORN-based SVM implementation. We implemented the well known parallel cascade SVM structure with a non-SV filtering process in SORN arithmetic. The first layers of the SORN-based cascade SVM start with an initial subset length of four training vectors and were trained by training datasets of length 32, 64, 128 and 256. The validation results were compared with an equivalent cascade SVM structure and a flat SVM in floating point representation.
A SORN data type was found to provide similar validation errors and for small numbers of input vectors also a similar performance of filtering non-SVs was achieved. For larger training datasets, the filtering with this data type does not reach the floating point performance.
Subject to the characteristics of SORN representation the SORN-based SVM is a promising approach for accelerating SVM training. Future research has to focus on SORN data types with reduced wordlength and hardware architectures.

REFERENCES

[1] I. Goodfellow, Y. Bengio and A. Courville, "Deep Learning," The MIT Press, 2016.
[2] C. Cortes and V. Vapnik, "Support Vector Networks," in *Machine Learning*, vol. 20, pp. 273-297, April 1995.
[3] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik, "Parallel support vector machines: The cascade SVM," in *Proc. Adv. Neural Inf. Process. Syst.*, 2004, pp. 521–528.
[4] O. Kramer "Cascade Support Vector Machines with Dimensionality Reduction," in *Applied Computational Intelligence and Soft Computing*, vol. 2015, December 2014.
[5] Microprocessor Standards Committee of the IEEE Computer Society, "IEEE Std 754™-2008 (Revision of IEEE Std 754-1985), IEEE Standard for Floating-Point Arithmetic."
[6] J. L. Gustafson, *The end of error: Unum computing*. Boca Raton: CRC Press, 2015.
[7] J. Gustafson, "A Radical Approach to Computation with Real Numbers," *Supercomputing Frontiers and Innovations*, vol. 3, no. 2, 2016.
[8] M. Bärthel, P. Seidel, J. Rust, and S. Paul, "SORN Arithmetic for MIMO Symbol Detection - Exploration of the Type-2 Unum Format," in *2019 17th IEEE International New Circuits and Systems Conference (NEWCAS)*, June 2019, pp. 1–4.
[9] I. N. Bronstein, K. A. Semendjajew, G. Musiol and H. Mühlig, "Taschenbuch der Mathematik," Verlag Harri Deutsch, 2013.
[10] J. Gonzio "Multiple centrality corrections in a primal dual method for linear programming," in *Computational Optimization and Applications*, vol. 6, no. 2, 1996, pp. 137-156.
[11] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," in *Data Mining and Knowledge Discovery*, vol. 2, no. 2, June 1998, pp. 121-167.
[12] MATLAB DigitDataset https://de.mathworks.com/help/wavelet/examples/digit-classification-with-wavelet-scattering.html access: 12.02.2020.