2013 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING, SEPT. 22–25, 2013, SOUTHAMPTON, UK

FPGA PROTOTYPE OF MACHINE LEARNING ANALOG-TO-FEATURE CONVERTER FOR EVENT-BASED SUCCINCT REPRESENTATION OF SIGNALS

Sergio Martin del Campo, Kim Albertsson, Joakim Nilsson, Jens Eliasson and Fredrik Sandin

SKF University Technology Center
Luleå University of Technology
971 87 Luleå, Sweden
E-mail: sergio.martindelcampo@ltu.se

EISLAB
Luleå University of Technology
971 87 Luleå, Sweden
E-mail: fredrik.sandin@ltu.se

ABSTRACT

Sparse signal models with learned dictionaries of morphological features provide efficient codes in a variety of applications. Such models can be useful to reduce sensor data rates and simplify the communication, processing and analysis of information, provided that the algorithm can be realized in an efficient way and that the signal allows for sparse coding. In this paper we outline an FPGA prototype of a general purpose "analog-to-feature converter", which learns an overcomplete dictionary of features from the input signal using matching pursuit and a form of Hebbian learning. The resulting code is sparse, event-based and suitable for analysis with parallel and neuromorphic processors. We present results of two case studies. The first case is a blind source separation problem where features are learned from an artificial signal with known features. We demonstrate that the learned features are qualitatively consistent with the true features. In the second case, features are learned from ball-bearing vibration data. We find that vibration signals from bearings with faults have characteristic features and codes, and that the event-based code enable a reduction of the data rate by at least one order of magnitude.

1. INTRODUCTION

Signal processing typically involves mathematical models that are imposed on the raw signal to obtain a representation of reduced dimensionality, which is suitable for interpretation or further analysis. Signal models are used for tasks like noise reduction, compression, estimation, solving inverse problems, morphological component analysis and compressed sensing. One modeling approach that has attracted significant interest in the last decade is *sparse* representation of signals [1, 2]. Sparse representations can be *succinct*, meaning that they require a minimum of information and allows for interpretation and analysis without intermediate decompression. It has been demonstrated empirically that features roughly similar to those observed in the primary visual cortex can be learned from natural images using a combination of sparse coding and Hebbian learning [3,4]. Similarly, cochlear impulse response functions (revcor filters) can be estimated from speech data using a similar learning approach [5], and efficient signal representations are empirically obtained in a variety of applications using sparse coding and feature learning [1, 2].

The sparse-coding approach is still in its infancy and further work on theory, models and applications is needed [2]. In particular, more work is needed to understand and formally justify sparse coding with overcomplete dictionaries of empirically learned features, but the promising results obtained make dictionary learning a prominent part of signal processing research and development. The possibility to create succinct representations of signals with data-driven models is interesting for technologies and application domains where resource constraints, modeling complexity and diversity are challenging aspects; wireless sensors, condition monitoring, automation and robotics are some examples. Our motivation comes from the application domain, where sparse coding and empirical dictionary learning can offer significant advantages over the present engineering-intensive approach to signal modeling. The goal is to develop an "analog-to-feature converter", which enable succinct event-based coding of signals in terms of learned morphological features.

Here we report on the first steps to realize an FPGA prototype and case-study results that illustrate how a device of that type can be used to reduce the data rate and enable interpretation of real-world signals in real time. This device provides event-based and parallel codes, which in principle are well suited for analysis using neuromorphic processors [6] and multicore processors. The model is demonstrated using an input signal with known features and ball bearing vibration data, respectively. Our results show that the resulting codes are different for normal and various faulty bearings, and that known features are recovered. See the work by Liu, Liu and Huang [7] for a similar case study with ball bearing vibration data. In addition to the empirical, data-driven learning of features, this approach can enable a significant reduction of the data rate required to represent the signal while maintaining high temporal precision and signal-to-noise ratio. This indicates that the proposed approach can be useful in applications such as condition monitoring and event detection.

2. MODEL

We adopt the model by Smith and Lewicki [5], which originates from the work on sparse visual coding by Olshausen and Field [3, 4]. The signal, $x(t)$, is modeled as a linear superposition of noise and features with compact support

$$x(t) = \epsilon(t) + \sum_i a_i \phi_{m(i)}(t - \tau_i). \quad (1)$$

The functions $\phi_m(t)$ are *atoms* that represent morphological features of the signal, where τ_i and a_i indicate the temporal position and weight of the atoms, respectively. The values of τ_i and a_i are determined with a matching pursuit algorithm [8, 9]. The set of atoms, Φ, is optimized in an unsupervised way by performing gradient ascent on the approximate log data probability

$$\frac{\partial}{\partial \phi_m} \log\left[p(x \mid \Phi)\right] = \frac{1}{\sigma_\epsilon^2} \sum_i a_i (x - \hat{x})_{\tau_i}, \quad (2)$$

where $(x - \hat{x})_{\tau_i}$ is the residual of the matching pursuit over the extent of atom ϕ_m at time τ_i and a_i is the event amplitude. This algorithm adapts the shape and length of each atom with a weighted average of the residuals of feature matches identified by the matching pursuit. This process is a form of Hebbian learning because adaptation results from the activation of atoms by the input signal. The algorithm can also be mapped on a neural network, where the notion of Hebbian learning becomes more evident [4]. Note that the resulting sparse code is not a linear function of the input signal because the matching pursuit is (weakly) non-linear. The termination criteria of the matching pursuit determines the sparseness and signal-to-residual ratio of the resulting event-based representation. Smith and Lewicki demonstrate [5] that efficient auditory codes that matches the input-output functions of the cochlea (revcor filters) are obtained using this approach.

2.1. Matching Pursuit with Dictionary Learning

The matching pursuit algorithm decomposes the signal $x(t)$ according to the model shown in Eq. (1). The result of the algorithm is a set of *atomic events*, which are defined by the occurrence of one specific atom, $\phi_{m(i)}(t - \tau_i)$, at time τ_i with weight a_i. All atoms, $\phi_m(t)$, belong to a finite dictionary, Φ, consisting of k elements.

$$\Phi = \{\phi_1, \ldots, \phi_k\}. \quad (3)$$

The matching pursuit algorithm is an iterative process that decomposes the signal in the dictionary of atoms, which can be overcomplete. The algorithm operates on the residual of the signal. Initially, the residual is the signal to be decomposed. The algorithm calculates the cross-correlation between the residual and all the elements of Φ. The atom with the maximum cross-correlation (inner product) for all possible time shifts triggers an atomic event. The time of the event is defined as τ_i and the inner product as a_i. The residual is updated by subtracting the atomic event, $a_i \phi_{m(i)}(t - \tau_i)$, from the residual. This process is repeated until a stopping criteria is fulfilled, which for example can be defined in terms of the maximum average event rate or the signal-to-noise ratio. For further details and a formal motivation of the algorithm, see the original work by Mallat and Zhang [8]. When the matching pursuit algorithm is implemented in an on-line fashion a sliding window of the signal is considered. The length of the window must be equal to or greater than the longest atom, and the finite window length need to be accounted for when defining the matching and stopping criteria.

The main challenge of the learning problem is to identify a dictionary of atoms, Φ, that maximizes the expectation of the log data probability

$$\Phi = \arg\max_\Phi \langle \log\left[p(x \mid \Phi)\right] \rangle, \quad (4)$$

where

$$p(x \mid \Phi) = \int p(x \mid a, \Phi) p(a) da. \quad (5)$$

The prior of the weights, $p(a)$, is defined to promote sparse coding in terms of statistically independent atoms [4]. The integral is approximated with the maximum a posteriori estimate resulting from the matching pursuit, which is motivated by the assumption that the code is sparse so that the integrand is peaked in a-space. This results in a learning algorithm that involves gradient ascent on the approximate log data probability defined by Eq. (2). The gradient of each particular atom in the dictionary is proportional to the sum of residuals corresponding to the matching-pursuit activation of that atom. The prefactor, $1/\sigma_e^2$, is the inverse variance of the residual that remains after matching pursuit. The step size, η, in the gradient ascent is a *learning rate* parameter. The gradient ascent update of the atoms then follows from Eq. (2)

$$\Delta\phi_m = \frac{\eta}{\sigma_e^2} \sum\nolimits_{i\,:\,m=m(i)} a_i (x - \hat{x})_{\tau_i}. \quad (6)$$

Therefore, the learning rate is dependent on the activation rate of atoms. This implies that the learning rate of different atoms need not be the same, and that there can be some atoms that do not learn at all. This is to be expected because there can be less features in the signal than allocated atoms in the dictionary. There are other approaches to dictionary learning [1], which for example can enable faster learning, orthogonality of atoms and globally optimal solutions. The main motivation of the approach taken here is the simplicity of the algorithm, which enables efficient online implementation on an FPGA /

ASIC, and the remarkable visual and auditory coding results obtained empirically with this approach [3–5]. The statistical independence of atomic events appears natural when searching for features of different phenomena in a system.

2.2. FPGA Implementation

The use of matching pursuit with dictionary learning in resource-constrained sensors and embedded systems requires that the model can be implemented in a compact and efficient way. Profiling of a C implementation of the algorithm shows that about 98% of the time is spent in the cross-correlation part of the algorithm. This suggests that a significant speed-up can be gained by the implementation of this part on an FPGA. The matching pursuit algorithm with dictionary learning is implemented on a ZedBoard development board for the Xilinx Zynq Z7020 FPGA. This FPGA has an embedded dual-core ARM® Cortex A9™ processor. The matching pursuit algorithm is implemented on the FPGA and dictionary learning is implemented in C on the A9 core. The communication between the processor and the FPGA is done using the AMBA® AXI4 interface protocol, which provides a shared memory between the FPGA and the processor.

Samples of the input signal are saved in an input buffer that has the same length as the atoms. A working buffer that is twice as long as the atoms is used for the matching pursuit. In principle, when the input buffer is full, the samples are shifted into the working buffer, so that half of the working buffer is shifted out and is discarded. Memory copies and shifts are practically avoided with the use of three buffers of the same length, two for the working buffer and one for the input buffer. A buffer manager reorganizes the role of the three buffers whenever the input buffer is filled. The cross-correlations between the working buffer and the atoms are calculated using a sliding window. Inner products are calculated in parallel within the window and sliding is implemented iteratively to reduce the number of logic blocks needed. A stopping condition determines whether the maximum cross-correlation is significant or not. If it is significant an atomic event is generated, which is subtracted from the working buffer before the matching pursuit continues. Otherwise the matching pursuit halts until the buffer manager has updated the working buffer. Figure 1 shows an overview of this implementation. The inner products in the cross-correlation part of the algorithm are similar to a bank of finite impulse response (FIR) filters, where each filter represents an atom. Parallel FIR filters can be designed by building a separate multiplier for each input-coefficient pair and an adder tree that sums the terms. This way the inner products can be calculated in one cycle with a constant delay that is introduced by the adder tree, see Figure 2. The delay is proportional to $\log_2(n)$, where n is the number of input elements.

One difference between Smith and Lewicki's original algorithm [5] and the FPGA implementation developed here is the logic for adapting the length of the atoms. Smith and Lewicki allowed the length of the atoms to vary by monitoring the tail amplitudes of the atoms. The atoms can grow in length if the temporal extent of features is larger than the initial size of the atoms, as indicated by the tail amplitudes of the atoms, or they can be shortened if the tail amplitudes are low. Our prototype uses a hard coded maximum atom length to enable parallel implementation on the FPGA. Also, our implementation is based on fixed-point numbers rather than floats in order to reduce the resources needed on the FPGA. We compare the results obtained with the prototype design with results obtained using a Matlab implementation of the model.

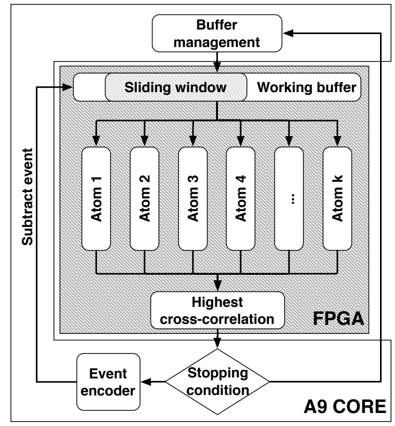

Fig. 1. Schematic view of the matching pursuit implementation. Operations performed by the FPGA and the A9 core are distinguished. Additionally, learning is done on the A9 core.

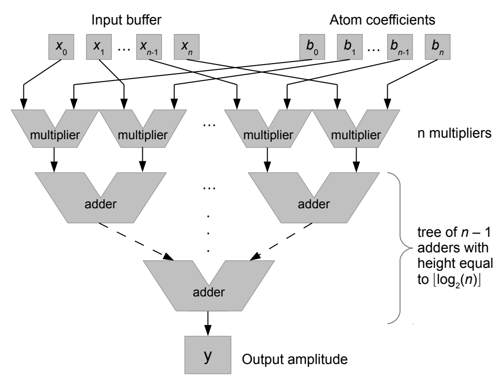

Fig. 2. Schematic view of the inner product design, which is used in the FPGA matching pursuit implementation.

3. CASE STUDIES

The implementation is first evaluated on a blind signal separation problem with an artificial input signal. The signal includes three known features. These features are a sine function with a period of 84 samples, a Morlet wavelet that is 70 samples long and an impulse with exponential rise and decay of the amplitude that is 90 samples long. The sine function is present during the entire input signal but the Morlet wavelet and the impulse features are present at random locations, corresponding to 10% of the entire signal for each feature. In addition, noise is added to the input signal so that the average SNR is 10 dB.

The matching pursuit with dictionary learning approach is evaluated also on ball-bearing vibration data. The data is taken from the bearing data center[1] at Case Western Reserve University. The test rig consisted of a motor, a torque transducer and a dynamometer. The evaluated bearings are mounted on the motor shaft. The data consists of accelerometer data recorded near the drive end of the motor. The data is collected at 48000 samples per second. Faults are introduced in the bearings at the inner raceway, the outer raceway and the ball. Data from normal, non-faulty bearings is used as reference.

3.1. Method

The matching pursuit algorithm with dictionary learning is applied to the artificially constructed input signal. First, the data is processed with a Matlab implementation of Smith and Lewicki's algorithm [5]. There are 8 atoms in the dictionary. Additional atoms are not needed since the input signal is formed by only 3 features plus noise. The atoms in the dictionary are initialized with normally distributed noise with zero mean, unit variance and vanishing tail amplitudes. The dictionary learning is done sequentially in blocks of length 1% of the original signal. The entire signal is processed 100 times so that the atoms can adapt to the features given the limited length of the input signal.

A similar procedure was used with the bearing data. First, the Matlab implementation of the algorithm was used to process the data. In this case there are 16 atoms in the dictionary. The next step is to use the fixed-point C implementation for the FPGA, which has some limitations compared to the Matlab implementation, see Section 2.2. There are 16 atoms in the dictionary and the atoms are initialized with zero-padded normally distributed noise with zero mean and unit variance. The atoms are 160 samples long. The matching pursuit and dictionary learning is done sequentially in blocks that are twice the length of an individual atom. The bearing vibration data is sampled under a variety of different test conditions. During acquisition of the data the load was changed between 0 HP and 3 HP. This cause a change of motor speed that range from 1800 to 1730 rpm. Similar tests are repeated with the various faulty bearings and the corresponding data is included in our case study.

[1] http://csegroups.case.edu/bearingdatacenter/

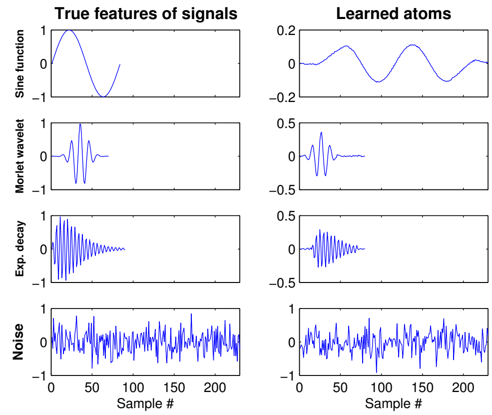

Fig. 3. Signal separation experiment. The mixed features include a sine function, a Morlet wavelet and an impulse function with exponential rise and decay. The comparison is done between the true features of the signal and the learned atoms. The 10 dB of noise introduced in the input signal is displayed on the left-hand side, while the final residual of matching pursuit is displayed on the right-hand side.

3.2. Results

The results for the known input signal are shown in Figure 3, which shows the true features in the input signal together with the learned atoms from the Matlab implementation. All features are shown with the same x-axis sample scale. The true features of the signal are shown in the column on the left-hand side of the figure, while the learned atoms corresponding to those features are shown in the right-hand column. The sine function shown as part of the true features is only one period formed by 84 samples. The learned atom have a period of about 79 samples. The difference in the amplitude of the features is related to the normalization / scaling of the learned atoms. Initially all eight atoms of the dictionary are adapting, but three atoms eventually represent the majority of the events and become similar to the features in the signal.

Figure 4 shows the results of the dictionary learning test with bearing data using the Matlab implementation of the model. It shows the 16 learned atoms with channel number 1 at the bottom and channel number 16 at the top. The size of these atoms range from 70 to 150 elements. The upper panel on the left-hand side shows the signal with the residual superimposed. The lower panel on the left-hand side shows the event-based representation of the signal. Each event is indicated by a triangle and denotes the activation of an atom at that particular channel and time. The signal can be approximately reconstructed as the weighted sum of activated atoms at the time offsets of the events. In this case only 19 events are required to represent 150 samples of the signal with a signal-to-residual ratio of 10 dB. The precision required to represent the weights of the events is typically lower than the precision required to represent the original amplitudes of the signal, and

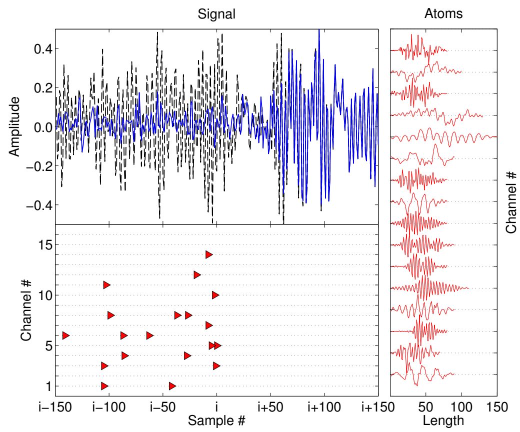

Fig. 4. Event-based representation of ball-bearing vibration signal using a dictionary of 16 learned atoms. The signal (dashed line) triggers events (triangles) on the 16 channels. The events are subtracted from the original signal, resulting in a residual (solid line). This bearing has a defect in one of the balls, which excites impulse-like atoms. The algorithm has been interrupted at sample i so that no events exist beyond that point.

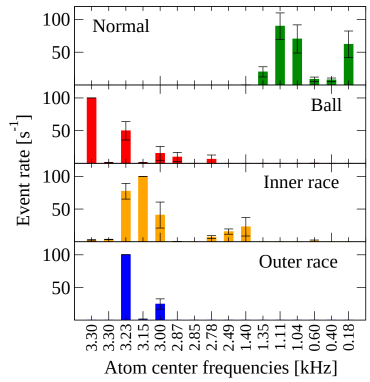

Fig. 5. Histograms of the channel event rates for normal and faulty bearings. The faulty bearings are divided in three categories depending on the geometrical location of the faults (ball, inner race or outer race). The type of defect is clearly characterized by the channel event rates, even though the rpm and load on the motor, and the shape of the faults are varying. Error bars denote standard deviations of event rates for different loads and fault shapes.

in some applications a binary weight is sufficient because a feature is either present or absent in the signal. Therefore, by adopting a parallel event-based representation of the signal, it is possible to reduce the data rate by at least one order of magnitude using this approach.

Different sets of data were evaluated with the Matlab implementation. The purpose was to compare the atoms corresponding to normal bearings and faulty bearings. Figure 5 shows a histogram of the channel event rates Normal bearings excite atoms with a low center frequency, while the faulty bearings excite atoms with a high center frequency. In addition, the location of the fault in the bearing affects the shape of the atoms so that the event rates on specific channels indicate the location of the fault. Figure 6 shows the results of the evaluation of one set of bearing data done with the C implementation for the FPGA. The format of the figure is similar to Figure 4. The size of all atoms is fixed to 160 elements long. The 300-samples signal segment shows the complete set of events required for its reconstruction. This shows that 45 events are required to reconstruct a 300 samples signal with a signal-to-residual ratio of 10 dB. Both implementations show an approximately similar data rate reduction. The size of the original signal was 60000 samples long. The Matlab implementation showed that only 8321 events were required to represent this signal. This gives an average of 20.8 events per every 150 samples. At the same time, the C implementation for the FPGA produced 8655 events with an average of 21.6 events per every 150 samples.

4. DISCUSSION

The possibility to enable new applications and study challenging real-world phenomena using an adaptive sparse-coding device is the motivation of this work. We show how an FPGA implementation of a machine learning analog-to-feature converter in principle can be realized and we discuss some limitations and benefits of such a device compared to conventional sampling with an ADC. We first illustrate that features can be identified using the implementation of the matching pursuit algorithm with dictionary learning. This is demonstrated using an artificially constructed input signal with known features. The implementation is able to learn approximations of the true features in the signal. It has been shown that this method reduces to independent component analysis under certain conditions [4]. This motivates the more challenging experiment to identify features in real-world ball bearing vibration data. The results obtained with the Matlab implementation shows that it is possible to reach about one order of magnitude reduction in data rate with sufficient signal-to-residual ratio to approximately reconstruct the original signal. This can possibly be further improved by encoding events in binary form, which is motivated in applications where features are either present in the signal or not. In addition, the channel event rates characterize the normal and various types of faulty bearings considered. This suggests that the algorithm and concept can be useful for condition monitoring. The same bearing vibration data is analyzed with the C implementation for the FPGA. This implementation gives a similar reduction of the data rate and comparable signal-to-residual ratios. There are some differences in the way the atoms are learned in the two implementations due to the limited flexi-

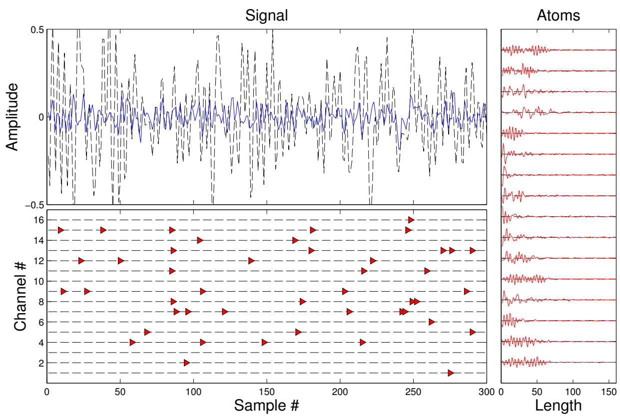

Fig. 6. Event-based representation of ball-bearing data on the implementation for the FPGA. There are sixteen learned atoms in the dictionary. The signal (dashed line) triggers events (triangles) on the sixteen channels. Also illustrated is the residual (solid line) that remains after subtraction of triggered atoms.

bility of the present FPGA prototype compared to the Matlab software. The Matlab implementation handles atom learning appropriately but the FPGA learning implementation needs to be improved because the atoms tend to grow towards one end of the buffer. This problem is caused by the way the zero padding of the atoms are handled in the FPGA design. Another issue is the limitation of the maximum atom length. A possible improvement is to design atoms of different lengths and let the learning algorithm allocate atoms in the FPGA of appropriate length for each particular waveform / feature. The non-zero amplitudes of the atoms should be centered in the atom buffers and learning of tail amplitudes should be subject to a constraint similar to that in [5]. When the first or last element of one atom buffer become non-zero, the corresponding atom should be reallocated to the center of another atom buffer of larger size. Other possible improvements are to parallelize the sliding window operation in the cross-correlation calculation, and to implement alternative learning algorithms.

The goal to develop a machine learning analog-to-feature converter that can replace conventional ADCs in some applications requires further work. There are open questions concerning the learning model(s) and the eventual hardware design. We need to ensure that the learning of atoms is well defined and that the limitations are clear. This first prototype is based on an FPGA development board and a custom daughter board with a high-speed and low-noise amplifier and ADC frontend. This approach is flexible and useful for our initial studies, but in order to realize the device conceptualized here a custom ASIC that implements the analog-to-event conversion should eventually be designed. That is necessary to reduce the power consumption, size and price of the device. Another application-specific problem is how to process the resulting events. In the context of condition monitoring, hybrid models that couple the empirically learned features to physical models and knowledge bases is an interesting possibility. However, the possibilities to automatically do feature extraction and detect emerging atoms far out in the sensor systems is already useful. Basic classifiers or pattern recognition models can also be used, and statistical spike pattern analysis can be applied to the output, for example in the form of neuromorphic processors that naturally use event-based codes.

5. ACKNOWLEDGMENTS

This work is partially supported by SKF through their University Technology Center at the Luleå University of Technology and the Swedish Foundation for International Cooperation in Research and Higher Education (STINT). We acknowledge the use of data from the ball bearing data center at the Case Western Reserve University.

6. REFERENCES

[1] A. Bruckstein, D. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Review*, vol. 51, no. 1, pp. 34–81, 2009.

[2] M. Elad, "Sparse and redundant representation modeling – what next?," *Signal Processing Letters, IEEE*, vol. 19, no. 12, pp. 922–928, 2012.

[3] B.A. Olshausen and D.J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, pp. 607–609, 1996.

[4] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?," *Vision Research*, vol. 37, pp. 3311–3325, 1997.

[5] E. C. Smith and M. S. Lewicki, "Efficient auditory coding," *Nature*, vol. 439, no. 7079, pp. 978–982, 02 2006.

[6] G. Indiveri and T. K Horiuchi, "Frontiers in neuromorphic engineering," *Frontiers in Neuroscience*, vol. 5, no. 118, 2011.

[7] H. Liu, C. Liu, and Y. Huang, "Adaptive feature extraction using sparse coding for machinery fault diagnosis," *Mechanical Systems and Signal Processing*, vol. 25, no. 2, pp. 558 – 574, 2011.

[8] S.G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Trans. Signal Processing*, vol. 41, no. 12, pp. 3397 –3415, dec 1993.

[9] E. Smith and M. S. Lewicki, "Efficient coding of time-relative structure using spikes," *Neural Computation*, vol. 17, no. 1, pp. 19–45, 2005.