Hardware-Accelerated Band-Power Feature Extraction for Supervised Learning on a Tactile Embedded System

Beiimbet Sarsekeyev, Joshua Osborne, Ahmad Patooghy, Olcay Kursun
Department of Computer Science, University of Central Arkansas, Conway, AR, 72035
bsarsekeyev1@cub.uca.edu, josborne@cub.uca.edu, apatooghy@uca.edu, okursun@uca.edu

***Abstract*—Real-time and energy efficient signal feature extraction have become increasingly important for machine-learning-enabled smart sensor systems in mobile and Edge applications. As considerable scientific and technological efforts have been devoted to developing tactile sensing with prospective applications in many fields, such as smart prosthetics, remote palpation, and robotic surgery with the sense of touch; in this paper, we develop a parallel hardware-software signal feature extraction method and apply it to a dataset of tactile texture classification. Being easily parallelizable, a set of passband-power feature extraction blocks compute signal power in various passbands and can be clock gated for accuracy-energy trade-offs controlled by a proposed feature summarization algorithm. Our experimental results on the tactile dataset have shown that the proposed method works at high levels of parallelization and realtimeness, performs with lower computational complexity, and achieves accuracy levels comparable to those of convolutional neural networks.**

***Index Terms*—Machine Learning, Feature Extraction, Energy Efficiency, Embedded Systems, Edge.**

I. Introduction

Machine-learning-enabled smart sensor systems have been increasingly used with applications ranging from prosthetics to telemedicine and assistive devices [1], [2]. Ultimately, such systems must process the noisy and multidimensional sensor data from various modalities under different response functions in real-time to detect patterns of interest and make inference using machine learning classifiers. Real-time data processing has become an increasingly important area of study as we have seen the large-scale implementation of machine learning and Edge devices in both corporate and user-based sectors. In order to decrease overall latency and energy-consumption, these mobile and Edge systems need real-time feature extraction methods that can be easily implemented in parallel hardware and support dynamic control of the resolution of the features by enabling/disabling them in runtime. In this paper, we have built upon the existing framework of the Cumulative Multi-Bandpower (CMB) feature extraction method. This memory-less, low-cost feature extraction method may be used in embedded systems to achieve highly accurate inferences when used as input by machine learning classifiers in embedded/Edge applications. To demonstrate the effectiveness of the CMB method, in [2], a tactile dataset was collected in [2]. The dataset was used to explore the tactile perception of the material properties in real-time using embedded systems, a recently popular task for achieving dexterous object manipulation in fields of robotics, prosthetics, and augmented reality [3]–[6]. On this dataset, the serial CMB algorithm was shown to perform with high accuracy and realtimeness [2]. The algorithm extracted CMB features serially as in Algorithm 1 and fed the feature vector to various pretrained classifiers for inference. The system used Random forests [7], [8], once trained, were easy to deploy on low-cost embedded boards for inference as they can be expressed by a set of simple if-then rules in code. It was shown that low-cost, highly accurate, and real-time tactile texture classification can be achieved on embedded systems using an ensemble of sensors, efficient feature extraction methods, and simple machine learning classifiers.

We improve the CMB method by designing a parallel computational structure that increases its throughput. Our parallel design supports energy efficiency by applying a feature selection/summarization algorithm that controls to reduce the number of features to be computed (in parallel). The feature summarization algorithm can decrease the number of CMB features computed via a histogram operation. The classifier predictions using fewer passband power features uses less computations and energy but can achieve nearly the same levels of accuracy in the inference of the class-labels. Starting with a large enough passbands, this approach can be used to cut down the number of passbands to find an effective balance between energy efficiency and classification accuracy. In an effort to improve this algorithm for embedded systems, this paper looks to discuss:

- Optimizing the CMB feature extraction algorithm for parallel processing on the co-designed hardware accelerator;
- Designing histogram based feature summarization algorithm for fusing CMB features for dynamic energy

conservation;
- Application of the convolutional neural networks to the tactile dataset for determining an accuracy baseline without taking into account any resource constraints.

The rest of this paper is organized as follows. Section II presents the serial CMB algorithm. Section III discusses the details of the proposed parallelization and feature fusion algorithm. We present and discuss our experimental results in Section IV, where we implement the proposed mechanism on a prototypical embedded system for tactile signal classification. Finally, Section V concludes the paper.

II. Review of the CMB Feature Extraction Method

CMB was introduced in [2] with the promise of real-time use of signal feature extraction in embedded systems. It has a light computational load that makes it particularly useful for embedded applications. Its formulation using Parseval's theorem for extracting frequency/power based features in time domain makes it useful real-time operation. Parseval's theorem [9] states that the total energy (thus, average power) of a signal can be calculated either using the amplitudes in the time domain or spectral power in the frequency domain. The CMB method works by employing exponential smoothing with different smoothing factors to the input sensor readings to create a bank of signals smoothened at different levels (leading to multiple versions of the original signal with different frequency passbands/cut-offs). Applying Parseval's theorem to these signals in real-time allows us to compute the power in their respective frequency bandwidths.

Let x[n] denote the discrete readings obtained from a sensor at time step t. Based on Parseval's relation, the average energy of a signal recording, $x[n]$, can be determined either by adding up the energy of the signal per each sample (i.e., $\sum \mid x[n] \mid^2$) at the time domain, or by taking the energy of signal in the frequency domain as summation of $\mid X(j\omega) \mid^2 /2\pi$ as shown in Eq. 1.

$$\sum_{t=-\infty}^{+\infty} \mid x[n] \mid^2 = \frac{1}{2\pi} \int_{2\pi} \mid X(j\omega) \mid^2 d\omega \tag{1}$$

Parseval's theorem (Eq. 1) states that the total energy (thus, average power) of a signal can be calculated either using the amplitudes in the time domain or spectral power in the frequency domain. More specifically, summing power-per-sample across time (i.e. sum of the squares of the amplitudes of the data samples) is another way of computing the total spectral power across frequency (Eq. 2). Therefore, the theorem offers a mechanism for staying in the time domain yet being able to do useful feature extraction in the frequency domain [9]. With this mechanism for real-time computation of the average power, $P[n]$, at time step t as the streaming samples of a given signal, $x[n]$ are received (Eq. 2).

$$\begin{aligned} P[0] &= x[0]^2 \\ P[n] &= \frac{1}{t+1}\big[n \times P[n-1] + x[n]^2\big], t > 0 \end{aligned} \tag{2}$$

Combining Parseval's theorem with the exponential smoothing [9] offers an efficient approach for computing a running average estimate for the power (Eq. 3), where α is a smoothing factor (e.g. $\alpha = 0.1$).

$$\begin{aligned} P[0] &\leftarrow x[0]^2 \\ P[n] &\leftarrow \alpha x[n]^2 + (1-\alpha)P[n-1], t > 0 \end{aligned} \tag{3}$$

To obtain different passbands of x[n], we apply exponential smoothing:

$$S^{\alpha_k}[n] = (1-\alpha_k) \times S^{\alpha_k}[n-1] + \alpha_k \times x[n] \tag{4}$$

for a set of K smoothing factor values, $1 = \alpha_0 > \alpha_1 > ... > \alpha_k > ... > \alpha_K > 0$, for $k = 1, ..., K$. Using lower smoothing factors, α_k, computes low-pass filters with lower cut-off frequencies (i.e. lower values of α_k actually increase the level of smoothing. The computation performs the following assignment:

$$S^{\alpha_k} \leftarrow (1-\alpha_k) \times S^{\alpha_k} + \alpha_k \times x[n] \tag{5}$$

The S^{α_k} array can then utilize Parseval's theorem to sum corresponding squares of these values such that narrower bands of frequencies are computed for continually-averaging power distributions. Let F^{α_k} denote the (average power) feature extracted for a given alpha value as in Eq. 6:

$$F^{\alpha_k}[n] = \frac{1}{n} \sum_i |S^{\alpha_k}[i]|^2 \tag{6}$$

Note that we can avoid buffering $F^{\alpha_k}[n]$ values and again use exponential smoothing to calculate the sum of power-per-sample, ${S^{\alpha_k}}^2$, as shown below:

$$F^{\alpha_k} \leftarrow (1-\beta) \times F^{\alpha_k} + \beta \times |S^{\alpha_k}|^2 \tag{7}$$

To make the CMB features DC-offset and scaling invariant, in [2] a simple normalization was applied to F^{α_k} features that yielded the so-called normalized R^{α_k} features sensitive only to frequency variations [2]:

$$F^{\alpha_k}[n] = \sum_i |S^{\alpha_k}[i] - x[i]|^2, k = 1, ..., K \tag{8}$$

$$R^{\alpha_k} = \frac{F^{\alpha_k}}{F^{\alpha_1}}, k = 1, ..., K \tag{9}$$

III. Proposed Hardware Accelerated CMB Feature Extraction Method: Eco-CMB

In this section, we propose a variation of the CMB feature extraction method co-designed with the parallelized hardware accelerator that we present to compute these features. The proposed method improves the accuracy of the original CMB features and allows CMB to be used in an adaptive energy-efficient setting; hence the name Eco-CMB. In this section, we first present our hardware accelerator architecture that can be used to compute either CMB or Eco-CMB features (passband power features) in parallel in realtime. The design of the architecture supports energy-savings by clock-gating under-utilized computing modules. Then we present the Eco-CMB algorithm that can use the full set or a subset of the passbands but still manage to sustain the accuracy of the classifier pretrained with the full set of passbands.

A. Hardware Accelerator Architecture

There has been an increasing demand for hardware acceleration in ML-enabled smart sensor systems. Parallel structure designs enable real-time processing of sensor readings in embedded systems. In intelligent embedded systems, where the data from sensors are collected in regular intervals and continually, custom hardware design can take advantage of inherent parallelism to deal with computational complexity of ML algorithms.

The serial/iterative CMB feature extraction algorithm given in Algorithm 1 supports a straightforward paralellization using hardware acceleration. Both time and space complexities of the serial CMB feature extraction algorithm (per sensor) are linear in K, $O(K)$, where K is the number of cumulative bands used by CMB. Using multiple sensors, the complexity of the algorithm is $O(Z \times K)$, where Z is the number of sensors (for example a 3D-accelerometer has $Z = 3$ displacement readings). Fast Fourier Transform (FFT), on the other hand, has $O(nlogn)$ time complexity and $O(n)$ space complexity per sensor, where n is the length of the FFT-window with n. Especially when the sampling rate is high (i.e. $n \gg K$), it becomes inefficient to buffer such a long window for the streaming input data. Although FFT features can lead to higher accuracy when using a single sensor, it is typical to combine features extracted from multiple sensors that closes this accuracy gap while allowing the use of real-time memory-less feature extraction algorithms like CMB.

Parallelization of CMB allows starting the feature extractor with a large number, K, of passbands without increasing the time needed to compute the set of K features of CMB. However, this leads to increased energy consumption and it requires the use of a smart mechanism to reduce K, if a smaller K value does not lead to large fluctuations at the output (prediction of the class-labels). The parallel CMB feature computation can utilize such enable/disable control signals to achieve energy efficiency along with the parallelized high accuracy computations. The CMB hardware accelerator takes advantages of parallel computations needed to compute different F^{α_k} and S^{α_k} values for various smoothing factors α_k. In fact, the accelerator implements $K+1$ parallel modules $M_0, M_2, ..., M_K$ each of which is capable of performing basic math operations to compute 7 in parallel. This results in $K+1$ times speedup in feature extraction. It is important to note that in our studies we have found out that the feature extraction is the bottleneck in computations, not the subsequent inference by the machine learning classifier (e.g. a random forest classifier).

Algorithm 1 Cumulative Multi-Bandpower (CMB) feature extraction algorithm for a system with Z sensors and K passbands per sensor.

for *(sensor* $z \leftarrow 1$ ***to*** Z*)* **do**
  $x = data[z]$
  Each $sensorz$ uses its own $\alpha[0..K]$, $S[0..K]$, $F[0..K]$, β
  for *(*$k \leftarrow 0$ ***to*** K*)* **do**
    $S[k] \leftarrow (1 - \alpha[k]) \times S[k] + \alpha[k] \times x$
    if *(*$k == 0$*)* **then**
      $F[k] \leftarrow (1 - \beta) \times F[k] + \beta \times x^2$
    else
      $F[k] \leftarrow (1 - \beta) \times F[k] + \beta \times |S[k] - x|^2$
      $R[k] \leftarrow F[k]/F[1]$
    end
  end
end

Figure 1 shows the architecture of the proposed accelerator hardware. The goal of the proposed accelerator is to enhance the classification efficiency in terms of throughput and energy consumption. The accelerator consists of $K + 1$ computing modules (M_k) to calculate F^{α_k} features, $0 \leq k \leq K$. As shown in Figure 1, each block of the accelerator consists of two floating point multipliers, a floating point adder, and a local buffer to keep the feature value. Module-based parallel execution (MBPE) processing allows initial operations to be performed in a continuous, non-sequential fashion, meaning that real-time sensor data may be gathered at much faster rates without the need for a storage-based buffer or queue. This method also decreases system-wide signal latency. Furthermore, by decreasing overall run times for feature extraction as well as eliminating need for significant buffering solutions, MBPE decreases total power draw on the system.

The control unit of each module decides if the module should work or clock-gated. If the module is in the working mode, the control unit applies the appropriate clock signals to the multipliers and the adder modules to lead the required

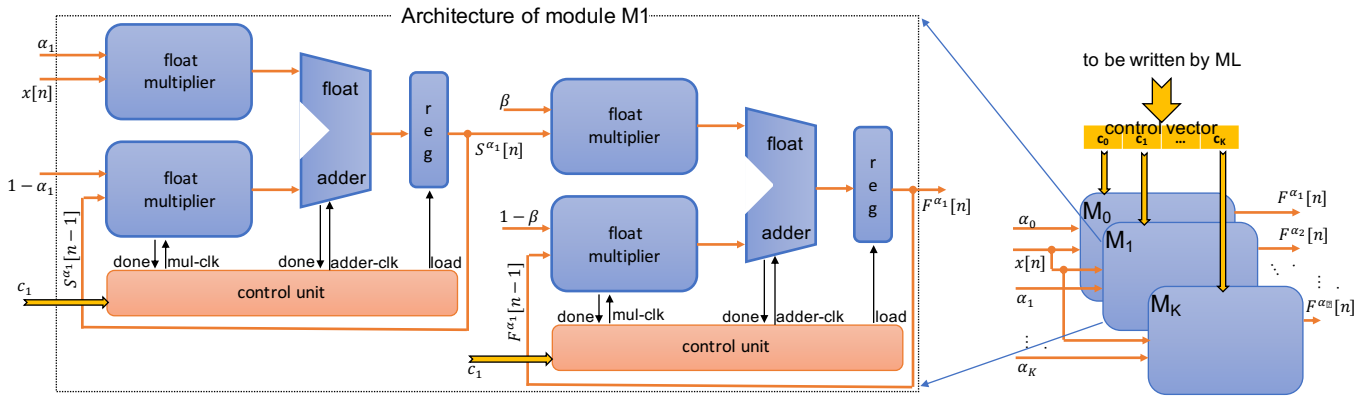


Fig. 1: The architecture of MPBE hardware accelerator is capable of computing CMB features in parallel.

floating point multiplication/addition operations. As the input data, intermediate variables computed and the features extracted are all represented in IEEE single precision, we need to have the hardware accelerator working with higher frequency than that of data reading to keep up with the input data stream. We have setup the hardware accelerator's clock $50x$ the frequency of input data. This give the accelerator enough time to perform floating point operations.

The hardware accelerator saves energy in two ways. First, if the input data is the same as previous data, the control unit will not trigger the clock signal for the multiplier/adder components; not being clocked means that all internal nets of these components will be protected from unnecessary signal activities, which in turn significantly reduces switching energy consumption. The second energy saving method works in a higher level i.e., the $K+1$ parallel modules of the accelerator can be controlled via a vector of control signals $C = [c_0, c_1, c_2, ..., c_K]$. If c_k is $'1'$ the module will work; otherwise the control unit will again clock-gate the module and accordingly the multiplier/adder components will not see unnecessary signal activities. The c_k control signals will be asserted by the machine learning structure for real-time regulation. The MBPE hardware will input these control signals to enable corresponding M_k blocks denoted by the subset $A = [\alpha_0, \alpha_1, \alpha_2, ..., \alpha_K]$. Our investigations to synthesize the proposed MBPE hardware accelerator show that each of the parallel computing modules, (M_k), would need hardware resources less than 2K CLBs and 900 DFFs when synthesized on Xilinx Virtex-5 5VFX200TFF1738 FPGA [10].

B. Eco-CMB Feature Extraction/Summarization Algorithm

The DC-offset and scaling invariant CMB features that are called the normalized R^{α_k} features in [2]. However, due to the computation in time domain using the quadratic terms negatively affect the invariances of these features especially in the presence of low frequencies. That is, simple classifiers like

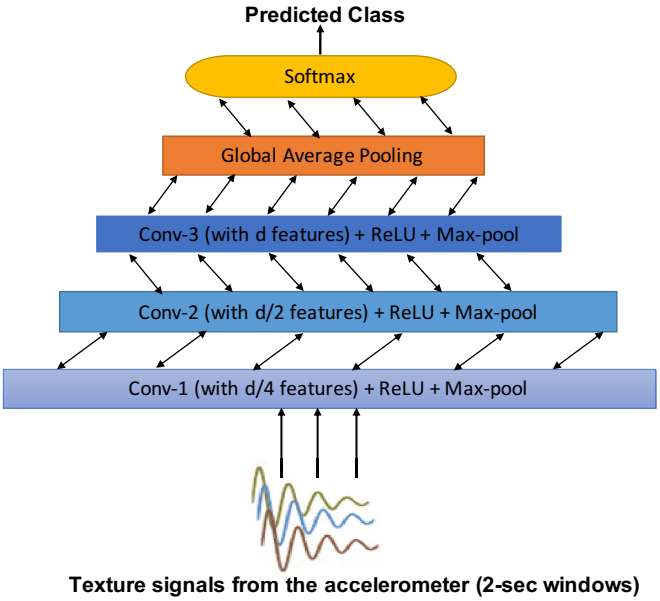


Fig. 2: The CNN architecture used in the experiments (three-convolutional CNN is shown as the representative). While the size of the receptive fields of the neurons double from one convolutional area to the next, we doubled the number of feature maps (e.g. for the 3-layer CNN, we used 25, 50, and 100 neurons in the first, second, and third layers, respectively.

K-Nearest Neighbor (KNN) [11] does not work well with the unnormalized or normalized CMB features because two short windows cropped from a long recording of a particular class may have very different R features. Therefore, we propose to further normalize the R features by the standard deviation of the window:

$$Eco^{\alpha_k} = \frac{R^{\alpha_k}}{s_x}, k = 2, ..., K \tag{10}$$

$$s_x = \frac{1}{n}\sum_i (x[i] - m_x)^2 \tag{11}$$

$$m_x = \frac{1}{n}\sum_i x[i] \tag{12}$$

Note that we omitted R^{α_1} feature (for k=1), because it is always equal to 1 by Eq. 9. Also note that Eco-CMB features can also be approximated by:

$$Eco^{\alpha_k} = \frac{R^{\alpha_k}}{\sqrt{F^{\alpha_1}}}, k = 2, ..., K \tag{13}$$

The advantage of Eco-CMB features is the proposed normalization not only improves the accuracy of complex classifiers but also significantly improves the accuracy of simple classifiers such as KNN, making them feasible options for the embedded applications, which also allows simple retraining on the embedded board. Complex classifiers such as multi-layer perceptrons or random forests, in their training, can handle dealing with the dynamic range of the signal-windows (cropped from longer recordings as in [2]) used as the training/test examples but their training algorithms are more difficult to be implemented on the embedded board as opposed to just using them pretrained (having them trained offline) for simply making inference on the test examples on the embedded system. In other words, adding new examples or new classes to update/re-train a random forest or a neural network may require the training to be performed from scratch, which may not be feasible on the embedded system (or the full dataset may not be available). When its accuracy is improved, on the other hand, the simplicity of KNN makes it a very good option as the classifier on the embedded system because it does not require any training: New training examples with their associated class-labels can be simply be stored in the memory of the embedded board to continue the training process.

Finally, we take a histogram of the Eco-CMB features by dividing the range of Eco-CMB features into equisized bins and return the values of the probability density function at the bins normalized such that the integral over the range is 1. For the range of the bins of the histogram, we use the min and max value of Eco-CMB features in the dataset. The use of the histogram achieves normalization of the length of the feature vector; that is, whether we use K Eco-CMB features or $K/2$ (or less) features, the histogram always produces the same length feature vector (equal to the number of the bins; e.g. 10). Therefore, a pretrained classifier (on the full set of K Eco-CMB features can still be utilized on the test set even when a smaller subset of Eco-CMB feature values are computed for energy-savings because the histogram of this smaller set of Eco-CMB features will have the same number of bins.

Fig. 3: The twelve texture classes available in the CMB tactile dataset.

IV. Experimental Evaluations

To evaluate the performance of the Eco-CMB feature extraction algorithm for tactile signal processing on Edge platforms, we first compare its accuracy with convolutional neural networks (CNNs) [12], [13] without considering resource limitations associated with their training/learning. We trained CNNs with various number of convolutional layers and number of features in each layer (we used PyTorch library [14] for implementing the CNN architecture shown in Figure 2). We compared resource-hungry CNNs and CMB features (CMB featured fed into a simple random forest classifier) on their application to the tactile dataset from [2]. The dataset was collected to explore the tactile perception of the material properties in real-time using embedded systems. The dataset has 12 texture classes, including sandpapers of various grits, Velcro strips with various thicknesses, aluminum foil, and rubber bands of various stickiness as shown in Figure 3 [2]. For each texture, 20 seconds of recordings are available. The sensors are attached to a probe that rubs against the surface of a rotating drum covered by the aforementioned textures. A 3-D accelerometer sensor with 200 Hz sampling rate and a microphone with 8 KHz sampling rate were used as the sensors to acquire the vibrotactile signals.

As shown in Table I, CMB accuracies closely match with those of CNNs with the same number of features. The 3-convolutional layer CNN, for example, starts with 25 feature maps, then transforms those to 50 feature maps in the second convolutional layer, and then finally transforms to 100 feature maps. The global average pooling layer extracts the final set of $d = 100$ features and feed them into the softmax layer for classification. Using 3-layer CNNs does not yield better results than 2-layer CNNs because of the high number of parameters (weights) to optimize on a small dataset. Comparing the accuracy obtained using 100 CMB features and 2-layer CNN shows that the CMB method offers the same levels of accuracy with less resource usage.

As 100 α smoothing factors (between 0 and 1) were used in [2] for CMB feature extraction, in this work, for Eco-CMB,

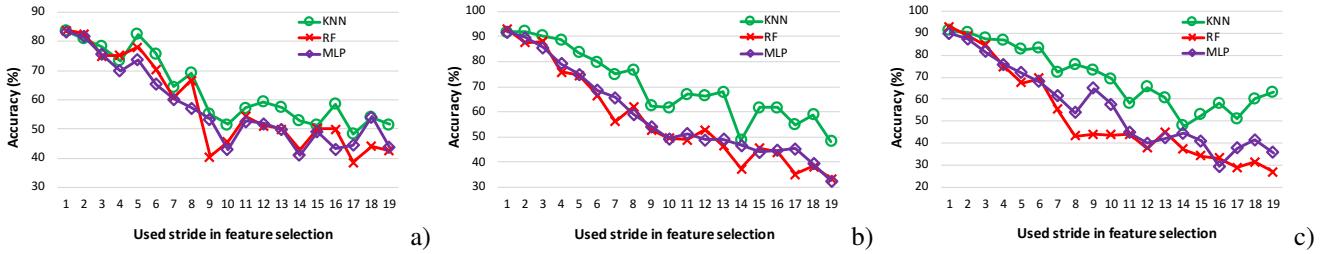


Fig. 4: The accuracy of texture classification using the sound recordings as a function of the stride (a bigger stride uses fewer Eco-CMB features by taking every second, third, forth feature and so on). Panels a, b, and c correspond to using 10, 20, and 30 bins in the histogram that summarizes the Eco-CMB features selected by the strides.

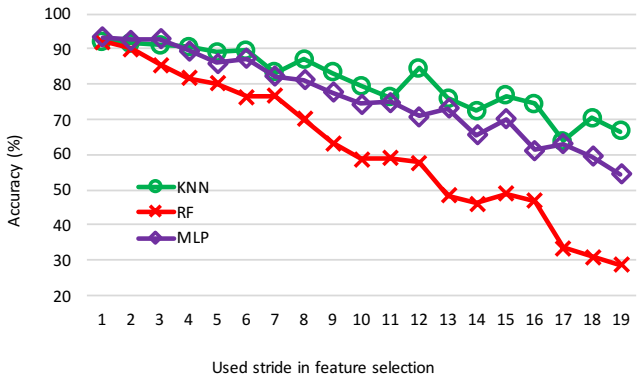


Fig. 5: The accuracy of texture classification using the 3D accelerometer sensor recordings as a function of the stride (reduction in the number of features used for the inference).

we also used $K = 100$ (for the full set of features). We reduced the K value as part of the proposed energy-saving feature. Although evenly spaced smoothing factors were used in [2], in this work we placed them more appropriately as follows: (i) As the dataset was collected such that different sensors cover different frequency ranges, we selected the list of $K = 100$ α values for computing the Eco-CMB features such that they correspond to an evenly spaced set of cut-off frequencies; (ii) we selected the smoothing factors so that the cut-off frequencies fall in reasonable ranges (for example, the microphone has a very high sampling frequency for use in tactile processing and therefore using α values near 1 would not lead to passbands with discriminative features). Therefore, for the 3D accelerometer sensor recordings (X, Y, Z), we used α values in the range of [0.76..0.05] that corresponds to cut-offs around [100KHz..2Hz]. For the sound recordings (the S signal recorded by the microphone), we used α values in the range of [0.39..0.01] that corresponds to cut-off frequencies around [800KHz..16Hz].

TABLE I: Accuracy on the test set using Convolutional Neural Networks versus the proposed Eco-CMB as a function of number of features extracted.

Number of features used (d)	CNN - (number of convolutional layers)			CMB features & Random Forest
	CNN-1	CNN-2	CNN-3	
d=25	92.7 ± 2.8	91.6 ± 3.8	85.2 ± 3.3	90.5 ± 3.4
d=50	94.8 ± 2.4	92.5 ± 3.9	89.3 ± 4.3	92.3 ± 3.5
d=100	93.8 ± 1.5	95.4 ± 2.6	91.2 ± 5.8	94.6 ± 3.4
d=200	94.3 ± 2.6	96.3 ± 0.8	92.6 ± 2.4	95.3 ± 2.9

For the histogram of the Eco-CMB, we have found that 20-30 bins in the histogram are optimal. The number of bins is not very sensitive; however, it needs to be chosen properly with respect to the number of classes of interest and how much precision is needed to achieve good classification accuracy. In our experiments on the tactile dataset, using fewer than 15 bins loses too much accuracy and using too many bins, greater than 30, does not lead to increase in the accuracy and it can even lead to the curse of dimensionality. We used 20 randomly cropped two-second-long recordings (windows) from each one of the texture classes for the training set. We used 50 such windows per class for the test-set. The reason we picked a relatively small number of training examples is to show that the proposed embedded system can achieve high accuracy with a small number of labelled examples.

We trained the classifiers using the training set that has the full set of features (all $K = 100$ Eco-CMB features, corresponding to using all the modules with their enable control signals set to 1). As the classifiers, we use KNN (K-Nearest Neighbor using the Euclidean distance and the number of neighbors is set to 1), Random Forest (with the default 100 estimators/trees and 50% of the features evaluated at every decision node), and MLP (multi-layer perceptron with two hidden layers with 50 units in each layer). All of these classifiers are implemented in Python using scikit-learn machine learning library [15]. We varied the number of Eco-CMB features on the test set, as shown in Fig. 4. The reduction in the number of Eco-CMB features is performed by taking strides of i. Every i^{th} feature is selected. In each stride the

enable control signal of the selected feature is set to 1 and the enables of the other $i-1$ modules are set to 0. For example, if stride is 5, then only $K/5$ features are selected. As $K = 100$ in our experiments, the strides of 15 and 16 for example end up using different subsets of Eco-CMB features but they both have only 6 of them. As shown in Figures 4, 5, and 6, using the full set of K features is the best in terms of accuracy; nevertheless, using a stride of 2, for example, uses only $K/2 = 50$ features that can achieve nearly the same accuracy with energy-efficiency because only the half of the parallel modules compute their Eco-CMB features.

In our experiments, we found that the random forest classifier works well using the full set of features (one advantage of random forests is that they can be simply implemented as a collection of if-then rules [2], [11] that can be easily translated to the embedded platform and deployed as the inference algorithm). MLP also offers a good option because MLP can be continued the training (a process called online training that will continue making weight updates [11]). However, when the number of Eco-CMB features are reduced (in the energy-saving mode), neither random forests or MLPs worked as well as KNN did. Using the proposed feature vector (histogram of the Eco-CMB features) as input, KNN achieves an accurate classification for the 12 texture classes that can be sustained at high levels while the number of features are reduced using larger strides.

V. Conclusions

In this paper, we co-designed a real-time signal feature extraction method and its hardware accelerator that compute these features in parallel. The proposed method/architecture improves the accuracy and applicability of the CMB (Cumulative Multi-Bandpower) feature extraction method and allows CMB to be used in an adaptive energy-efficient setting. The proposed hardware acceleration for real-time energy-efficient signal feature extraction can be employed in IoT devices with sensors for processing and classification of streaming inputs. The CMB feature extraction method works by computing the passband powers in gradually decreasing cut-off frequencies. According to the proposed algorithm, after applying normalization and histogram steps to CMB features, a mechanism in the co-designed hardware accelerator becomes applicable to enable/disable these parallel modules computing the passband features. We performed our simulations on a tactile dataset for texture classification. We showed that using the histogram approach offers robustness in accuracy against the reductions in the number of passbands. The highest accuracy and energy efficiency combination is achieved by the nearest-neighbor (KNN) classifier. In addition to serving as a pretrained classifier for inference on the test examples, as a lazy classifier, KNN offers the most practical mechanism of simply collecting/storing more training examples for the continuation of the training process on the embedded system implementation.

Acknowledgment

This work was supported by the Arkansas INBRE program with a grant from the National Institute of General Medical Sciences (NIGMS) P20 GM103429 from the National Institutes of Health and by the DART (Data Analytics That Are Robust and Trusted) grant from NSF EPSCoR RII Track-1.

References

[1] N. Ha, K. Xu, G. Ren, A. Mitchell, and J. Z. Ou, "Machine learning-enabled smart sensor systems," *Advanced Intelligent Systems*, vol. 2, no. 9, p. 2000063, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202000063

[2] O. Kursun and A. Patooghy, "An embedded system for collection and real-time classification of a tactile dataset," *IEEE Access*, vol. 8, pp. 97 462–97 473, 2020.

[3] O. Oballe-Peinado, J. A. Hidalgo-Lopez, J. Castellanos-Ramos, J. A. Sanchez-Duran, R. Navas-Gonzalez, J. Herran, and F. Vidal-Verdu, "Fpga-based tactile sensor suite electronics for real-time embedded processing," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 12, pp. 9657–9665, 2017.

[4] W. Duchaine, "Why tactile intelligence is the future of robotic grasping," in *IEEE Spectrum Automaton*. IEEE, 2016.

[5] C. Chi, X. Sun, N. Xue, T. Li, and C. Liu, "Recent progress in technologies for tactile sensors," *Sensors*, vol. 18, no. 4, p. 948, 2018.

[6] A. Moringen, W. Aswolinkiy, G. Buscher, G. Walck, R. Haschke, and H. Ritter, "Modeling target-distractor discrimination for haptic search in a 3d environment," in *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (BIOROB)*, Aug 2018, pp. 845–852.

[7] M. Fernandez-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *Journal of Machine Learning Research*, vol. 15, pp. 3133–3181, 2014. [Online]. Available: http://jmlr.org/papers/v15/delgado14a.html

[8] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals & Systems (2nd Ed.)*. USA: Prentice-Hall, Inc., 1996.

[10] M. Al-Ashrafy, A. Salem, and W. Anis, "An efficient implementation of floating point multiplier," in *2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)*, 2011, pp. 1–5.

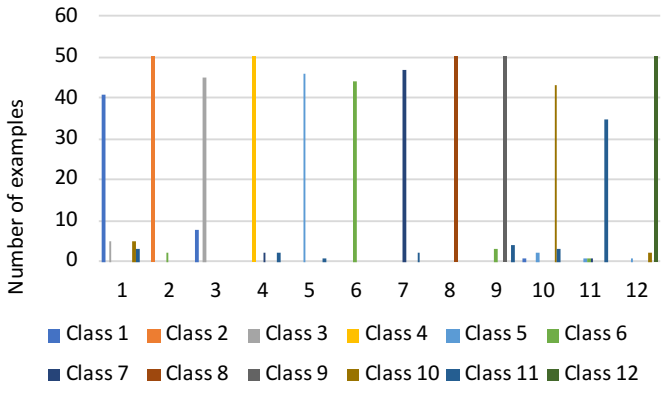
[11] E. Alpaydin, *Introduction to Machine Learning, Third Edition*. The MIT Press, Cambridge, 2014.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, Cambridge, 2016.

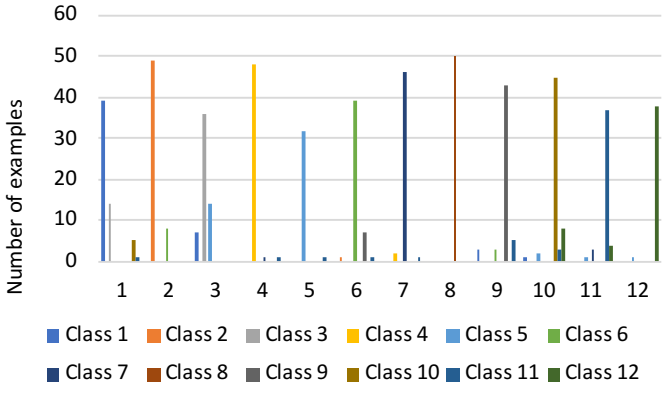
[13] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2019.

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035, [Online documentation for the transforms package is available at: https://pytorch.org/docs/stable/torchvision/transforms.html].

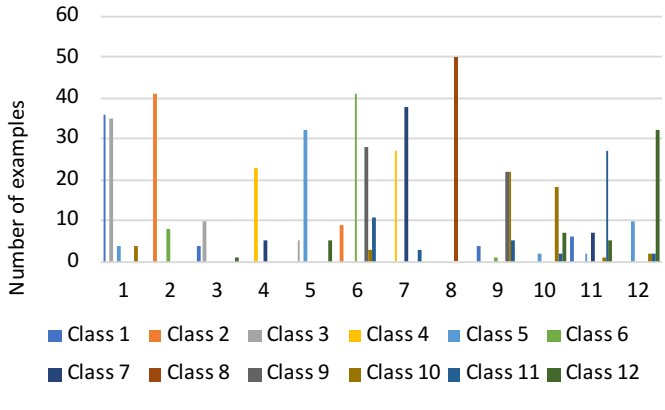
[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.



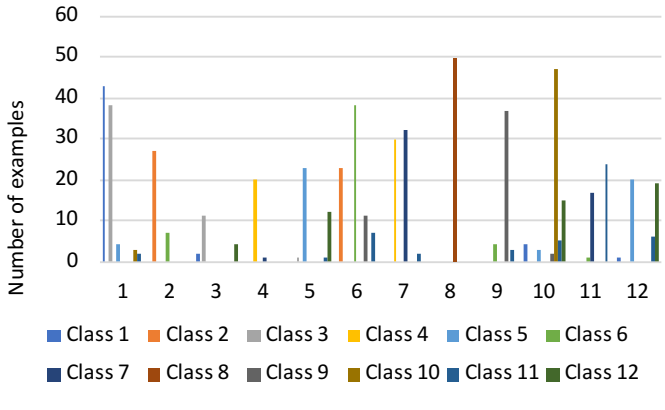
a)



b)



c)



d)

Fig. 6: Confusion matrix visualizations for the classification using the sound recordings. Panels a to d correspond to strides of 1, 5, 10, and 20, respectively. Each panel shows the number of examples correctly classified in each class (with 50 examples per class).