CanYouHearMeNow: A Minimum Voice Activity Detection System for Always-on Speech Classifiers

Braedon Salz, *Student Member, IEEE*
University of Illinois at Urbana-Champaign
Urbana, IL, 61820
Website: http://bradysalz.com

***Abstract*—We present two minimal, efficient voice classifiers that are able to maintain reasonable voice activity detection numbers within a certain SNR bound. An analog system only requires six bits of precision to maintain 90% detection accuracy assuming at least 10dB of SNR. A digital system can use fractions of the primary ADC's range while also maintaining similar levels of accuracy. The primary motivation of this work is to minimize area, power, and complexity of intelligent classifiers, as well as linearizing the power-complexity trade-offs.**

I. Introduction

Systems are becoming more and more intelligent in order to better serve their user bases. Computational devices began with no programmability, only able to execute their hard-coded features. Input streaming began with large mechanical switches, and output streaming began with gears and pulleys to visualize results. That clearly does not scale, so programmability swiftly increased to allow users to interact with results via a shared memory. Once systems became intelligent to operate on their own, other streaming data inputs can be added. Modern systems today combine many results using sensor fusion in order to create intelligent systems. As personal computation devices such as smartphones become more and more popular, the amount of information available increases exponentially. This however presents a problem to the end user, as computational speed and battery life have not enjoyed the same scaling.

In this paper, we present a introductory solution to what is becoming one of the most common inputs for intelligent systems: voice detection and classification. Some work has already been done to overcome this, namely, intelligent voice assistants such as Microsoft's Cortana or Apple's Siri offload the voice related tasks from the CPU to separate digital signal processing (DSP) chips. These chips, while enjoying lower power relative to a full CPU, still draw significant amounts of power when operating. Furthermore, given the sporadic nature of voice input as well as need for rapid response, these chips are effectively always on. Work has also been done to minimize the data acquisition characteristics through use of compressed sensing in the analog domain before handing it off to a digital processor. Unfortunately, these systems are not robust to changes in volume, dynamic range, or other context changes. Implementing dynamic control over these systems is effectively the same workload as operating a full DSP at best, and similar to a CPU at worst. This severely limits the system's usability.

The clear method for power and performance optimization involves breaking up the task into detection and classification. To an always-on classifier, all data is equal, and it will spend equal amount of energy trying to classify an input. This is not the case for the human brain however. Imagine the following situation: a user is attempting to talk to his smartphone in public in a foreign country. The phone will waste time continuously processing the background noise for no reason. The human brain will recognize that it does not understand the background noise, and will gradually filter it out. This is where inspiration for our project began, by separating useless data from useful data. We seek to create a novel architecture for detecting voice activity in order to linearize the power-complexity curve (as shown in Fig. 1). Current systems implement a "step function", as they threshold the input for any signal, then activate the entire classifier. An ideal power-complexity proportional system would ideally have a linear power demand as a function of the complexity of the input task.

For this project, we propose two architectures for a voice activity detection circuit. One system is an analog front-end inspired by previous work [1] which focuses on minimizing power and robust performance in high signal-to-noise (SNR) scenarios. This system involves a bank of bandpass filters in the analog domain to extract relevant features, then running through a decision tree trained a priori. The other system is a digital architecture focused on re-using existing area and online linear. This system uses a sub-set of the system ADC for feature extraction and allows dynamic scaling in order to best adapt to changes in the classification context.

The paper is organized as follows. In Section II, we present an overview of classification using decision trees, as well as architectures for each system. In Section III, we present the datasets chosen for this project and the motivation behind them. In Section IV, we show simulated results of both classifiers. Lastly, we summarize our results and present our conclusions in Section V.

II. System Overview

Currently machine learning and deep learning is a very popular topic. Research in those areas is showing exponential

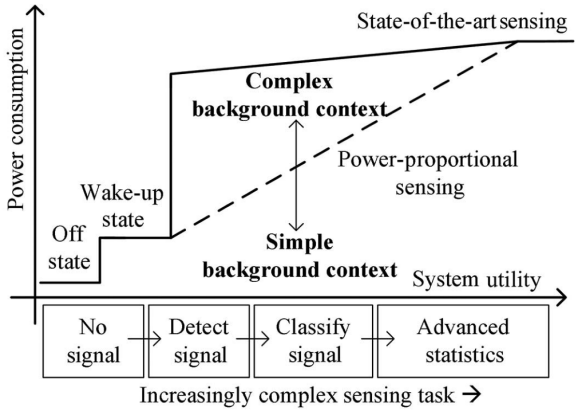

Fig. 1. Power-complexity proportional classification (dashed line) compared to traditional classification (solid line).

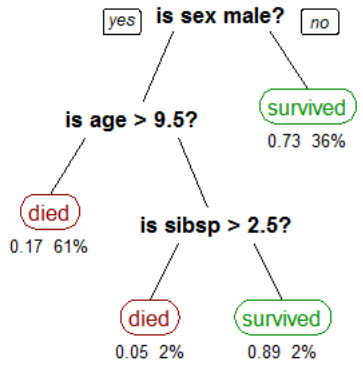

Fig. 2. A decision tree classifier built using the Titanic passengers dataset.

improvements over traditional systems in terms of classification accuracy in all sorts of problems. However, as classification accuracy increases more and more, there is an unspoken power cost that also increases. One of the most impressive achievements has been DeepMind's AlphaGo beating a world champion in the game of Go, a game long thought nearly impossible to train a computer to play at all, let alone at a world championship level. What was less impressive was the hardware required to do - 1,202 CPUs and 176 GPUs [2]. This is beyond reasonable to use on a mobile device, as such a system can not provide that amount of power instantaneously, let alone continuously.

For this project, we focused on the simplest architecture to both train and implement in hardware: a decision tree classifier. A simple example is shown in Fig. 2. This tree was constructed with the Titanic passengers dataset, which attempts to predict the survival rate of an individual on the Titanic given certain features. A decision tree classifier works by making a simple inequality check at each node, iterating down the tree based on the decision. If the node is a leaf, then the tree classifies the data accordingly.

We trained our decision tree by using the Gini impurity metric. Other metrics are equally popular, such as entropy minimization or variance reduction, but they do not lend themselves well to online learning in hardware as they either require large amounts of calculation or use difficult to implement operations like logarithms. The Gini impurity metric (I_G) is straight-forward in comparison. For each item x, the metric aims to maximize the probability p that each item is assigned to the correct class k out of N possible classes:

$$I_G(x) = \sum_{i=1}^{N} p(i|x)(1 - p(i|x)) = 1 - \sum_{i=1}^{N} p(i|x)^2 \quad (1)$$

In our case, we use binary levels - either audio is detected or not. This makes solving for the optimal feature to use at each node quite simple, as we merely have to count the incorrectly classified labels once per feature per node. Upon iterating over the number of available features in a node, and we choose the one that provides the largest Gini impurity as the splitting feature. We descend into each node and continue this manner recursively until sufficient accuracy is reached. For both systems, the base classifier was a full binary decision tree with a maximum depth of 5, which corresponds to $2^5 = 64$ decision nodes. This number was chosen through empirical testing to be a reasonable threshold for accuracy-complexity curves.

Many voice activity detection algorithms have been previously used for feature extraction, but they are again to complex to reasonably implement in real-time on a low power classifier. One of the most popular features to use is Mel frequency cepstrum coefficients (MFCCs). However, calculating MFCCs involves taking a Fourier transform, binning the spectrum into M bins, taking M logarithms, then taking a discrete cosine transform on those values. The other popular metric is to measure harmonicity, which is a normalized autocorrelation over a wide window. Both of these are reasonable calculations for a DSP voice classifier, but not for a voice detection algorithm, as they are too computationally complex.

Instead, we simply use the average energy in several frequency bins. We construct the frequency bins by running the input data through a set of bandpass filters, then integrating the result to get an approximate measure of the energy in each bin. These are the only inputs to our classifier. The method in which we obtain these inputs for both systems is described below. The last important metric is the number of samples to take in one decision, which we'll refer to as the feature window size. There is a clear trade-off here between the confidence of the decision and the latency of the decision when it comes to the feature window size. A common range is 15 to 30 ms. Any shorter than that, and there is not enough data to computer. Any longer, and the classifier will possibly miss relevant data for its task. We chose 20 ms as balancing point for this, although this was not tested to find if it was optimum.

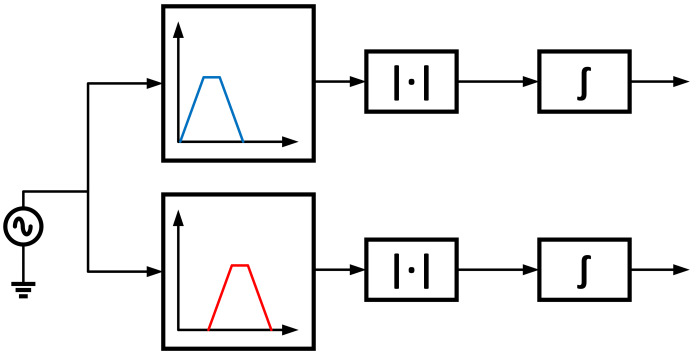

Fig. 3. The analog feature extractor system **add integrator not comparator**.

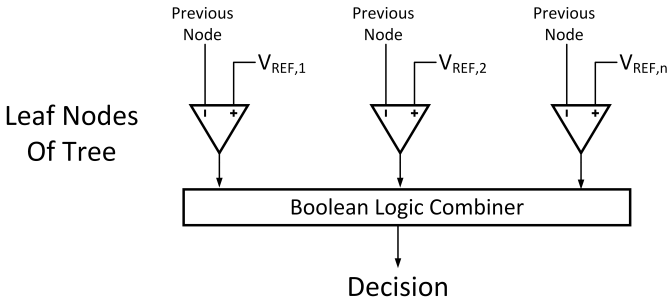

Fig. 4. A subset of the analog classifier chain, showing an intermediate node and several leaf nodes.

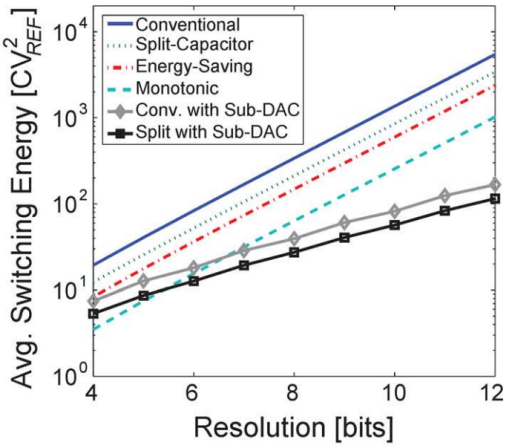

Fig. 5. SAR ADC average switching energy as a function of resolution.

A. Analog System

The analog system was the first architecture designed, as it fit in very well with the goal of minimizing power in classification. Rather than use any complex digital calculations, we simply construct a bank of analog bandpass filters. From there, we take the output of each bandpass filter, rectify the signal, then put it into a clocked integrator. The integrator resets every feature window size time period to zero. A theoretical implementation of this is shown in Fig. 3.

The filters are designed as first order bandpass filters with 0dB passband gain and a Q of 2. The filter bank's center frequencies cover one decade: 400Hz to 4kHz, distributed logarithmically. We use an ideal rectifier passed into an integrator with 10Hz bandwidth to perform the summing. Following that, each weight is passed to the decision tree. Each node of the decision tree consists of a low-offset comparator that compares a hard-coded V_{ref} to the relevant feature node. A simple combinatorial logic circuit would then decode the output of all leaf nodes into a single binary label - 1 or 0. In the event voice is detected, a flag is sent high to turn on the ADC and voice classifier processor.

While this is the most power-efficient implementation, it also leaves the classifier quite rigid, which motivated the digital subsystem discussed in Section II-B. If we later find an optimal set of filters or coefficients that significantly improve our classification accuracy, it will be difficult to update. One possible solution is to have an on-chip DAC to generate the V_{ref} biases. For the level of precision required for the decision tree classifier biases, the DAC step size would need to quite small, on the order of 10 μV. Another option would be to selectively trim voltages the bias voltages within a range, this is often done for high-precision chips as a one-time calibration (OTP). This leads to a clear trade-off between trimming offset and classifier area.

The only analog discontinuity modeled is noise, in which we add uniform Gaussian noise to model input referred noise (discussed in Section III). Modeling passband gain variation, Q variation, and comparator offset are metrics that were outside the scope of this project, but are likely not too impactful to the project's results. Previous work has shown that voice activity training algorithms are robust to constant offsets, as they simply show up as bias terms in the training algorithms [3].

B. Digital System

One downside of the aforementioned analog system is the area required. In order to support these low frequencies, larger capacitors are required to provide the low frequency corners. For reference, in 90nm, the capacitor area required is approximately 80μm $\times$ 50μm, per filter. This adds up quite quickly, motivating us to explore smaller solutions as well. Similarly, the rigidness of the above system prompts an investigation into creating a similar architecture, but with more tunability. We note that regardless of whether or not a voice classifier is being used, the hardware is still there. Dark silicon, the idea that a majority of the chip is simply not used, is a rising problem in multi-core CPU and GPU architectures [4]. Again, while it is clear that the minimal power solution maximizes dark silicon, as that is saved power, it is worth investigating silicon reuse opportunities.

Obviously using the voice classifier ADC at full rate and full input range is an already implemented solution. Here, we instead opt to only use a select number of the bits, effectively sub-sampling the precision of the ADC. For our argument, we assume the ADC is implemented as a successive approximation register (SAR) ADC. Previous work has shown

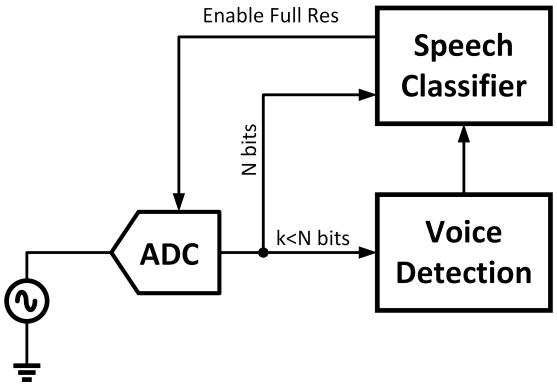

Fig. 6. A system overview of the "precision sub-sampling" ADC.

quite clearly that the energy spent in switching a SAR ADC is directly proportional to the resolution of the ADC [5].

The digital classifier system we construct will, at a high-level, sub-sample the voice classifier ADC to detect voice activity. We implement the same feature extraction described above, by constructing a bandpass filter bank, rectifying each bin, then summing digitally to get a total bin power. Once voice activity has been detected, we will "activate" the full resolution of the ADC, and allow voice classification to occur. Since the training thresholds are fully digital, we open the option for online training. The system can scale its input resolution as well as adjust the threshold values quite easily, as they are just digital control words. Adapting the tree structure itself is actually fairly simple as well, meaning we can train the full classifier in hardware.

The reason it's simple to retrain a binary decision tree classifier is due to the decision metrics as it is robust against poor training data organization. If the training set were consist of 50k positive samples followed by 50k negative samples, that would result in low test set accuracy when using a batch or stochastic gradient descent trainer. Since the Gini impurity metric only concerns itself with the quantity of each class per feature, not the ordering, online learning is significantly easier.

III. Datasets + Training

A. Datasets

Any classifier is only as good as the data it can train on, so it's critically important to have a large amount of valid data. We used four primary audio sources for this: TIMIT [6], NOIZEUS [7], KitchenSound [8], and UrbanSound [9]. Each dataset however, had a unique recording situation, requiring significant amount of data cleaning in order to maintain equality amongst all input data.

The TIMIT dataset consists of 630 speakers of eight different dialects of American English. Each speaker reads approximately Each file consists of an approximately 2 second long clip of a speaker reading a sentence, with no noticeable noise in the background. This dataset was recorded at 16kHz sampling rate with 16-bit accuracy. Only two of the This data was used to provide the majority of the positive examples for our training set.

The NOIZEUS dataset contains 30 sentences repeated by six speakers (three male, three female) across seven background noise scenarios: onboard-train, babble, car, exhibition hall, restaurant, street, airport, and train station. This data was recorded at 25kHz, 16-bit and digitally downsampled to 8kHz. Furthermore, the same data is duplicated across all scenarios for varying SNR values: 0dB, 5dB, 10dB, and 15B SNR, as well as a "clean" set with no added noise. This data was used to provide robustness across many environments and contexts.

The KitchenSound dataset consists of 22 different kitchen sounds repeated 20 times. The data was recorded at 44.1kHz and 16-bit audio. The UrbanSound dataset consists of 8 different urban sound environments such as: jackhammer, siren, car horn, and air conditioner. This data was also recorded at 44.1kHz and 16-bit audio. None of these files contain human voice, which makes this an excellent dataset for training against false positives. While our primary classification goal is to minimize the number of false negatives, we would also like to not waste power by turning the classifier on for no reason.

Lastly, all data is available in its sampled and uncompressed form. By ensuring this, we can hope to avoid any inaccuracies from audio noise or distortion due to compression. Of course, since all the data is sampled and therefore digital, it will be difficult to get a true approximation of the analog classifier's performance. It is our assumption that the accuracy of the 16-bit audio recorders is much larger than the accuracy of an implemented analog classifier, so that way we have established an accuracy ceiling.

B. Training

The classifiers were built in Python using the `scikit-learn` [10] and `Scipy` [11] packages. An initial `DecisionTreeClassifier` was trained using approximately 100k window frames of data. While this was not all of the available data to train on, it represented a reasonably large fraction of the data. The dataset was equally balanced into positive and negative classes in order avoid balancing issues later on.

Each training file had the relevant class generated through a simple thresholding method. The TIMIT dataset only consists of human voice, so we can simply check the mean-squared value of the frame-window compared to a predetermined threshold. An example of this training labeling is shown in Fig. 7. For the NOIZEUS dataset, simply thresholding isn't valid for the high SNR cases, as the noise will appear above the threshold. For this dataset, a look-up table of references was created based on the "clean" files, then it was referenced by the high SNR files.

When training the classifier, care was taken to shuffle the input data, as to prevent any unintentional bias or overfitting through training order. The training data was broken into an $80/20$ split of training and cross-validation samples. This was also split in a way that gave balanced classes across both datasets. We picked the tree that gave the highest floating

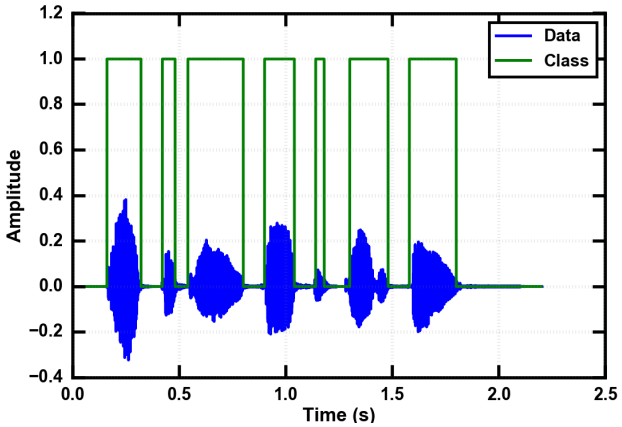

Fig. 7. An example audio clip with the labeling overlaid on top of it.

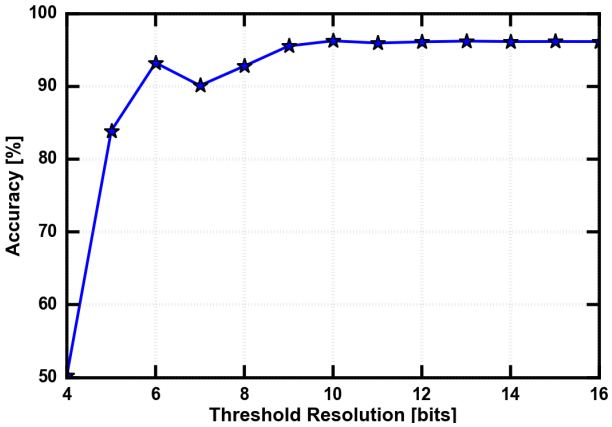

Fig. 8. Decision tree threshold resolution using ideal data compared to decision accuracy.

point accuracy across 10 cross-validation iterations as our baseline. From there, we constructed quantized decision trees by quantizing every threshold of the decision tree.

IV. Simulation Results

The test dataset consisted of the same items for all results shown below. For our negative classes, we tested on 10 UrbanSound files (one from each class) and three KitchenSound test files. The KitchenSound test files are each two minutes long and hold one example of every sound trained on. For our positive results, we tested using the `airport`, `babble`, and `car` environments in the NOIZEUS dataset. We note that the human voice data for all environments is the same in the NOIZEUS dataset, so our classifier may have lower accuracy when compared to unseen audio data.

A. Threshold Quantization

For both classifiers, the first significant metric is the accuracy in the decision thresholds. Whether these are analog precisions coming from a DAC or digital word sizes in a DSP core, the resolution compared to the input data is important metric. For this test, we only use the ideal floating point data to simulate uncompressed inputs. These results are shown in Fig. 8.

We obtain a minimum accuracy of 90% after using at least six bits of resolution. This suggests that both systems can reasonably expect to use low-precision hardware without large sacrifices in data performance. Additionally, it shows that high-precision hardware does not necessarily lead to higher accuracy rates. Beyond 10 bits, the accuracy varies by less than 1%, which is not significant enough to record. Our initial assumption is that the classifier is primarily dominated by SNR, as here the quantization noise scales exponentially with resolution.

B. Noise Performance

Instead of comparing accuracy here, we instead compare the F1 score of our classifier. The F1 score is defined based on the metrics *precision* and *recall*:

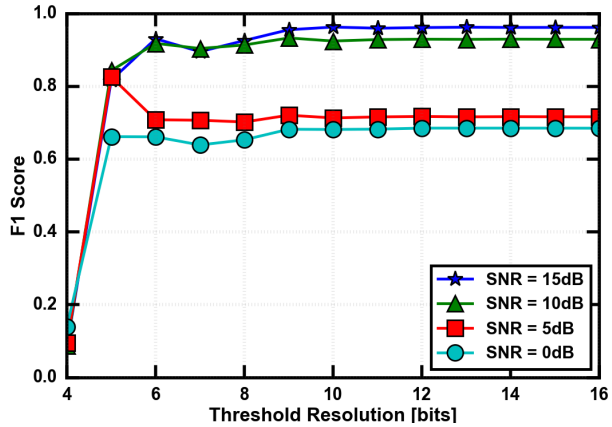

Fig. 9. Decision tree threshold resolution compared to F1 score as a function of input data SNR.

$$\text{Precision} = \frac{tp}{tp + fp} \tag{2}$$

$$\text{Recall} = \frac{tp}{tp + fn} \tag{3}$$

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4}$$

In order to validate our previous guess of noise being the dominant factor, we swap from the ideal data set to the NOIZEUS dataset and test it across all available SNR values. These results are shown in Fig. 9. We suspect there is an error in the 5-bit resolution tree, and ignore it as an outlier. Following that, it's clear that SNR of our does indeed directly affect our output accuracy. However, it's not clear if the fall-off from the previous test is due to quantization noise or SNR, we perform another test.

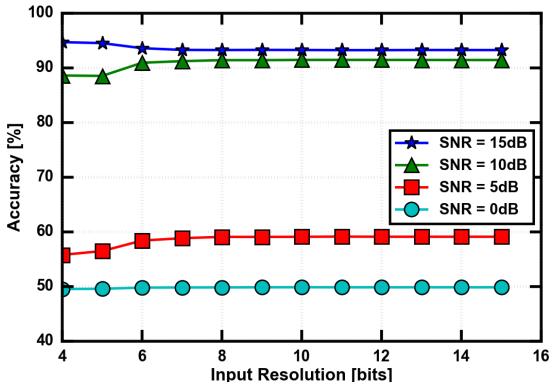

Fig. 10. Input resolution compared to

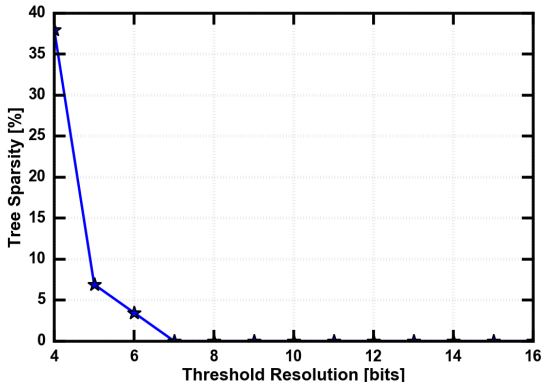

Fig. 11. Viewing data sparsity as a function of threshold quantization

C. Input Quantization

Here, we are interested in testing our input quantization effects. This is for implementation with the digital system, where we are sub-sampling the precision of the ADC. For this test, the accuracy of the thresholds is double the accuracy of the input quantization. This ensures the threshold can always be between two input quantization labels, assuming we maintain the same precision for our digital operations (filtering and integrating).

This confirms that the primary reason for noise was due to the input SNR, not any artifacts of input quantization. The magnitude of this result is surprising and certainly merits more testing.

D. Sparsity

Lastly, we test the sparsity of our tree classifier. We'll define the sparsity S of our classifier as:

$$S = \sum w_{tree} = 0 \tag{5}$$

Which is to say, the number of "zeros" in our tree nodes. We know that each energy value will be positive during the feature window, so we can safely say a zero node has no impact on the tree. We would like to minimize the sparsity of the tree, as will result in extra hardware that serves no function. Our results for sparsity as a function of threshold quantization are shown in Fig. 11.

This is where the maximum tree depth of 5 came from. Any deeper tree resulted in significantly higher sparsity, as more nodes did not provide any more accuracy after a certain depth.

V. Conclusion

In conclusion, we present two minimal, efficient voice classifiers that are able to maintain reasonable voice activity detection numbers within a certain SNR bound. An analog system only requires six bits of precision to maintain 90% detection accuracy assuming at least 10dB of SNR. A digital system can use fractions of the primary ADC's range while also maintaining similar levels of accuracy. We hope this work will continue to motivate explorations into low-power intelligent systems.

References

[1] K. M. H. Badami, S. Lauwereins, W. Meert, and M. Verhelst, "A 90 nm CMOS, 6uW Power-Proportional Acoustic Sensing Frontend for Voice Activity Detection," *IEEE Journal of Solid-State Circuits*, vol. PP, no. 99, pp. 1–12, 2015.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016.

[3] M. Verhelst and A. Bahai, "Where Analog Meets Digital: Analog-to-Information Conversion and Beyond," *IEEE Solid-State Circuits Magazine*, vol. 7, no. 3, pp. 67–80, Summer 2015.

[4] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pp. 365–376, June 2011.

[5] M. Yip and A. P. Chandrakasan, "A Resolution-Reconfigurable 5-to-10-Bit 0.4-to-1 V Power Scalable SAR ADC for Sensor Applications," *IEEE Journal of Solid-State Circuits*, vol. 48, pp. 1453–1464, June 2013.

[6] J. Garofolo, "TIMIT Acoustic-Phonetic Continuous Speech Corpus - Linguistic Data Consortium." https://catalog.ldc.upenn.edu/ldc93s1, 1993. Garofolo, John, et al. TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1. Web Download. Philadelphia: Linguistic Data Consortium, 1993.

[7] Y. Hu and P. C. Loizou, "Subjective Comparison of Speech Enhancement Algorithms," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, vol. 1, pp. I–I, May 2006.

[8] J. A. Stork, L. Spinello, J. Silva, and K. O. Arras, "Audio-based human activity recognition using non-markovian ensemble voting," in *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pp. 509–514, IEEE, 2012.

[9] J. Salamon, C. Jacoby, and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research," pp. 1041–1044, ACM Press, 2014.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and others, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[11] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science Engineering*, vol. 13, pp. 22–30, Mar. 2011.