DSP-Inspired Deep Learning: A Case Study Using Ramanujan Subspaces

Srikanth V. Tenneti*
Amazon Web Services
Palo Alto, California 94303
Email: stenneti@amazon.com

P. P. Vaidyanathan
Department of EE, Caltech
Pasadena, California 91125
Email: ppvnath@systems.caltech.edu

***Abstract*—Can Deep Learning be used to augment DSP techniques? Algorithms in DSP are typically developed starting from a mathematical model of an application. In some cases however, simplicity of the model can result in deterioration of performance when there is a severe modeling mis-match. This paper explores the idea of implementing a DSP technique as a computational graph, so that hundreds of parameters can jointly be trained to adapt to any given dataset. Using the specific example of period estimation by Ramanujan Subspaces, significant improvement in estimation accuracies under high noise and very short datalengths is demonstrated.** [1]

***Keywords*—Ramanujan Sums, Ramanujan Subspaces, Deep Learning, Periodicity, Computational Graphs.**

I. Introduction

In data science, applications involving parameter estimation can broadly be classified into two categories. The first consists of situations where the underlying mechanics of the problem can be modeled reasonably well mathematically. Examples include Direction of Arrival estimation in RADAR [10], analysis of bio-medical metrics such as heart-rate from ECG, pitch estimation in speech [3], detection of gravitation waves [1], and so on. For such problems, DSP tools and techniques have proven successful for more than six decades now.

On the other hand, there is a second category of applications where the underlying mechanics is practically impossible to capture using explicit mathematical models. These include applications such as credit score prediction, navigation strategies in self-driving cars, emotion characterization from facial images, and so on. The exceptional growth in computational and data storage technology in the last decade has allowed us to address several such rather abstract estimation applications using a new generation of Machine Learning (ML) and AI algorithms [2], [5].

Our hypothesis in this paper is as follows: Although many parameter estimation problems easily fall under one of the above two categories, there exists a large class of applications where combining DSP and ML can lead to very beneficial results. For instance, in DSP we often start with some simplifying assumptions on the data, such as the noise being additive, the signal being perfectly sinusoidal or periodic, i.i.d. noise samples, availability of *long enough* data length, etc., for the sake of analytical tractability. As long as these modeling assumptions are reasonably valid, we can devise optimal algorithms (such as say, maximum likelihood estimates) for many applications. But what if we need to use a DSP model in a situation where such modeling assumptions turn out to be too simplistic?

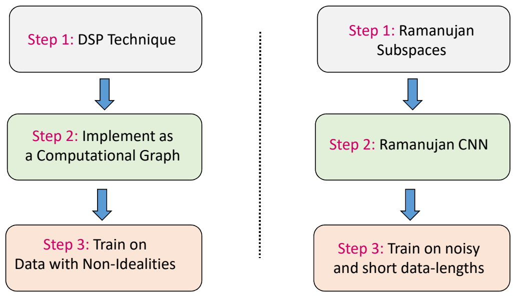
Step 1: DSP Technique
Step 2: Implement as a Computational Graph
Step 3: Train on Data with Non-Idealities
Step 1: Ramanujan Subspaces
Step 2: Ramanujan CNN
Step 3: Train on noisy and short data-lengths

Fig. 1. (Left) The proposed framework to use deep learning for training DSP techniques for non-ideal datasets. (Right) The specific context of Ramanujan Subspaces demonstrated in this paper.

For instance, a new family of period estimation techniques were recently proposed based on number theoretic sequences known as Ramanujan Sums [8], [23]. These included compressed sensing based dictionaries [13], filterbanks [17], [21] and eigen-space approaches [14]. These methods were shown to outperform several existing techniques for applications involving integer periods, such as in DNA microsatellites [18], protein repeats [19], absence seizure detection [20], brain-machine interfacing [9] and so on. Nevertheless, one of the main challenges with these Ramanujan sums based methods is that, the theoretical framework on which they were designed assumes long enough datalength [15] or negligible noise. The performance of these techniques deteriorates significantly when the data becomes severely corrupted. For example, large extents of insertion-deletions, and colored, non-stationary or signal-dependent noise in the data can degrade their estimation accuracy. On the other hand, mathematically adapting these techniques for each new type of data non-ideality is practically impossible.

Motivated by this, we propose the following general approach (Fig. 1): Start with a DSP technique, which possibly has modeling mismatches with the data at hand. Implement this DSP technique as a trainable Machine Learning model. This model can then be trained in a supervised manner on data with non-idealities, so that all its parameters are automatically tailored for that specific data.

As one can imagine, if such an approach is indeed feasible, it can offer several advantages over both plain DSP, and using arbitrary machine learning architectures. The former is due to adaptation of the algorithm to data non-idealities. The latter is because, as opposed to arbitrary machine learning models, we are already starting from an initialization that works to a certain extent, and just needs fine-tuning. This paper demonstrates this idea using the specific case of period estimation using Ramanujan Sums. It will be shown that, for highly noisy and

* This work was done when Srikanth V. Tenneti was a Post-Doctoral Researcher at California Institute of Technology, prior to joining Amazon Web Services.

[1]This work was supported in parts by the NSF grant CCF-1712633, the ONR grant N00014-18-1-2390, and an Amazon AI fellowship facilitated through IST, California Institute of Technology.

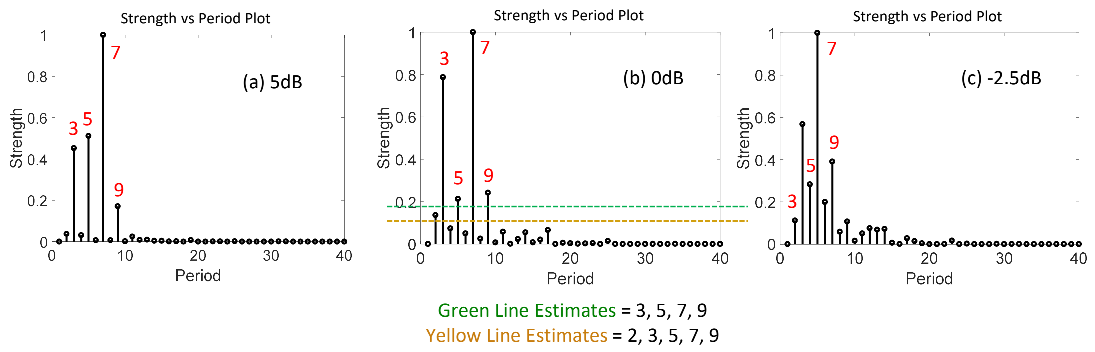

Fig. 2. Strength vs Period Plots. See Sec. II for details.

very short signal lengths, such a hybrid DSP-ML model offers remarkable advantages. Before we proceed, it is worth noting that different ways of bridging together DSP knowledge and machine learning models is quickly gaining popularity across many other domains as well, for e.g. [4], [7] and [11].

Paper Outline: Sec. II presents a brief overview of period estimation using Ramanujan Subspaces. A motivation of how machine learning can be incorporated with Ramanujan subspaces is also discussed. Sec. III develops the structure of a computational graph to implement Ramanujan Subspaces. Sec. IV includes multiple experimental results and their discussion.

II. Problem Motivation

In 1918, the Indian mathematician Ramanujan proposed the following sequences known as Ramanujan sums, for every integer $q > 0$:

$$c_q(n) = \sum_{\substack{k=1 \\ \gcd(k,q)=1}}^{q} e^{j2\pi kn/q} \tag{1}$$

where gcd is the greatest common divisor. It can be shown that $c_q(n)$ has period q, and is integer valued for all n [8]. In this work, by periodicity, we mean signals of the form:

$$x(n+P) = x(n) \qquad \forall\, n \in \mathbb{Z} \tag{2}$$

Here, the integer P is known as a *repetition index* of $x(n)$, and the smallest positive repetition index is known as the *period*.

Following are some examples of Ramanujan sums, with one period shown in each case:

$$\begin{aligned} &c_1(n) = 1,\ c_2(n) = \{1, -1\},\ c_3(n) = \{2, -1, -1\}, \\ &c_4(n) = \{2, 0, -2, 0\},\ c_5(n) = \{4, -1, -1, -1, -1\} \end{aligned} \tag{3}$$

Ramanujan originally proposed these sequences to show that several arithmetic functions in number theory, such as the Euler Totient function, Von Mangoldt function etc. can be expanded in terms of the Ramanujan sums. More recently in [23], these sequences were shown to have many useful properties in the context of periodic signals. For instance, [23] generalizes (1) to an entire subspace known as a Ramanujan Subspace as follows:

$$\mathcal{S}_q = \text{span}\{e^{j2\pi kn/q} :\ \gcd(k,q) = 1\} \tag{4}$$

The Ramanujan subspaces are orthogonal for different q [22], [23]. Moreover, they satisfy the following (see Theorem 12 in [22]):

Theorem 1. The LCM property *let P be the period of a signal $x(n)$. If $P_1, P_2, \ldots, P_N$ are the indices of Ramanujan subspaces on which $x(n)$ has non-zero projections, then:*

$$P = \text{lcm}(P_1, P_2, \ldots, P_N) \tag{5}$$

While theoretically, the above is true only for infinitely long noise-free signals, it is still approximately true for finite duration signals with mild noise. For instance Fig. 2(a) plots the projection energies vs the index of Ramanujan subspaces for a signal of length 100 samples. The signal was generated by adding randomly generated signals with periods 5, 7 and 9, so that the net period is $5 \times 7 \times 9 = 315$. The true periods are clearly seen in Fig. 2(a). We will refer to such plots as **Strength vs Period plots** in the rest of the paper. In the applications mentioned in Sec. I, such strength vs period plots using various Ramanujan Subspaces based algorithms [13], [17] were shown to outperform DFT-based and other currently used techniques.

A. Why Machine Learning?

As long as the signal is reasonably ideal (low noise, large enough data-length etc.), the strength vs period plot is clean as in Fig. 2(a). But Fig. 2(b) shows the strength vs period plot for the same signal when the SNR (with additive Gaussian noise) decreases to 0dB. Notice that the period estimate in this case depends critically on how we pick a threshold that distinguishes the true peaks from spurious ones due to noise. The green line gives the correct period estimates while the yellow line estimates a spurious peak at period 2. How then does one choose an optimal threshold?

As the noise increases further, there may in fact be no way to pick a constant threshold that separates true peaks from the spurious ones. Fig. 2(c) shows an example. The spurious peak at period 4 is larger than the true peak at 3. But notice that we only need to identify the true peaks so that the LCM property can be used. Can we somehow generalize the LCM property so that we do not have to explicitly separate the true peaks from the spurious peaks in the strength vs period plot? As one can imagine, this idea is almost impractical to explore theoretically[2].

This is where Machine Learning seems very promising. The hope is that, if we can implement Ramanujan Subspacess as a trainable model, then the thresholding operations, the desired generalization of the LCM property etc. can automatically be *learned* from the data. Not just that, the projection matrices and the basis vectors of the Ramanujan subspaces themselves can be adapted to the particular dataset. This not only avoids the need to theoretically reanalyze each new dataset, but also avoids using any explicit modeling assumptions for the non-idealities.

Does this idea actually work? The following sections reveal interesting results in this regard. While we limit this paper's scope to the case of additive noise and small data-lengths, one can easily use the same idea for datasets having other

[2]The LCM property is quite involved theoretically [22], [23].

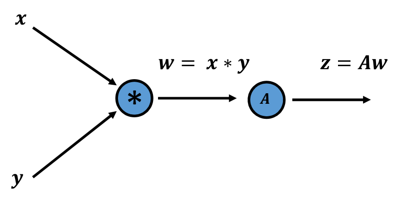

Fig. 3. A simple example of a computational graph depicting the operation $\mathbf{A} \times (\mathbf{x} * \mathbf{y})$. Here $\mathbf{A}$ is a fixed (not trained) matrix, $\mathbf{x}$ for instance could be the input vector, and $\mathbf{y}$ a filter whose coefficients are to be trained. See Sec. III.

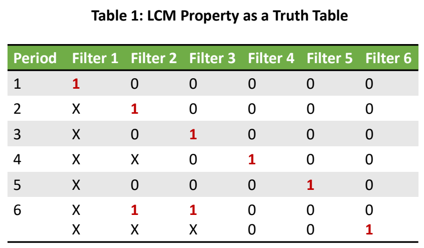

Table 1: LCM Property as a Truth Table

Period	Filter 1	Filter 2	Filter 3	Filter 4	Filter 5	Filter 6
1	1	0	0	0	0	0
2	X	1	0	0	0	0
3	X	0	1	0	0	0
4	X	X	0	1	0	0
5	X	0	0	0	1	0
6	X	1	1	0	0	0
	X	X	X	0	0	1

non-idealities such as insertion deletion errors, time varying amplitudes, changing periodicity, non-integer periods etc.

III. Ramanujan Subspace as a Machine Learning Model

Which ML model is ideally suited for implementing Ramanujan Subspaces? In our experience, computational graphs are well suited, not just here, but for implementing a number of signal processing methods.

A. *Computational graphs for implementing DSP algorithms*

Most signal processing techniques can be viewed as a sequence of operations/transformations from an input signal to the output. In deep learning literature, a computational graph is essentially a directed graph representing sequences of operations. A simple example is shown in Fig.3. Each edge represents an intermediary variable that occurs in the computational pipeline between the input and the output variables. Each node in the graph represents a piecewise differentiable operation on variables entering that node. The most important property of a computational graph is that it is amenable to parameter training using backpropagation through the graph. As is evident, such a model is flexible enough to capture the steps of many signal processing techniques. The standard multi-layer neural networks and convolutional neural networks (CNNs) are just two examples of computational graphs. In the following, we will design a CNN-like computational graph in a layer-by-layer fashion to implement the period estimation pipeline using Ramanujan Subspaces.

We will start by assuming that the input signal is a $D \times 1$ vector, where D is the datalength of the signal. We further assume that the input's period is known to be $\leq P_{max}$ for some large enough P_{max}. Specifically in our simulations in Sec. IV, we consider $P_{max} = 15$ for demonstration.

B. *Convolutional Layer*

Our first step is to project the $D \times 1$ input signal onto the Ramanujan Subspaces (Fig. 4). It turns out that the projection matrices for Ramanujan Subspaces are in fact circulant, meaning that they can be implemented using convolutions. This is proved in [22], [23], [21]. The impulse responses of the required filters turn out to be just truncated version of Ramanujan sums [17], [21]. We would need P_{max} filters in this layer, one for every Ramanujan Subspace (shown in Fig. 4). The output of each filter is a $D \times 1$ vector corresponding to the projection vector.

C. *Square Activation and Average Pooling Layers*

The next step is to compute the energies in the projection vectors from the previous layer. This can be achieved using the following two layers:

1) Square Activation Layer: Computing point-wise squares of the outputs of the conv layer. The resulting vectors are still P_{max} in number, each of size $D \times 1$.
2) Average Pooling: Computes the average of each $D \times 1$ vector from the square activation layer.

Notice that plotting the outputs of the Average Pooling layer would essentially give us a strength vs period plot similar to Fig. 2(a). The LCM of those periods that have a large energy must be the estimate of the period. Our subsequent layers must hence be built to identify peaks in the strength vs period plot and computing the LCM.

D. *Neural Layers: LCM computation as a Truth Table*

Determining period from the strength vs period plot using the LCM property (Theorem 1) can be represented as a truth table. Table 1 illustrates the idea for the simple case of $P_{max} = 6$. A '1' in the table indicates that the corresponding conv layer filter has a significant peak in the strength vs period plot, and '0' indicates otherwise. An 'X' indicates that the state of that filter output does not matter in the LCM computation. For instance, if filter 4 has a peak, then the LCM is unaffected by whether filter 2 has a peak or not. Notice that some periods, such as 6, can occur in multiple ways.

Table 1 can be used to write logic expressions for indicator variables for each possible period. For instance, consider the following:

$$P_6 = F_2.F_3.\overline{F_4}.\overline{F_5}.\overline{F_6} + \overline{F_4}.\overline{F_5}.F_6 \tag{6}$$

Here '.' is the logical AND, and '+' the logical OR operation. P_6 will be 1 only when the LCM of the significant peaks is 6, i.e, when the input's period is 6. Using similar indicator variables for each period in the range $1 \leq P \leq P_{max}$, we can estimate the input's period from the strength vs period plot.

Furthermore, it is well known that any truth table can be implemented as a two layer neural network. For example, the first term in the R.H.S. of (6) can be implemented as a neuron:

$$t_1 = \text{sign}(f_2 + f_3 + (1 - f_4) + (1 - f_5) + (1 - f_6) - 4.5) \tag{7}$$

where each f_i is a real number with value 1 or 0 corresponding to the binary variable F_i. The second term in (6) similarly is:

$$t_2 = \text{sign}((1 - f_4) + (1 - f_5) + f_6 - 2.5) \tag{8}$$

and they can be combined using a third neuron as:

$$p_6 = \text{sign}(t_1 + t_2 - 0.5) \tag{9}$$

For trainability of the previous layers using back-propagation, we need to replace each $\text{sign}(\cdot)$ in the above expressions with a sigmoidal function $\sigma(\gamma(\cdot))$, where γ can be a trainable parameter. It can be shown that for our case of $P_{max} = 15$, the first LCM layer needs 21 neurons, and the second 15, as indicated in Fig. 4.

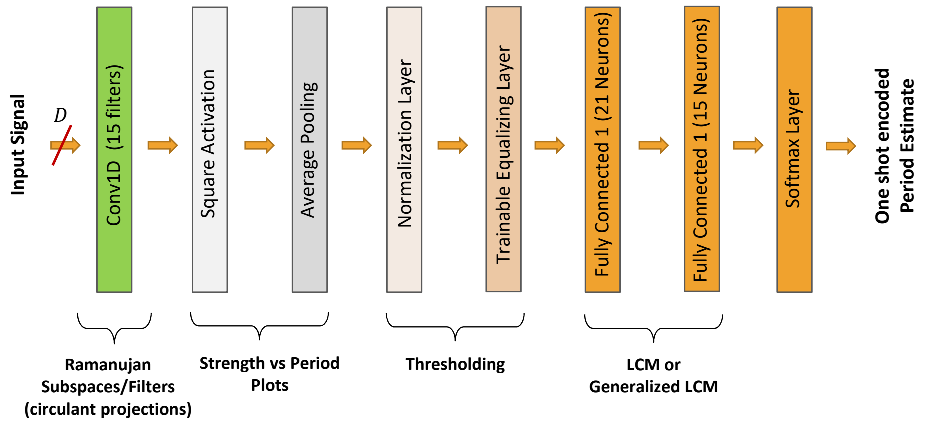
Input Signal
D
Conv1D (15 filters)
Square Activation
Average Pooling
Normalization Layer
Trainable Equalizing Layer
Fully Connected 1 (21 Neurons)
Fully Connected 1 (15 Neurons)
Softmax Layer
One shot encoded Period Estimate
Ramanujan Subspaces/Filters (circulant projections)
Strength vs Period Plots
Thresholding
LCM or Generalized LCM

Fig. 4. The handcrafted Ramanujan Computational Graph. See Sec. III for details.

Can we directly connect these neural layers at the output of the average pooling layer? Unfortunately this does not work. This is because, these neural layers as designed above expect the inputs to be exactly either 1 or 0. In a typical strength vs period plot (Fig. 2(a)), all the detected component periods may not be equal to 1. Similarly, there may be a small noise floor for the non-periods, so that they are not exactly 0. To take care of this, we need two additional operations as follows.

E. *Normalizing and Thresholding Layers*

Our first operation is normalizing the strength vs period plot. This can be done by a layer which, for every channel i, computes:

$$\bar{x}_i = \frac{x_i}{\max_{1\leq j\leq P_{max}} x_j} \tag{10}$$

Here x_i is the output of the i^{th} average pooling operation. This ensures that all the resulting outputs are in the range $[0, 1]$. However, notice in Fig 2(a) that not all component periods have the same strength. We need an additional operation that thresholds all peaks above a certain threshold to 1, and everything else to 0. This can be implemented as:

$$\sigma(\alpha(\bar{x}_i - \beta)) \tag{11}$$

where $\sigma(\cdot)$ is the sigmoidal function as before and α is a trainable scale parameter. The parameter β is exactly the thresholding operation used to identify the peaks from non-peaks in Fig. 2, which we can now train from the data. Once we add these layers after the average pooling layer, we can then connect the two-layer neural network as discussed above. We finally add a softmax layer at the end of the neural layers. Notice that the way we have designed this structure, the period estimate is one-shot encoded at the output of the softmax layer. That is, for each estimate of the period, there is one correspondding output of the softmax layer that attains value 1, while all other outputs are 0. This makes categorical cross entropy an appropriate loss function to minimize while training. Cross entropy, which is a popular metric from Information Theory to quantify distances between probability distributions [12], has gained wide popularity in machine learning for categorical classification applications [2], [5].

IV. Experiments and Discussion

We will divide our experiments into three categories:

A. *Very small datalength and low SNR*

Table 2 shows the performance of the architecture developed in the previous section. The SNR is low (0dB), and the data length D is quite small, varied between 30 and 50 samples. The training, validation and test data consisted of randomly generated periodic signals with the indicated datalength values, with periods in the range 1 to 15. These datasets also included sums of periodic signals, for example period 3 + period 4, whose resulting periods are ≤ 15. There were 5000 synthetic training signals for each combination of periods. The percentages shown in Table 2 are the fraction of test data for which the period was correctly estimated.

Ram UT refers to the untrained (UT) Ramanujan CNN developed in the previous section. **Ram T** is the same, but after training. **Random UT** and **Random T** are the same architecture that was developed in the previous section, but with randomly initialized weights for all the layers instead of carefully chosen values as in Ram UT. Random T can be interpreted as a ML model, whose architecture is fully DSP inspired, but weights are random. The training was performed using back-propagation (stochastic gradient descent implemented using ADAM [6]). For both Ram T and Random T, the model with the best validation error was picked at the completion of the training process.

As evident from Table 2, this experiment clearly establishes the main message of this paper: Combining DSP and Deep Learning can result in remarkable benefits. Ram T (the hybrid DSP/ML) shows that the percentage of success almost triples over Ram UT (just DSP). The Random UT also seems to be training to reasonably good accuracies. What subspaces do the conv layer of Random T represent? Do they imitate Ramanujan Subspaces? This direction of investigation leads to some interesting observations, which we will present elsewhere due to space limitations in this paper.

B. *Good Datalengths and SNR*

Table 3 considers a regime where the design assumptions of Ramanujan Subspace models are reasonably accurate. The datalength D is 200 samples, and the SNR is varied between 2.5 to 10 dB. Ram UT performs quite well especially at 5 and 10 dB, since its modeling assumptions match the data quite well. It can be seen that as the SNR drops to 2.5 dB, the training process does improve the performance of Ram T. It is also very interesting to see that Random T can also reach comparable performances, although starting from a random initializations.

Table 2: Small Datalengths, SNR 0dB

Datalength	Ram UT	Ram T	Random UT	Random T
50	29.5%	**88%**	4%	**85%**
45	27.9%	**87%**	4%	**84%**
30	21.6%	**78%**	4%	**73%**
		(**Hybrid DSP/ML**)		(**Hybrid DSP/ML**)

Table 3: Datalength 200, relatively high SNRs

SNR	Ram UT	Ram T	Random UT	Random T
10	98.1%	**98.1%**	4.7%	**94%**
5	96.5%	**96.5%**	4.7%	**94%**
2.5	89.6%	**95.5%**	4.7%	**93.5%**
		(**Hybrid DSP/ML**)		(**Hybrid DSP/ML**)

C. Comparison to other ML models

What if we directly use machine learning models, which are not necessarily inspired from DSP? Table 4 shows some arbitrarily chosen architectures of 2 and 3 layered neural networks (indicated as NN's, along with the number of neurons in each layer), and a CNN with a perturbed architecture than Ram T[3]. While our hybrid DSP/ML model Ram T does outperform the models shown here, and also several other models that we tested in our experiments, we must emphasize the following.

We are not claiming that there doesn't exist any ML model that outperforms Ram T. Rather, it is highly non-obvious how one can find such an architecture. More generally, how does one come up with a principled way of choosing a Machine Learning model for a given estimation problem? The approach proposed in this paper, of building computational graphs implementing DSP algorithms, and using them as initializations in deep learning seems like a promising step in this direction.

V. Concluding Remarks

This paper aims to demonstrate the benefits of combining the tools and techniques of DSP with the versatile trainability of deep learning architectures. A simple case study using Ramanujan Subspaces was used as an example. We postulate that a similar approach for other DSP algorithms will also result in remarkable performance improvements over datasets that were traditionally considered *too non-ideal* in DSP. Furthermore with the help of DSP, one might even obtain good insights on what ML architecture to use, the minimum architecture size (number of layers, number of neurons per layer) required etc., for a particular application. For e.g., [16] explores the minimum number of convolutional filters needed to identify periodicity in data. Such insights are quite rare in the world of deep learning today, and insights from DSP might be promising in this regard in the coming years [4], [11], [7].

Table 4: Comparison with other ML models

Method	Accuracy
Ram T (15, 21, 15) **(Ours)**	88.0 %
Multiclass Logistic Regression	16.2 %
2 layer NN (100, 15)	79.8 %
2 layer NN (200, 15)	80.4 %
3 layer NN (15, 15, 15)	45.0 %
Arbitrary CNN (15, 42, 15)	14.3%

References

[1] Abott et al, "GW170817: Observation of Gravitational Waves from a Binary Neutron Star Inspiral", Physical Review Letters, vol. 119, no. 16, Oct 2017.

[2] Y. S. Abu-Mostafa, M. M.-Ismail, H.-T. Lin, *Learning From Data*, AMLbook, 2012.

[3] A. de Cheveigna and H. Kawahara, "YIN, a fundamental frequency estimator for speech and music", J. Acoust. Soc. of America, vol. 111, pp. 1917-30, 2002.

[4] Demba Da, "Deeply-Sparse Signal rePresentations (DS^2P)", arXiv:1807.01958.

[5] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, The MIT Press, 2016.

[6] D. P. Kingma, J. Ba, "Adam: A Method for Stochastic Optimization", arXiv:1412.6980.

[7] A. B. Patel, T. Nguyen and R. G. Baraniuk, "A Probabilistic Theory of Deep Learning", arXiv:1504.00641.

[8] S. Ramanujan, "On certain trigonometrical sums and their applications in the theory of numbers, Trans. of the Cambridge Phil. Soc., vol. XXII, no. 13, pp. 259-276, 1918.

[9] P. Saidi, G. Atia and A. Vosoughi, "Detection of Visual Evoked Potentials using Ramanujan Periodicity Transform for real time brain computer interfaces," IEEE ICASSP, New Orleans, 2017.

[10] R. O. Schmidt, "Multiple emitter location and signal parameter estimation," in Proc. RADC Spectral Est. Workshop, 1979, pp. 243- 258.

[11] J. Sulam, V. Papyan, Y. Romano and M. Elad, "Multi-Layer Convolutional Sparse Modeling: Pursuit and Dictionary Learning", arXiv:1708.08705v2.

[12] J. A. Thomas and T. M. Cover, *Elements of Information Theory*, John Wiley and Sons, NJ, 2006.

[13] S. Tenneti and P. P. Vaidyanathan, "Nested Periodic Matrices and Dictionaries: New Signal Representations for Period Estimation," IEEE Trans. Sig. Proc., vol. 63, no. 14, pp. 3736-50, July, 2015.

[14] S. V. Tenneti and P. P. Vaidyanathan, "iMUSIC: A Family of MUSIC-Like Algorithms for Integer Period Estimation," IEEE Trans. Sig. Proc., vol. 67, no. 2, pp. 367-382, Jan 2019.

[15] S. V. Tenneti and P. P. Vaidyanathan, "Minimum Data Length for Integer Period Estimation," IEEE Trans. Sig. Proc., vol. 66, no. 10, pp. 2733-2745, May, 2018.

[16] S. V. Tenneti and P. P. Vaidyanathan, "Period estimation and tracking: Filter bank design using truth tables of logic," Asilomar Conf. on Signals, Systems and Computers, USA, 2015.

[17] S. Tenneti and Vaidyanathan P.P., "Ramanujan Filter Banks for Estimation and Tracking of Periodicities", Proc. IEEE Int. Conf. on Acoust., Speech, and Sig. Proc., 2015.

[18] S. Tenneti and P. P. Vaidyanathan, "Detecting Tandem Repeats in DNA Using the Ramanujan Filter Bank", Proc. IEEE Int. Symp. on Circuits and Systems, Canada, 2016.

[19] S. Tenneti and P. P. Vaidyanathan, "Detection of Protein Repeats Using Ramanujan Filter Bank", Asilomar Conference on Signals, Systems and Computers, USA, 2016.

[20] S. Tenneti and P. P. Vaidyanathan, "Absence Seizure Detection Using Ramanujan Filter Banks," Asilomar Conf. on Signals, Systems, and Computers, USA, 2018, pp. 1913-1917.

[21] P. P. Vaidyanathan and S. Tenneti, "Properties of Ramanujan Filter Banks", European Signal Processing Conference, 2015.

[22] P. P. Vaidyanathan, "Ramanujan sums in the context of signal processing: Part I: Fundamentals" IEEE Trans. Sig. Proc., Vol. 62, No. 16, pp. 4145-57, Aug. 2014.

[23] P. P. Vaidyanathan, "Ramanujan sums in the context of signal processing: Part II: FIR representations and applications" IEEE Trans. Sig. Proc., Vol. 62, No. 16, pp. 4158-72, Aug. 2014.

[3]with a linear activation layer after the conv instead of the squaring, and no thresholding layer, but with 42 neurons as opposed to 21 in Ram T in the hidden neural layer.