Comparison of Neural Network Architectures for Spectrum Sensing

Ziyu Ye
Dept. ECE
UC San Diego
San Diego, United States
ziy076@kiwi-ml.ucsd.edu

Andrew Gilman
Sch. of Nat. and Comp. Sci.
Massey University
Auckland, New Zealand
A.Gilman@massey.ac.nz

Qihang Peng
Sch. of Info. and Comm. Eng.
UESTC
Chengdu, China
anniepqh@uestc.edu.cn

Kelly Levick
Dept. ECE
UC San Diego
San Diego, United States
klevick@ucsd.edu

Pamela Cosman
Dept. ECE
UC San Diego
San Diego, United States
pcosman@eng.ucsd.edu

Larry Milstein
Dept. ECE
UC San Diego
San Diego, United States
milstein@ece.ucsd.edu

***Abstract*—Different neural network (NN) architectures have different advantages. Convolutional neural networks (CNNs) achieved enormous success in computer vision, while recurrent neural networks (RNNs) gained popularity in speech recognition. It is not known which type of NN architecture is the best fit for classification of communication signals. In this work, we compare the behavior of fully-connected NN (FC), CNN, RNN, and bi-directional RNN (BiRNN) in a spectrum sensing task. The four NN architectures are compared on their detection performance, requirement of training data, computational complexity, and memory requirement. Given abundant training data and computational and memory resources, CNN, RNN, and BiRNN are shown to achieve similar performance. The performance of FC is worse than that of the other three types, except in the case where computational complexity is stringently limited.**

***Index Terms*—cognitive radio, spectrum sensing, neural network**

I. Introduction

A cognitive radio aims to exploit white space in existing licensed communication systems. Spectrum sensing, or detection of the white space, is a key focus of research in this field. Classic spectrum-sensing techniques, such as energy detection, matched-filter-based detection, cyclostationary-feature detection, and covariance-matrix-based detection were established decades ago. More advanced detection schemes such as cooperative sensing were proposed to further improve the performance by exploiting spatial, temporal and/or spectral correlations. A review of conventional spectrum sensing techniques can be found in [1]. More recently, machine learning has been applied in spectrum sensing [2]. Our previous work [3], [4] shows potential benefits of using artificial neural networks (NNs) in spectrum sensing, and reviews relevant literature.

This research was supported by the Data Science/Machine Learning Initiative of UC San Diego's ECE Dept.

Andrew Gilman was supported by Massey University Research Fund.

In this work, we address the choice of NN architectures for spectrum sensing. The two most well-studied NN architecture types today are the convolutional neural network (CNN) and the recurrent neural network (RNN). CNNs consist of stacked shift-invariant local filters (kernels) and are particularly effective in capturing spatially-local features arranged hierarchically. An RNN uses recurrent connections that allows it to extract and utilize empirical autocorrelations in sequential data. Modern RNNs are equipped with gated operations and are able to capture correlations across long intervals of time in the input sequence. The NN architectures have different levels of popularity in different tasks: Variants of CNN dominate in computer vision, while RNNs are widely used in speech recognition and natural language processing (NLP) .

Various examples of NN architecture comparison for specific applications can be found in the literature. In [5], [6], CNN and RNN are compared on NLP tasks. The authors of [5] report that bi-directional RNN (BiRNN) outperforms CNN in relation classification, while in [6], CNN and RNN each show advantages in different tasks/scenarios. Authors of both works attribute their observations to the intuition that CNN puts more emphasis on local features, while RNN is able to learn long-term dependencies. In [7], CNN and RNN are compared on environmental-sound-based scene classification. They found that RNN is less effective on average, presumably because of the lack of long-term correlations in the natural environmental sound. In [8] and [9], CNN and RNN are trained to predict stock price changes. Both works report CNN as the winner. In [10], RNN is found to be superior than CNN in detecting internet attacks based on payload data, and [11] finds that a fully-connected NN (FC) performs comparably to an RNN in a power grid identification task.

In spectrum sensing, a preference among the NN architectures has not yet been established. Communication signals have both, similarities to and differences from, the types of signals considered in typical applications of CNN and RNN. Similar to images and speech signals, a communication signal sampled in time consists of ordered and correlated samples. However, it may lack some common characteristics present in images and speech signals. Without assuming a specific transmitted data sequence (for example, a known pilot signal) or a specific channel-coding scheme, it is uncertain whether the communication signal has strong long-term correlations, or whether it contains strong local features, such as corners and edges in images. Like many other applications, the choice among NN architectures for spectrum sensing is not obvious and requires empirical comparisons.

This work aims to provide a preliminary performance comparison of CNN, RNN, BiRNN, and FC in spectrum sensing. The communication signal simulation setup and the performance metrics are explained in Section II. In Section III, we compare the best performance of each of the NN architecture types, and the computational complexity and memory requirements to achieve this performance. In Section IV, we compare performance of NN architectures under a computational complexity constraint. We summarize and conclude in Section V.

II. Communication Simulation and Performance Metrics

A. Simulation Setup

Simulation data generated with GNU Radio are used to train and test the NNs. Consider the following spectrum sensing task:

$$\boldsymbol{x} = \begin{cases} \boldsymbol{s} + \boldsymbol{n} & , \quad y = 1 \\ \boldsymbol{n} & , \quad y = 0 \end{cases} \tag{1}$$

where the complex vector $\boldsymbol{x}$ is the sampled low-pass equivalent of the received waveform in a sensing interval, and $\boldsymbol{s}$ and $\boldsymbol{n}$ are the signal and noise components of $\boldsymbol{x}$. $\boldsymbol{x}$ may either contain the signal component $\boldsymbol{s}$ or not, reflected by the label y. The received waveform is assumed to have been filtered by an ideal bandpass filter, whose passband matches the primary signal's bandwidth exactly. Parameters of the simulation are listed in Table I. The sampling rate is set beyond the Nyquist rate of the band-limited $\boldsymbol{x}$ so that no information loss is incurred by sampling. Equal numbers of busy ($y = 1$) and idle ($y = 0$) examples are generated. The dataset is divided into three parts: a training set containing 8E+06 examples, and validation and test sets, each containing 1E+05 examples. We train different types of NNs on the training set to predict label y from $\boldsymbol{x}$, after fine-tuning architecture and training hyperparameters on a validation set and compare final performance on the test set.

B. Performance Metrics

As suggested in [12], besides the detection performance, resource consumption such as computation and memory are also important NN performance aspects. We evaluate each NN's performance on four characteristics: detection performance, amount of training data required, forward-pass (inference) computational complexity, and forward-pass memory requirements.

TABLE I
Communication signal simulation parameters

Primary Signal	**Modulation**	QPSK
	Pulse Shape	root-raised-cosine (RRC) pulse with roll-off factor 0.35
	Source Data	random, uncorrelated bits
Noise	AWGN	
SNR	3dB	
Sampling Rate	10 times the symbol rate	
Sensing Duration	111 samples	

a) Detection Performance: The detection performance is reflected by the detection probability, P_d, which is the probability of correct decision conditional on ground truth $y = 1$, and the false alarm probability, P_{fa}, which is the probability of incorrect decision conditional on $y = 0$. In this work, we fix P_{fa} at 1% by choosing the classifier's threshold (on the validation set), and evaluate the detection performance using the false dismissal probability $P_{fd} = 1 - P_d$.

b) Amount of Training Data: In the real world, training data could be a precious resource, depending on the expense of data collection and labeling. The amount of training data could be limited, which may render NN models that require huge training sets impractical. To examine the influence of the training dataset size on performance, we train each NN not only on the entire 8E+06 training set, but also on subsets of size 1E+03 and 1E+05.

c) Computational Complexity: Computational complexity is closely related to energy consumption and decision latency, both of which are important for spectrum sensing. The operation count, which is the number of floating point operations (FLOPS) in one forward pass on a single input, is used as a performance metric. We ignore the computation cost associated with non-linear activation functions as it is negligible in comparison to the number of FLOPS in the linear operations. The operation counts of FC, CNN, RNN and BiRNN are listed below. The symbols are defined in Table II.

$$N_{op}^{(FC)} = \sum_{k=1}^{K} (2N_{k-1}N_k) + 2N_K N_{out} \tag{2}$$

$$\begin{aligned} N_{op}^{(CNN)} = & \sum_{k \in \mathcal{K}_{conv}} 2C_{k-1}C_k N_k^{(kernel)} N_k \\ & + \sum_{k \in \mathcal{K}_{pool}} C_k N_k (N_k^{(pool)} - 1) \\ & + \sum_{k \in \mathcal{K}_{bn}} 2C_k N_k \\ & + 2C_K N_K N_{dense} + 2N_{dense} N_{out} \end{aligned} \tag{3}$$

$$\begin{aligned} N_{op}^{(RNN)} = & L \cdot \sum_{k=1}^{K} (8(N_{k-1} + N_k)N_k + 4N_k) \\ & + 2N_K N_{out} \end{aligned} \tag{4}$$

$$N_{op}^{(BiRNN)} = 2L \cdot \sum_{k=1}^{K} \left(8(N_{k-1}+N_k)N_k + 4N_k\right) + 4N_K N_{out} \tag{5}$$

TABLE II
NOTATION

Symbol	Definition
K	Number of hidden layers
N_k	Layer width / feature vector length of the k-th hidden layer
N_0, N_{out}	Input and output size
N_{dense}	Size of the dense layer (CNN only)
$N_k^{(kernel)}$	Kernel size of the k-th hidden layer (CNN only)
$N_k^{(pool)}$	Pooling factor in the k-th hidden layer (CNN only)
C_k	Layer depth / number of filters (output channels) in the k-th hidden layer (CNN only)
$\mathcal{K}_{conv}$	Set of indices of convolutional layers (CNN only)
$\mathcal{K}_{pool}$	Set of indices of pooling layers (CNN only)
$\mathcal{K}_{bn}$	Set of indices of batch-normalization layers (CNN only)
L	Input sequence length (RNN only)

d) Memory Requirement: NN memory requirements depend heavily on the specific implementation. We consider two memory requirement metrics corresponding to two extreme cases. The peak instantaneous memory requirement, M_{peak}, reflects the peak memory requirement of the most memory-efficient implementation, which reallocates memory after each operation and holds only necesary content in memory at each moment. The total memory requirement, M_{total}, is the amount of memory that would be rquired if no reallocation happens and memory is pre-allocated for all parameters and intermediate states in advance. To avoid ambiguity, we assume that operations such as addition of bias term, nonlinear activation, and batch normalization are performed in place. We assume maximum parallelism, so operations in the same layer are executed in parallel. The unit of the memory metrics is the size of a floating point variable. The expressions for M_{peak} and M_{total} for FC, CNN, RNN, and BiRNN are listed below.

$$M_{peak}^{(FC)} = \max_k N_{k-1} \cdot N_k + 2N_k + N_{k-1} \tag{6}$$

$$\begin{aligned} M_{peak}^{(CNN)} &= \max\left\{M_{max}^{(conv)}, M_{max}^{(bn)}, M^{(dense)}\right\} \\ M_{max}^{(conv)} &= \max_{k\in\mathcal{K}_{conv}} \{C_{k-1}N_{k-1} + C_k N_k \\ &\quad + C_k(C_{k-1}N_k^{(kernel)} + 1)\} \\ M_{max}^{(bn)} &= \max_{k\in\mathcal{K}_{bn}} 3C_k N_k \\ M^{(dense)} &= C_K N_K N_{dense} + C_K N_K + 2N_{dense} \end{aligned} \tag{7}$$

$$M_{peak}^{(RNN)} = \max_k 4(N_{k-1}+N_k)N_k + N_{k-1} + 6N_k \tag{8}$$

$$M_{peak}^{(BiRNN)} = 2M_{max}^{(RNN)} \tag{9}$$

$$M_{total}^{(FC)} = N_0 + \left(\sum_{k=1}^{K}(N_{k-1}\cdot N_k + 2N_k)\right) + (N_K \cdot N_{out} + 2N_{out}) \tag{10}$$

$$\begin{aligned} M_{total}^{(CNN)} &= C_0 N_0 \\ &+ \sum_{k\in\mathcal{K}_{conv}} \left(C_k(C_{k-1}N_k^{(kernel)} + 1) + C_k N_k\right) \\ &+ \sum_{k\in\mathcal{K}_{pool}} (C_{k-1}N_{k-1} + C_k N_k) + \sum_{k\in\mathcal{K}_{bn}} 2C_k N_k \\ &+ (C_K N_K N_{dense} + 2N_{dense}) + (N_{dense}N_{out} + 2N_{out}) \end{aligned} \tag{11}$$

$$M_{total}^{(RNN)} = N_0 + \sum_k \left(4(N_{k-1}+N_k)N_k + 10N_k\right) + (N_K N_{out} + 2N_{out}) \tag{12}$$

$$M_{total}^{(BiRNN)} = N_0 + 2\sum_k \left(4(N_{k-1}+N_k)N_k + 10N_k\right) + (2N_K N_{out} + 2N_{out}) \tag{13}$$

III. COMPARISON OF OPTIMIZED NNS

As performance is multi-dimensional, ideally, the performance of each NN architecture type should be characterized by the Pareto frontier in the multi-dimensional space. To obtain one point on the Pareto frontier requires solving a constrained optimization problem, in which the training data size, computational complexity and memory requirement are fixed, and the architecture and learning hyperparameters are optimized to maximize the detection probability. Due to limited time and computational resources, instead of searching for the Pareto frontier, we resolved to the following protocol: For each of the four NN architecture types, the hyperparameters are manually tuned to maximize the detection probability on three training set sizes (1E+03, 1E+05, and 8E+06). Complexity and memory were ignored during this tuning process, but are compared on the best set of hyperparameters.

In tuning the hyperparameters, the following constraints are imposed. For CNN, we consider a type of architecture adopted from VGGNet [13]. The CNN consists of multiple homogeneous blocks in a sequence, followed by a single dense layer. Each block consists of two convolutional layers (kernel size=3, stride=1, padding='same' mode, ReLU activations) alternating with batch normalization layers and followed by a max-pooling layer with a small pooling factor (2-4). Both convolutional layers within a block contain the same number of channels and the number of channels increases by a factor of 2 in each consecutive block. The output of the last block is flattened into a vector and passed to a dense layer of the same size. The output of the dense layer is fed to the output layer. For RNN, we confine our search within the long-short-term-memory (LSTM) architecture. Because VGG and LSTM are state-of-the-art models in the CNN and RNN families respectively, we believe that these constraints do not cause much performance loss in detection probability.

We manually optimized the following hyperparameters to the best of our ability: the number of hidden layers (FC and RNNs only), the size of hidden layers (FC and RNNs only), number of blocks (CNN only), the pooling factor (CNN only), the learning rate, and the batch size (see Table III for details). Adam optimizer with learning-rate scheduling and early termination are used for all training. The learning rate is reduced by a factor of 10 each time when the validation loss sees no notable decrease in 10 consecutive epochs. Early termination is triggered if the validation loss sees no notable decrease in 15 consecutive epochs. Due to the stochastic nature of initialization and training, we repeat training on the two smaller training sets 10 times, and 5 times on the largest training set. The maximum and median detection probabilities are then computed.

TABLE III
TUNED HYPER-PARAMETERS

Arch. Type	Training Data Size	Learning Rate	Batch Size	Model Specification
FC	1E+03	1e-3	20	4 hidden layers, each of size 64
	1E+05	5e-4	1000	Same as above
	8E+06	5e-4	1000	Same as above
CNN	1E+03	1e-3	1000	2 blocks with 32, 64 filters respectively. Pooling factor is 4.
	1E+05	5e-4	1000	Same as above
	8E+06	5e-4	1000	3 blocks with 16, 32, 64 filters respectively. Pooling factor is 2.
RNN	1E+03	1e-4	50	1 hidden layer of size 64
	1E+05	5e-4	100	1 hidden layer of size 128
	8E+06	5e-4	100	Same as above
BiRNN	1E+03	5e-4	50	1 hidden layer of size 64
	1E+05	5e-4	1000	Same as above
	8E+06	5e-4	1000	1 hidden layer of size 128

The false dismissal probabilities of the NNs with tuned hyperparameters are compared in Fig. 1. The FC gives the worst performances among the four architecture types across all of the training set sizes. The FCs trained on the two smaller training sets perform either worse than or similar to the energy detection, and have no practical value considering that the energy detection is much simpler and requires no training. Unlike CNN, RNN or BiRNN, the FC does not observe the inherent order and adjacency of samples in the input, which makes it much harder to learn local patterns, no matter how wide/deep the FC is. The performances of the CNN, RNN and BiRNN are notably better than that of the FC, and can be considerably better than energy detection when the training set is large enough. The false dismissal probabilities of the CNN, RNN and BiRNN trained on the size-8E+06 training set are lower than that of the energy detection by at least a factor of 8. The performance gaps between CNN, RNN, and BiRNN are less defined, and not consistent across the training set sizes. Considering that our hyperparameter tuning is relatively coarse, no distinction can be comfirmed between CNN, RNN and BiRNN's abilities in this detection task.

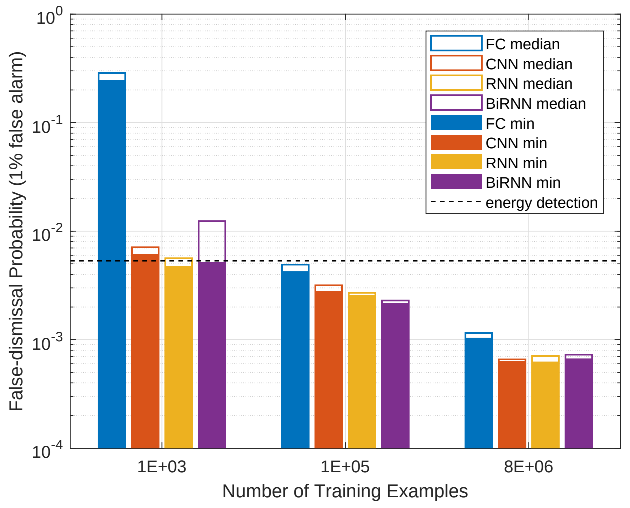

Fig. 1. False dismissal probabilities of the optimized NNs tested on SNR=3dB, QPSK test data

The operation counts and memory requirements of the tuned NNs are shown in Figs. 2 and 3. These metrics were not taken into consideration in the hyperparameter tuning process, so the tuned NNs could be highly inefficient in computation and memory. It is likely that by carefully changing the hyperparameters, the operation counts and memory requirements shown in Figs. 2 and 3 could be somewhat reduced without impairing detection performance. Despite these limitations, some qualitative observations can be made from the figures. First, in Fig. 2, the operation count of the FC is lower than that of the other NNs by 2 orders of magnitude on average. While the tuned FC gives the worst detection performance in Fig. 1, it also consumes the least amount of computational resources. Inspired by this observation, further comparisons between the FC and the other NN architecture types are conducted in Section IV. Second, while the RNNs and BiRNNs require more computation than the CNNs (Fig.2), their memory requirements are notably lower than that of the CNNs (Fig. 3). Considering that the memory for storing the parameters constitutes most of the total memory, this observation likely implies that the RNN has a higher level of parameter sharing than the CNN, which would potentially make the RNN more memory efficient. On the 1E+05 training set, the RNN requires more computation and memory than the BiRNN. This is because the RNN tuned on the 1E+05 training set has a larger layer size than the BiRNN tuned on the same training set.

IV. Comparison under a Computation Constraint

In the previous section, we compared the NNs whose hyperparameters were tuned with no constraint on their computational complexity or memory. The comparison shows a large gap between the computational complexity of the FC and that of the other NNs. While the FC clearly has the disadvantage that it was not able to approach the same level

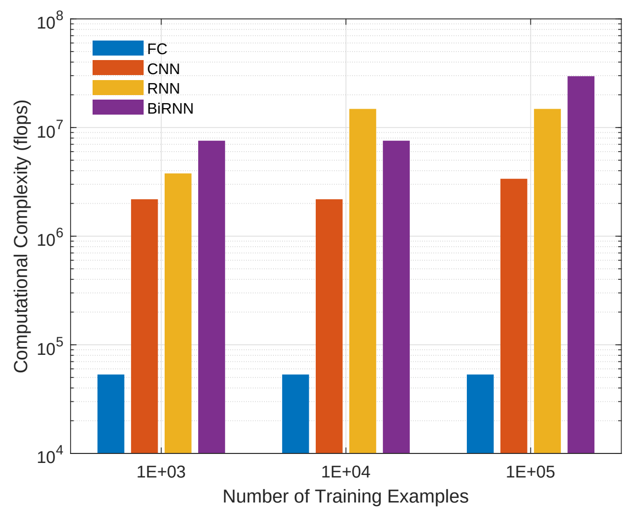

Fig. 2. Operation counts of the optimized NNs

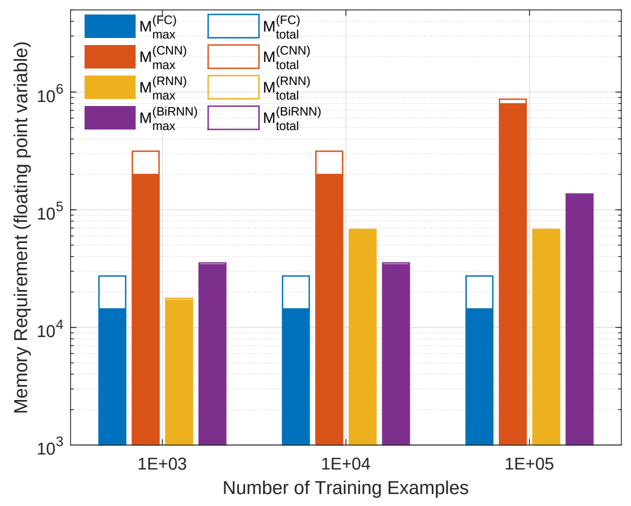

Fig. 3. Memory requirements of the optimized NNs

of performance of the other NNs in the hyperparameter tuning process, it is unclear how its performance compares to CNN, RNN and BiRNN with the same computational complexity. We scaled down the CNN, RNN and BiRNN used in the previous section so that their operation counts roughly match that of the FC. The modified NNs are described in Table IV. The four NNs with roughly the same level of computational complexity are trained on the three training sets, and their false dismissal probabilities are plotted in Fig. 4. The CNN, RNN, and BiRNN perform worse through the modification, particularly on the largest training set, where they all perform worse than the FC.

Because of the limitations of our parameter tuning process, the NNs which gave the performances in Fig. 4 are not necessarily the most computationally efficient in the CNN, RNN and BiRNN families. In particular, the constraints we imposed on the CNN and RNN architectures in Section III could have limited their computational efficiency. It is highly possible that there exist some CNN, RNN and BiRNN which can achieve lower false dismissal probability without violating the computation constraint. However, the observation in Fig. 4 suggests that unlike in the general case where the FC is clearly worse than the other architecture types, there is a possibility that an FC can achieve a performance comparable to that of the more advanced architectures when the computational resources are stringently limited.

TABLE IV
HYPER-PARAMETERS MODIFIED TO MEET THE COMPUTATIONAL COMPLEXITY CONSTRAINT

Arch. Type	**Model Specification**
FC	4 hidden layers, each of size 64
CNN	1 CONV-CONV-POOL block with 4 filters. Pooling factor is 4.
RNN	1 hidden layer of size 6
BiRNN	1 hidden layer of size 4

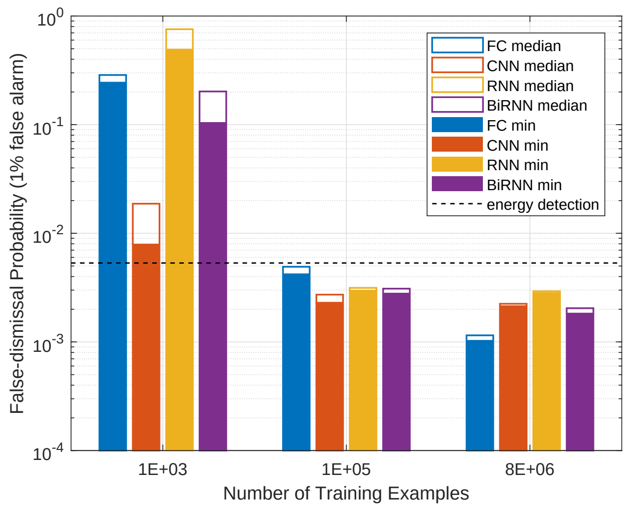

Fig. 4. False dismissal probability of NNs under computational complexity constraint

V. Conclusion

In this work, we compared detection performance, computational complexity, and memory requirements of four different types of NN architectures in a spectrum sensing task. We found that with abundant computation and memory resources, CNN, RNN and BiRNN are able to achieve a performance significantly better than that of the energy detector. CNN, RNN and BiRNN architectures resulted in very similar detection performance. The RNN/BiRNN possibly have an advantage over the CNN in terms of memory efficiency. Experimental results also show that FC should not be used in spectrum sensing unless in the special case where the computational resource is stringently limited. One factor not considered in this work is the correlation in the data carried by the signal, which commonly exists because of the error correction coding and correlations in the source content. The effect of these correlations on detection performance of the NN architectures is a potential topic of future research.

REFERENCES

[1] T. Yucek and H. Arslan, " A Survey of Spectrum Sensing Algorithms for Cognitive Radio Applications," IEEE Communications Surveys & Tutorials, vol. 11, pp 116-130, 2009.

[2] M. Bkassiny, Y. Li, and S. K. Jayaweera, "A Survey on Machine-Learning Techniques in Cognitive Radios," IEEE Communications Surveys & Tutorials, vol. 15, pp 1136-1159, 2013.

[3] Q. Peng, A. Gilman, V. Nuno, P. Cosman, and L. Milstein, "Robust deep sensing through transfer learning in cognitive radio," submitted to Wireless Communications Letters.

[4] Z. Ye, Q. Peng, K. Levick, H. Rong, A. Gilman, P. Cosman and L. Milstein, "A Neural Network Detector for Spectrum Sensing under Uncertainties", 2019 IEEE Global Communications Conference (GLOBECOM), 2019.

[5] D. Zhang and D. Wang, "Relation Classification: CNN or RNN?," Natural Language Understanding and Intelligent Applications Lecture Notes in Computer Science, pp. 665-675, 2016.

[6] W. Yin, K. Kann, M. Yu, and H. Schtze, "Comparative study of CNN and RNN for natural language processing," CoRR, vol. abs/1702.01923, 2017.

[7] J. Li, W. Dai, F. Metze, S. Qu, and S. Das, "A comparison of Deep Learning methods for environmental sound detection," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017.

[8] L. Persio and O. Honchar, "Artificial Neural Networks architectures for stock price prediction: comparisons and applications," International Journal of Circuits, Systems and signal processing, vol. 10, pp 403-413, 2016

[9] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon, and K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017.

[10] H. Liu, B. Lang, M. Liu, and H. Yan, "CNN and RNN based payload classification methods for attack detection," Knowledge-Based Systems, vol. 163, pp. 332-341, 2019.

[11] D. Molina, J. Liang, R. Harley, and G. K. Venayagamoorthy, "Comparison of TDNN and RNN performances for neuro-identification on small to medium-sized power systems," 2011 IEEE Symposium on Computational Intelligence Applications In Smart Grid (CIASG), 2011.

[12] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," CoRR, vol. abs/1605.07678, 2016.

[13] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," International Conference on Learning Representations (ICLR) 2015, 2015.