Deep Learning for Automatic Modulation Classification using Data Driven Subsampling

Sharan Ramjee, *Student Member, IEEE*, Shengtai Ju, *Student Member, IEEE*, Diyu Yang, *Student Member, IEEE*, Xiaoyu Liu, *Student Member, IEEE*, Aly El Gamal, *Member, IEEE* and Yonina C. Eldar, *Fellow, IEEE*

Abstract

In this work, we propose a novel subsampling technique to facilitate the use of deep learning for automatic modulation classification in wireless communication systems. Unlike traditional approaches that rely on pre-designed strategies that are solely based on expert knowledge, the proposed Data Driven Subsampling strategy employs deep neural network architectures - that we call *Subsampler Nets* - to simulate the effect of removing candidate combinations of samples from each training input vector. The subsampled data is then processed by another deep learning classifier that recognizes each of the considered 10 modulation types. We show that the proposed subsampling strategy not only introduces drastic reduction in the classifier training time, but can also improve the classification accuracy to levels never reached before for the considered dataset. An important feature herein is exploiting the transferability of learning by showing that using a combination of Subsampler Nets delivers a superior performance to using only a single network in the combination.

I. Introduction

Automatic modulation classification plays an important role in modern wireless communication. It finds applications in various commercial and military areas. For example, Software Defined Radios (SDR) use blind recognition of the modulation type to quickly adapt to various communication systems, without requiring control overhead. In military settings, friendly signals should be securely received, while hostile signals need to be efficiently recognized typically without prior information. Under such conditions, advanced real time signal processing and blind modulation recognition techniques are required. Modulation recognition may also prove to be an essential capability for identifying the source(s) of received wireless signals, which can enable various intelligent decisions for a context-aware autonomous wireless communication system.

A typical modulation classifier consists of two steps: signal preprocessing and classification algorithms. Preprocessing tasks may include noise reduction and estimation of signal parameters such as carrier frequency and signal power. In the second step, three popular categories of modulation recognition algorithms are conventionally selected: Likelihood-Based

This work is supported in part by DARPA and AFRL under grant no. 108818, and was presented in part at the Asilomar Conference on Signals, Systems and Computers [1].

S. Ramjee, S. Ju, D. Yang, X. Liu and A. El Gamal are with the Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, 47907 USA (e-mail: sramjee, ju10, yang1467, liu1962, elgamala@purdue.edu).

Y. C. Eldar is with the Department of Math and Computer Science, Weizmann Institute of Science, Rechovot, Israel (e-mail: yonina@weizmann.ac.il).

(LB) [2]–[7], Feature-Based (FB) [8]–[13] or using an Artificial Neural Network (ANN) [14]–[18]. The first compares the likelihood ratio of each possible hypothesis against a threshold, which is derived from the probability density function of the observed wave. Multiple likelihood ratio test (LRT) algorithms have been proposed: Average LRT [19], Generalized LRT [20], Hybrid LRT [7] and quasi-hybrid LRT [2]. For the FB approach, the classification decision is based solely on a subset of selected features. Both LB and FB methods require precise estimates in the first step and have only been derived to distinguish between few modulation types [4], [19], [21], [22]. ANN structures such as Multi-Layer Perceptrons (MLP) have been widely used as modulation type classifiers [14]. Traditional MLP has performed well on modulation types such as AM, FM, ASK, and FSK. Furthermore, recent work has shown that deep neural networks with cutting-edge structures could greatly improve the classification process (see e.g., [23] and [24]), and deliver superior performance to the state of the art by enabling modulation recognition in presence of a wide variety of modulation types, and with little or no requirements from the preprocessing step.

Deep neural networks have played a significant role in the research domain of video, speech and image processing over the past few years. The recent success of deep learning algorithms is associated with applications that suffer from inaccuracies in existing mathematical models and enjoy the availability of large data sets. Recently, the idea of deep learning has been introduced for modulation classification using a Convolutional Neural Network (CNN) for distinguishing between 10 different modulation types [23]. Simulation results show that a CNN not only demonstrates better accuracy results, but also provides more flexibility in detecting various modulation types compared to current expert-based approaches. Other deep neural network architectures like the Residual Network (ResNet) [24] were also recently introduced to strengthen feature propagation in the deep neural network by creating shortcut paths between different layers in the network. By adding the bypass connections, an identity mapping is created, allowing the deep network to learn simple functions. A ResNet architecture was shown to be successful for distinguishing between 24 different modulation types in [25]. Also, a Convolutional Long Short-term Deep Neural Network (CLDNN) has been recently introduced in [26], by combining the architectures of the CNN and the Long Short-Term Memory (LSTM) into a deep neural network by taking advantage of the complementarity of CNNs, LSTMs, and conventional deep neural network architectures. The LSTM unit is a memory unit of a Recurrent Neural Network (RNN). RNNs are neural networks with memory that are suitable for learning sequence tasks such as speech recognition and handwritten recognition. LSTM optimizes the gradient vanishing problem in RNNs by using a forget gate in its memory cell, which enables the learning of long-term dependencies. The authors in [27] demonstrated the potential of LSTM units for accurately recognizing a wide range of modulation types.

In this work, we first present four different architectures that deliver higher classification accuracy than the CNN introduced in [23]. We design our own CNN and CLDNN architectures for the modulation recognition task, as well as derive optimized versions of the ResNet architecture of [25] and the LSTM architecture of [27], by tuning the number of residual stacks for ResNet and the hyperparameters for LSTM. However, we find that the performance of all these architectures, as well as the one in [23], suffers a degradation due to confusions between similar modulation types, in particular those of QAM16 and QAM64 and those of AM-DSB and WBFM. Another major challenge facing machine learning algorithms based on deep neural network architectures is the long training time. For example, for the problem at hand, even the simple CNN architecture in [23] would take approximately 40 minutes to train using three Nvidia Tesla P100 GPU chips. This creates a serious obstacle towards the feasibility of applying such algorithms in real time, where online

training is needed to adapt the network architecture to changing environmental conditions. In particular, applying deep learning to the autonomous wireless communication systems anticipated in next-generation networks require significant reductions in the training time compared to the state of the art methods. In such systems, it is likely that training of the machine learning algorithms will be frequently needed to accommodate new environmental conditions. Hence, the issue of reducing the training time becomes essential for the success of these algorithms. The third major challenge is the computational overhead due to sampling the received signal, which can be cumbersome in real time, particularly in wideband settings.

We tackle the three aforementioned challenges by introducing a novel data driven subsampling stategy that relies on multiple deep neural network classifiers, as well as a final deep neural network classifier that recognizes the modulation type. Our strategy relies on the learning transferability property of deep neural networks, as we determine the optimal set of samples based on simulations that employ a diverse set of architectures, all of which are suitable for the considered classification task. The obtained results demonstrate that not only the proposed data driven subsampling strategy leads to significant reductions in the required training time, but it also leads to achieving unprecedented classification accuracy values and almost fully resolve the confusions - suffered by traditional methods as well as previous deep learning-based methods - between similar pairs of modulation types like QAM16 and QAM64 as well as AM-DSB and WBFM at higher SNR values (above 2 dB).

The rest of this document is organized as follows. In Section II, we detail the experimental setup. We then provide a performance analysis in Section III of a Gaussian naive Bayes classifier to shed light on the difficulty of the considered modulation classification task, and the inadequacy of traditional likelihood based techniques. In Section IV, we introduce novel deep neural architectures with superior performance to the state of the art. We then review the state of the art subsampling and feature selection techniques in Section V. We provide a description of the proposed data driven subsampling method and analyze its performance in Section VI. Finally, we conclude with a discussion in Section VII and concluding remarks in Section VIII.

II. Experimental Setup

In this work, we consider the classification of the modulation type of received wireless signals, using deep neural network classifiers and a novel subsampling technique. A general expression for the received baseband complex envelope is

$$r\left(t\right) = s(t; \boldsymbol{u_i}) + n\left(t\right), \tag{1}$$

where

$$s(t; \boldsymbol{u_i}) \;=\; a_i e^{j2\pi \triangle f t} e^{j\theta} \sum_{k=1}^{K} e^{j\phi_k} s_k^{(i)} g(t-(k-1)T-\varepsilon T),\; 0 \leq t \leq KT \tag{2}$$

is the noise-free baseband complex envelope of the received signal, and $n(t)$ is the instantaneous channel noise at time t. In (2), a_i is the received signal amplitude, Δf is the carrier frequency offset, θ is the time-invariant carrier phase introduced by the propagation delay, ϕ_k is the phase jitter, $\{s_k^{(i)}, 1 \leq k \leq K\}$ denotes K complex symbols taken from the i^{th} modulation format, T represents the symbol period, ε is the normalized epoch for time offset between the transmitter and signal receiver, $g(t) \;=\; P_{pulse}(t) \otimes h(t)$ is the composite effect of the residual channel with $h(t)$ denoting the channel impulse response and $\otimes$ denoting mathematical convolution, and $P_{pulse}(t)$ is the transmit pulse shape. Here,

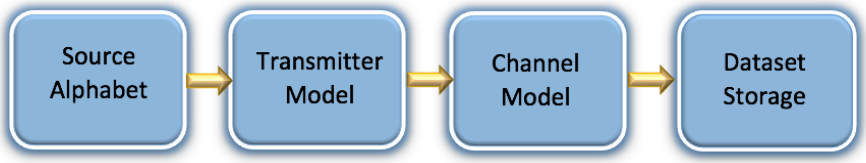


Fig. 1: High-level framework for data generation.

$\boldsymbol{u_i} = \{a_i,\ \Delta f,\ \theta,\ \varepsilon,\ g(t),\ \{\phi_k\}_{k=1}^{K},\ \{s_k^{(i)}\}_{k=1}^{K}\}$ is the multidimensional vector that includes the deterministic unknown signal or channel parameters for the i^{th} modulation type. **The goal is to recognize the modulation type i from the received signal $r(t)$.**

We use the RadioML2016.10b dataset generated in [23] as the input data. Details about the generation of this dataset can be found in [28]. Figure 1 shows a high-level framework for the data generation process. Ten widely used modulation schemes are chosen: eight digital and two analog modulations. These consist of BPSK, QPSK, 8PSK, QAM16, QAM64, BFSK, CPFSK, and PAM4 for digital modulations, and WBFM, and AM-DSB for analog modulations. For digital modulations, the entire Gutenberg works of Shakespeare in ASCII is used, with whitening randomizers applied to ensure equiprobable symbols and bits. For analog modulations, a continuous voice signal consisting of acoustic voice speech with some interludes and off times is used as input.

The dataset is split equally among all considered modulation types. For the channel model, physical environmental noises like thermal noise and multipath fading were simulated. The models for generating random channel and device imperfections, that determine the parameters in (2), are detailed in [28][1]. When packaging data, the output stream of each simulation is randomly segmented into vectors as the original dataset with a sample rate of 1M sample per second. Similar to the way that an acoustic signal is windowed in voice recognition tasks, a sliding window extracts 128 samples with a shift of 64 samples, which forms a sample vector in the dataset we are using. 160,000 sample vectors generated using the GNU-radio library developed in [28] are segmented into training and testing datasets. The training examples - each consisting of 128 samples - are fed into the neural network in 2×128 vectors with real and imaginary parts separated in complex time samples, except for the pure LSTM architecture, which is fed samples in polar form (amplitude and phase). The SNR of the samples is uniformly distributed from -20dB to +18dB, with a step size of 2 dB, i.e., the dataset is equally split among all SNR dB values in $\{-20, -18, -16, \cdots, 18\}$.

For all our experiments, we used Keras with TensorFlow as a backend. We used a GPU server equipped with three Tesla P100 GPUs with 16 GB memory. The Adam optimizer was used for all architectures, and the loss function was the categorical cross entropy function. We also used ReLu activation functions for all layers, except the last dense layer, where we used Softmax activation functions. For all architectures except the LSTM, we used a batch size of 1024, and a learning rate of 0.001. For the LSTM architecture, we used a batch size of 400, and a learning rate of 0.0018.

[1]Dataset generation parameters are also available at https:// github.com/radioML/dataset

We note from the frequency domain representation of the input waveform depicted in Fig. 2 that the sampling rate of the input waveform is around 6 times the Nyquist rate.

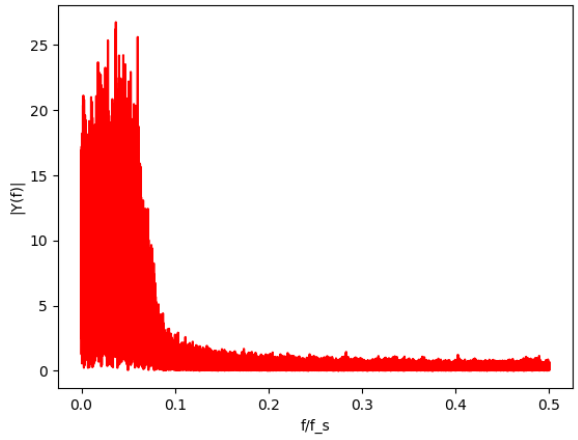


Fig. 2: One-sided normalized FFT for a BPSK signal. A value of 0.5 on the horizontal axis corresponds to the Nyquist rate (Bandwidth is half the sampling rate). Most of the signal energy is within a band of around $\frac{1}{12}$ of the sampling rate.

III. Gaussian Naive Bayes Classifier

In this section, we discuss the performance of a Gaussian naive Bayes classifier, to assess the difficulty of the considered modulation classification task. In subsequent sections, it will be clear how deep learning significantly outperforms the naive Bayes classifier, which demonstrates the effectiveness of employing deep learning for this problem. The Gaussian naive Bayes classifier can be described through the conditional probabilities:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}}, \tag{3}$$

where $P(x_i|y)$ is the likelihood of an observed instance x_i belonging to a certain class y, σ_y^2 is the observed variance of class y, and μ_y is the observed mean of class y. The predicted output of x_i is the class that maximizes the likelihood function. Instead of trying to classify all ten modulation types, we used pairs of modulation schemes to demonstrate the performance of the Gaussian naive Bayes classifier more clearly.

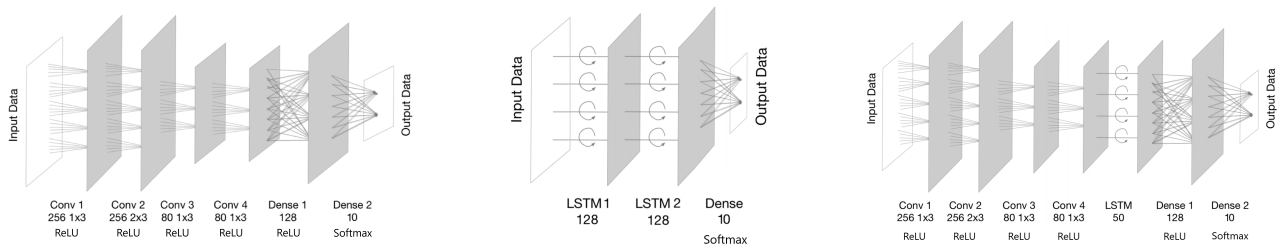
Several pairs of modulation schemes were investigated, including both commonly-confused pairs and not commonly-confused pairs. We will list out a few pairs to illustrate the results. The classification accuracy for PAM4 vs QAM64 was 68.5%, while it dropped to only 50% for QAM16 vs QAM64. The classification accuracy for AM-DSB vs BPSK was 70%, while it became 53% for AM-DSB vs WBFM. We then tried SNR selection, where we used the Gaussian naive Bayes classifier over the dataset corresponding only to the single SNR under consideration. Using 18 dB SNR, the accuracy for PAM4 vs. QAM64 increased to 82%. The accuracy for QAM 16 vs QAM64 still remained at around 50%, which demonstrates the difficulty of distinguishing between these two pairs even at high SNR. The accuracy for AM-DSB vs BPSK increased to 81%, while the accuracy for the commonly-confused AM-DSB vs WBFM remained at 53%.

TABLE I: Gaussian naive Bayes classifier results for different modulation pairs at all SNRs and at 18 dB SNR.

Modulation Pairs	Classification Accuracy (All SNR)	Classification Accuracy (18 dB)
PAM4 vs QAM64	68.5%	82%
QAM16 vs QAM64	50%	51%
AM-DSB vs BPSK	70%	81%
AM-DSB vs WBFM	53%	53%

Based on the collected results, we found the Gaussian naive Bayes classifier to deliver relatively good performance at high SNR for classifying pairs of not commonly-confused modulation schemes. However, it does not perform well for commonly misclassified modulation pairs such as AM-DSB/WBFM and QAM16/QAM64. When using the high SNR data at 18 dB, the classification accuracy significantly increased for all tested modulation pairs except the commonly misclassified modulation pairs AM-DSB/WBFM and QAM16/QAM64. In the following section, we will present deep neural network architectures whose performance for modulation classification is significantly better then the examined naive Bayes method.

IV. Deep Neural Network Architectures



(a) (b) (c)

Fig. 3: Architecture Diagrams of (a) CNN, (b) LSTM, and (c) CLDNN

A. CNN Architecture

The Convolutional Neural Network (CNN) known as CNN2, that was proposed in [23], achieved an accuracy of 75% at high SNR for modulation classification on the considered dataset. After experimenting with various network depths and filter settings, we were able to obtain an accuracy of approximately 83.8%, which was the highest observed accuracy after extensive hyperparameter tuning. We used a CNN architecture with four convolutional layers, a dense layer with ReLU, and a final dense layer with Softmax, as depicted in Figure 3a.

The first parameter below each convolutional layer in the figure represents the number of filters in that layer, while the second and third numbers show the size of each filter. For the two dense layers, we use 128 and 10 neurons in order of their depth in the network. Here, we note that the improved performance is due to the two extra convolutional layers as compared to the CNN2.

B. LSTM Architecture

A pure LSTM architecture that had no convolutional layers was proposed in [29] for modulation classification, and achieved a significant increase in the classification accuracy on the considered dataset. The input samples to this architecture are in polar form, instead of the rectangular form that is used for all other considered architectures. The polar form representation is obtained by computing the amplitude and phase of the input I/Q sample at each sampling time step. The architecture consists of two LSTM layers, each with 128 cells, followed by a dense layer with Softmax. The LSTM units extract the temporal dependencies of the amplitude and phase characteristics of different modulation schemes. The dense layer with Softmax activation function serves as the last hidden layer to project the output of the second LSTM layer onto the final probability output space $\mathbb{P}(\text{Classes})$. The LSTM architecture is depicted in Figure 3b.

We fine tuned the hyperparameters of the LSTM network in [29] and found out that it achieves a classification accuracy of 92% at high SNR. The great performance of the LSTM network further demonstrates that RNNs provide a good choice for the task of modulation classification in terms of classification accuracy, due to their ability for extracting long-term temporal relationships in time series data, which could be useful for identifying patterns of symbol-to-symbol transitions. However, RNNs have several drawbacks such as long training times and difficulty to parallelize computations. The training time for the LSTM structure is approximately four times longer than that of the four-layer CNN. Furthermore, the temporal features that RNNs attempt to extract are likely to break down either when the sampling rate of the arriving signals is not fixed, or when a dimensionality reduction or subsampling process is necessary for achieving faster learning.

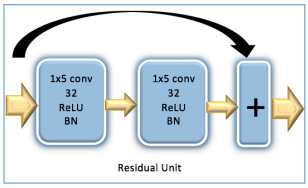
C. CLDNN Architecture

Recurrent Neural Networks (RNN) have been proved to Inspired by [26], we proposed a CLDNN in [1] by adding an LSTM layer into the CNN architecture. The detailed architecture considered for the CLDNN is shown in Figure 3c. The extra LSTM layer is placed between the convolutional layers and the dense layers. In our experiments, an LSTM layer with 50 cells provided the best accuracy. A classification accuracy of 88.5% at higher SNR values (above 2 dB) was obtained using this architecture.

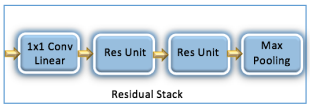
D. ResNet Architecture

Layer	Output dimensions
Input	2x128
Residual Stack	32x64
Residual Stack	32x32
Residual Stack	32x16
FC/ReLU	128
FC/ReLU	128
FC/Softmax	10

(a)



(b)



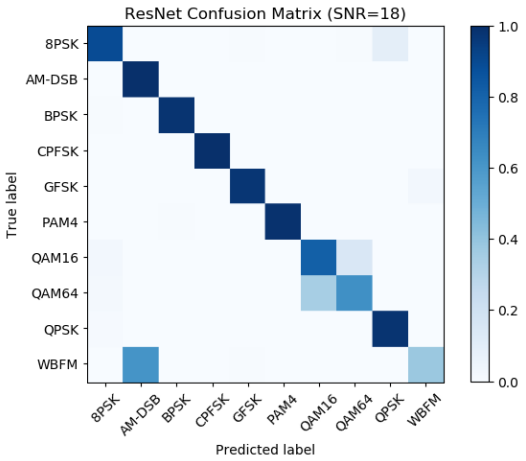
(c)

Fig. 4: (a): The Resnet Architecture, (b) A Residual Unit, and (c) A Residual Stack

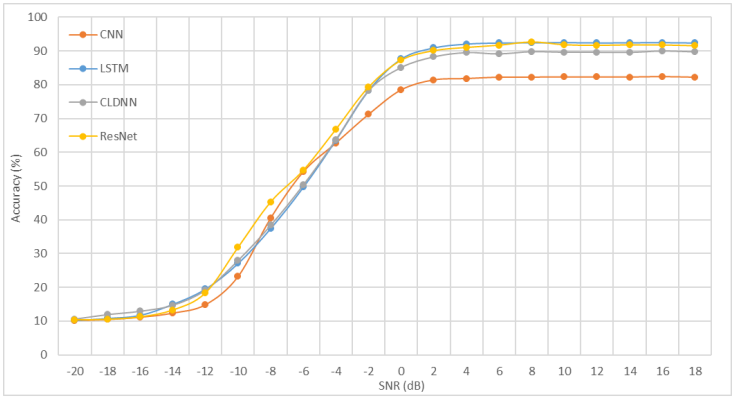
As neural networks grow deeper, their learning performance has been challenged with problems like vanishing or exploding gradient and overfitting, and therefore both training and testing accuracy for a deep neural network start to degrade after the network reaches a certain depth. The Deep Residual Network (ResNet) architecture that was introduced in the ImageNet and COCO 2015 competitions [24], resolves accuracy degradation issues in deeper neural networks and has been shown to be a robust choice for a wide range of machine learning tasks. Inspired by the ResNet architecture in [25], we designed a similar ResNet but with three residual stacks instead of six, as we found that choice to lead to increasing the classification accuracy. In our network, three residual stacks are followed by three fully connected layers, where each residual stack consists of one convolutional layer, two residual units, and one max-pooling layer. As seen in [25], for each residual unit, a shortcut connection is created by adding the input of the residual unit with the output of the second convolutional layer of the residual unit. Finally, each convolutional layer in the residual unit uses a filter size of 1x5 and is followed by a batch normalization layer to prevent overfitting. The overall architecture can be observed in Figure 4.

E. Results

In Figure 5b, we plot the classification accuracy at each SNR value for each of the aforementioned four deep neural network architectures. We notice that even though our CNN architecture delivers better performance than the CNN2 architecture of [23], as discussed in Section IV-A, the other three proposed architectures deliver a noticeably superior performance at high SNR. We further note that - similar to previous work on deep learning for modulation classification - most of the misclassifications at high SNR are due to confusions between the AM-DSB/WBFM and the QAM16/QAM64 pairs, which is evident from the ResNet 18 dB confusion matrix depicted in Figure 5a. We will show in Section VI how the proposed Data Driven Subsampling method leads deep neural network classifiers to clear this confusion.



ResNet Confusion Matrix at 18 dB SNR.



Accuracy vs SNR for each of the Architectures.

Fig. 5: Deep Learning Results for Modulation Classification

V. Conventional Subsampling and Feature Selection Techniques

In this section, we discuss candidate conventional subsampling techniques, where signals are subsampled with the goal of minimizing loss in information relevant to the classification task. Since subsampling is essentially a feature selection problem, we will also explore state of the art feature selection methods and apply them to perform subsampling. In the rest of the paper, we will use the ResNet as our choice of architecture for training and classification of the data with the selected samples as it delivered the best relative performance among the architectures examined in Section IV. Similar trends in the Accuracy vs SNR curves have also been observed by the other architectures in this paper, unless we explicitly mention otherwise.

A. *Conventional Subsampling Techniques*

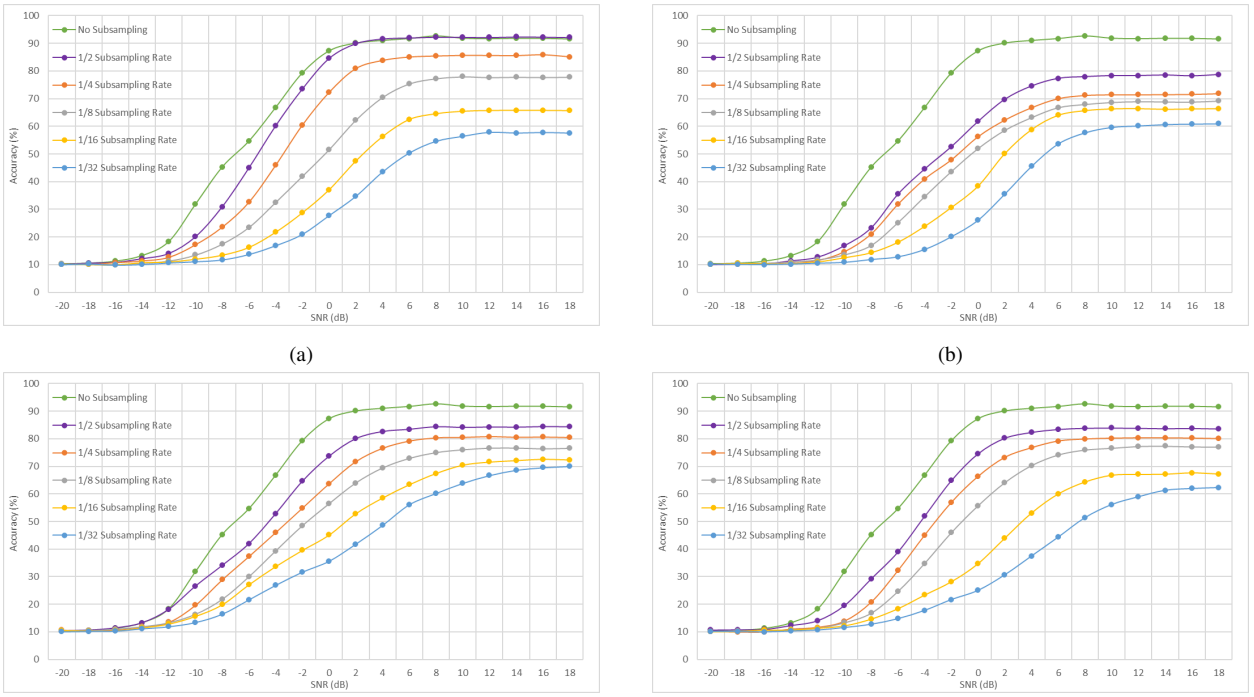


Fig. 6: Accuracy vs SNR for ResNet with (a) Uniform, (b) Random, (c) Magnitude Rank, and (d) PCA-based Subsampling

1) ***Uniform Subsampling****:*

Uniform Subsampling is the selection of samples at equal intervals of time [30]. Here, we sample the input sample vector at regular intervals and train and test the deep neural network architectures based on the subsampled vector. The results obtained by Uniform Subsampling can be observed in Figure 6a. We observe that the performance of ResNet slightly increases when using half the samples at high SNR values compared to no subsampling, as the dataset consists of oversampled data as illustrated in Figure 2. Therefore, since the loss of information is low when subsampling above the Nyquist Rate, we believe that the accuracy increase for the ResNet comes as a result of overfitting reduction.

2) ***Random Subsampling****:*

Random Subsampling is the selection of samples at random intervals of time [30]. An important detail to note here is that we perform random subsampling such that the order in which the samples appear is maintained, which means that if two samples are collected at time t and time $t + t_1$, where $t_1 > 0$, then the sample collected at $t + t_1$ must come after the sample collected at time t in the resulting subsampled vector. This is to make sure that the ability of the ResNet to form temporal relations of repetitive patterns in time as examined in [24] is not affected. Further, the randomization is performed only once to select a set of indices to subsample at, and then all training and testing vectors are subsampled at the same indices. The results obtained by random subsampling of the input to the ResNet can be observed in Figure 6b.

We note that the results are based on one random choice for the indices, as we obtained very similar results when trying multiple other choices.

The uniformly sampled data leads to higher classification accuracy than the randomly sampled data when the resulting sampling rate is close to Nyquist $\left(\frac{1}{8}\right.$ subsampling$\left.\right)$ or above $\left(\frac{1}{4}\right.$ and $\left.\frac{1}{2}\right.$ subsampling$\left.\right)$. However, random subsampling actually leads to higher classification accuracies for sampling rates well below the Nyquist rate $\left(\frac{1}{16}\right.$ and $\left.\frac{1}{32}\right.$ subsampling$\left.\right)$. This is consistent with the intuition in [30], where typically sub-Nyquist strategies that are non-uniform are superior.

3) ***Magnitude Rank Subsampling****:*

Inspired by the sub-Nyquist rate sampling techniques discussed in [30], we formulated a new subsampling technique that is based on the magnitudes of the samples. First, the real and imaginary parts of the samples are used to calculate the magnitudes of the samples. Then, samples corresponding to each vector are then ranked in the descending order of their magnitudes. Finally, the top samples with the highest magnitudes are collected based on the subsampling rate and are rearranged back in the sequence that they were present in as observed in the original dataset, which is similar to the maintenance of the order in which the samples appear in the case of Random Subsampling. The results obtained by the described Magnitude Rank Subsampling of the input to the ResNet can be observed in Figure 6c. We observe that Magnitude Rank Subsampling performs significantly better than both uniform and random subsampling when operating well below Nyquist rates $\left(\frac{1}{16}\right.$ and $\left.\frac{1}{32}\right.$ subsampling rates$\left.\right)$. Again, this is consistent with the intuition discussed in [30] for sub-Nyquist sampling.

4) ***Principal Component Analysis-based Subsampling****:*

Principal Component Analysis (PCA) is a dimensionality reduction algorithm where the principal components of a dataset are computed by performing the eigendecomposition of its covariance matrix [31]. PCA uses an orthogonal transformation to project the dataset onto a set of linearly uncorrelated directions known as principal components [32]. This allows us to reduce the dimensionality of the dataset by capturing as much of the data variability in as few dimensions as possible. We use this to formulate a new technique called Principal Component Analysis-based Subsampling (PCS), where we first use PCA to obtain the transformation coefficient of each sample. The coefficients for the real and complex parts of each sample are computed and the sum of the square of these coefficients constitute of the coefficient of each sample. We then sort these coefficients by order of decreasing magnitudes and the top samples with the highest coefficients are collected based on the subsampling rate and are rearranged back in the sequence that they were present in as done in case of Random and Magnitude Rank Subsampling. The results for PCS, as observed in Figure 6d, show a very good performance compared to other subsampling techniques described in Section V-A. Magnitude Rank Subsampling comes close to PCS at high SNR values and is slightly better when the subsampling rate is low $\left(\frac{1}{16}\right.$ and $\left.\frac{1}{32}\right)$. However, unlike the other techniques mentioned in this section, Magnitude Rank Subsampling suffers from the drawback that all the samples are needed to be collected in order to extract the top k samples with the highest magnitudes for both the training and testing datasets. For the other techniques, once the sample indices are learned from the training dataset, the same samples can be collected from the testing dataset without having to examine all the samples of the testing dataset, which allows for a more practical implementation in real world scenarios where the employed hardware needs to be power efficient.

B. *Feature Selection Techniques*

Some of the techniques described in Section V-A are popular compressed sensing [33] methods that are specialized for subsampling. However, in the context of machine learning, compressed sensing and subsampling, in general, are essentially feature selection paradigms where samples are selected similar to the way features are selected. The subsampling techniques thus far employed, with the exception of PCA-based Subsampling, have been naive in the sense that they do not exploit the global characteristics of the data in the training set to assess feature importance. Therefore, here we will discuss popular filter, wrapper, and embedded feature selection techniques [34].

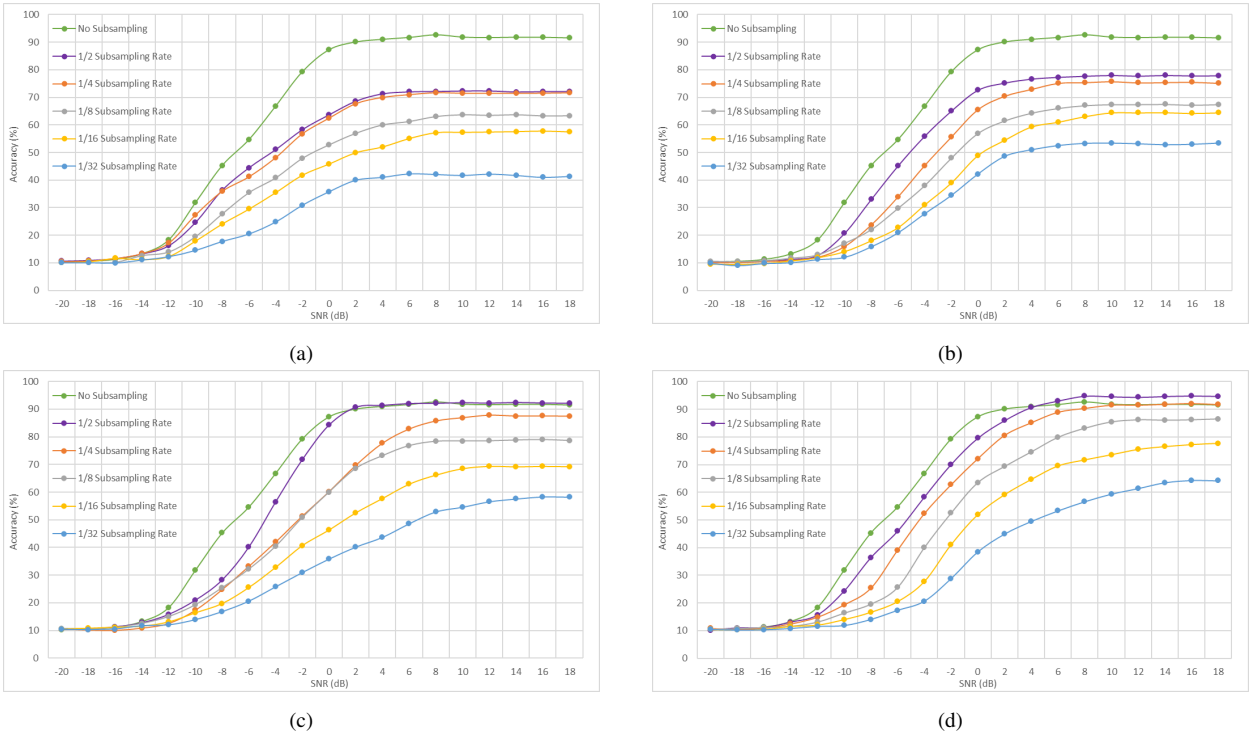


Fig. 7: Accuracy vs SNR for ResNet with (a) Laplacian Score, (b) Fisher Score, (c) RFS and (d) FQI Techniques

Filter methods do not rely on the employed learning algorithm and use proxy measures to evaluate and rank features instead of using the classification error. They are computationally efficient but due to the lack of a specific learning algorithm supervising the feature selection process, the selected features may not be good for the task at hand. Wrapper methods, on the other hand, use the predictive performance of a predefined learning algorithm to evaluate the quality of selected features. This is an iterative process and thus, these methods are more computationally inefficient compared to filter methods. Finally, Embedded methods perform feature selection by embedding feature selection into model learning, and hence attempt to combine the computational efficiency of filter methods and the learning algorithm incorporation of wrapper methods [35].

An important point to note when using feature selection is that each sample consists of two features: a real and an imaginary part. Thus, each of the examples in the dataset consists of 128 samples or 256 features. The issue here arises when we have two rankings for each of the samples in the dataset, as only whole samples can be selected. This implies

that only pairs of features, each of which belong to the same sample, can be selected. Thus, for each of the techniques examined, except FQI where we have sample scores, we add the two scores obtained for the two features belonging to a sample to obtain a sample score before proceeding to rank the samples. The techniques that we chose to employ are ones that have become popular (see e.g., [36]) for their superior ability to perform feature selection on time-domain data with temporal dependencies, which is the type of data that we are dealing with for modulation classification. Additionally, the implementations of these techniques can be found in the scikit-feature[2] package created by [37].

1) ***Laplacian Score****:*

The Laplacian Score [38] is an unsupervised filter feature selection algorithm that selects features such that the data manifold structure is preserved to the greatest possible extent [36]. The data manifold structure is captured through a graph whose vertices are the samples and edges represent sample similarity. We start by constructing the graph Gaussian kernel matrix S with unity standard deviation, and edge weights inversely proportional to the distance between two samples:

$$S(i,j) = e^{-\|x_i - x_j\|_2^2}, \tag{4}$$

if x_i is one of the five nearest neighbors of x_j. If this is not the case, then we set $S(i,j)$ as 0. The diagonal matrix D represents node degrees and can be represented as $D(i,i) = \sum_{j=1}^{n} S(i,j)$, where n is the number of training examples. The Laplacian matrix L is represented as $L = D - S$. Let $\mathbf{1} = [1, ..., 1]^T$, where we use x^T to denote the transpose of a vector x. The Laplacian Score of each feature f_i is computed according to (5).

$$LaplacianScore(f_i) = \frac{\tilde{f}_i^T L \tilde{f}_i}{\tilde{f}_i^T D \tilde{f}_i}, \text{ where } \tilde{f}_i = f_i - \frac{f_i^T D \mathbf{1}}{\mathbf{1}^T D \mathbf{1}} \mathbf{1}, \tag{5}$$

and with a slight abuse of notation, we use f_i to denote the n-long vector consisting of the values of the i^{th} feature for all n training examples. Using (5), we computed the top k features by selecting the k features with the k smallest Laplacian scores. Observing the results in Figure 7a, we can see that the accuracy curves for feature selection using the Laplacian Score are worse than the ones for the Fisher Score and the ReliefF feature selection techniques. This poor performance can be attributed to the fact that the Laplacian Score is an unsupervised algorithm and while preserving the data manifold structure allows the algorithm to mitigate the generic overall loss in information, it fails to effectively remove features in the training data that allow deep neural network classifiers to distinguish between training examples belonging to different classes.

2) ***Fisher Score****:*

The Fisher Score [39] is a supervised filter feature selection algorithm. The features are selected such that the features of samples within the same modulation type are similar while the features of samples belonging to other modulation types are as distinct as possible [36]. The Fisher Score of each feature f_i is computed according to (6).

[2]https://github.com/jundongl/scikit-feature

$$FisherScore(f_i) = \frac{\sum_{j=1}^{c} n_j(\mu_{ij} - \mu_i)^2}{\sum_{j=1}^{c} n_j \sigma_{ij}^2}, \tag{6}$$

where c is the number of classes, n_j represents the number of training examples in class j, μ_i represents the mean value of feature f_i, μ_{ij} represents the mean value of feature f_i for training examples in class j, and σ_{ij}^2 represents the variance of feature f_i for training examples in class j. Using (6), we compute the top k sample indices by selecting the samples with the largest Fisher Scores, and these results can be observed in Figure 7b. We observe that using the Fisher Score for sample selection leads to poor classification accuracies. We believe that this is because the algorithm selects each feature independently without taking into account the interactions between different features [39].

3) ***RFS****:*

Efficient and Robust Feature Selection (RFS) [40] is a fast and computationally efficient embedded feature selection method that exploits the noise robustness property of the joint $\ell_{2,1}$-norm loss function, by applying the $\ell_{2,1}$-norm minimization on both the loss function and its associated regularization function. This can be attributed to the rotational invariant property [41] of the $\ell_{2,1}$-norm loss function. [42] define RFS's objective function as

$$\min_{W} \|XW - Y\|_{2,1} + \alpha\|W\|_{2,1}, \tag{7}$$

where X is the data matrix, Y is the one-hot label indicator matrix, W is a matrix indicating feature contributions to classes, and α is the regularization parameter. Features are then ranked by the ℓ_2 norm values of the corresponding row in the optimal matrix W. The value of α for our experiments was chosen by performing RFS on a wide range of values and picking the value that led to the highest accuracy on the cross-validation set. We show the results of RFS in Figure 7c. We believe that the relatively poor performance of RFS compared to FQI is because the AM-DSB/WBFM and the QAM16/QAM64 classes have samples that are localized in the same vicinity of the sample space as observed in the PCA and t-SNE plots of the training examples that belong to these classes in Figures 14 and 15. Thus, RFS overfits these classes during training and is unable to determine the samples that can be used to distinguish one class from another when these samples are clustered together in the same region of the sample space.

4) ***FQI****:*

The Feature Quality Index (FQI) [43] is a wrapper feature selection algorithm that serves as the main inspiration for our proposed technique and utilizes the output sensitivity of the considered model to changes in the input to rank features. The FQI of feature f_i is computed as

$$FQI(f_i) = \sum_{j=1}^{n} \|o_j - o_j^i\|^2, \tag{8}$$

where n is the total number of training examples, o_j is the output of our ResNet architecture when the j^{th} training example is the input, and o_j^i is the output of the neural network when the j^{th} training example, with the value of the i^{th} feature set to 0, is the input [44]. Features that are less important impact the output to a lesser extent and possess lower FQIs. Using (8), we computed the best k features by selecting the k features with the k largest FQIs. Observing the results in Figure 7d, we can see that the accuracy curves for feature selection using FQI values are better than the other methods. As stated

earlier, each sample consists of two features and features can only be selected in pairs where both features correspond to the same sample. We can compute the FQI for one sample by setting the inputs for both real and imaginary parts as 0. This allows us to avoid taking into account the feature interactions within the pairs of features that belong to the same sample, which is what allows the FQI to outperform the other techniques as observed in 7d.

VI. Data Driven Subsampling

As observed in Figures V-A and V-B, all the subsampling techniques mentioned in Section V suffer from low classification accuracies that are caused by the loss of information during subsampling and the inability to resolve the AM-DSB/WBFM and the QAM16/QAM64 misclassification errors. Our proposed approach overcomes these drawbacks by using deep neural networks in order to find the set of samples whose removal leads to the lowest loss in classification accuracy. Data Driven Subsampling, as its name suggests, utilizes the training data to search for optimal samples in the sample space using Subsampler Nets. In this section, we will demonstrate how to use the architectures mentioned in Section IV to build these Subsampler Nets, and also show that the sets of samples chosen by these Subsampler Nets lead to classification accuracies that are drastically higher than the state of the art.

A. Subsampler Nets: A Wrapper Feature Selection Approach

In this section, we propose a supervised wrapper feature selection algorithm that we call the **Subsampler Net**, which uses a deep neural network to rank the importance of each sample in accordance to the relative drop in classification accuracy that occurs when that sample is removed (set to 0). Similar to other wrapper feature selection methods, a Subsampler Net relies on a classifier to rank sample importance. We will refer to this classifier as the **Ranker Model** that can be any state-of-the-art deep learning model that has already been trained using all the samples in the training set for the considered modulation classification task.

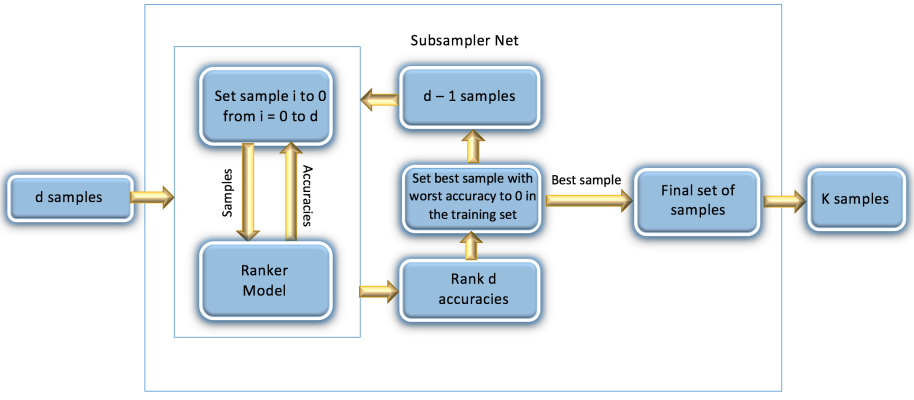


Fig. 8: Subsampler Net that chooses k representative samples out of a pool of d samples.

Suppose we want to obtain k samples from a pool of d samples. As shown in Figure 8, the Subsampler Net first starts by setting a sample to 0 (setting both the real and imaginary parts of that sample to 0) in a batch of training examples

and evaluating it using the Ranker Model, which will then provide us with a classification accuracy for that batch. Setting a sample to 0 will mean that the input neurons to the model for the two features corresponding to that sample are dead, which allows us to simulate the removal of that sample from the signal as all the weights from these neurons in the input layer will not contribute to the outcome of the model, as examined later on in this section. This is done d times by setting each of the d samples to 0. After evaluating each of the d samples, we are left with d classification accuracies that correspond to the ability of the model to classify the signal if each of the samples were to be removed. A low accuracy means that the sample was crucial for classification and a high accuracy means that the ability of the model to classify the modulation type was not significantly affected by the removal of the sample. Thus, the most important sample, which is the sample whose removal results in the lowest classification accuracy, is then permanently set to 0 for this batch of training examples and added to the final set of samples. Now, we are left with $d-1$ samples and this process is repeated until we are finally left with k samples.

This is similar to FQI as described in Section V-B4 and other weight-based feature selection methods [45] that rely on the property of neural networks that more salient features possess weights of higher magnitude. This property has been extensively documented in [46], [47], and [48] and is also demonstrated through a toy example in Section VII-C. Wrapper feature selection methods rank the set of features by removing each feature and evaluating the performance of a classifier trained with the remaining features. This is done separately for each feature by removing that feature from the training set, training the model on the training set consisting of the remaining features, and evaluating the performance on the cross-validation set also consisting of the remaining features. Thus, to remove one feature from a set of d features, the classifier has to be trained d times as performed in Recursive Feature Elimination (RFE) [49]. This is done iteratively and since the classifier has to be trained a large number of times at each iteration, wrapper methods are computationally inefficient. For instance, in the case of greedy forward wrapper feature selection schemes, the classifier has to be trained $\sum_{i=d-k+1}^{d} i$ times, where d is the total number of features and k is the number of features to be selected. That being said, we can simulate the removal of a sample by setting the sample in the input data to 0 and evaluating it on a pre-trained model trained with all the samples. The weights connected to this sample in the input layer will not contribute to the outcome of the model and if these weights significantly influenced the output of the pre-trained model, then the pre-trained model will experience a degradation in its ability to classify the input data. Since more salient samples possess weights of higher magnitude, these weights influence the output to a greater extent and setting the values of more salient samples to 0 in the input will result in a greater degradation in the ability of the pre-trained model to classify the input compared to when the same is done for less salient features. This can be measured using the loss or the accuracy given in the output layer, where a greater loss or a lower accuracy corresponds to a greater degradation. Therefore, the Subsampler Net iteratively uses the same pre-trained Ranker Model to rank samples by simulating the degradation in the ability to classify the input when that sample is removed, where a sample whose removal results in a worse classification has a better ranking. The main differences between the Subsampler Net and FQI are that FQI does not iteratively select samples, and that FQI always uses the squared error, whereas the Subsampler Net's ranking depends on the loss function of the Ranker Model. Moreover, we found that normalizing the data by setting the mean to 0 and the variance to 1 helps this process because when we set the input of a sample to 0, we are effectively setting the input to the mean, and lower variances in the data now manifest as lower weights in the input layer [50].

While attempting this method, we observed that the sample indices chosen for batches of training examples belonging to the same SNR value were the same in most cases, while they were likely to be distinct from the sample indices chosen for batches of training examples belonging to different SNR values. Therefore, we first divide the training set into 20 batches based on the SNR value (-20 dB to 18 db) and run each batch through the Subsampler Net and obtain a set of k sample indices for each SNR. Finally, we train a new architecture based on the training set with k samples obtained after subsampling. Once we know the sample indices that belong to the set of salient features, the same intervals in time can be used to subsample all signals for the testing dataset, which allows us to reduce the high power requirement burden of high sampling rates when implemeting this in hardware. Note that to apply the presented algorithm in a real system, the SNR value has to be estimated in order to know the correct set of samples to feed the classifier. We provide a pseudo code for the Subsampler Net in Algorithm 1.

Algorithm 1: Subsampler Net

Inputs: k: Final No. of samples; trainSet: Training Dataset; rankerModel: Trained Model that ranks samples

Outputs: sampleList: List of k selected sample indices

function SUBSAMPLERNET(k, trainSet, rankerModel)
    Initialize sampleList to an empty list
    for $i = 0$ *to* k **do**
        Initialize accList to an empty list
        Set candidateList as set of sample indices not in sampleList
        for j *in candidateList* **do**
            Set sample with index j to 0 in trainSet
            accuracy = rankerModel(trainSet)
            Append (j, accuracy) to accList
            Set sample with index j back to original value
        end
        Sort accList by order of increasing accuracy
        Append sample index with lowest accuracy in accList to sampleList
        Permanently set this sample to 0 for all examples in trainSet
    return (sampleList)
end function

We attempted to perform the proposed method using each of the four architectures described in Section IV as the Ranker Model and then trained the ResNet architecture using the subsampled data. These results can be observed in Figure 9. Note that we obtained similar results when using the CNN, LSTM, and CLDNN architectures for training and classification of the subsampled data. At this point, it is also worth noting that we attempted - for fair comparison - to perform the subsampling and feature selection techniques described in Section V, while applying each technique separately for each SNR value. Also note that for the Uniform, Random, and Magnitude Rank subsampling techniques stated in Section V, this would make no difference for the obvious reason that the subsampling strategy does not depend on other examples

in the dataset. Also, the accuracy values remained approximately the same for PCS. As for the feature selection methods, the accuracies dropped for the Laplacian Score, Fisher Score, and RFS while the accuracy slightly increased at high SNR for FQI.

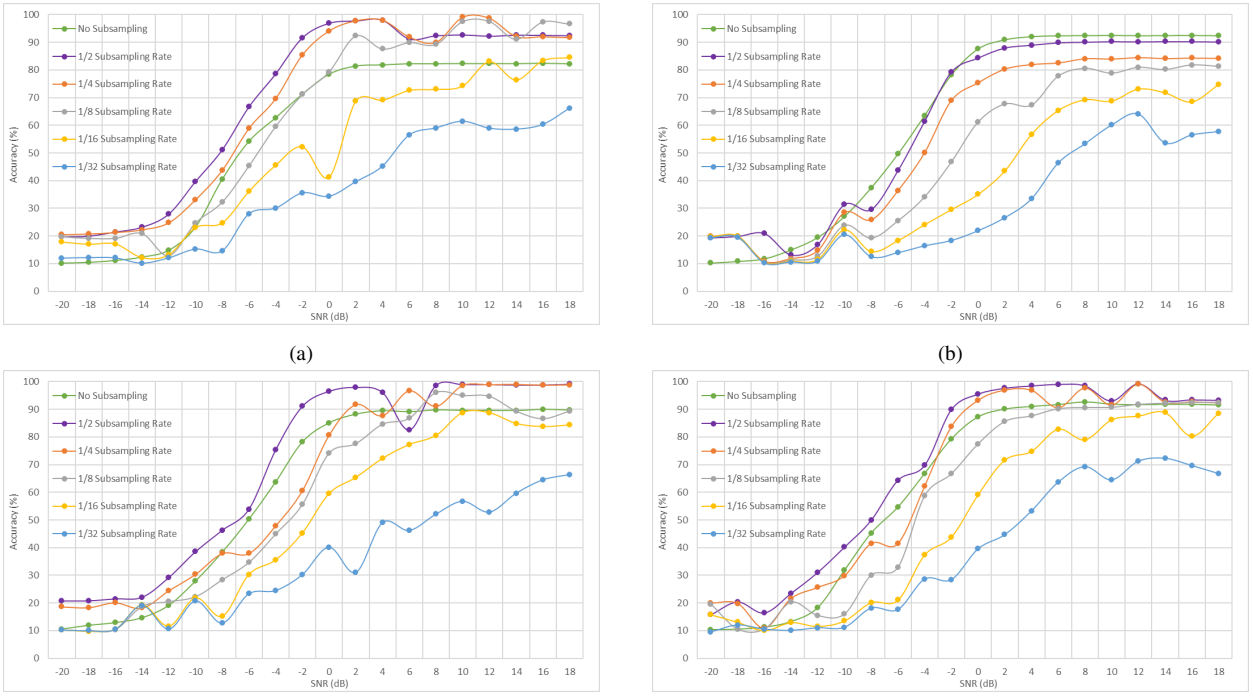


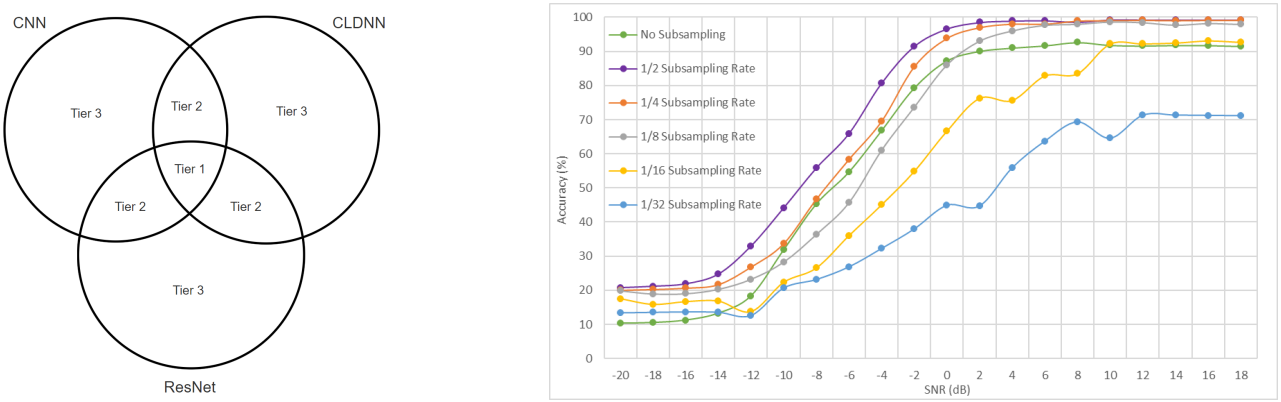
Fig. 9: Accuracy vs SNR for ResNet with Data Driven Subsampling using (a) CNN, (b) LSTM, (c) CLDNN, and (d) ResNet as the Ranker Models

B. The Holistic Subsampler: The Best of All Worlds

As observed in Figure 9, using the CNN subsampler leads to good performance at low SNR values, using the CLDNN subsampler leads to good performance at high SNR values, while using the ResNet subsampler performs well at intermediate SNR values. The LSTM subsampler, on the other hand, leads to inferior performance at all SNR values as the classification accuracy significantly drops for lower subsampling rates. As will be illustrated in Section VII, Subsampler Nets where the Ranker Model overfits the training data are expected to be better at evaluating the relative importance of the samples. The LSTM, unlike the other architectures, is less deep and as such, is less likely to overfit, which we believe is the reason behind its poor performance as the Ranker Model in a Subsampler Net. At high SNR values, where the noise is relatively low, the temporal relations across the signal samples are more prominent, and hence, are more critical for modulation classification as examined in [26]. We believe that this is what allows the CLDNN to perform well as the Ranker Model at high SNR values. On the other hand, at low SNR values, where the noise power is relatively high, long-term temporal relations across samples belonging to the same signal are obscure, and are less likely to facilitate accurate pattern recognition, whereas short term patterns - that can be captured by convolutional kernels - are more likely to distinguish one modulation type from another [23]. We think that this is why the CNN performs well as the Ranker

Model at low SNR values. At intermediate SNR values, both long-term and short-term temporal relations could be equally pronounced and as such, the ResNet performs well as the Ranker Model due to its flexibility in detecting both types of patterns [51].

We hence observe from the presented results that there is a need to benefit from the different Ranker Models across the entire range of SNR values. This is why we introduce the notion of the Holistic Subsampler, which combines the ability of all three Ranker Models in order to achieve high accuracies at all SNR values. A Holistic Subsampler contains three Ranker Models, which comprises of the CNN, the CLDNN, and the ResNet. After the set of samples are collected for each of the three Ranker Models, we divide these samples into three tiers.



(a) (b)

Fig. 10: (a): Venn diagram that depicts the division of the tiers for sample ranking. (b): Accuracy vs SNR for the ResNet classifier with Data Driven Subsampling using the Holistic Subsampler.

The Venn diagram in Figure 10a illustrates the division of the tiers. Tier 1 consists of the intersection of all three sets of samples, Tier 2 consists of the samples that belong to two of the three sets of samples, and Tier 3 consists of samples that are not common among any of the three sets of samples. The samples within each of the tiers are sorted according to the sum of the classification accuracy values that occur when that sample is removed using the corresponding Ranker Models. For example, for Tier 2 samples, we sort the samples using the sum of the two obtained classification accuracy values when the sample is removed (set to 0). The Holistic Subsampler is a recursive algorithm that first selects the best sample, then sets the value of the corresponding sample index to 0 for the whole training set, and calls itself again to find the next best sample. This is done until the desired number of samples k is reached. To find the best sample, the top sample - corresponding to the lowest sum of classification accuracy values - is selected from Tier 1. If Tier 1 is empty, then the top sample from Tier 2 is selected. If Tier 1 and Tier 2 are both empty, then the top sample from Tier 3 is selected.

C. ϵ-Greedy Search: The Final Piece of The Puzzle

Observing Figure 10b, we can see that below the Nyquist rate, there are certain ranges of SNR values at which the Holistic Subsampler is unable to prevent sudden drops in classification accuracy. We believe that this is because unlike conventional wrapper feature selection techniques that take into account potential complex dependencies between features,

the Subsampler Net merely selects the best sample without regard to how the selection of a subsequent sample will affect the importance of the currently selected sample to the classification task. We chose to do this in the interest of saving training time of Subsampler Nets in order to make the implementation of the proposed data driven subsampling method feasible using low-power communication devices. Now, we are left with the problem of dealing with these large drops in classification accuracy. We implement a variant of the ϵ-greedy algorithm [52] in order to explore candidate combinations for subsequent best samples while taking into account dependence relationships between the selected samples.

Algorithm 2: ϵ-Greedy Search

Inputs: k: Final No. of samples; ϵ: exploration factor that is the fraction of total samples to be explored; prevSnrAcc: Classification accuracy at the preceding SNR value; trainSet: Training Dataset

Outputs: finalIdx: Set of sample indices whose removal leads to the lowest accuracy among combinations explored; finalSNRAcc: Accuracy obtained using trainSet when the sample indices in sampleIdx are selected

function ϵ-GREEDY(k, ϵ, prevSnrAcc, trainSet)

Call SubsamplerNet using CNN, CLDNN, and ResNet as the Ranker Models

Set sampleIdx as the set of the k sample indices selected by the Holistic Subsampler

Set currSnrAcc as accuracy of ResNet architecture when trained with selected samples

if $k = 0$ **then**

Return (sampleIdx, currSnrAcc) if currSnrAcc $>$ prevSnrAcc

Return (NULL, NULL) otherwise

else

for i = *0 to* $\min(k,\epsilon d)$ **do**

Set trainSet[sampleIdx[i]] to 0

Set (finalIdx, finalSnrAcc) = ϵ-GREEDY($k-1$, ϵ, prevSnrAcc, trainSet)

Set trainSet[sampleIdx[i]] back to original value

Add sampleIdx[i] to finalIdx

Return (finalIdx, finalSnrAcc) if returned values are not NULL

done

Return (NULL, NULL)

end

end function

According to Algorithm 2, we introduce ϵ, the exploration factor that determines the number of candidate samples considered for selection at each step. If $\epsilon = 0.1$, then in every step, we explore the 10% best samples according to the ranking provided by the Holistic Subsampler. This is unlike the conventional ϵ-greedy algorithm, where ϵ represents the probability that the decision taken deviates from the top greedy choice. Our variant of the algorithm explores all of the top routes and ϵ is the parameter that determines the number of top routes explored. The Multi-Armed Bandit problem [53], which is one of the most popular applications of the traditional ϵ-Greedy Algorithm, is based on a scenario where there is a single top choice and all the other choices are of the same importance before exploration. However, in our case, the choices apart from the top choice have rankings that reflect their relative importance. Therefore, unlike the Multi-Armed

Bandit problem, we do not need to waste time exploring unknown paths, and this is the rationale behind our deviation from the conventional ϵ-Greedy Algorithm.

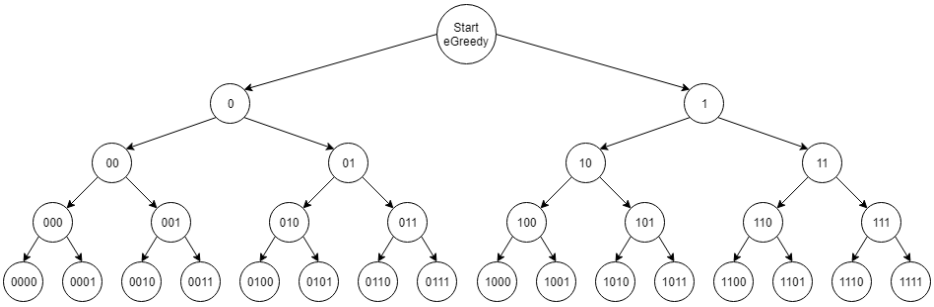


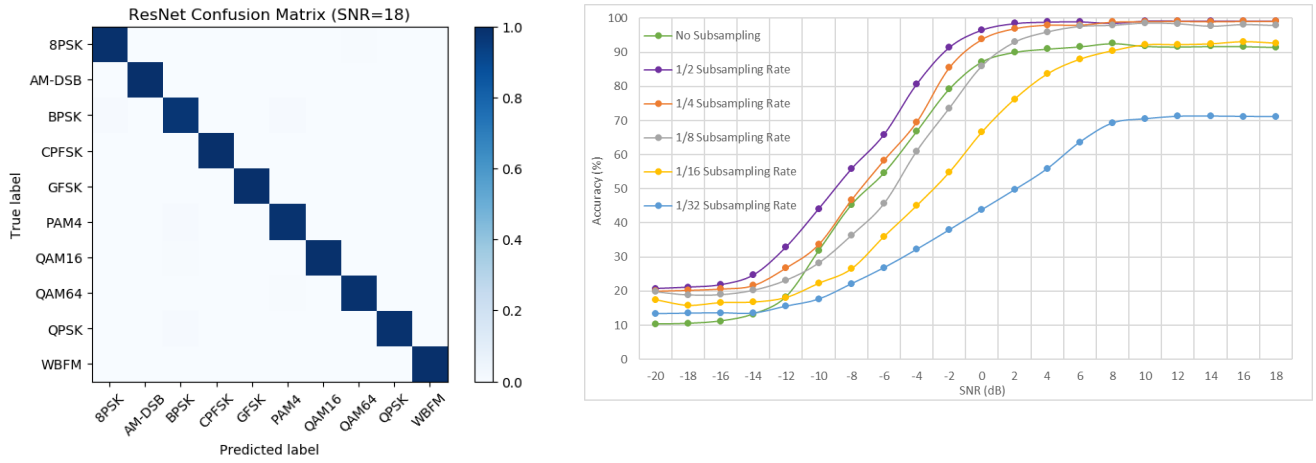
Fig. 11: Illustration of ϵ-Greedy Search with $\epsilon = \frac{1}{64}$ and a Subsampling rate of $\frac{1}{32}$. A 0 corresponds to removing the best sample, while a 1 corresponds to removing the second best sample, according to the ranking of the Holistic Subsampler. The samples are re-ranked after each removal.

As observed in Figure 11, the ϵ-Greedy Search can be represented as a traversal algorithm over an ϵd-ary tree whose depth is equal to the desired number of samples k, where the subsampling rate is $\frac{k}{d}$. Each node in the tree corresponds to the selection of a combination of sample indices. The nodes at the same depth are arranged by increasing order of accuracy when removed, which implies that the combination of samples corresponding to the left child of a node has higher priority than that corresponding to the right child of the same node, for the case when $\epsilon d = 2$. The combination of samples at depth k contains k samples, represented by an ϵd-ary vector of length k. For the example in Figure 11, $\epsilon d = \frac{128}{64} = 2$, and hence, each node is represented by a binary vector. A 0 corresponds to removing the best sample, while a 1 corresponds to removing the second best sample according to the ranking of the Holistic Subsampler, and the samples are re-ranked after each removal. For example, the leaf node 1001 corresponds to first removing the second best sample (node 1), then re-ranking the samples, then removing the best sample (node 10), then re-ranking the samples, then removing the best sample (node 100), then re-ranking the samples, and finally selecting the second best sample (node 1001).

It is worth mentioning **the Holistic Subsampler, on its own, is just a special case obtained by setting** $\epsilon = \frac{1}{d} = \frac{1}{128}$. The Holistic Subsampler greedily chooses the best sample at each iteration, which leads to the leftmost leaf of the tree in Figure 11. This corresponds to node 0000 because when $\epsilon = \frac{1}{128}$, only a tree with a single branch is formed, which leads to the formation of the nodes 0, 00, 000, 0000, in that order. We are merely expanding the scope of exploration of the Holistic Subsampler when we increase the value of ϵ.

The root node does not represent any sample and is added just for the sake of illustration of how the tree is formed. The leaf nodes are searched from left to right until a classification accuracy that is satisfactory is reached. This is expected to stop faster than conventional wrapper feature selection methods because the left child of a node is of higher importance than the right child and thus, we are expected to find a combination leading to satisfactory accuracy before traditional wrapper feature selection techniques. This is primarily due to the fact that most conventional wrapper feature selection methods do not leverage the power of transferability of deep neural networks, as will be discussed in Section VII. These feature selection methods retrain the classifier used to perform the ranking of the features after each feature is greedily removed, which takes an extremely large amount of time when the number of features is large.

In Figure 11, the subsampling rate used is $\frac{1}{32}$, which leads to the selection of four samples since each sample vector is composed of 128 samples. Here, a sample is selected at each level in the tree, which is why the tree has a depth of four. Additionally, $\epsilon = \frac{1}{64}$, which leads to the ϵ-Greedy algorithm searching for the best two samples at each level of the tree. Observing Figure 10b, we see that the slope of the classification accuracy curve for the Holistic Subsampler becomes negative at -12, 4, and 8 dB with a $\frac{1}{16}$ subsampling rate and at -12, 2, and 10 dB with a $\frac{1}{32}$ subsampling rate. For the 4 and 8 dB SNR values with $\frac{1}{16}$ subsampling rate, we employed ϵ-Greedy Search with an $\epsilon = \frac{1}{64}$ to fix this issue, as we suspected that the drop in accuracy with a higher SNR value is due to unexplored sample dependencies. We did the same for 10 dB SNR values with $\frac{1}{32}$ subsampling rate. For the -12 dB SNR value with both $\frac{1}{16}$ and $\frac{1}{32}$ subsampling rates, we had to employ ϵ-Greedy Search with an $\epsilon = \frac{1}{32}$ because having $\epsilon = \frac{1}{64}$ was insufficient. We did the same for the 2 dB SNR value at the $\frac{1}{32}$ subsampling rate. After the application of ϵ-Greedy Search, we were finally able to resolve the sample dependencies issue and the final results can be observed in Figure 12b.



(a) (b)

Fig. 12: (a): ResNet Confusion Matrix after Data Driven Subsampling using a Subsampling Rate of $\frac{1}{2}$ at 18 dB SNR. (b): Accuracy vs SNR for ResNet with Data Driven Subsampling using the Holistic Subsampler after ϵ-Greedy Search.

From our results, we can draw the conclusion that the sample interaction issue is prominent at low subsampling rates. We belive that this is because of underfitting at such rates. Fortunately, ϵ-Greedy Search is ideal at low subsampling rates because the depth of the tree is equal to the final number of samples that are to be collected and as such, the depth of the tree decreases as the subsampling rate decreases. That being said, the ϵ-Greedy Search could become infeasible for higher subsampling rates, unless the search stops early as we found to be the case in our experiments. For example, with $\epsilon = \frac{1}{64}$, we will deal with binary trees and as a result, with every extra sample that is to be collected, the time taken to perform ϵ-Greedy Search doubles as the tree grows twice in size, if all combinations corresponding to leaf nodes were to be explored. Therefore, our variant of the ϵ-Greedy Search has a time complexity of the Order $O((\epsilon d)^k)$, if the number of explored combinations has the same order as the number of the tree leaves.

D. Putting Everything Together

Now that we have described the concepts of the Holistic Subsampler and ϵ-Greedy Search, we can specify the proposed approach. To recapitulate, we first initialize ϵ to $\frac{1}{128}$ and invoke the ϵ-Greedy Search. As mentioned earlier, this is the same as invoking the Holistic Subsampler on its own, which greedily selects the best k samples in a recursive manner.

Next, we proceed to the next SNR value's training set, where the accuracy obtained for the previous SNR value will serve as the stopping criterion. For the -20 dB SNR case, there is no stopping criterion as the dataset does not contain data for SNR values lower than -20 dB.

Now, the ϵ-Greedy Search is invoked with an ϵ value of $\frac{1}{128}$ for this next training set belonging to the next SNR value. If the accuracy is lower than the accuracy for the previous SNR value, then ϵ-Greedy Search is invoked again after doubling the ϵ value to $\frac{1}{64}$. The ϵ-Greedy Search stops exploring once the accuracy is greater than the accuracy obtained for the previous SNR value. We repeatedly double ϵ and invoke ϵ-Greedy Search until this stopping criterion is met. The pseudocode for this strategy is given in Algorithm 3.

Algorithm 3: Data Driven Subsampling

Inputs: k: Final No. of samples; trainSet: Training Dataset

Outputs: idxDict: Dictionary with SNR as key and sets of k sample indices each as values

function DATADRIVENSUBSAMPLING(k, trainSet)
    Divide trainSet based on SNR
    Initialize idxDict as an empty dictionary
    Initialize prevSnrAcc to 0
    for *snrValue in set of SNR values* **do**
        Initialize $\epsilon = \frac{1}{128}$
        Initialize snrIdx as NULL
        while *snrIdx is NULL* **do**
            Set (snrIdx, snrAcc) = ϵ-GREEDY(k, ϵ, prevSnrAcc, trainSet)
            Set $\epsilon = 2\epsilon$
        done
        Set snrIdx as the value to snrValue key in idxDict
        Set prevSnrAcc as snrAcc
    done
    return idxDict
end function

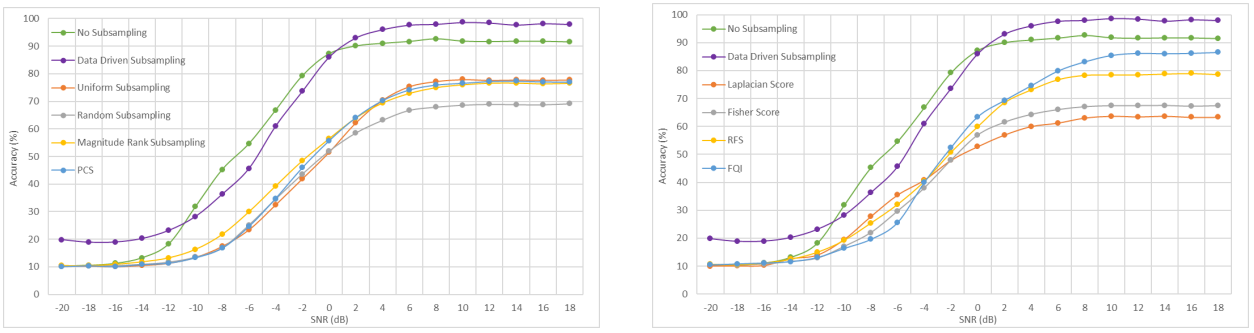
The function described in Algorithm 3 will return the selected set of k sample indices for each SNR value. Note that every time the ϵ-Greedy Search is invoked, the ϵ value is doubled. As described in Section VI-C, this is the same as searching for combinations of sample indices in a tree that has twice the arity. Based on the loss in classification accuracy, we can choose the smallest set of k samples that gives us a classification accuracy higher than the given classification accuracy requirement. For instance, 20 is the smallest number of samples that can be selected such that the classification

accuracy (99.08% with 20 samples) has to be higher than 99% at 18 dB SNR. Similarly, 8 is the smallest number of samples that can be selected given that the classification accuracy (91.67% with 8 samples) has to be higher than 90% at 18 dB SNR. Additionally, as a result of subsampling, the training time of the classifier is reduced. We show the reduction in training times and high SNR classification accuracy of the ResNet classifier for different subsampling rates in Table II. **Note that a subsampling rate as low as $\frac{1}{16}$, which corresponds to the sub-Nyquist regime and results in approximately $\frac{1}{3}$ of the original training time, still results in a classification accuracy higher than that without subsampling**.

TABLE II: Comparison of Training Times for the ResNet after Data Driven Subsampling.

Samples	Time per Epoch	Epochs	Total Training Time	Accuracy (18 dB SNR)
All	32.0642s	37	1186.3754s	91.49%
1/2	26.9615s	33	889.7295s	99.27%
1/4	24.2615s	29	703.5835s	99.13%
1/8	21.8361s	27	589.5747s	97.94%
1/16	17.1167s	26	445.0342s	92.67%
1/32	11.6828s	24	280.3827s	71.14%

Additionally, the Accuracy vs SNR comparisons of Data Driven Subsampling with the other techniques mentioned in Section V can be observed in Figure 13 for a subsampling rate of $\frac{1}{8}$, which falls in the sub-Nyquist regime. We observe that the performance of Data Driven Subsampling is superior to all the considered conventional Subsampling and Feature Selection techniques at all SNR values.



(a) (b)

Fig. 13: Accuracy vs SNR comparisons of Data Driven Subsampling with (a) Conventional Subsampling Techniques and (b) Feature Selection Techniques for the ResNet classifier at $\frac{1}{8}$ subsampling rate.

VII. Discussion

A. *Exploiting Transfer Learning for Data Driven Subsampling*

We note how the Holistic Subsampler achieves better results than any individual Subsampler Net, even though the final classifier relies on a single ResNet architecture. Furthermore, even though each of these architectures is trained to classify the data when all samples are present at the input, when used as Subsampler Nets, one or more of these samples are

set to 0. Hence, we use the trained deep neural network classifiers in two ways other than their intended application that they are trained on: 1- They are used to select samples for another classifier, 2- They are used with only a subset of samples present. This is only possible because of the transferability property of these deep neural network architectures. In general, we believe that exploiting transferability has great potential for various wireless communication tasks that rely on processing received signal samples, and could be beneficial in guiding the designs of special purpose hardware chips for using deep learning in wireless communication systems.

B. Subsampling Leads to Higher Accuracies

In the context of Machine Learning, subsampling is essentially the same thing as feature selection, which is a concept that allows us to build simpler and more comprehensible models. Since feature selection facilitates the removal of less prevalent features or the selection of more prominent features, it is widely used for reducing overfitting as a technique that can be applied for input preprocessing. The Holistic Subsampler carries out the same task, but unlike conventional feature selection techniques, is able to leverage the power of three deep learning models - that are all good for the considered modulation classification task - to find the most important samples by analyzing the training dataset. In Section IV, we saw that all the deep learning architectures suffered from the same drawback of the AM-DSB/WBFM and QAM16/QAM64 misclassification. In this section, we will be using Principal Component Anaysis (PCA) [54] and t-Distributed Stochastic Neighbor Embedding (t-SNE) [55] to visualize how subsampling allows us to reduce overfitting, which leads to higher classification accuracies with fewer samples, particularly for the aforementioned class pairs.

As mentioned in Section V-A4, PCA is a classic dimensionality reduction algorithm where the principal components of a dataset are computed by performing the eigendecomposition of its covariance matrix. The estimated covariance matrix Σ of the training dataset X with n training examples is computed using (9) below.

$$\Sigma = \frac{(X - \bar{X})^T (X - \bar{X})}{n - 1}, \tag{9}$$

where $\bar{X}$ is the mean of X. We then compute the matrix of eigenvectors P of this covariance matrix Σ, and can compute the complete principal components decomposition D of the training data X using simple matrix multiplication.

$$D = XP. \tag{10}$$

If we want to reduce the d-dimensional training data X to k-dimensional data, we compute (10) by replacing P with the first k columns of P. Therefore, PCA simply uses an orthogonal transformation to project the dataset onto a set of linearly independent directions known as principal components [32].

t-SNE is a commonly used non-linear dimensionality reduction algorithm that aims at preserving the local neighborhood structure when reducing a high dimensional space to a low dimensional space by converting pairwise distances to pairwise joint distributions. Additionally, t-SNE optimizes low dimensional embeddings to match the high and low dimensional joint distributions [56]. If $\{x_i\}_{i=1}^n$ are high dimensional data points (training examples), and $\{y_i\}_{i=1}^n$ are the corresponding low dimensional embedding points, t-SNE defines the low dimensional joint distribution of points i, j as follows:

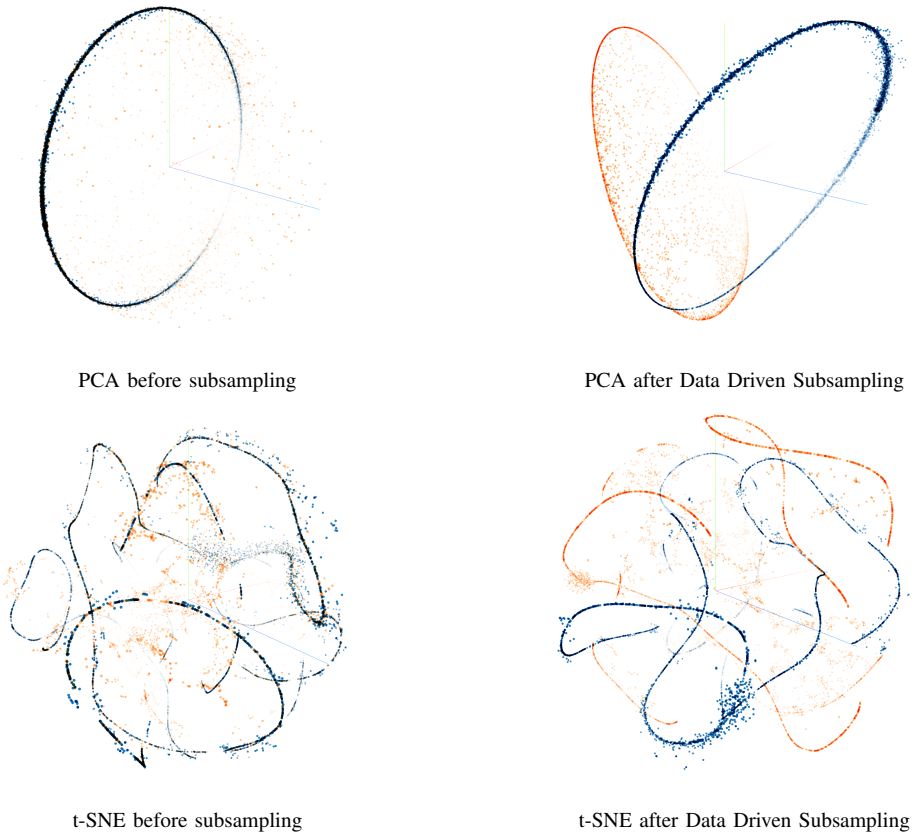
$$q_{i,j} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{s \neq t} \left(1 + \|y_s - y_t\|^2\right)^{-1}}. \tag{11}$$

Here, the high dimensional joint distribution is defined as the symmetrized conditional $p_{i,j} = \frac{p_{i|j}+p_{j|i}}{2n}$, where

$$p_{i|j} = \frac{e^{\frac{-\|x_i - x_j\|^2}{2\sigma_j^2}}}{\sum_{s \neq j} e^{\frac{-\|x_s - x_j\|^2}{2\sigma_j^2}}}, \tag{12}$$

and σ_j determines the radius of the significant local neighborhood around x_j. A *perplexity* value determines the values of these radii. t-SNE then optimizes $\{y_i\}_i$ to minimize the Kullback-Leibler distance between the low dimensional distribution Q and the high dimensional distribution P, defined in (13).

$$\text{KL}(P||Q) = \sum_{i \neq j} p_{i,j} \log \frac{p_{i,j}}{q_{i,j}}. \tag{13}$$



PCA before subsampling

PCA after Data Driven Subsampling

t-SNE before subsampling

t-SNE after Data Driven Subsampling

Fig. 14: PCA and t-SNE visualizations of the training dataset for the AM-DSB (blue) and WBFM (orange) classes before and after Data Driven Subsampling with a subsampling rate of $1/2$ at 18 dB SNR.

To demonstrate that Data Driven Subsampling leads to higher classification accuracies with fewer samples, we first subsample the training dataset at 18 dB SNR with a subsampling rate of $1/2$. Before subsampling, we had 128 samples, each with a real and complex part, which is a total of 256 features. After subsampling, we have 64 samples, which is a total of 128 features. Finally, we implement PCA using (9) and (10) to reduce the dimensions of the training dataset to 3-Dimensions so that they can be plotted for better visualization. Similarly, we implement t-SNE using (11), (12), and (13).

We chose to implement both PCA and t-SNE because PCA portrays a better distinction between AM-DSB and WBFM in Figure 14 while t-SNE depicts a better distinction between QAM16 and QAM64 in Figure 15. Note that in order to avoid confusion, we only plot the training examples for the modulation classes that are under consideration. Additionally, these can also be visualized on TensorFlow Embedding Projector[3].

In Figure 14, we can see that the training examples that belong to the AM-DSB (blue) and WBFM (orange) classes are blended together in the PCA and t-SNE plots before subsampling. However, after Data Driven Subsampling, the PCA and t-SNE plots show that these training examples can now be more distinctly differentiated as the AM-DSB (blue) and WBFM (orange) training examples are now clustered within the same class instead of being fused with training examples of the other class. Here, both the t-SNE plots were generated with a perplexity value of 30, a learning rate of 10, and were run for 250 iterations.

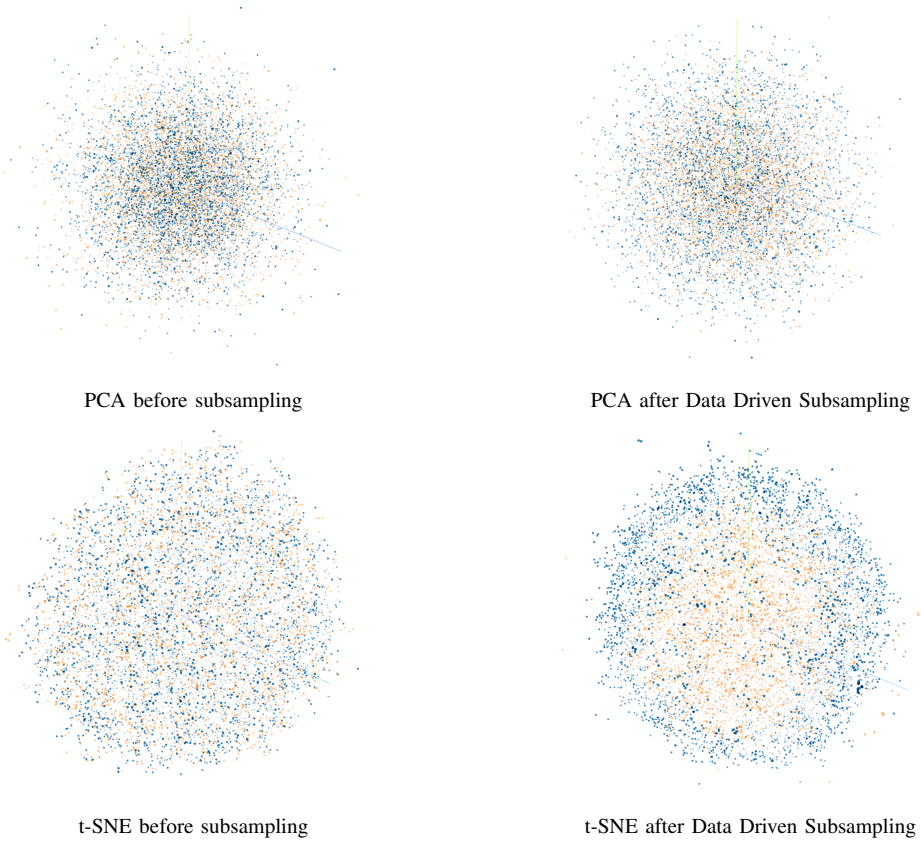


Fig. 15: PCA and t-SNE visualizations of the training dataset for the QAM16 (blue) and QAM64 (orange) classes before and after Data Driven Subsampling with a Subsampling Rate of $1/2$ at 18 dB SNR.

Similarly, in Figure 15, we can see that the training examples of the QAM16 (blue) and QAM64 (orange) modulation classes are more distinguishable after Data Driven Subsampling. Particularly in the t-SNE plots, we can clearly see that the training examples are clustered together within the same class with QAM16 training examples clustered at the center and

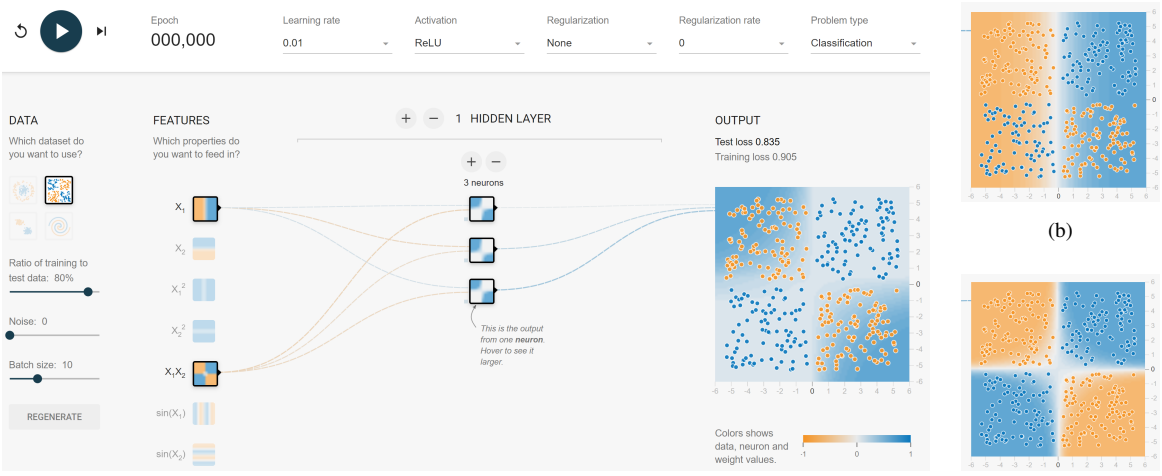
[3]https://projector.tensorflow.org/

QAM64 training examples clustered around the QAM 16 training examples. Here, both the t-SNE plots were generated with a perplexity value of 20, a learning rate of 10, and were run for 250 iterations.

We believe that the classification accuracies of the considered deep neural network architectures - described in Section IV - are only approximately 93% at high SNR, because they tend to overfit the training data for these two pairs of classes. However, using Data Driven Subsampling, we are able to select the samples that are the most different among these pairs of classes, thereby reducing the overfitting that takes place. In other words, since these modulation classes are more distinguishable after Data Driven Subsampling, deep neural network architectures can form better decision boundaries, which is what allows them to achieve a higher classification accuracy using Data Driven Subsampling.

C. Overfitting facilitates better subsampling

In Section VI-A and as shown by other work regarding weight-based feature selection (see e.g., [57] and [50]), we elaborated on how more salient features possess higher magnitudes of weights in the input layer than features that are less salient, which is the property of neural networks that serves as the basis for the Subsampler Net. The performance of a Subsampler Net heavily depends on the performance of the RM that ranks the features. In some cases, however, even the state of the art models do not have high classification accuracies. In such cases, we can obtain better feature selection results with a Subsampler Net by overfitting the RM on the training set.

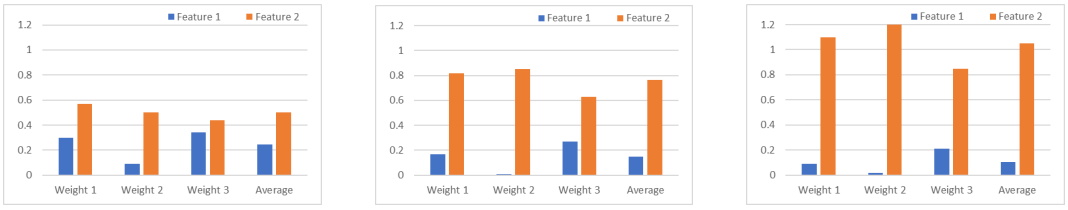


(a) (b) (c)

Fig. 16: (a): Toy example from TensorFlow Playground; (b) Feature 1 and (c) Feature 2.

We will demonstrate this using the toy example illustrated in Fig. 16 that portrays the architecture used for the RM along with the corresponding hyperparameters. Here, feature 1 is x_1 and feature 2 is x_1x_2. Feature 2 is more salient than feature 1 as it is able to form a decision boundary that allows for better classification of the data (shown in (c)), while feature 1 cannot (shown in (b)). Each of these features have three weights in the input layer and as expected, the weights connected to feature 2 manifest into weights of higher average magnitude than those belonging to feature 1 as shown in Fig. 17. As the number of training epochs increases, the difference in the average magnitudes of the weights increases.

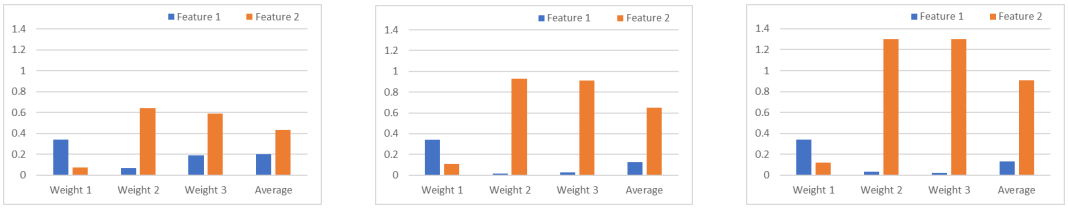
This implies that the RM will be able to better rank the saliency of features because the difference between the accuracies of more and less salient features increases. Thus, we can overfit the RM on the training set by training it for a large number of epochs without regularization to give the RM greater discriminative power that enables better subsampling.



(a) (b) (c)

Fig. 17: Input layer weight magnitudes after training for (a) 10, (b) 100, and (c) 1000 epochs.

However, the values that these weights manifest to are dependent on their initialization before training begins. In some rare cases, as seen in Figure 18, all the weights that belong to the more salient feature might not always possess magnitudes higher than the corresponding weights of the less salient feature after training. We attempted to randomly initialize weights and train the neural networks about 15 times and this scenario occurred only once. It is also worth noting that the trend of increase in the difference between the average magnitude of the weights for features 1 and 2 still holds in such cases. That being said, we can also follow other weight initialization strategies to reduce the probability of this scenario occurring, and this is an issue that we plan to investigate in future work.



(a) (b) (c)

Fig. 18: Input layer weight magnitudes after training for (a) 10, (b) 100, and (c) 1000 epochs.

D. Sensitivity to SNR Estimate

Even though both the final classifier and the ranker models used in Data Driven Subsampling are trained using the whole training set across all considered SNR values, the selected set of sample indices is different for each SNR value, and hence, we expect a real time system employing this method to have an accurate estimate of the SNR value, in order to know the right set of sample indices. We made this choice, as we found it to deliver a significantly superior performance to the extreme alternative, where the same set of sample indices is selected for all SNR values. In future work, we plan to investigate the impact of small errors in such an estimate, by comparing the different sets of sample indices selected by Data Driven Subsampling for adjacent SNR values. We plan to also benefit from analyzing these sets of sample indices, to better understand the roles of different ranker models at different SNR values.

VIII. Concluding Remarks

In this work, we considered the problem of recognizing one of ten modulation types using the RadioML2016.10b dataset of [23]. We first presented four deep neural network architectures that deliver superior classification accuracy to the state of the art, namely a CNN, LSTM, CLDNN and ResNet. We then presented Data Driven Subsampling: a subsampling method that employs three of these architectures for selecting a set of samples that maximize the classification accuracy via recursive simulations. Our results indicate that using Data Driven Subsampling with a ResNet classifier leads to very high classification accuracy values, even in the sub-Nyquist regime, where we achieve an almost perfect classification (accuracy above 99%) at high SNR. We also noted the drastic reduction in the classifier's training time as a result of subsampling. We plan to further investigate in future work the potential of employing deep learning for subsampling in wireless communication systems, as we believe that the insights distilled from this work carry practical significance beyond the considered modulation classification task.

References

[1] X. Liu, D. Yang, and A. El Gamal, "Deep neural network architectures for modulation classification," in *Proc. IEEE Asilomar Conference on Signals, Systems and Computers*, 2017.

[2] J. Sills, "Maximum-likelihood modulation classification for psk/qam," in *Proc. IEEE Military Communications Conference (MILCOM)*, 1999.

[3] A. Polydoros and K. Kim, "On the detection and classification of quadrature digital modulations in broad-band noise," *IEEE Transactions on Communications*, vol. 38, no. 8, pp. 1199–1211, 1990.

[4] P. Sapiano and J. Martin, "Maximum likelihood PSK classifier," in *Proc. IEEE Military Communications Conference (MILCOM)*, 1996.

[5] B. F. Beidas and C. L. Weber, "Asynchronous classification of MFSK signals using the higher order correlation domain," *IEEE Transactions on communications*, vol. 46, no. 4, pp. 480–493, 1998.

[6] P. Panagiotou, A. Anastasopoulos, and A. Polydoros, "Likelihood ratio tests for modulation classification," in *MILCOM 2000 Proceedings. 21st Century Military Communications. Architectures and Technologies for Information Superiority (Cat. No. 00CH37155)*, vol. 2. IEEE, 2000, pp. 670–674.

[7] L. Hong and K. Ho, "Antenna array likelihood modulation classifier for BPSK and QPSK signals," in *Proc. IEEE Military Communications Conference (MILCOM)*, 2002.

[8] S.-Z. Hsue and S. S. Soliman, "Automatic modulation recognition of digitally modulated signals," in *Proc. IEEE Military Communications Conference (MILCOM)*, 1989.

[9] L. Hong and K. Ho, "Identification of digital modulation types using the wavelet transform," in *Proc. IEEE Military Communications Conference (MILCOM)*, 1999.

[10] A. Swami and B. M. Sadler, "Hierarchical digital modulation classification using cumulants," *IEEE Transactions on communications*, vol. 48, no. 3, pp. 416–429, 2000.

[11] G. Hatzichristos and M. P. Fargues, "A hierarchical approach to the classification of digital modulation types in multipath environments," in *Proc. IEEE Asilomar Conference on Signals, Systems, and Computers*, 2001.

[12] S. S. Soliman and S.-Z. Hsue, "Signal classification using statistical moments," *IEEE Transactions on Communications*, vol. 40, no. 5, pp. 908–916, 1992.

[13] L. Lichun, "Comments on signal classification using statistical moments," *IEEE Transactions on Communications*, vol. 50, no. 2, p. 195, 2002.

[14] L. Mingquan, X. Xianci, and L. Lemin, "AR modeling-based features extraction of multiple signals for modulation recognition," in *Proc. IEEE International Conference on Signal Processing*, 1998.

[15] B. G. Mobasseri, "Digital modulation classification using constellation shape," in *Proc. IEEE International Conference on Signal Processing*, 2000.

[16] L. Mingquan, X. Xianci, and L. Leming, "Cyclic spectral features based modulation recognition," in *Proc. International Conference on Communication Technology (ICCT)*, 1996.

[17] E. E. Azzouz and A. K. Nandi, "Modulation recognition using artificial neural networks," *Signal Processing*, vol. 56, no. 2, pp. 165–175, 1997.

[18] K. E. Nolan, L. Doyle, D. O'Mahony, and P. Mackenzie, "Modulation scheme recognition techniques for software radio on a general purpose processor platform," in *Proc. Joint IEI/IEE Symposium on Telecommunication Systems, Dublin*, 2001.

[19] K. Kim and A. Polydoros, "Digital modulation classification: the BPSK versus QPSK case," in *Proc. IEEE Military Communications Conference (MILCOM)*, 1988.

[20] N. E. Lay and A. Polydoros, "Per-survivor processing for channel acquisition, data detection and modulation classification," 1994.

[21] C.-S. Park, J.-H. Choi, S.-P. Nah, W. Jang, and D. Y. Kim, "Automatic modulation recognition of digital signals using wavelet features and SVM," in *Proc. International Conference on Advanced Communications Technology*, 2008.

[22] L. De Vito, S. Rapuano, and M. Villanacci, "Prototype of an automatic digital modulation classifier embedded in a real-time spectrum analyzer," *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 10, pp. 2639–2651, 2010.

[23] T. O'Shea, J. Corgan, and T. Clancy, "Convolutional radio modulation recognition networks," in *Proc. International conference on engineering applications of neural networks*, 2016.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[25] T. O'Shea, T. James, T. Roy, and T. Clancy, "Over-the-air deep learning based radio signal classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

[26] T. N. Sainath, O. Vinyals, A. W. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[27] N. E. West and T. O'Shea, "Deep architectures for modulation recognition," in *International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2017.

[28] T. O'Shea and N. West, "Radio machine learning dataset generation with gnu radio," in *Proc. GNU Radio Conference*, 2016.

[29] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, and S. Pollin, "Deep learning models for wireless signal classification with distributed low-cost spectrum sensors," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 3, pp. 433–445, 2018.

[30] Y. C. Eldar, *Sampling Theory: Beyond Bandlimited Systems*. Cambridge University Press, 2015.

[31] C. Boutsidis, D. Garber, Z. Karnin, and E. Liberty, "Online principal component analysis," Available at: http://cs-www.cs.yale.edu/homes/el327/papers/opca.pdf.

[32] M. Ringnér, "What is principal component analysis?" *Nature biotechnology*, vol. 26, no. 3, p. 303, 2008.

[33] M. F. Duarte and Y. C. Eldar, "Structured compressed sensing: From theory to applications," *IEEE Transactions on Signal Processing*, vol. 59, pp. 4053–4085, 2011.

[34] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[35] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[36] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Comput. Surv.*, vol. 50, pp. 94:1–94:45, 2017.

[37] J. Li, K. Cheng, S. Wang, F. Morstatter, T. Robert, J. Tang, and H. Liu, "Feature selection: A data perspective," *arXiv:1601.07996*, 2016.

[38] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *Advances in neural information processing systems*, 2006, pp. 507–514.

[39] Q. Gu, Z. Li, and J. Han, "Generalized fisher score for feature selection," *arXiv:1202.3725*, 2012.

[40] F. Nie, H. Huang, X. Cai, and C. H. Ding, "Efficient and robust feature selection via joint 2, 1-norms minimization," in *Advances in neural information processing systems*, 2010, pp. 1813–1821.

[41] C. Ding, D. Zhou, X. He, and H. Zha, "R 1-pca: rotational invariant l 1-norm principal component analysis for robust subspace factorization," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 281–288.

[42] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection," *ACM Computing Surveys*, vol. 50, no. 6, p. 145, Dec 2017. [Online]. Available: http://dx.doi.org/10.1145/3136625

[43] K. De Rajat, N. R. Pal, and S. K. Pal, "Feature analysis: Neural network and fuzzy set theoretic approaches," *Pattern Recognition*, vol. 30, no. 10, pp. 1579–1590, 1997.

[44] A. Verikas and M. Bacauskiene, "Feature selection with neural networks," *Pattern Recognition Letters*, vol. 23, no. 11, pp. 1323–1335, 2002.

[45] T. N. Lal, O. Chapelle, J. Weston, and A. Elisseeff, "Embedded methods," in *Feature extraction*. Springer, 2006, pp. 137–165.

[46] K. W. Bauer Jr, S. G. Alsing, and K. A. Greene, "Feature screening using signal-to-noise ratios," *Neurocomputing*, vol. 31, no. 1-4, pp. 29–44, 2000.

[47] L. M. Belue and K. W. Bauer Jr, "Determining input features for multilayer perceptrons," *Neurocomputing*, vol. 7, no. 2, pp. 111–121, 1995.

[48] K. L. Priddy, S. K. Rogers, D. W. Ruck, G. L. Tarr, and M. Kabrisky, "Bayesian selection of important features for feedforward neural networks," *Neurocomputing*, vol. 5, no. 2-3, pp. 91–103, 1993.

[49] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[50] S. Ramjee and A. E. Gamal, "Efficient wrapper feature selection using autoencoder and model based elimination," *arXiv:1905.11592*, 2019.

[51] Q. Liao and T. A. Poggio, "Bridging the gaps between residual learning, recurrent neural networks and visual cortex," *CoRR*, vol. abs/1604.03640, 2016.

[52] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," *arXiv:1507.00814*, 2015.

[53] S. Bubeck, N. Cesa-Bianchi *et al.*, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends® in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

[54] G. Ivosev, L. Burton, and R. Bonner, "Dimensionality reduction and visualization in principal component analysis," *Analytical chemistry*, vol. 80, no. 13, pp. 4933–4944, 2008.

[55] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[56] Y. Cao and L. Wang, "Automatic selection of t-sne perplexity," *CoRR*, vol. abs/1708.03229, 2017.

[57] X. Wang, Y. Wang, and L. Wang, "Improving fuzzy c-means clustering based on feature-weight learning," *Pattern recognition letters*, vol. 25, no. 10, pp. 1123–1132, 2004.