Contributors

Jithin Jagannath, Nicholas Polosky, Anu Jagannath ANDRO Advanced Applied Technology, ANDRO Computational Solutions, LLC, Rome, New York, U.S.A

Francesco Restuccia, Tommaso Melodia Wireless Networks and Embedded Systems Laboratory, Northeastern University, Boston, MA, U.S.A

Acronyms

AM amplitude modulation

AMC automatic modulation classification

ANN artificial neural network

AXI Advanced eXtensible Interface

BP back-propagation

BPSK binary phase shift keying

CART classification and regression trees

CPFSK continuos phase frequency shift keying

CPU central processing unit

CDMA code division multiple access

CMAC cerebellar model articulation controller

CNN convolutional neural networks

DCNN deep convolutional neural network

DMA direct memory access

DRL deep reinforcement learning

DSA dynamic spectrum access

DSB double-sideband modulation

DL deep learning

DNN deep neural network

DP dynamic programming

FFT fast Fourier transform

FIFO first-in first-out

FPGA field-programmable gate array

FSK frequency shift keying

GFSK Gaussian frequency shift keying

GMSK Gaussian minimum shift keying

HDL hardware description language

HLS high-level synthesis

II initiation interval

IoT Internet of things

I/Q in-phase/quadrature

LO local oscillator

LSTM long short term memory

ML machine learning

MLP multi-layer perceptron

MMSE minimum mean square error

MST multi-stage training

M-QAM M-ary quadrature amplitude modulation

NLP natural language processing

OFDM orthogonal frequency-division multiplexing

PAM pulse-amplitude modulation

PCA Principal component analysis

PL programmable logic

PS processing system

PSD power spectral density

PSK phase shift keying

QAM quadrature amplitude modulation

QoS quality of service

QPSK quadrature phase shift keying

RAM random access memory

RBF radial basis function

RBFNN radial basis function neural network

RF radio frequency

RN residual network

RNN recurrent neural network

SGD stochastic gradient descent

SoC System on Chip

SNR signal-to-noise-ratio

SSB single-sideband modulation

SVM support vector machine

UF unrolling factor

WBFM wideband Frequency Modulation

WIC wireless interference classification

Chapter 13

Neural Networks for Signal Intelligence: Theory and Practice

The significance of robust wireless communication in both commercial and military applications is indisputable. The commercial sector struggles to balance the limited spectral resources with the ever growing bandwidth demand which includes multimedia support with specific quality of service (QoS) requirements. In tactical scenarios, it has always been challenging to operate in a hostile congested and contested environment. Both these scenarios can benefit from efficient spectrum sensing and signal classification capabilities. While this problem has been studied for decades, the recent rejuvenation of machine learning has made a significant footprint in this domain. Accordingly, this chapter aims to provide readers with a comprehensive account of how machine learning techniques, specifically artificial neural networks have been applied to solve some of the key problems related to gathering signal intelligence. To accomplish this, we begin by presenting an overview of artificial neural networks. Next, we discuss the influence of machine learning on the physical layer

in the context of signal intelligence. Thereafter, we discuss directions taken by the community towards hardware implementation. Finally, we identify the key hurdles associated with the applications of machine learning at the physical layer.

13.1. Introduction

According to the latest Ericsson's mobility report, there are now 5.2 billion mobile broadband subscriptions worldwide, generating more than 130 exabytes per month of wireless traffic [Ericsson Incorporated, 2018]. Moreover, it is expected that by 2020, over 50 billion devices will be absorbed into the Internet, generating a global network of "things" of dimensions never seen before [Cisco Systems, 2017]. Given that only a few radio frequency (RF) spectrum bands are available to wireless carriers [Federal Communications Commission [2016]], technologies such as RF spectrum sharing through beamforming [Shokri-Ghadikolaei et al., 2016, Vázquez et al., 2018, Lv et al., 2018], dynamic spectrum access (DSA) [Jin et al., 2018, Chiwewe and Hancke, 2017, Jagannath et al., 2018a, Federated Wireless, 2018, Agarwal and De, 2016] and anti-jamming technologies [Zhang et al., 2017, Huang et al., 2017, Chang et al., 2017] will become essential in the near future.

Software defined radios were introduced as a solution to the limitation associated with a rigid radio hardware design that prevents reconfigurability and operational flexibility. Equipping radios with the ability to learn and observe the operational scenarios to make cognitive decisions can improve spectrum sharing and spectral situation awareness. Spectrum sharing will allow radios to sense and utilize unused/underutilized spectrum to avoid spectrum conges-

tion caused by unintentional/intentional interferences. Further, signal sensing and classification can bolster the spectral knowledge of the radios to reinforce and foster situational awareness. Commercial and tactical military operators can exploit this cognitive ability to maximize the spectrum utility and provide more robust communications links.

The recent introduction of machine learning (ML) to wireless communications has in part to do with the new-found pervasiveness of ML throughout the scientific community and in part to do with the nature of the problems that arise in wireless communications. With the advent of advances in computing power and ability to collect and store massive amounts of data, ML techniques have found their way into many different scientific domains in an attempt to put both of the aforementioned to good use. This concept is equally true in wireless communications. Additionally, problems that arise in wireless communication systems are frequently formulated as classification, detection, estimation, and optimization problems; all of which ML techniques can provide elegant and practical solutions to. In this context, the application of ML to wireless communications seems almost natural and presents a clear motivation [Bkassiny et al., 2013, Jiang et al., 2017, Chen et al., 2017].

The objective of this chapter is to provide a detailed insight on the influence artificial neural network (ANN) has had on the physical layer. To begin, we provide an overview of the ANN in Section 13.2. In Section 13.3, we discuss the applications of ANN to physical layer specifically to acquire signal intelligence. Next, in Section 13.4, we discuss the implications of hardware implementations in context of ML. Finally, in Section 13.5, we discuss the open problems that may be currently debilitating the application of ML in wireless systems.

13.2. Overview of Artificial Neural Network

Before we begin, we would like to introduce some standard notations that will be used throughout this chapter. We use boldface upper and lower-case letters to denote matrices and column vectors, respectively. For a vector $\mathbf{x}$, x_i denotes the i-th element, $\|\mathbf{x}\|$ indicates the Euclidean norm, $\mathbf{x}^\intercal$ its transpose, and $\mathbf{x} \cdot \mathbf{y}$ the Euclidean inner product of $\mathbf{x}$ and $\mathbf{y}$. For a matrix $\mathbf{H}$, H_{ij} will indicate the (i,j)-th element of $\mathbf{H}$. The notation $\mathcal{R}$ and $\mathcal{C}$ will indicate the set of real and complex numbers, respectively. The notation $\mathbb{E}_{x \sim p(x)} \left[f(x) \right]$ is used to denote the expected value, or average of the function $f(x)$ where the random variable x is drawn from the distribution $p(x)$. When a probability distribution of a random variable, x, is conditioned on a set of parameters, θ, we write $p(x; \theta)$ to emphasize the fact that θ parameterizes the distribution and reserve the typical conditional distribution notation, $p(x|y)$, for the distribution of the random variable x conditioned on the random variable y. We use the standard notation for operations on sets where $\cup$ and $\cap$ are the infix operators denoting the union and intersection of two sets, respectively. We use $S_k \subseteq S$ to say that S_k is either a strict subset of or equal to the set S and $x \in S$ to denote that x is an element of the set S. $\varnothing$ is used to denote the empty set and $|S|$ the cardinality of a set S. Lastly, the convolution operator is denoted as $*$.

13.2.1. Feedforward Neural Networks

The original formulation of feedforward neural networks was proposed by Rosenblatt [1962]. It can be seen as an extension to the perceptron algorithm,

originally developed by Rosenblatt [1957], with an element-wise nonlinear transition function applied to the linear classifier. This nonlinear transition function allows the hyperplane decision boundary to take a nonlinear form, allowing the model to separate training data that is not linearly separable. The formulation for a single layer is as follows,

$$y = \sigma(\mathbf{w}^T\mathbf{x} + b) \tag{13.1}$$

where $\mathbf{x}$ is the training example input, y is the layer output, $\mathbf{w}$ are the layer weights, b is the bias. One common approach to handling the bias is to add an additional parameter to the weight vector and append a 1 to the input vector. When a bias term is omitted this formulation can be assumed unless otherwise stated throughout the section.

The nonlinear transition function, σ, is also referred to the activation function throughout literature. This is often chosen from a handful of commonly used nonlinear functions for different applications. The most widely used activation functions are the following,

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \tag{13.2}$$

$$ReLU = \max(0, z), \text{ and} \tag{13.3}$$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{13.4}$$

Additionally, the radial basis function (RBF) kernel function can be used as an activation function and doing so gives rise to the radial basis function neural network (RBFNN), as introduced by Broomhead and Lowe [1988]. To increase the complexity of the model, and thus its ability to learn more complex

relationships between the input features, network layers can be subsequently added to the model that accepts the previous layer's output as input. Doing so results in a deep neural network (DNN). The function of the network as a whole $\phi(x)$ thus becomes,

$$\phi(\mathbf{x}) = \mathbf{W}^{(3)}\sigma(\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\mathbf{x})) \tag{13.5}$$

where the weight matrices $\mathbf{W}^{(i)}$ are indexed according to the layer they belong to. Intuitively, this allows the first layer to learn linear functions between the input features, the second layer to learn nonlinear combinations of these functions, and the third layer to learn increasingly more complex nonlinear combinations of these functions. This formulation additionally gives rise to a nice graphical interpretation of the model, which is widely used in literature and given in Figure 13.1.

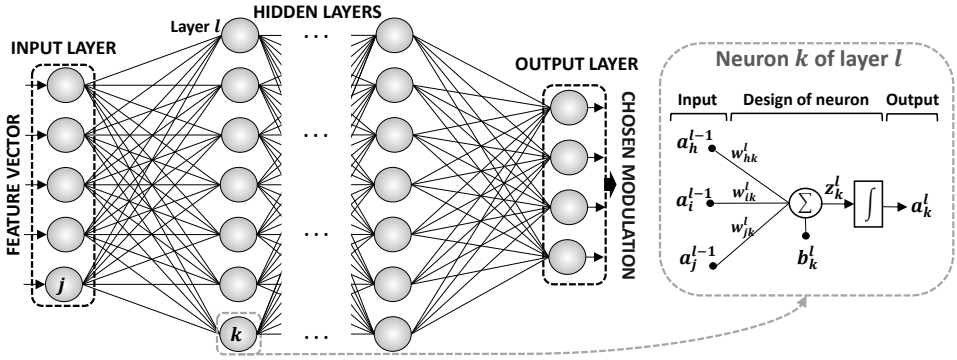

Figure 13.1: Standard Framework of Feed Forward Neural Network

This graphical interpretation is also where the feedforward neural network gets its loose biological interpretation. Each solid line in Figure 13.1 denotes a weighted connection in the graph. The input, output, and hidden layers are denoted as such in the graph and a close up of one node in the graph is

provided. This close up calls the single node, a neuron, but it can equivalently be referred to simply as a unit in this text and throughout literature. The close up also shows the inputs to the neuron, the weighted connections from the previous layer, the weighted sum of inputs, and the activation value, denoted as a_i^{l-1}, w_{ik}^l, z_k^l, and a_k^l, respectively. Occasionally, a neuron employing a given activation function may be referred to as that type of unit in this text and throughout literature, i.e. a unit with a $ReLU$ activation function may be called a "$ReLU$ unit".

The most common way to train most kinds of neural networks is the optimization method called stochastic gradient descent (SGD). SGD is similar to well known gradient descent methods with the exception that the true gradient of the loss function with respect to the model parameters is not used to update the parameters. Usually, the gradient is computed using the loss with respect to a single training example or some subset of the entire training set, which is typically referred to as a mini-batch, resulting in mini-batch SGD. This results in the updates of the network following a noisy gradient, which in fact, often helps the learning process of the network by being able to avoid convergence on local minima which are prevalent in the non-convex loss landscapes of neural networks. The standard approach to applying SGD to the model parameters is through the repeated application of the chain rule of derivation using the famous back-propagation algorithm developed by Rumelhart et al. [1986].

The last layer in a given neural network is called the output layer. The output layer differs from the inner layers in that the choice of the activation function used in the output layer is tightly coupled with the selection of the loss function and the desired structure of the output of the network. Generally, the following discussion of output layers and loss functions applies to all neural

networks, including the ones introduced later in this section.

Perhaps the simplest of output unit activation functions is that of the linear output function. It takes the following form,

$$\hat{y} = \mathbf{W}^T\mathbf{h} + b \tag{13.6}$$

where $\mathbf{W}$ is the output layer weight matrix, $\mathbf{h}$ are the latent features output from the previous layer, and $\hat{y}$ are the estimated output targets. Coupling a linear output activation function with a mean squared error loss function results in the maximizing the log-likelihood of the following conditional distribution,

$$p(y|\mathbf{x}) = N(y; \hat{y}, I) \tag{13.7}$$

Another task prominent among ML problems is that of binary classification. In a binary classification task, the output target assumes one of two values and thus can be characterized by a Bernoulli distribution, $p(y = 1|x)$. Since the output of a purely linear layer has a range over the entire real line, we motivate the use of a function that "squashes" the output to lie in the interval $[0, 1]$, thus obtaining a proper probability. The logistic *sigmoid* does exactly this and it is, in fact, the preferred method to obtain a Bernoulli output distribution. Accordingly, the output layer becomes,

$$\hat{y} = \sigma(\mathbf{W}^T\mathbf{x} + b) \tag{13.8}$$

The negative log-likelihood loss function, used for maximum likelihood estimation, of the above output layer is given as,

$$L(\theta) = -log(p(y|x)) = f((1 - 2y)z) \tag{13.9}$$

where $f(x) = log(1 + e^x)$ is called the *softplus* function and $z = \mathbf{W}^T\mathbf{x} + b$ is called the activation value. The derivation of Eq. (13.9) is not provided here but can be found in [Goodfellow et al., 2016a] for the interested reader.

In the case when the task calls for a multi-class classification, we want a Multinoulli output distribution rather than a Bernoulli output distribution. The Multinoulli distribution assigns a probability that a particular example belongs to a particular class. Obviously, the sum over class probabilities for a single example should be equal to 1. The Multinoulli distribution is given as the conditional distribution: $\hat{y}_i = p(y = i|\mathbf{x})$. It is important to note that the output, $\hat{\mathbf{y}}$, is now an n-dimensional vector containing the probability that $\mathbf{x}$ belongs to class $i \in [0, n]$ at each index i in the output vector. The targets for such a classification task are often encoded as an n-dimensional vector containing $(n-1)$ 0's and one 1, located at an index j which denotes that the associated training example belongs to the class j. This type of target vector is commonly referred to as a one-hot vector. The output function that achieves the Multinoulli distribution in the maximum likelihood setting is called the *softmax* function and is given as,

$$softmax(\mathbf{z})_i = \frac{e^{\mathbf{z}}}{\sum_j e^{z_j}} \tag{13.10}$$

where z_j is the linear activation at an output unit j. *Softmax* output units are almost exclusively coupled with a negative log-likelihood loss function. Not only does this give rise to the maximum likelihood estimate for the Multinoulli output distribution but the log in the loss function is able to undo the exponential in the *softmax* which keeps the output units from saturating and allows the gradient to be well-behaved, allowing the learning to proceed [Goodfellow et al., 2016b].

13.2.2. Convolutional Neural Networks

The convolutional neural networks (CNN) was originally introduced by LeCun et al. [1989] as a means to handle grid-like input data more efficiently. Input of this type could be in the form of a time-series but is more typically found as image-based input. The formulation of CNNs additionally has biological underpinnings related to the human visual cortex.

CNNs are very similar to the feedforward networks introduced previously with the exception that they use a convolution operation in place of a matrix multiplication in the computation of a unit's activation value. In this section, we assume that the reader is familiar with the concept of the convolution operation on two continuous functions, where one function, the input function, is convolved with the convolution kernel. The primary differences from the aforementioned notion of convolution and convolution in the CNN setting are that the convolution operation is discretized (for practical implementation purposes) and that it is often truly the cross-correlation operation that is performed in CNNs rather than true convolution. This means that the kernel is not typically flipped before convolving it with the input function. This is primarily done for practical implementation purposes and does not typically affect the efficacy of the CNN in practice.

Convolution in the context of CNNs is thus defined as the following, for an input image I,

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{13.11}$$

where K is the convolution kernel and the output, S, is often referred to as the feature map throughout literature. It is important to note that the above

formulation is for two dimensional convolution but can be extended to input data of different dimensions. The entries of K can be seen as analogous to the weight parameters described previously (Section 13.2.1) and can be learned in a similar manner using SGD and the back-propagation (BP) algorithm [Rumelhart et al., 1986]. Intuitively, one can imagine having multiple K kernels in a single CNN layer being analogous to having multiple neurons in a single feedforward neural network layer. The output feature maps will be grid-like and subsequent convolutional layers can be applied to these feature maps after the element-wise application of one of the aforementioned nonlinear activation functions.

In addition to convolutional layers, CNNs often employ a separate kind of layer called pooling layers. The primary purpose of a pooling layer is to replace the output of the network at a certain location with a type of summarization of the outputs within a local neighborhood. Examples of pooling layers include max pooling [Zhou and Chellappa, 1988], average pooling, L^2 norm pooling, and distance weighted average pooling. A max pooling layer would summarize some rectangular region of the input image by selecting only the maximum activation value present in the region as output from the pooling layer. Pooling layers improve the efficacy of CNNs in a few different ways. First, they help make the learned representation of the input invariant to small translations, which is useful when aiming to determine the presence of a feature in the input rather than its location. Second, pooling layers help condense the size of the network since convolutional layers do not inherently do so. A binary classification task taking image data with size $256 \times 256 \times 3$ will need to reduce the size of the net to a single output neuron to make use of the output layer and cost function pairs described previously in Section 13.2.1. Lastly, pooling layers

lead to infinitely strong prior distributions making the CNN more statistically efficient [Goodfellow et al., 2016a].

Some common adaptations applied to CNNs come in the form of allowing information flow to skip certain layers within the network. While the following adaptions were demonstrated on CNNs and long short term memorys (LSTMs) (a type of recurrent neural network (RNN)), the concepts can be applied to any of the networks presented in this chapter. A residual network (RN), or ResNet He et al. [2015], is a neural network which contains a connection from the output of a layer, say L_{i-2}, to the input of the layer L_i. This connection allows the activation of the L_{i-2} layer to skip over the layer L_{i-1} such that a "residual function" is learned from layer L_{i-2} to layer L_i. The RN uses an identity operation on the activation of the L_{i-2} layer, meaning the values are unchanged, prior to adding them to the values input to layer L_i. Conversely, a highway neural network Srivastava et al. [2015], allows a similar skip connection over layers but additionally applies weights and activation functions to these connections so a nonlinear relationship can be learned. Lastly, a dense neural network Huang et al. [2016] is a network that employs such weighted connections between each layer and all of its subsequent layers. The motivation behind each of these techniques is similar in that they attempt to mitigate learning problems associated with vanishing gradients Hochreiter et al. [2001]. For each of these networks, the BP algorithm that [Rumelhart et al., 1986] used must be augmented to incorporate the flow of error over these connections.

13.3. Neural Networks For Signal Intelligence

ML techniques for signal intelligence typically manifest themselves as solutions to discriminative tasks. That is, many applications focus on multi-class or binary classification tasks. Perhaps the most prevalent signal intelligence task solved using ML techniques is that of automatic modulation classification (AMC). In short, this task involves determining what scheme was used to modulate the transmitted signal, given the raw signal observed at the receiver. Other signal intelligence tasks that employ ML solutions include wireless interference classification. In this section, different state of the art ML solutions to these signal intelligence tasks are discussed in further detail.

13.3.1. Modulation Classification

The deep learning (DL) solutions to modulation classification tasks have received significant attention in the last two years [O'Shea et al., 2018, O'Shea and Hoydis, 2017, Wang et al., 2017, West and O'Shea, 2017, Kulin et al., 2018, Karra et al., 2017]. O'Shea et al. [2018] present several DL models to address the modulation recognition problem, while Karra et al. [2017] train hierarchical deep neural networks to identify data type, modulation class and modulation order. Kulin et al. [2018] present a conceptual framework for end-to-end wireless DL, followed by a comprehensive overview of the methodology for collecting spectrum data, designing wireless signal representations, forming training data and training deep neural networks for wireless signal classification tasks.

The task of AMC is pertinent in signal intelligence applications as the modulation scheme of the received signal can provide insight to what type of communication frameworks and emitters are present in the local RF environment. The problem at large can be formulated as estimating the conditional distribution, $p(y|x)$, where y represents the modulation structure of the signal and x is the received signal.

Traditionally, AMC techniques are broadly classified as maximum likelihood based approaches [Ozdemir et al., 2013, 2015, Wimalajeewa et al., 2015, Foulke et al., 2014, Jagannath et al., 2015], feature-based approaches [Azzouz and Nandi, 1996, Hazza et al., 2012, Kubankova et al., 2010] and hybrid techniques [Jagannath et al., 2017]. Prior to the introduction of ML, AMC tasks were often solved using complex hand engineered features computed from the raw signal. While these features alone can provide insight about the modulation structure of the received signal, ML algorithms can often provide a better generalization to new unseen data sets, making their employment preferable over solely feature based approaches. The logical remedy to the use of complex hand engineered feature based classifiers are models that aim to learn directly from received data. Recent work done by O'Shea and Corgan [2016] show that deep convolutional neural networks (DCNNs) trained directly on complex time domain signal data outperform traditional models using cyclic moment feature based classifiers. In the work done by Shengliang Peng and Yao [2017], the authors propose a DCNN model trained on the two-dimensional constellation plots generated from the received signal data and show that their approach outperforms other approaches using cumulant based classifiers and support vector machines (SVMs).

While strictly feature based approaches may become antiquated with the

advent of the application of ML to signal intelligence, expert feature analysis can provide some useful inputs to ML algorithms. In [Jagannath et al., 2018b], we compute hand engineered features directly from the raw received signal and apply a feedforward neural network classifier to the features to provide an AMC. The discrete time complex valued received signal can be represented as,

$$y(n) = h(n)x(n) + w(n), \qquad n = 1, ..., N \tag{13.12}$$

where $x(n)$ is the discrete-time transmitted signal, $h(n)$ is the complex valued channel gain that follows a Gaussian distribution and $w(n)$ is the additive complex zero-mean white Gaussian noise process at the receiver with two-sided power spectral density (PSD) $N_0/2$. The received signal is passed through an Automatic Gain Control prior to the computation of feature values.

The first feature value computed from the received signal is the variance of the amplitude of the signal and is given by,

$$Var(|y(n)|) = \frac{\sum_{N_s}(|y(n)| - \mathbb{E}(|y(n)|))^2}{N_s} \tag{13.13}$$

where $|y(n)|$ is the absolute value of the over-sampled signal and $\mathbb{E}(|y(n)|)$ represents the mean computed from N_s samples. This feature provides information which helps distinguish frequency shift keying (FSK) modulations from the phase shift keying (PSK) and quadrature amplitude modulation (QAM) modulation structures also considered in the classification task. The second and third features considered are the mean and variance of the maximum value of the power spectral density of the normalized centered-instantaneous

amplitude, which is given as,

$$\gamma_{max} = \frac{max\left|FFT(a_{cn}(n))\right|^2}{N_s}, \tag{13.14}$$

where $FFT(.)$ represents the fast Fourier transform (FFT) function, $a_{cn}(n) \triangleq \frac{a(n)}{m_a} - 1$, $m_a = \frac{1}{N_s}\sum_{n=1}^{N_s} a(n)$, and $a(n)$ is the absolute value of the complex-valued received signal. This feature provides a measure of the deviation of the PSD from its average value. The mean and variance of this feature computed over subsets of a given training example are used as two separate entries in the feature vector input into the classification algorithm, corresponding to the second and third features, respectively.

The fourth feature used in our work was computed using higher order statistics of the received signal. Namely, cumulants, are known to be invariant to the various distortions commonly seen in random signals and are computed as follows,

$$C_{lk} = \sum_{p}^{\text{No. of partitions in } l} (-1)^{p-1}(p-1)! \prod_{j=1}^{p} \mathbb{E}\{y^{l_j - k_j} y^{*k_j}\}, \tag{13.15}$$

where l denotes the order and k denotes the number of conjugations involved in the computation of the statistic. We use the ratio, C_{40}/C_{42} as the fourth feature which is computed using,

$$C_{42} = \mathbb{E}(|y|^4) - |\mathbb{E}(y^2)|^2 - 2\mathbb{E}(|y|^2)^2, \tag{13.16}$$

$$C_{40} = \mathbb{E}(y^4) - 3\mathbb{E}(y^2)^2. \tag{13.17}$$

The fifth feature used in our work is called the in-band spectral variation as it allows discrimination between the FSK modulations considered in the task.

We define $Var(f)$ as,

$$Var(f) = Var\Big(\mathcal{F}\big(y(t)\big)\Big), \tag{13.18}$$

where $\mathcal{F}(y(t)) = \big\{Y(f) - Y(f - F_0)\big\}_{f=-f_i}^{+f_i}/F_0$, F_0 is the step size, $Y(f) = FFT(y(t))$, and $[-f_i, +f_i]$ is the frequency band of interest.

The final feature used in the classifier is the variance of the deviation of the normalized signal from the unit circle, which is denoted as $Var(\Delta_o)$. It is given as,

$$\Delta_o = \frac{|y(t)|}{\mathbb{E}(|y|)} - 1. \tag{13.19}$$

This feature helps the classifier discriminate between PSK and QAM modulation schemes.

The modulations considered in the work are the following: binary phase shift keying (BPSK), quadrature phase shift keying (QPSK), 8PSK, 16QAM, continuos phase frequency shift keying (CPFSK), Gaussian frequency shift keying (GFSK), and Gaussian minimum shift keying (GMSK). This characterizes a seven class classification task using the aforementioned six features computed from each training example. To generate the data set, a total of 35,000 examples were collected: 1,000 examples for each modulation at each of the five signal-to-noise-ratio (SNR) scenarios considered in the work. Three different feedforward neural network structures were trained at each SNR scenario using a training set consisting of 80% of the data collected at the given SNR and a test set consisting of the remaining 20%. The three feedforward nets differed in the number of hidden layers, ranging from one to three. Qualitatively, the feedforward network with one hidden layer outperformed the other models in all but the least favorable SNR scenario, achieving the highest classification accuracy of 98% in the most favorable SNR scenario. The seemingly paradox-

ical behavior is attributed to the over-fitting of the training data when using the higher complexity models, leading to poorer generalization in the test set.

This work has been further extended to evaluate other ML techniques using the same features. Accordingly, we found that training a random forest classifier for the same AMC task yielded similar results to the feedforward network classifier. Additionally, the random forest classifier was found to outperform the DNN approach in scenarios when the exact center frequency of the transmitter was not known, which was assumed to be given in Jagannath et al. [2018b]. The random forest classifier was comprised of 20 classification and regression treess (CARTs) constructed using the gini impurity function. At each split a subset of the feature vectors with cardinality equal to 3 was considered.

An alternative approach to the previously described method is to learn the modulation of the received signal from different representations of the raw signal. Kulin et al. [2018] train DCNNs to learn the modulation of various signals using three separate representations of the raw received signal. In the work, the raw complex valued received signal training examples are denoted as $r_k \in C^N$, where k indexes the procured training data set and N is the number of complex valued samples in each training example. We inherit this notation for presentation of their findings. The data set in the work was collected by sampling a continuous transmission for a period of time and subsequently segmenting the received samples into N dimensional data vectors.

Kulin et al. [2018] train separate DCNNs on three different representations of the raw received signal and compare their results to evaluate which representation provides the best classification accuracy. The first of the three signal representations are given as a $2 \times N$ dimensional in-phase/quadrature (I/Q) matrix containing real valued data vectors carrying the I/Q information of the

raw signal, denoted x_i and x_q, respectively. Mathematically,

$$x_k^{IQ} = \begin{bmatrix} x_i^T \\ x_q^T \end{bmatrix} \tag{13.20}$$

where $x_k^{IQ} \in R^{2\times N}$. The second representation used is a mapping from the complex values of the raw received signal into two real valued vectors representing the phase, Φ and the magnitude, A,

$$x_k^{A/\Phi} = \begin{bmatrix} x_A^T \\ x_\Phi^T \end{bmatrix} \tag{13.21}$$

Where $x_k^{A/\Phi} \in R^{2\times N}$ and the phase vector $x_\Phi^T \in R^N$ and magnitude vector $x_A^T \in R^N$ have elements,

$$x_{\Phi_n} = \arctan\left(\frac{r_{q_n}}{r_{i_n}}\right), x_{A_n} = (r_{q_n}^2 + r_{i_n}^2)^{\frac{1}{2}} \tag{13.22}$$

respectively. The third representation is a frequency domain representation of the raw time domain complex signal. It is characterized by two real valued data vectors, one containing the real components of the complex FFT, $\Re(X_k)$, and the other containing the imaginary components of the complex FFT, $\Im(X_k)$, giving,

$$x_k^F = \begin{bmatrix} \Re(X_k)^T \\ \Im(X_k)^T \end{bmatrix} \tag{13.23}$$

Using these three representations of the raw signal, three DCNNs with identical structure are trained on each representation and the accuracy of the resultant models are compared to determine which representation allows for learning the best mapping from raw signal to modulation structure.

Each training example comprised of $N = 128$ samples of the raw signal sampled at 1 MS/s (mega-samples per seconds) and the following 11 modulation formats were condsidered in the classification task: BPSK, QPSK, 8-PSK, 16-QAM, 64-QAM, CPFSK, GFSK, 4-pulse-amplitude modulation (PAM), wideband Frequency Modulation (WBFM), amplitude modulation (AM)-double-sideband modulation (DSB), and AM-single-sideband modulation (SSB). Thus, the training targets $y_k \in R^{11}$ are encoded as one-hot vectors where the index holding an i corresponds to the modulation of the signal. A total of 220,000 training examples $x_k \in R^{2\times 128}$ were acquired uniformly over different SNR scenarios ranging from $-20dB$ to $+20dB$.

The DCNN structure used for each signal representation is the same and consists of two convolutional layers, a fully connected layer, and a *softmax* output layer trained using the negative log-likelihood loss function. The activation function used in each of the convolutional layers and the fully connected layer is the $ReLU$ function. The DCNNs were trained using a training set comprised of 67% of the total data set, with the rest of the data set being used as test and validation sets. An Adam optimizer [Kingma and Ba, 2014] was used to optimize the training processes for a total of 70 epochs. The metrics used to evaluate each of the models include the precision, recall, and F1 score of each model. In the work, a range of values is provided for the three aforementioned metrics for the CNN models trained on different data representations for three different SNR scenarios: high, medium, and low, corresponding to $18dB$, $0dB$, and $-8dB$, respectively. In the high SNR scenario, it is reported that the precision, recall, and F1 score of each of the three CNN models fall in the range of $0.67 - 0.86$. For the medium and low SNR scenarios, the same metrics are reported in the ranges of $0.59 - 0.75$ and $0.22 - 0.36$, respectively. This relatively

low performance can be attributed to the choice of the channel model used when generating the data, namely, a time-varying multipath fading channel.

Furthermore, what is evaluated also includes the classification accuracy of each of the three models trained using different data representations under similar SNR conditions. Qualitatively, each of the three DCNN models performs similarly at low SNR, while the DCNN trained on the I/Q representation of data yields a better accuracy at medium SNR, and the DCNN trained on the amplitude and phase representation yields a better accuracy at high SNR. Interestingly, the DCNN trained on the frequency domain representation of the data performs significantly worse than the I/Q and A/ϕ DCNNs at high SNR. This could potentially be due to the similar characteristics exhibited in the frequency domain representation of the PSK and QAM modulations used in the classification problem. The primary takeaways from this work are that learning to classify modulation directly from different representations of the raw signal can be an effective means of developing a solution to the AMC task; however, the efficacy of the classifier is dependent on how the raw signal is represented to the learning algorithm.

13.3.2. Wireless Interference Classification

The wireless interference classification (WIC) is a classification task that essentially refers to identifying what type of wireless emitter exists in the environment. The motivation behind such a task is that it can often be helpful to know what type of emitters are present (WiFi, Zigbee, Bluetooth, etc.) so that you can effectively attempt to avoid interference and coexist with other emitters sharing the resources. Recent work done by Selim et al. [2017] show the use of

DCNNs to classify radar signals using both spectrogram and amplitude-phase representations of the received signal. In the work presented by Akeret et al. [2017], DCNN models are proposed to accomplish interference classification on two-dimensional time-frequency representations of the received signal to mitigate the effects of radio interference in cosmological data. Additionally, the authors of Czech et al. [2018] employ DCNN and LSTM models to achieve a similar end.

Kulin et al. [2018] propose to employ DCNNs for the purpose of the wireless interference classification of three different wireless communication systems based on the WiFi, Zigbee, and Bluetooth standards. They look at five different channels for each of the three standards and construct a fifteen class classification task for which they obtain $225,225$ training vectors consisting of 128 samples each, where samples were collected at 10 MS/s. A flat fading channel with additive white Gaussian noise is assumed for this classification task.

Three DCNNs were trained and evaluated using the wireless interference classification data set described above. Each of the three DCNNs was trained on one of the representations of the data that were presented in the previous section that discussed AMC. The DCNN architectures were also the same as presented previously in Section 13.3.1.

Each of the three DCNNs trained using different data representations was evaluated in a similar fashion to the evaluation method described in Section 13.3.1, namely, using precision, recall, and F1 score under different SNR scenarios. For the wireless interference classification task, the precision, recall, and F1 score of each of the three DCNNs all fell in the interval from $0.98 - 0.99$ under the high SNR scenario. For the medium and low SNR scenarios, the analogous intervals were from $0.94 - 0.99$ and $0.81 - 0.90$, respectively.

Additionally, Kulin et al. [2018] provide an analysis of classification accuracy for each of the three DCNN models at varying SNRs. For the task of wireless interference classification, the DCNN model trained on the frequency domain representation of the data outperforms the other models at all SNRs and especially so in lower SNR scenarios. These findings are due to the fact that the wireless signals that were considered have more expressive features in the frequency domain as they have different bandwidth, modulation, and spreading characteristics.

Youssef et al. [2017] take a different approach to the wireless interference classification task and primarily compare different types of learning models rather than different types of data representation. The proposed models include deep feedforward networks, deep convolutional networks, support vector machines using two different kernels, and a multi-stage training (MST) algorithm using two different learning algorithms. In the work, 12 different transmitters are considered and 1,000 packets from each transmitter are collected for a total of 12,000 packets which comprise the entire data set. Each transmitter transmitted the same exact 1,000 packets, which were generated using pseudo-random values injected into the modem. All of the transmitters used a proprietary orthogonal frequency-division multiplexing (OFDM) protocol with a QPSK modulation scheme and a baseband transmitter sample rate of 1.92 MS/s. At the receiver, each packet is represented by 10,000 time domain I/Q samples. Each of the models was trained on data sets consisting of training examples made up of 32, 64, 128, 256, 512, and 1024 samples from each packet, and their performance is compared across data sets. Given the

complex valued received signal,

$$f = (f_1, f_2, ..., f_N) \tag{13.24}$$

N samples were selected by skipping the first N_0 samples of a packet where $|\mathcal{R}(f_i)| < \tau$ for some $\tau > 0$ yielding the signal vector g,

$$g = (f_{N_0}, f_{N_0+1}, ..., f_{N_0+N-1}) \tag{13.25}$$

For the DNN, SVM, and MST models each training example was constructed by concatenating the real and imaginary parts of the signal vector, yielding a vector of dimension $2N$. For the DCNN model the real and imaginary parts of the signal vector were stacked to generate $2 \times N$ dimensional training vectors.

The DNN architecture considered in the work consisted of two fully connected hidden layers, comprised of 128 *ReLU* units each and an output layer consisting of logistic *sigmoid* units. The network was trained using the Adam optimizer [Kingma and Ba, 2014] and mini-batch size of 32.

The DCNN model used by the authors was composed of two convolutional layers using 64 (8×2) and 32 (16×1) filters, respectively. Each convolutional layer was input into a max-pool layer with a pool size of 2×2 and 2×1, respectively. The output of the second max-pool layer was fed into a fully-connected layer consisting of 128 *ReLU* units. An output layer employing logistic *sigmoid* units was used on top of the fully-connected layer.

The two SVM architectures analyzed in the work differ only in the kernel function used. The first architecture employed the polynomial kernel and the second employed the Pearson VII Universal Kernel [Üstün et al., 2005]. Both architectures used Platt's Minimization Optimization algorithm to compute

the maximum-margin hyperplanes.

Furthermore, an analysis of the performance of MST multi-layer perceptrons (MLPs) trained using first order and second order methods is provided. A high level description of MST MLP is presented here and we refer the interested reader to Youssef et al. [2015] for a more rigorous derivation. The MST method to training neural networks, as presented in the work, is essentially a hierarchical way to solve an optimization problem by solving smaller constituent optimization problems. To this end, in what is called the first stage, a number of separate MLPs would be trained on different subsets of the training data set. This can be seen in the lowest layer of the hierarchical representation adapted from Youssef et al. [2017], and provided in Figure 13.2.

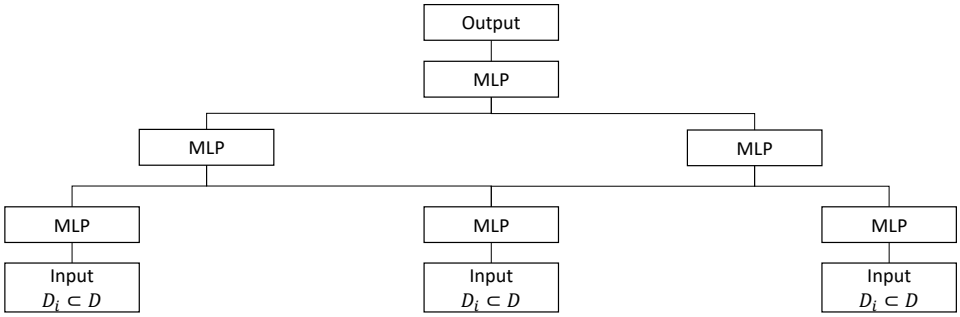

Figure 13.2: Adaptation of MST MLP used by Youssef et al. [2017].

Once the first stage is trained, a second stage is trained by taking the concatenation of the network outputs from the first stage as input. Training can continue in this fashion for subsequent stages. One of the advantages of training networks in this way is that the many smaller MLPs comprising the larger classifier can be efficiently trained using second order optimization methods. Second order optimization methods such as Newton, Gauss-Newton, or Levenberg-Marquardt methods are usually intractable due to the size of typical networks but can provide better convergence when applicable. Two 3-stage

MST systems were trained in the work, one using the first order method of SGD, and the other using the second order Accelerated Levenberg-Marquardt method [K. Youssef]. Each MST system had an identical structure where stage 1 consisted of 60 MLPs with 2 hidden layers and 10 units in each layer. Stage 2 and 3 had the same architecture and were comprised of 30 MLPs with each MLP consisting of 2 hidden layers made up of 15 units each. All hidden units employed the *tanh* activation function and all output layers contained linear units.

All of the models described above were trained on 10 different iterations of the collected data set and their performance was compared. Five data sets were constructed using training examples made up of 32, 64, 128, 256, and 512 samples and then each model was trained twice, using a training set comprised of 90% and 10% of the total data set, for a total of 10 different data sets for each model. In general, the MST system trained using second order methods on 90% of the training data performed best across all sizes of training examples, yielding a classification accuracy of 100% for each data set. All of the models performed better when trained using 90% of the data set as opposed to 10% of the training data set. Generally, each model performed better when provided with training examples that contained more samples, with the exception of the deep feedforward network model, which could be attributed to the fact that longer sequences of samples may contain an increasing number of artifacts which the DNN may not be robust to.

A summarization of the different models presented in this section is provided in Table 13.1.

Table 13.1: Summary of ML Solutions for Signal Intelligence

Classifiers	**Task**	**Representation**	**Model**
Jagannath et al. [2018b]	AMC	Feature-Based	DNN
Kulin et al. [2018]	AMC	I/Q, A/Φ, FFT	DCNN
O'Shea and Corgan [2016]	AMC	I/Q	DCNN
Shengliang Peng and Yao [2017]	AMC	Constellation	DCNN
Kulin et al. [2018]	WIC	I/Q, A/Φ, FFT	DCNN
Selim et al. [2017]	WIC	2D time-frequency, A/Φ	DCNN
Akeret et al. [2017]	WIC	2D time-frequency	DCNN
Czech et al. [2018]	WIC	2D time-frequency	DCNN, LSTM
Youssef et al. [2017]	WIC	I/Q	DNN, DCNN, SVM, MST

13.4. Neural Networks for Spectrum Sensing

One of the key challenges in enabling real-time inference from spectrum data is how to *effectively* and *efficiently* extract *meaningful* and *actionable* knowledge out of the tens of millions of I/Q samples received every second by wireless devices. Indeed, a single 20 MHz-wide WiFi channel generates an I/Q stream rate of about 1.28 Gbit/s, if I/Q samples are each stored in a 4-byte word. Moreover, the RF channel is significantly time-varying (*i.e.*, in the order of milliseconds), which imposes strict timing constraints on the *validity* of the extracted RF knowledge. If (for example) the RF channel changes every 10ms, a knowledge extraction algorithm must run with latency (much) less than 10ms to both (i) offer an accurate RF prediction and (ii) drive an appropriate physical-layer response; for example, change in modulation/coding/beamforming vectors due

to adverse channel conditions, local oscillator (LO) frequency due to spectrum reuse, and so on.

As discussed earlier, DL has been a prominent technology of choice for solving classification problems for which no well-defined mathematical model exists. It enables the analysis of unprocessed I/Q samples without the need of application-specific and computational-expensive feature extraction and selection algorithms [O'Shea et al., 2018], thus going far beyond traditional low-dimensional ML techniques. Furthermore, DL architectures are application-insensitive, meaning that the same architecture can be retrained for different learning problems.

Decision-making at the physical layer may leverage the spectrum knowledge provided by DL. On the other hand, RF DL algorithms must execute in *real-time* (*i.e.*, with static, known-a-priori latency) to achieve this goal. Traditional central processing unit (CPU)-based knowledge extraction algorithms [Abadi et al., 2016] are unable to meet strict time constraints, as general-purpose CPUs can be interrupted at-will by concurrent processes and thus introduce additional latency to the computation. Moreover, transferring data to the CPU from the radio interface introduces unacceptable latency for the RF domain. Finally, processing I/Q rates in the order of Gbit/s would require CPUs to run continuously at maximum speed, and thus consume enormous amounts of energy. For these reasons, RF DL algorithms must be closely integrated into the RF signal processing chain of the embedded device.

13.4.1. Existing work

Most of the existing work is based on traditional low-dimensional machine learning [Wong and Nandi, 2001, Xu et al., 2010, Pawar and Doherty, 2011, Shi and Karasawa, 2012, Ghodeswar and Poonacha, 2015], which requires (i) extraction and careful selection of complex features from the RF waveform (*i.e.*, average, median, kurtosis, skewness, high-order cyclic moments, etc.); and (ii) the establishment of tight decision bounds between classes based on the current application, which are derived either from mathematical analysis or by learning a carefully crafted dataset [Shalev-Shwartz and Ben-David, 2014]. In other words, since feature-based machine learning is (a) significantly application-specific in nature; and (b) introduces additional latency and computational burden due to feature extraction, its application to real-time hardware-based wireless spectrum analysis becomes impractical, as the wireless radio hardware should be changed according to the specific application under consideration.

Recent advances in DL [LeCun et al., 2015] have prompted researchers to investigate whether similar techniques can be used to analyze the sheer complexity of the wireless spectrum. For a compendium of existing research on the topic, the reader can refer to Mao et al. [2018]. Among other advantages, DL is significantly amenable to be used for real-time hardware-based spectrum analysis, since different model architectures can be reused to different problems as long as weights and hyper-parameters can be changed through software. Additionally, DL solutions to the physical layer modulation recognition task have been given much attention over recent years, as previously discussed in this chapter. The core issue with existing approaches is that they leverage DL to perform offline spectrum analysis only. On the other hand, the opportunity of real-time hardware-based spectrum knowledge inference remains substantially

uninvestigated.

13.4.2. Background on System-on-Chip Computer Architecture

Due to its several advantages, we contend that one of the most appropriate computing platform for RF DL is a System on Chip (SoC). An SoC is an integrated circuit (also known as "IC" or "chip") that integrates all the components of a computer, *i.e.*, CPU, random access memory (RAM), input/output (I/O) ports and secondary storage (*e.g.*, SD card) – all on a single substrate [Molanes et al., 2018]. SoCs have low power consumption [Pete Bennett (EE Times), 2004] and allow the design and implementation of *customized hardware* on the field-programmable gate array (FPGA) portion of the chip, also called programmable logic (PL). Furthermore, SoCs bring unparalleled flexibility, as the PL can be reprogrammed at-will according to the desired learning design. The PL portion of the SoC can be managed by the processing system (PS), *i.e.*, the CPU, RAM, and associated buses.

SoCs use the Advanced eXtensible Interface (AXI) bus specification [Xilinx Inc., 2011] to exchange data (i) between functional blocks inside the PL; and (ii) between the PS and PL. There are three main AXI sub-specifications: *AXI-Lite*, *AXI-Stream* and *AXI-Full*. AXI-Lite is a lightweight, low-speed AXI protocol for register access, and it is used to configure the circuits inside the PL. AXI-Stream is used to transport data between circuits inside the PL. AXI-Stream is widely used, since it provides (i) standard inter-block interfaces; and (ii) rate-insensitive design, since all the AXI-Stream interfaces share the same bus clock, the high-level synthesis (HLS) design tool will handle the

handshake between DL layers and insert first-in first-outs (FIFOs) for buffering incoming/outgoing samples. AXI-Full is used to enable burst-based data transfer from PL to PS (and *vice versa*). Along with AXI-Full, direct memory access (DMA) is usually used to allow PL circuits to read/write data obtained through AXI-Stream to the RAM residing in the PS. The use of DMA is crucial since the CPU would be fully occupied for the entire duration of the read/write operation, and thus unavailable to perform other work.

13.4.3. A Design Framework for Real-time RF Deep Learning

One of the fundamental challenges to be addressed is how to transition from a software-based DL implementation (*e.g.*, developed with the Tensorflow Abadi et al. [2016] engine) to a hardware-based implementation on an SoC. Basic notions of high-level synthesis and a hardware design framework are presented in Sections 13.4.3.1 and 13.4.3.2, respectively.

13.4.3.1. High-level Synthesis

HLS is an automated design process that interprets an algorithmic description of a desired behavior (*e.g.*, C/C++) and creates a model written in hardware description language (HDL) that can be executed by the FPGA and implements the desired behavior [Winterstein et al., 2013]. Designing digital circuits using HLS has several advantages over traditional approaches. First, HLS programming models can implement almost any algorithm written in C/C++. This allows the developer to spend less time on the HDL code and focus on the algorithmic portion of the design, and at the same time avoid bugs and

increase efficiency, since HLS optimizes the circuit according to the system specifications. The clock speed of today's FPGAs is a few orders of magnitude slower than CPUs (*i.e.*, up to 200-300 MHz in the very best FPGAs). Thus, parallelizing the circuit's operations is crucial. In traditional HDL, transforming the signal processing algorithms to fit FPGA's parallel architecture requires challenging programming efforts. On the other hand, an HLS toolchain can tell how many cycles are needed for a circuit to generate all the outputs for a given input size, given a target parallelization level. This helps to reach the best trade-off between hardware complexity and latency. In addition, as shown in the following, loop pipelining and loop unrolling could be used for a better silicon convergence in terms of performance, power consumption and latency.

Loop Pipelining: In high-level languages (such as C/C++), the operations in a loop are executed sequentially and the next iteration of the loop can only begin when the last operation in the current loop iteration is complete. Loop pipelining allows the operations in a loop to be implemented in a concurrent manner.

Figure 13.3 shows an example of loop pipelining, where a simple loop of three operations, *i.e.*, read (RD), execute (EX), and write (WR), is executed twice. For simplicity, we assume that each operation takes one clock cycle to complete. Without loop pipelining, the loop would take 6 clock cycles to complete. Conversely, with loop pipelining, the next RD operation is executed concurrently to the EX operation in the first loop iteration. This brings the total loop latency to 4 clock cycles. If the loop length were to increase to 100, then the latency decrease would be even more evident: 300 versus 103 clock cycles, corresponding to a speedup of about 65%. An important term for loop pipelining is called initiation interval (II), which is the number of clock cycles

```
for (int i=0; i<2;i++) {
    Op_Read;      /* RD */
    Op_Execute;   /* EX */
    Op_Write;     /* WR */
}
```

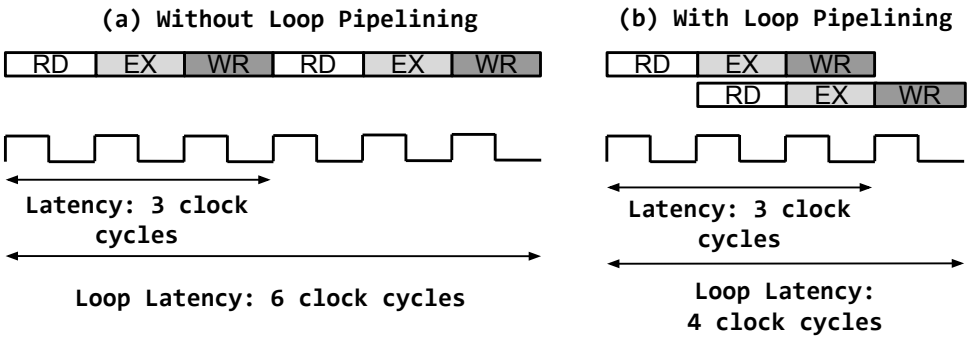

Figure 13.3: Loop pipelining.

between the start times of consecutive loop iterations. In the example of Figure 13.3, the II is equal to one, because there is only one clock cycle between the start times of consecutive loop iterations.

Loop Unrolling: Loop unrolling creates multiple copies of the loop body and adjusts the loop iteration counter accordingly. For example, if a loop is processed with an unrolling factor (UF) equal to 2 (*i.e.*, two subsequent operations in the same clock cycle as shown in Figure 13.4), it may reduce a loop's latency by a factor of 50%, since a loop will execute in half the iterations usually needed. Higher UF and II may help achieve low latency but at the cost of higher hardware resource consumption. Thus, the trade-off between latency and hardware consumption should be thoroughly explored.

13.4.3.2. Design Steps

Our framework presents several design and development steps, which are illustrated in Figure 13.5. Steps that involve hardware, middleware (*i.e.*, hardware

```
for(int i = 0; i < 10; i++) {
    sum += a[i];
}

for(int i = 0; i < 10; i+=2) {
    sum += a[i];
    sum += a[i+1];
}
```

Figure 13.4: Loop unrolling.

description logic, or HDL), and software have been depicted with different shades of grey.

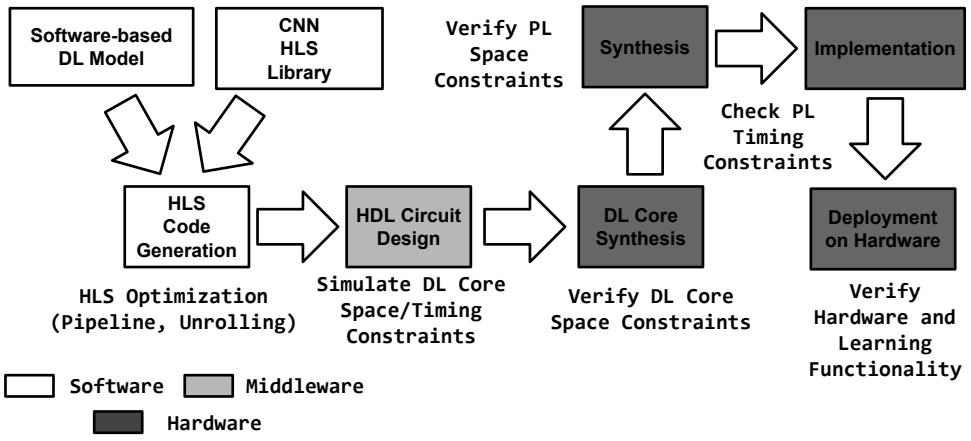

Figure 13.5: A Hardware Design Framework for RF Deep Learning.

The first major step of the framework is to take an existing DL model and convert the model in HLS language, so it can be optimized and later on synthesized in hardware. Another critical challenge is how to make the hardware implementation fully reconfigurable, *i.e.*, the weights of the DL model may need to be changed by the *Controller* according to the specific training. To address these issues, we distinguish between (i) the DL model architecture, which is the set of layers and hyper-parameters that compose the model itself, and (ii) the parameters of each layer, *i.e.*, the neurons' and filters' weights.

To generate the HLS code describing the software-based DL model, an *HLS Library*, which provides a set of HLS functions that parse the software-based DL model architecture and generates the HLS design corresponding to the desired architecture. The *HLS Library* supports the generation of convolutional, fully-connected, rectified linear unit, and pooling layers, and operated on fixed-point arithmetic for better latency and hardware resource consumption. The HLS code is subsequently translated to HDL code by an automated tool that takes into account optimization directives such as loop pipelining and loop unrolling. At this stage, the HDL describing the DL core can be simulated to (i) calculate the amount of PL resources consumed by the circuit (*i.e.*, flip-flops, BRAM blocks, etc); and (ii) estimate the circuit latency in terms of clock cycles. After a compromise between space and latency as dictated by the application has been found, the DL core can be synthesized and integrated with the other PL components, and thus total space constraints can be verified. After implementation (*i.e.*, placing/routing), the PL timing constraints can be verified, and finally the whole system can be deployed on the SoC and its functionality tested.

13.5. Open Problems

In this section, we discuss a set of open challenges overcoming which will accelerate the induction of ML techniques to future wireless communications and networking.

13.5.1. Lack of Large-scale Wireless Signal Datasets

It is well known that learning algorithms require a considerable amount of data to be able to effectively learn from a training dataset. Moreover, to compare the performance of different learning models and algorithms, it is imperative to use the same sets of data. More mature learning fields, such as computer vision and natural language processing (NLP) already have standardized datasets for these purposes [Deng, 2012, Deng et al., 2009]. However, literature still lacks large-scale datasets for RF ML.

This is not without a reason. Although the wireless domain allows the synthetic generation of signals having the desired characteristics (*e.g.*, modulation, frequency content, and so on), problems such as RF fingerprinting and jamming detection require data that captures the unique characteristics of devices and wireless channels. Therefore, significant research effort must be put forth to build large-scale wireless signal datasets to be shared with the research community at large.

13.5.2. Choice of I/Q Data Representation Format

It is still subject of debate within the research community what is the best data representation for RF deep learning applications. For example, an I/Q sample can be represented as a tuple of real numbers or a single complex number, while a set of I/Q samples can be represented as a matrix or a single set of numbers represented as a string. It is a common belief that there is no one-size-fits-all data representation solution for every learning problem and that

the right format might depend, among others, on the learning objective, choice of the loss function, and the learning problem considered [O'Shea et al., 2018].

13.5.3. Choice of Learning Model and Architecture

While there is a direct connection between images and tensors, the same cannot be concluded for wireless signals. For example, while 3-D tensors have been proven to effectively model images (*i.e.*, red, green, and blue channels), and kernels in convolutional layers are demonstrably powerful tools to detect edges and contours in a given image, it is still unclear if and how these concepts can be applied to wireless signals. Another major difference is that, while images can be considered as stationary data, RF signals are inherently stochastic, non-stationary and time-varying. This peculiar aspect poses significant issues in determining the right learning strategy in the wireless RF domain. For example, while CNN seems to be able to effective at solving problems such as modulation recognition [West and O'Shea, 2017, Karra et al., 2017, O'Shea et al., 2018], it is still unclear if this is the case for complex problems such as RF fingerprinting. Moreover, DL has traditionally been used in static contexts [Krizhevsky et al., 2012, Hinton et al., 2012], where the model latency is usually not a concern. Another fundamental issue absent in traditional deep learning is the need to satisfy strict constraints on resource consumption. Indeed, models with high number of neurons/layers/parameters will necessarily require additional hardware and energy consumption, which are clearly scarce resources in embedded systems. Particular care must be devoted, therefore, when designing learning architectures to solve learning problems in the RF domain.

13.6. Conclusion

This chapter provides a comprehensive account of advancements in physical layer rendered by the application of ANN. To accomplish this, we first provide readers with an overview of the most prevalent ANNs that are employed in the wireless communication networks. Next, we discuss the impact of ANN on designing physical layer for gathering signal intelligence. Realizing the importance of extending these techniques to hardware implementation, we discuss some steps that can be taken in those directions to ensure a rapid transition of these techniques to commercial hardware. Finally, we discuss some of the open problems that need to be tackled to further ease the adoption of ANN for wireless networks. The overarching goal of this chapter is to enable researchers with the fundamental tool to understand the applications of ANN in the context of signal intelligence in wireless communication and apprise them of the latest advancements that will consequently motivate new and existing works.

Bibliography

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

Satyam Agarwal and Swades De. eDSA: energy-efficient dynamic spectrum access protocols for cognitive radio networks. *IEEE Transactions on Mobile Computing*, 15(12):3057–3071, 2016.

J. Akeret, C. Chang, A. Lucchi, and A. Refregier. Radio frequency interfer-

ence mitigation using deep convolutional neural networks. *Astronomy and Computing*, 18:35–39, January 2017. doi: 10.1016/j.ascom.2017.01.002.

Elsayed Elsayed Azzouz and Asoke Kumar Nandi. *Automatic Modulation Recognition of Communication Signals*. Kluwer Academic Publishers, Norwell, MA, 1996. ISBN 0792397967.

M. Bkassiny, Y. Li, and S. K. Jayaweera. A survey on machine-learning techniques in cognitive radios. *IEEE Communications Surveys & Tutorials*, 15(3):1136–1159, Third 2013. ISSN 1553-877X. doi: 10.1109/SURV.2012.100412.00017.

David Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. *ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (UNITED KINGDOM)*, RSRE-MEMO-4148, 03 1988.

Guey-Yun Chang, Szu-Yung Wang, and Yuen-Xin Liu. A jamming-resistant channel hopping scheme for cognitive radio networks. *IEEE Transactions on Wireless Communications*, 16(10):6712–6725, 2017.

Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and Mérouane Debbah. Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks. *CoRR*, abs/1710.02913, 2017. URL http://arxiv.org/abs/1710.02913.

Tapiwa Moses Chiwewe and Gerhard Petrus Hancke. Fast convergence cooperative dynamic spectrum access for cognitive radio networks. *IEEE Transactions on Industrial Informatics*, 2017.

Cisco Systems. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016-2021 White Paper. http://tinyurl.com/zzo6766, 2017.

D. Czech, A. Mishra, and M. Inggs. A CNN and LSTM-based approach to classifying transient radio frequency interference. *Astronomy and Computing*, 25:52–57, October 2018. doi: 10.1016/j.ascom.2018.07.002.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Ericsson Incorporated. Ericsson Interim Mobility Report, February 2018. https://www.ericsson.com/assets/local/mobility-report/documents/2018/emr-interim-feb-2018.pdf, 2018.

Federal Communications Commission [2016]. Spectrum Crunch. https://www.fcc.gov/general/spectrum-crunch.

Federated Wireless. Citizens Broadband Radio Service (CBRS) Shared Spectrum: An Overview. https://www.federatedwireless.com/wp-content/uploads/2017/09/CBRS-Spectrum-Sharing-Overview.pdf, 2018.

S. Foulke, J. Jagannath, A. L. Drozd, T. Wimalajeewa, P. K. Varshney, and W. Su. Multisensor Modulation Classification (MMC) Implementation considerations USRP case study. In *Proc. of IEEE Conf. on Military Communications (MILCOM)*, Baltimore, MD, USA, October 2014.

S. Ghodeswar and P. G. Poonacha. An SNR estimation based adaptive hierarchical modulation classification method to recognize M-ary QAM and M-ary PSK signals. In *Proc. of International Conference on Signal Processing, Communication and Networking (ICSCN)*, pages 1–6, Chennai, India, March 2015. doi: 10.1109/ICSCN.2015.7219867.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016a. http://www.deeplearningbook.org.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016b.

Alharbi Hazza, Mobien Shoaib, Saleh AlShebeili, and Alturki Fahd. Automatic modulation classification of digital modulations in presence of HF noise. *EURASIP Journal on Adv. in Signal Processing*, 2012:238, 2012.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jrgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993.

Jen-Feng Huang, Guey-Yun Chang, and Jian-Xun Huang. Anti-jamming rendezvous scheme for cognitive radio networks. *IEEE Transactions on Mobile Computing*, 16(3):648–661, 2017.

J. Jagannath, H. M. Saarinen, and A. L. Drozd. Framework for Automatic Signal Classification Techniques (FACT) for Software Defined Radios. In *Proc. of IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA)*, Verona, NY, USA, May 2015.

J. Jagannath, D. O'Connor, N. Polosky, B. Sheaffer, L. N. Theagarajan, S. Foulke, P. K. Varshney, and S. P. Reichhart. Design and Evaluation of Hierarchical Hybrid Automatic Modulation Classifier using Software Defined Radios. In *Proc. of IEEE Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, January 2017.

J. Jagannath, S. Furman, T. Melodia, and A. Drozd. Design and experimental evaluation of a cross-layer deadline-based joint routing and spectrum allocation algorithm. *IEEE Transactions on Mobile Computing*, pages 1–1, 2018a. ISSN 1536-1233. doi: 10.1109/TMC.2018.2866093.

J. Jagannath, N. Polosky, D. OConnor, L. Theagarajan, B. Sheaffer, S. Foulke, and P. Varshney. Artificial Neural Network based Automatic Modulation Classifier for Software Defined Radios. In *Proc. of IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA, May 2018b.

C. Jiang, H. Zhang, Y. Ren, Z. Han, K. C. Chen, and L. Hanzo. Machine

learning paradigms for next-generation wireless networks. *IEEE Wireless Communications*, 24(2):98–105, April 2017. ISSN 1536-1284. doi: 10.1109/MWC.2016.1500356WC.

Xiaocong Jin, Jingchao Sun, Rui Zhang, Yanchao Zhang, and Chi Zhang. SpecGuard: spectrum misuse detection in dynamic spectrum access systems. *to appear, IEEE Transactions on Mobile Computing*, 2018.

L-S. Bouchard K. Youssef. Training artificial neural networks with reduced computational complexity. URL https://gtp.autm.net/public/project/34861/.

K. Karra, S. Kuzdeba, and J. Petersen. Modulation recognition using hierarchical deep neural networks. In *Proc. of IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–3, Baltimore, MD, USA, March 2017. doi: 10.1109/DySPAN.2017.7920746.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

A. Kubankova, J. Prinosil, and D. Kubanek. Recognition of Digital Modulations Based on Mathematical Classifier. In *Proc. of the European Conference of Systems (ECCS)*, Stevens Point, WI, 2010.

M. Kulin, T. Kazaz, I. Moerman, and E. De Poorter. End-to-end learning from spectrum data: A deep learning approach for wireless signal identification in

spectrum monitoring applications. *IEEE Access*, 6:18484–18501, 2018. doi: 10.1109/ACCESS.2018.2818794.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

Yann LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, pages 143–155, 1989.

Lu Lv, Jian Chen, Qiang Ni, Zhiguo Ding, and Hai Jiang. Cognitive non-orthogonal multiple access with cooperative relaying: A new wireless frontier for 5g spectrum sharing. *IEEE Communications Magazine*, 56(4):188–195, 2018.

Q. Mao, F. Hu, and Q. Hao. Deep learning for intelligent wireless networks: A comprehensive survey. *to appear, IEEE Communications Surveys & Tutorials*, 2018. doi: 10.1109/COMST.2018.2846401.

R. F. Molanes, J. J. Rodrguez-Andina, and J. Faria. Performance characterization and design guidelines for efficient processor - FPGA communication in Cyclone V FPSoCs. *IEEE Transactions on Industrial Electronics*, 65(5): 4368–4377, May 2018. ISSN 0278-0046. doi: 10.1109/TIE.2017.2766581.

T. J. O'Shea, T. Roy, and T. C. Clancy. Over-the-air deep learning based radio signal classification. *IEEE Journal of Selected Topics in Signal Processing*, 12 (1):168–179, Feb 2018. ISSN 1932-4553. doi: 10.1109/JSTSP.2018.2797022.

Timothy J. O'Shea and Johnathan Corgan. Convolutional radio modulation recognition networks. *CoRR*, abs/1602.04105, 2016. URL http://arxiv.org/abs/1602.04105.

Timothy James O'Shea and Jakob Hoydis. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):563–575, 2017.

O. Ozdemir, Ruoyu Li, and P.K. Varshney. Hybrid Maximum Likelihood Modulation Classification Using Multiple Radios. *IEEE Communications Letters*, 17(10):1889–1892, October 2013.

O. Ozdemir, T. Wimalajeewa, B. Dulek, P. K. Varshney, and W Su. Asynchronous Linear Modulation Classification with Multiple Sensors via Generalized EM Algorithm. *IEEE Transactions on Wireless Communications*, 14 (11):6389–6400, November 2015.

S. U. Pawar and J. F. Doherty. Modulation recognition in continuous phase modulation using approximate entropy. *IEEE Transactions on Information Forensics and Security*, 6(3):843–852, Sept 2011. ISSN 1556-6013. doi: 10.1109/TIFS.2011.2159000.

Pete Bennett (EE Times). The Why, Where and What of Low-Power SoC Design. https://www.eetimes.com/document.asp?doc_id=1276973, 2004.

F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957. URL https://books.google.com/books?id=P_XGPgAACAAJ.

F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962. URL https://books.google.com/books?id=7FhRAAAAMAAJ.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. ¡a href="absps/sutherlandbp.pdf"¿Commentary from News and Views section of Nature¡/a¿.

Ahmed Selim, Francisco Paisana, Jerome A. Arokkiam, Yi Zhang, Linda Doyle, and Luiz A. DaSilva. Spectrum monitoring for radar bands using deep convolutional neural networks. *CoRR*, abs/1705.00462, 2017. URL http://arxiv.org/abs/1705.00462.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Huaxia Wang Hathal Alwageed Shengliang Peng, Hanyu Jiang and Yu-Dong Yao. Modulation classification using convolutional neural network based deep learning model. *WOCC*, 2017.

Q. Shi and Y. Karasawa. Automatic modulation identification based on the probability density function of signal phase. *IEEE Transactions on Communications*, 60(4):1033–1044, April 2012. ISSN 0090-6778. doi: 10.1109/TCOMM.2012.021712.100638.

Hossein Shokri-Ghadikolaei, Federico Boccardi, Carlo Fischione, Gabor Fodor, and Michele Zorzi. Spectrum sharing in mmwave cellular networks via cell association, coordination, and beamforming. *IEEE Journal on Selected Areas in Communications*, 34(11):2902–2917, 2016.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL http://arxiv.org/abs/1505.00387.

Bülent Üstün, Willem J. Melssen, and Lutgarde M. C. Buydens. Facilitating the application of support vector regression by using a universal pearson vii function based kernel. 2005.

Miguel Angel Vázquez, Luis Blanco, and Ana I Pérez-Neira. Hybrid analog–digital transmit beamforming for spectrum sharing backhaul networks. *IEEE transactions on signal processing*, 66(9):2273, 2018.

Tianqi Wang, Chao-Kai Wen, Hanqing Wang, Feifei Gao, Tao Jiang, and Shi Jin. Deep learning for wireless physical layer: Opportunities and challenges. *China Communications*, 14(11):92–111, 2017.

N. E. West and T. O'Shea. Deep architectures for modulation recognition. In *Proc. of IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–6, Baltimore, MD, USA, March 2017. doi: 10.1109/DySPAN.2017.7920754.

T. Wimalajeewa, J. Jagannath, P. K. Varshney, A. L. Drozd, and W. Su. Distributed Asynchronous Modulation Classification Based on Hybrid Maximum Likelihood Approach. In *Proc. of IEEE Conf. on Military Communications (MILCOM)*, Tampa, FL, USA, October 2015.

Felix Winterstein, Samuel Bayliss, and George A Constantinides. High-level synthesis of dynamic data structures: A case study using vivado hls. In *Proc. of International Conference on Field-Programmable Technology (FPT)*, pages 362–365, Kyoto, Japan, 2013.

M. L. D. Wong and A. K. Nandi. Automatic digital modulation recognition using spectral and statistical features with multi-layer perceptrons. In *Proc. of the Sixth International Symposium on Signal Processing and*

its Applications (Cat.No.01EX467), volume 2, pages 390–393, 2001. doi: 10.1109/ISSPA.2001.950162.

Xilinx Inc. AXI Reference Guide, UG761 (v13.1) March 7, 2011. https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf, 2011.

J. L. Xu, W. Su, and M. Zhou. Software-defined radio equipped with rapid modulation recognition. *IEEE Transactions on Vehicular Technology*, 59(4): 1659–1667, May 2010. ISSN 0018-9545. doi: 10.1109/TVT.2010.2041805.

K. Youssef, N. N. Jarenwattananon, and L. Bouchard. Feature-preserving noise removal. *IEEE Transactions on Medical Imaging*, 34(9):1822–1829, Sept 2015. ISSN 0278-0062. doi: 10.1109/TMI.2015.2409265.

K. Youssef, L.-S. Bouchard, K. Z. Haigh, H. Krovi, J. Silovsky, and C. P. Vander Valk. Machine Learning Approach to RF Transmitter Identification. *ArXiv e-prints*, November 2017.

Liyang Zhang, Francesco Restuccia, Tommaso Melodia, and Scott Pudlewski. Learning to detect and mitigate cross-layer attacks in wireless networks: Framework and applications. In *Proc. of IEEE Conf. on Communications and Network Security*, Las Vegas, NV, USA, October 2017.

Yi Ting Zhou and Rama Chellappa. Computation of optical flow using a neural network. *IEEE 1988 International Conference on Neural Networks*, pages 71–78 vol.2, 1988.