Automatic Machine Learning for Multi-Receiver CNN Technology Classifiers

Amir-Hossein Yazdani-Abyaneh
yazdaniabyaneh@email.arizona.edu
ECE Department, University of Arizona
Tucson, Arizona, USA

Marwan Krunz
krunz@email.arizona.edu
ECE Department, University of Arizona
Tucson, Arizona, USA

ABSTRACT

Convolutional Neural Networks (CNNs) are one of the most studied family of deep learning models for signal classification, including modulation, technology, detection, and identification. In this work, we focus on technology classification based on raw I/Q samples collected from multiple synchronized receivers. As an example use case, we study protocol identification of Wi-Fi, LTE-LAA, and 5G NR-U technologies that coexist over the 5 GHz *Unlicensed National Information Infrastructure (U-NII)* bands. Designing and training accurate CNN classifiers involve significant time and effort that goes to fine-tuning a model's architectural settings (e.g., number of convolutional layers and their filter size) and determining the appropriate hyperparameter configurations, such as learning rate and batch size. We tackle the former by defining architectural settings themselves as hyperparameters. We attempt to automatically optimize these architectural parameters, along with other preprocessing (e.g., number of I/Q samples within each classifier input) and learning hyperparameters, by forming a *Hyperparameter Optimization (HyperOpt)* problem, which we solve in a near-optimal fashion using the *Hyperband* algorithm. The resulting near-optimal CNN (OCNN) classifier is then used to study classification accuracy for OTA as well as simulations datasets, considering various SNR values. We show that using a larger number of receivers to construct multi-channel inputs for CNNs does not necessarily improve classification accuracy. Instead, this number should be defined as a preprocessing hyperparameter to be optimized via Hyperband. OTA results reveal that our OCNN classifiers improve classification accuracy by 24.58% compared to manually tuned CNNs. We also study the effect of min-max normalization of I/Q samples within each classifier's input on generalization accuracy over simulated datasets with SNRs other than training set's SNR, and show an average of 108.05% improvement when I/Q samples are normalized.

CCS CONCEPTS

• **Machine Learning** → **Neural Networks**; *Automatic Machine Learning*; HyperOpt; • **Wireless communication** → Spectrum sensing.

This work is licensed under a Creative Commons Attribution-NonCommercial International 4.0 License.

WiseML '22, May 19, 2022, San Antonio, TX, USA.
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9277-8/22/05.
https://doi.org/10.1145/3522783.3529524

KEYWORDS

Signal classification, CNN, AutoML, multi-receiver, HyperOpt, Hyperband, Wi-Fi, LTE-LAA, 5G NR-U, SDR

ACM Reference Format:
Amir-Hossein Yazdani-Abyaneh and Marwan Krunz. 2022. Automatic Machine Learning for Multi-Receiver CNN Technology Classifiers. In *Proceedings of the 2022 ACM Workshop on Wireless Security and Machine Learning (WiseML '22), May 19, 2022, San Antonio, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3522783.3529524

1 INTRODUCTION

Wi-Fi, LTE Licensed Assisted Access (LAA) and 5G New Radio Unlicensed (NR-U) can operate over the 5 GHz Unlicensed National Information Infrastructure (U-NII) bands. To coexist harmoniously, these technologies use variants of Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) to access unlicensed channels. However, the standards for each technology specify different CSMA/CA parameters, e.g., maximum backoff value, clear channel access (CCA) thresholds, etc., resulting in heterogeneous channel access behavior [1, 2]. Studies have shown that for these technologies to achieve fair coexistence, they have to adapt their channel access parameters and schedule their transmissions based on the transmission activities of all coexisting technologies [3]. Fair coexistence can be greatly facilitated via the classification of active transmissions into their respective technologies, which require technology-specific receivers that search for distinct features within the received waveforms (e.g., 802.11 preamble, LTE OFDM grid structure, etc.). However, to ease the burden of having multiple technologies implemented and interconnected on commercial devices, an alternative approach adopted in this paper, is to use a universal classifier based on Deep Neural Networks (DNNs), such as *Convolutional Neural Networks (CNNs).*

CNN-based signal classification has been studied to fulfill different goals, including modulation classification [4, 5], technology (protocol) identification [6, 7], transmitter authentication [8, 9], LTE spectrogram generation [10], and generation of synthetic modulated waveforms via Generative Adversarial Networks (GANs) [11]. These classifiers suggest classes based on received raw I/Q samples. Directly working with I/Q samples bypasses the need for specialized signal processing (e.g., correcting for carrier frequency offset and channel effects), enabling classification to be done within microseconds [12].

In this paper, we focus on unlicensed transmissions in the 5 GHz U-NII frequency bands, where Wi-Fi, LTE-LAA, and 5G NR-U technologies coexist. Gaining insight from prior research conducted on signal classification using deep learning models, we also choose CNNs as our base classifiers and the baseband I/Q samples as inputs for such classifiers. In a rich scattering environment, including the

5 GHz U-NII bands, multi-path fading results in sufficient spatial diversity between the I/Q samples collected at differently located receivers, including multiple receive chains of the same device. Accordingly, we consider a classifier whose I/Q samples are obtained from multiple synchronized receivers. This is analogous to an image classification problem, where images are fed to CNNs through multiple color channels, i.e., red, green, and blue [13]. We study the implications of employing multiple I/Q streams on the classification accuracy.

Much time and effort go into manually designing the architecture of a CNN classifier and tuning its hyperparameters (e.g., learning rate, batch size, etc.). Using *Automatic Machine Learning (AutoML)* techniques, specifically *Hyperparameter Optimization (HyperOpt)* [14], we define the architectural settings of our CNN model as hyperparameters, and along with other learning and preprocessing hyperparameters, run a HyperOpt algorithm called *Hyperband* [15] to find near-optimal hyperparameters and architectural design settings. Hyperband has shown a 20× increase in performance (e.g., validation loss) over random search, which is often used for HyperOpt problems with high-dimensional spaces (e.g., DNN design). Random search is often used instead of exhaustive search, which has an exponential complexity.

Considering different ranges of SNR values, we generate two datasets, one obtained over the air (OTA) via USRP experiments and the other using MATLAB simulations. OTA results show that our near-optimal CNNs improve classification accuracy by 24.58%, on average, compared to manually tuned CNN classifiers. For our simulation datasets, we show that normalizing the classifier's inputs improves generalization accuracy over datasets with SNR values other than the training set's SNR by 108.05%. Moreover, our evaluation results suggest that using the maximum available number of receivers to collect I/Q samples from does not improve classification accuracy for some CNN architectures, and this number should be set as a hyperparameter to be optimized within HyperOpt.

The paper is organized as follows. Section 2 provides a summary of related works. In Section 3, we introduce our multi-receiver signal classification problem. HyperOpt, Hyperband, and hyperparameter settings of near-optimal CNN structures are presented in Section 4. We provide our evaluation results in Section 5 and conclude the paper in Section 6.

2 RELATED WORKS

Zhang et al. [6] studied a similar technology classification problem of Wi-Fi, LTE, and 5G signals, where they evaluated the classification performance of CNNs and LSTMs using I/Q samples received from a single receiver. In [7], Bitar et al. studied the classification of I/Q samples for Bluetooth, ZigBee, and 802.11n in the context of IoT. O'Shea et al. [4] studied the problem of modulation classification and showed improved classification performance for CNNs compared with conventional ML classification algorithms (e.g., SVMs and decision trees). Their work was extended in [5] by studying different wireless communication imperfections, such as carrier frequency offset (CFO) and symbol rate offset (SRO) in SDR experimental setups. Shi et al. [16] considered primary/secondary user (PU/SU) signal classification in a dynamic spectrum access scenario as a form of modulation classification problem, where PUs and SUs adopt different modulations for their transmissions, and utilized CNN classifiers. Sankhe et al. [9] proposed a transmitter authentication framework for identifying a set of known transmitters using the fact that I/Q samples transmitted by different transmitters are affected differently by hardware impairments (e.g., CFO). The authors in [17] used multiple streams of I/Q samples from different receivers for modulation classification. They trained individual CNNs for individual receivers, and used cooperative decision rules (e.g., ensemble average) to suggest a final classification outcome. In contrast, in our work, we form a single input from all I/Q streams (i.e., multi-channel inputs) and feed it to one CNN classifier. All the above works used manually tuned architectural and hyperparameter settings.

3 MULTI-RECEIVER SIGNAL CLASSIFICATION

We consider a technology classification problem, where several Wi-Fi, LTE-LAA, and 5G NR-U transmitters coexist over a shared channel. We assume an interleaving *spectrum sharing* scenario, where at any given time, only one type of transmission is possible. At the classifier, signals are received over N synchronized receivers, $N \geq 1$, at a sampling rate of f_s (see Figure 1). Let x denote the baseband transmitted signal. The set of received baseband signals $\boldsymbol{y}$ at the classifier are given by:

$$\boldsymbol{y} = \boldsymbol{H}x + \boldsymbol{\mathcal{N}} \tag{1}$$

where $\boldsymbol{y} = [y_1, \cdots, y_N]^T$ and y_i represents the baseband I/Q samples at the ith receiver. Throughout the paper, we refer to the set of I/Q samples at given receiver as an *I/Q stream*. $\boldsymbol{H} = [H_1, \cdots, H_N]^T$ where $H_i = M_i e^{j\phi_i}$, $i = 1, \cdots, N$, is the baseband-equivalent complex channel between the transmitter and the ith receiver; M_i and ϕ_i represent the amplitude of channel fading and its phase shift, respectively. $\boldsymbol{\mathcal{N}} = [\mathcal{N}_1, \cdots, \mathcal{N}_N]^T$, and $\mathcal{N}_i$ models the AWGN noise at the ith receiver. Over a time window T, f_sT I/Q samples are collected at each receiver. Then, n I/Q streams are selected for further processing, where $1 \leq n \leq N$ (n is defined as a hyperparameter in Section 4). Next, a non-overlapping window of size w is applied to these I/Q streams, and each window of I/Q samples is then min-max normalized to $[0, 1]$. After normalization, inputs of size $w \times 2 \times n$ are created[1]. We denote such an input by $I_{w\times2\times n}$:

$$I_{w\times2\times n} = \begin{bmatrix} [y_1^{(I)}[1] \cdots y_n^{(I)}[1]] & [y_1^{(Q)}[1] \cdots y_n^{(Q)}[1]] \\ [y_1^{(I)}[2] \cdots y_n^{(I)}[2]] & [y_1^{(Q)}[2] \cdots y_n^{(Q)}[2]] \\ \vdots & \vdots \\ [y_1^{(I)}[w] \cdots y_n^{(I)}[w]] & [y_1^{(Q)}[w] \cdots y_n^{(Q)}[w]] \end{bmatrix} \tag{2}$$

where $y_i^{(I)}[l]$ and $y_i^{(Q)}[l]$ are the lth in-phase and quadrature samples received from the ith receiver. After normalization, each $I_{w\times2\times n}$ is fed to a CNN model to generate a class probability, and the signal type with the highest class probability is considered as the observed technology. Windows of I/Q streams are normalized to improve CNN's classification accuracy over datasets with SNR settings different than the training set's (i.e., *unobseved datasets*), resulting in improved generalization performance. This is desirable since it is

[1] Samples collected over a period T result in $\lfloor \frac{Tf_s}{w} \rfloor$ I/Q inputs of size $w \times 2 \times n$.

Table 1: BCNN's manually tuned hyperparameters.

Activation function	ReLU
Loss	Categorical cross-entropy
Overfitting algorithm	Early stopping (patience = 6)
Batch size	128
Optimizer	Adam (learning rate = 10^{-4})
Window size (w)	512
Max training epochs	100

not feasible to gather training datasets for all possible SNR settings. Also, proper configuration of w and n (i.e., size of each $I_{w\times 2\times n}$) can improve classification and generalization accuracies greatly. We shed more light on this in the next section.

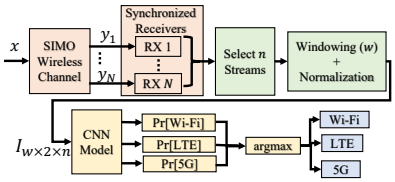

Figure 1: Classification pipeline.

4 NEAR-OPTIMAL CNN DESIGN: AN AUTOML APPROACH

Although CNN models have been used to provide high accuracy for modulation and technology classification, significant effort is spent on fine-tuning the hyperparameters and setting various design parameters of such models. Accordingly, we seek to optimize the CNN's architectural design, preprocessing (e.g., w), and learning hyperparameters (e.g., learning rate). We aim at providing near-optimal values for these parameters by forming/solving a *Hyperparameter Optimization (HyperOpt)* problem. Along with *Neural Architecture Search (NAS)* and *Meta Learning*, HyperOpt belongs to the field of AutoML [14]. One type of NAS, which is also applied in our work, is to treat the architectural design features, such as the number of layers, kernel sizes, etc., as hyperparameters for HyperOpt. Specifically, we explore a systematic way for HyperOpt, called *Hyperband* [15].

To compare our near-optimal CNN classifiers with a manually designed CNN, we use a single-convolutional layer CNN model called *Baseline CNN (BCNN)*, see Figure 2. We evaluate the performance of BCNN for different values of n in Section 5. Table 1 shows BCNN's manually tuned learning hyperparameters.

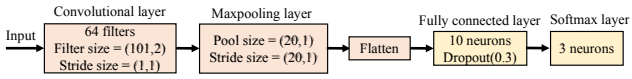

Figure 2: BCNN architecture.

4.1 Hyperparameter Optimization (HyperOpt)

The goal of HyperOpt is to find an optimal hyperparameter configuration for a learning algorithm in a resource-efficient manner in terms of computational resources and training-time budget. To formally define a HyperOpt problem, we follow the approach in [14] and introduce the following notation:

- M: Machine learning algorithm with H hyperparameters.
- Δ_h: Domain of the hth hyper parameter, $h = 1, \cdots, H$. The domain can consist of real values (e.g., learning rate), integers (e.g., number of fully connected layers), binary values (e.g., whether to shuffle the data at each training epoch), or a categorical value (e.g., choice of learning optimizer).
- Δ : Hyperparameter configuration space, $\Delta = \prod_{h=1}^{H} \Delta_h$.
- μ: A vector of hyperparameter settings ($\mu \in \Delta$).
- M_μ: Machine learning algorithm M with a hyperparameter configuration of μ.

Given a dataset D, the goal of HyperOpt is to find an optimal hyperparameter realization μ^* by solving the following bi-level optimization problem:

$$\mu^* = \underset{\mu\in\Delta}{\operatorname{argmin}}\, \mathbb{E}_{(D^{(\text{Train})}, D^{(\text{Valid})})\sim D}[L(M_\mu, D^{(\text{Train})}, D^{(\text{Valid})})]. \quad (3)$$

$L(M_\mu, D^{(\text{Train})}, D^{(\text{Valid})})$ is the validation loss (e.g., cross-validation error) of the model generated by M_μ, which is trained on $D^{(\text{Train})}$, and validated on $D^{(\text{Valid})}$, where $D^{(\text{Train})}$ and $D^{(\text{Valid})}$ are disjoint subsets of dataset D. To solve (3), we can use black-box optimization techniques (e.g., grid search, which is exponential in the number of hyperparameters), random search, population-based methods (e.g., genetic algorithms), and Bayesian optimization [18]. These methods are expensive to evaluate for large datasets and complex models such as CNNs. The convergence time of HyperOpt can be reduced by applying multi-fidelity optimization methods that train an algorithm with a vector of hyperparameters μ on a small subset of data, train for a few iterations, or run only on a subset of features. Multi-fidelity methods tend to minimize the low fidelity approximations of the actual loss function. Hyperband [15] is a pure exploration Multi-Armed Bandit (MAB) algorithm that improves on the *successive halving* algorithm, which is another bandit-based strategy for multi-fidelity optimization. Successive halving finds the best hyperparameters by initially associating computational (e.g., number of iterations to train) or storage (e.g., memory) budgets to different hyperparameter configurations of an ML algorithm/model, and trains their respective models for the number of associated training iterations. Models are then evaluated and the worst performing half of the models/algorithms are dropped, while the budget for the remaining half is doubled. The former process continues until one model/algorithm with a specific hyperparameter setting is left. See Figure 3 for an example of successive halving applied on eight number of hyperparameter configurations. Successive halving requires the user to decide whether to try many configurations and assign a small budget to each or to evaluate a relatively smaller set of configurations with larger assigned budgets. Assigning large budgets to poor configurations is a waste of resources, while assigning small budgets to competent configurations prematurely terminates their evaluations. Hyperband is designed to improve successive halving by systematizing budget partitioning for different hyperparameter configurations. It divides the total budget to several combinations

of number of configurations and their associated budgets. The number of different configurations represent the number of arms in the MAB problem. For each arm (i.e., number of hyperparameters to consider), Hyperband calls the successive halving algorithm and keeps track of the validation loss for each evaluated hyperparameter. Finally, Hyperband recommends the hyperparameter setting with the lowest validation loss as a solution for (3). It is faster and more efficient than the successive halving algorithm, and due to its cheap low-fidelity evaluations, Hyperband has shown improved performance over random search and black-box Bayesian optimization for data subsets, feature subsets and iterative algorithms such as SGD that is used for training CNNs.

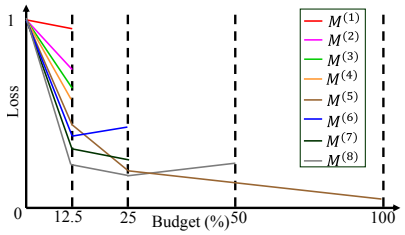

Figure 3: Successive halving for eight arbitrary hyperparameter configurations. The approach starts with eight models with different configurations and consecutively applies successive halving until only one model remains.

4.2 Hyper-CNN & Hyperparameter Settings

We define the architectural design settings of our CNN model to be optimized by Hyperband as hyperparameters. We call this hypothetical architecture *Hyper-CNN*. In image classification, convolutional and max-pooling layers have been used for feature extraction. These layers are often followed by a set of fully connected layers that extract nonlinear relations from the former extracted features to perform classification. We follow the same intuition and let Hyper-CNN to consist of two parts (see Figure 4): *Feature Extractor (FE)* and *Classifier (CL)*. FE consists of one or more *Convolutional Blocks (Convblocks)*. Each Convblock includes a convolutional layer, a dropout layer, and a Maxpooling layer. Within each convolutional layer, the number of filters and their sizes, the dropout rate, Maxpooling layer's pool size and stride size, and the choice of activation function are defined as hyperparameters. We also let the number of Convblocks in FE to be a hyperparameter. CL contains fully connected layers, each followed by a dropout layer with a tunable rate. We set as hyperparameters the number of fully connected layers, number of neurons in each layer, the dropout rate, and the choice of activation function. Moreover, we define as hyperparameters the choice of our training optimizer, its learning rate, batch size, and whether to shuffle training data at each epoch or not. We let the window size, w, and the number of selected I/Q streams, n, to be hyperparameters to get optimized by Hyperband, as well. Hyperband requires the domain of each hyperparameter, i.e., Δ_h to be bounded. Table 2 states the bounds for each hyperparameter. We fix the filter stride size of convolutional filters to $(1, 1)$ and the maximum number of training epochs to 100.

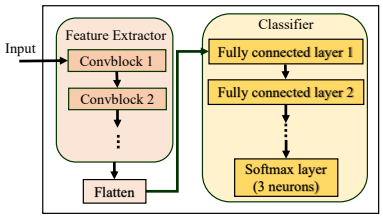

Figure 4: Hyper-CNN architecture.

Table 2: Hyper-CNN's hyperparameter domains ("Fully connected" is abbreviated by FC).

Hyperparameter	Domain
FE: Number of Convblocks	$\{1, 2, 3\}$
FE: Num. filters at convolutional layer	$\{16, 32, \cdots, 160\}$
FE: Filter size at convolutional layer	$\{16, 32, \cdots, 160\} \times \{2\}$
FE: Pool size at the Maxpooling layer	$\{5, 15, \cdots, 75\}$
FE: Stride size at the Maxpooling layer	$\{5, 15, \cdots, 60\}$
CL: Num. FC layers	$\{1, 2, 3, 4\}$
CL: Num. neurons at FC layer	$\{10, 20, \cdots, 100\}$
CL: Num. neurons at FC layer	$\{10, 20, \cdots, 100\}$
FE and CL: Activation function	{ReLU, tanh, sigmoid}
FE and CL: Dropout rate	$\{0, 0.15, \cdots, 0.6\}$
Preprocessing: Window size (w)	$\{128, 160, \cdots, 512\}$
Preprocessing: Normalization	{True, False}
Preprocessing: Num. I/Q streams	$\{1, 2, \cdots, N\}$
Learning: Optimizer	{ADAM, SGD, RMSProp}
Learning: Learning rate	$\{0.01, 0.001, \cdots, 10^{-5}\}$
Learning: Early stopping	{True, False}
Learning: Batch size	$\{32, 64, \cdots, 320\}$
Learning: Shuffle data	{True, False}

Table 3: Waveform generation settings.

Technology	Modulation Scheme	Code Rate
Wi-Fi (MC=6, 802.11 ac)	64-QAM	3/4
LTE (RC9)	64-QAM	3/4
5G DL FR1 (SCS= 15 kHz)	64-QAM	1/2

To get near-optimal classifiers for a dataset D, we tune Hyper-CNN via Hyperband using $D^{(\text{Train})}$ and $D^{(\text{Valid})}$ as described in (3). We call the tuned Hyper-CNN structure *Optimal CNN (OCNN)*. Hyperband recommends different OCNNs for different datasets.

5 EVALUATION

We construct labeled datasets by generating Wi-Fi, LTE, and 5G waveforms using MATLAB WLAN [19], LTE [20], and 5G [21] communication toolboxes. See Table 3 for a detailed description of generated waveforms, where all waveforms are 20 MHz wide. For each waveform type, we generate 500 different waveforms by randomizing their payloads. Datasets are generated via OTA Software Defined Radio (SDR) experiments and simulations for different SNR ranges. Each generated waveform is then passed through a multi-path fading wireless channel and received by N number of receivers with a sampling rate of f_s MHz – these settings vary for OTA experiments and simulations. I/Q streams are then divided into inputs of size $w \times 2 \times n$ based on the settings of w and n. Depending on its underlying technology, each $I_{w\times 2\times n}$ is labeled as a one-hot coded vector of size 1×3.

Our real-world OTA experiments are based on SDRs. We use National Instruments (NI) SDRs NI-USRP 2921's. Our USRP setup consists of one transmitter and three receivers (i.e., $N = 3$) each

approximately 3 m away from the transmitter (see Figure 5). Adjacent receivers are separated by 50 cm and synchronized via an Ettus Octoclock CDA-2990 with a maximum sampling rate of f_s =10 MHz. We observe lots of uncontrolled interference over the 5 GHz band. Hence, to not interfere with ongoing Wi-Fi transmissions, we choose a center frequency of 2.495 GHz (ISM band) for these experiments. The three receiving USRPs are connected to the same host PC via a switch. We use MATLAB SDR toolbox to transmit and receive generated waveforms. We construct multiple datasets for different transmission gains in the set {0, 5, 10, 15, 20} dB which translates to average (over all three receivers) received SNR values of {1.5, 3.5, 5.5, 8.5, 11.5} dB. Each generated dataset has 32,000 inputs for each technology.

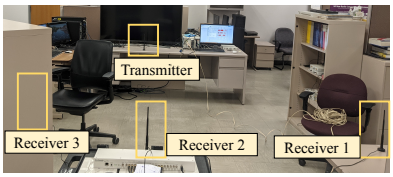

Figure 5: USRP experiment setup (Receiver 3 is inside a metallic book shelve).

To evaluate our approach for a larger number of receivers and a higher sampling rate, we simulate the wireless channel considering an 802.11ax™ (TGax) indoor MIMO channel model [22], see Table 4. We generate datasets for SNR values in the set {−10, −5, 0, 5, 10, 15, 20} dB. For each simulated dataset, we randomly select 100,000 inputs for each technology type.

Table 4: SIMO wireless channel settings.

Parameter	Value
Sampling rate	20 Msps
Delay profile	Model-D
Channel bandwidth	20 MHz
Carrier frequency	5 GHz
Environmental speed	0 m/s
Transmitter-receiver distance	3.8 m
Number of receive antennas (N)	6
Large-scale fading effects	Pathloss and shadowing

For each OTA and simulated dataset considered for evaluations, we use 60%, 20%, and 20% of inputs for training, validation, and testing, respectively.

5.1 Optimal CNN Architectures

We run Hyperband after formulating the HyperOpt problem as presented in (3) for all generated datasets using their respective training and validation sets. Table 5 shows a summary of optimal hyperparameter settings for both OTA and simulated datasets. All OCNNs select ReLU as their activation function, and ADAM as their learning optimizer. They all choose to normalize their inputs, use early stopping, and shuffle the training set at the beginning of each training epoch.

It can be concluded from the table that most OCNN structures prefer having multi-channel inputs, i.e., selecting I/Q streams from multiple receivers, rather than using single-channel inputs. Moreover, on average, OCNNs choose w to be 347, select 1.58 number of Convblocks and 1.83 number of fully connected layers. Due to space limit we can not show other detailed architectural settings (e.g., filter size). However, Figure 6 shows the detailed architecture of OCNN tuned for the simulation dataset with an SNR of 0 dB. In Section 5.3, we show that this OCNN architecture achieves the highest average generalization performance compared to other OCNNs.

Table 5: OCCN hyperparameter settings for different datasets.

SNR (dB)	Hyperparameter Settings			
	n	w	Convblocks	Fully Connected
1.5 (OTA)	2	384	2	3
3.5 (OTA)	3	256	2	2
5.5 (OTA)	1	448	2	3
8.5 (OTA)	3	224	1	1
11.5 (OTA)	2	448	1	2
−10 (Simulations)	6	320	1	1
−5 (Simulations)	1	416	1	2
0 (Simulations)	5	448	2	1
5 (Simulations)	5	228	2	1
10 (Simulations)	3	352	1	2
15 (Simulations)	6	320	2	3
20 (Simulations)	6	320	2	3

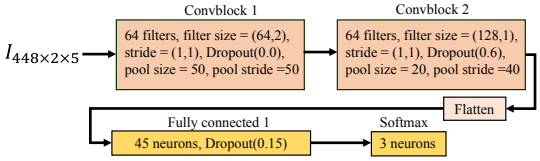

Figure 6: OCNN architecture for simulation dataset with an SNR value of 0 dB.

5.2 OCNN vs. BCNN

In Figures 7(a) and 7(b), we show classification accuracy of BCNN with different number of selected I/Q streams and OCNN architectures vs. received SNR, for OTA experiments and simulations, respectively. On average, OCNNs improve classification accuracy of best-performing BCNNs by 24.58% and 46.15%, for OTA experiments and simulations, respectively. Also it can be concluded that increasing the number of selected I/Q streams does not necessarily increase the accuracy and it is better to define it as a hyperparameter to be optimized by Hyperband. Moreover, OCNNs' classification accuracy over simulation datasets shows that the classification task becomes easier to solve as SNR is increased; whereas, over low SNR values, OCNNs improves accuracy by a greater margin.

5.3 Generalization over Unobserved Datasets

Generalization accuracy is the classification accuracy of a model when it is tested on datasets with SNR values other than the training set's SNR. In Figure 8 we show the average generalization accuracy vs. the SNR of the training using simulations. It can be seen that normalization improves the generalization accuracy by 108.05%, on average. We also show the generalization accuracy evaluated on

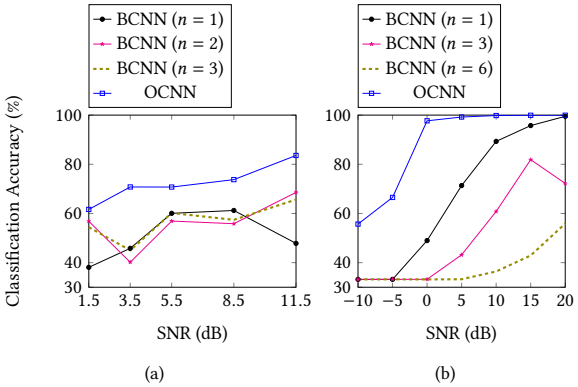

Figure 7: BCNN and OCNN classification accuracy vs. SNR for (a) OTA USRP and (b) simulation datasets.

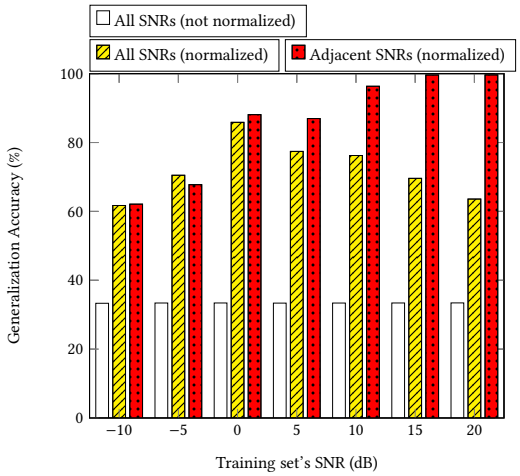

Figure 8: Average generalization accuracy with/without normalized inputs vs. SNR setting which the model was trained on.

datasets with SNR values that are 5 dB higher or lower than the training set's (except for −10 and 20 dB, where we only test on −5 and 15 dB, respectively). It can be seen that on average models are 19.7% more accurate for close SNR values.

6 CONCLUSIONS

In this work, we considered the protocol classification problem of raw received I/Q samples for Wi-Fi, LTE-LAA, and 5G NR-U. We presented a multi-receiver technology classification method that uses a CNN model to classify multiple streams of I/Q samples collected from differently located synchronized receivers. We provided an AutoML approach to jointly optimize CNN architectures, preprocessing hyperparameters, such as number of selected I/Q streams, and learning configurations (e.g., choice of optimizer). Our USRP experiments showed 20.58% gain when optimal CNNs are used instead of manually tuned CNNs. We also showed that increasing the number of I/Q streams does not necessarily increase classification accuracy and it should be optimized as a part of our HyperOpt problem. For our simulated datasets, we found that normalizing I/Q samples improves generalization accuracy by 108.05%.

7 ACKNOWLEDGMENTS

This research was supported by the U.S. Army Small Business Innovation Research Program Office and the Army Research Office under Contract No. W911NF-21-C-0016, by NSF (grants CNS-1563655, CNS-1731164, and IIP-1822071), and by the Broadband Wireless Access Applications Center (BWAC). Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of NSF or ARO.

REFERENCES

[1] B. Chen, J. Chen, Y. Gao, and J. Zhang, "Coexistence of LTE-LAA and Wi-Fi on 5 GHz with corresponding deployment scenarios: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 7–32, 2016.
[2] S. Muhammad, H. H. Refai, and M. O. Al Kalaa, "5G NR-U: Homogeneous coexistence analysis," in *Proc. of IEEE GLOBECOM'20*, pp. 1–6, 2020.
[3] Y. Gao and S. Roy, "Achieving proportional fairness for LTE-LAA and Wi-Fi coexistence in unlicensed spectrum," *IEEE Transactions on Wireless Communications*, vol. 19, no. 5, pp. 3390–3404, 2020.
[4] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," in *International conference on engineering applications of neural networks*, pp. 213–226, Springer, 2016.
[5] 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*, vol. 12, no. 1, pp. 168–179, 2018.
[6] W. Zhang, M. Feng, M. Krunz, and A. H. Yazdani Abyaneh, "Signal detection and classification in shared spectrum: A deep learning approach," in *Proc. of the IEEE INFOCOM'21*, pp. 1–10, 2021.
[7] N. Bitar, S. Muhammad, and H. H. Refai, "Wireless technology identification using deep convolutional neural networks," in *Proc. of IEEE PIMRC'17*, pp. 1–6, 2017.
[8] T. Jian, B. C. Rendon, E. Ojuba, N. Soltani, Z. Wang, K. Sankhe, A. Gritsenko, J. Dy, K. Chowdhury, and S. Ioannidis, "Deep learning for RF fingerprinting: A massive experimental study," *IEEE Internet of Things Magazine*, vol. 3, no. 1, pp. 50–57, 2020.
[9] K. Sankhe, M. Belgiovine, F. Zhou, L. Angioloni, F. Restuccia, S. D'Oro, T. Melodia, S. Ioannidis, and K. Chowdhury, "No radio left behind: Radio fingerprinting through deep learning of physical-layer hardware impairments," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 1, pp. 165–178, 2019.
[10] T. Roy, T. O'Shea, and N. West, "Generative adversarial radio spectrum networks," in *Proc. of ACM WiseML'19*, pp. 12–15, 2019.
[11] M. Patel, X. Wang, and S. Mao, "Data augmentation with conditional GAN for automatic modulation classification," in *Proc. of ACM WiseML'20*, pp. 31–36, 2020.
[12] S. Soltani, Y. E. Sagduyu, R. Hasan, K. Davaslioglu, H. Deng, and T. Erpek, "Real-time and embedded deep learning on FPGA for RF signal classification," in *Proc. of IEEE MILCOM'19*, pp. 1–6, 2019.
[13] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4. Springer, 2006.
[14] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges.* Springer Nature, 2019.
[15] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.
[16] Y. Shi, K. Davaslioglu, Y. E. Sagduyu, W. C. Headley, M. Fowler, and G. Green, "Deep learning for RF signal classification in unknown and dynamic spectrum environments," in *Proc. of IEEE DySPAN'19*, pp. 1–10, IEEE, 2019.
[17] Y. Wang, J. Wang, W. Zhang, J. Yang, and G. Gui, "Deep learning-based cooperative automatic modulation classification method for MIMO systems," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4575–4579, 2020.
[18] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," in *International Conference on Machine Learning*, pp. 1437–1446, PMLR, 2018.
[19] "WLAN toolbox version 3.2," R2021a. The MathWorks, Natick, MA, USA.
[20] "LTE toolbox version 3.5," R2021a. The MathWorks, Natick, MA, USA.
[21] "5G toolbox version 2.2," R2021a. The MathWorks, Natick, MA, USA.
[22] R. P. e. a. Jianhan, L., "TGax channel model. IEEE 802.11-14/0882r4," R2021a. September 2014.