RF Fingerprinting Unmanned Aerial Vehicles with Non-standard Transmitter Waveforms

Nasim Soltani, Guillem Reus-Muns, Batool Salehi, Jennifer Dy, Stratis Ioannidis, and Kaushik Chowdhury

***Abstract*—The universal availability of unmanned aerial vehicles (UAVs) has resulted in many applications where the same make/model can be deployed by multiple parties. Thus, identifying a specific UAV in a given swarm, in a manner that cannot be spoofed by software methods, becomes important. We propose RF fingerprinting for this purpose, where a neural network learns subtle imperfections present in the transmitted waveform. For UAVs, the constant hovering motion raises a key challenge, which remains a fundamental problem in previous works on RF fingerprinting: Since the wireless channel changes constantly, the network trained with a previously collected dataset performs poorly on the test data. The main contribution of this paper is to address this problem by: (i) proposing a multi-classifier scheme with a two-step score-based aggregation method, (ii) using RF data augmentation to increase neural network robustness to hovering-induced variations, and (iii) extending the multi-classifier scheme for detecting a new UAV, not seen earlier during training. Importantly, our approach permits RF fingerprinting on manufacturer-proprietary waveforms that cannot be decoded or altered by the end-user. Results reveal a near two-fold accuracy in UAV classification through our multi-classifier method over the single-classifier case, with an overall accuracy of 95% when tested with data under unseen channel. Our multi-classifier scheme also improves new UAV detection accuracy to a near perfect 99%, up from 68% for a single neural network approach.**

***Index Terms*—UAV RF fingerprinting, deep neural networks, multi-classifier**

I. Introduction

Agriculture, construction, insurance, and telecommunications are being transformed by the explosive growth of small unmanned aerial vehicles (UAVs). Several industry estimations predict that this segment will grow to $17 Billion by 2024 [1]. Companies servicing these market segments, casual users, and hobbyists have benefited from widespread UAV availability at affordable price points. However, this has also raised the possibility of new attack vectors. For example, when UAVs serve as mobile wireless access points (APs) for ground nodes, malicious users may masquerade legitimate APs by falsely advertising recorded SSIDs [2]. Classical differentiation methods like angle of arrival and time of flight of the signal may no longer be possible, as even the legitimate APs move around the region of interest. To address this important issue of trust, we propose deep learning based RF fingerprinting for UAVs that can complement other secure detection methods [3],

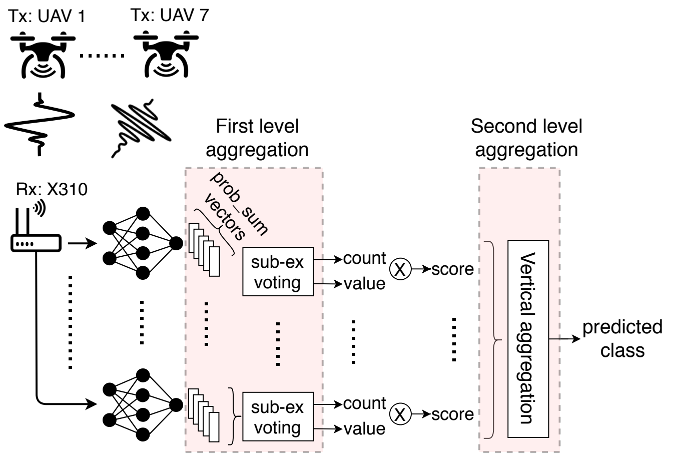


Figure 1: UAV classification using multi-classifier scheme with two levels of aggregation.

[4]. Specifically, our approach is composed of increasing robustness in the training pipeline and combining test outputs of multiple trained deep neural networks, each being trained on a different portion of a large training set.

• **Problem.** RF fingerprinting relies on identifying discriminating transmitter-generated features at the receiver. These features include artifacts such as nonlinearities in the power amplifier gain, I/Q phase imbalance, clock and frequency offsets, etc., mainly arising from slight variations in the operating points of the electronic components. While RF fingerprinting using deep learning has shown to be very successful for static devices [5]–[10], to the best of our knowledge, there are no works on applying this technique for classifying identical hovering UAVs. We note that this is different from the well-investigated problem of UAV type detection [11], where the objective is to distinguish between different make/models. Since the wireless transmitters, typically WiFi interface cards, are from different providers, fingerprinting these cards (and hence identifying the UAV) reduces to a simpler problem than classifying UAVs of the *same* make/model.

Furthermore, the constant UAV hovering introduces complex channel variations between the transmitter UAV and the ground-based receiver, which needs to be carefully studied. Our previous experimental studies show that the standard deviation in position around the target location can be as high as 0.85 m for DJI M100 UAV using on-board GPS modules [12]. While we focus *only* on RF fingerprinting in this paper, our technique can be combined with multimodal sensors detecting acoustic or infrared patterns for enhanced classification accuracy. A recent work jointly uses WiFi and

The authors are with the Institute for the Wireless Internet of Things, Electrical and Computer Engineering Department, Northeastern University, Boston, MA, USA.

Corresponding author: Kaushik Chowdhury, email: krc@ece.neu.edu.

Bluetooth emissions [13], although this increases complexity for both the sensing hardware and the training/classification process.

• **Approach.** As shown in Figure 1, we assume a network of a number of UAVs that may coexist in the same airspace. We form a dataset by flying 7 identical DJI M100 UAVs inside an RF anechoic chamber, at different distances from a receiver that collects I/Q samples from DJI's non-standard, proprietary waveform. Thus, the setup captures real-world signal variations, as would be seen in practical deployments. We address a more generalized fingerprinting problem than previous works for classifying UAVs and stationary devices, where the waveform was known, modifiable and decodable [5], [14], [15]. Moreover, our prior solutions of *introducing* an artificial fingerprint by (i) intentionally injecting distortions of type I/Q imbalance in a software-defined-radio-enabled transmitter [5] or (ii) applying a specially designed FIR filter to enhance the transmitter fingerprint [15] are not feasible. We previously used *partial equalization* for WiFi signals, which removes the channel-induced distortions before training, so as to capture the *pure* fingerprint [14]. Equalization is proven to be helpful in the case of hovering UAV emitters, where the dataset is heavily impacted by the channel [16]. However, as we consider a proprietary waveform, equalization is not applicable, either. In summary, we seek to design a method that relies only on raw I/Q samples, which further motivates us to explore approaches such as *data augmentation* to train more robust deep learning models.

Finally, in real world scenarios, detecting a *new* (out-of-library) device (i.e., the device whose signal does not exist in the training set) is of paramount importance. Since the output of the neural network is a probability vector, there is always going to be a non-zero probability of identifying the new UAV as one of the previously trained, legitimate UAVs. Thus, the new UAV will be classified *wrongly* as one of the known classes. To address this issue, we design an approach that uses statistics observed in the known device set to guide a decision process in the new device test set, assuming that there are inherent similarities in the information distribution. We start from the approach in [17] that showed promising outcomes for standard WiFi frames, and then extend that approach within the multi-classifier approach in this new domain of UAV identification.

• **Contributions.** Our contributions are as follows:

- We empirically show the effect of aerial hovering on the accuracy of deep learning-based UAV RF fingerprinting. We quantify the degradation in the test performance of a neural network classically trained on past sequences of data, due to these slight hovering motions (Section III).
- We propose a novel architecture composed of multiple classifiers, each being trained on different portions of past sequences and learned a different channel-distorted fingerprint. In the test phase, the predictions from all the neural networks are aggregated to make a final decision for each transmission (Section IV-A).
- To combine the probability vectors at the output of the neural networks, we propose a two-level score-based aggregation method. In the first level, output vectors of each individual neural network are combined to make predictions per neural network. In the second level, predictions of all the neural networks are combined to make a joint prediction by all the neural networks for each transmission (Section IV-B).
- We propose an algorithm for determining the number of neural networks that should compose the multi-classifier scheme (Section IV-C).
- We propose a data augmentation scheme for training robust individual neural networks in our multi-classifier scheme, which further improves the accuracy (Section IV-D).
- For detecting a new UAV, we propose a new device detection method with our multi-classifier scheme. The result is a robust framework for detecting new UAVs that the neural network is not previously trained to classify (Section IV-E).

In the rest of the paper, Section II discusses more related works, Section V presents performance evaluation results, and Section VI concludes the paper.

II. Related Work

A wide variety of techniques for UAV detection and classification, such as passive/active RF, acoustic, or image-based sensing have been investigated previously [12], [18]–[20]. However, the primary focus of this work is UAV classification using RF fingerprinting, and we survey below RF-based approaches that are directly relevant to the scope of this paper.

UAV manufacturers may use different RF technologies for signaling, with variations in transmitter parameters such as frame interval or frequency of operation, which can be exploited for the comparatively simpler problem of identifying the make/model [21], [22]. For example, authors in [11] propose a UAV detection method using K-Nearest Neighbours, which has less computational overhead compared to neural networks, for UAVs with different make/models.

There are methods to extract Hash Fingerprinting features from the preamble with an SVDD-based classifier to identify different UAV vendors [23]. Moreover, authors in [24] use two separate phases of feature extraction and machine-learning-based UAV identification. Both these works rely on the assumption that the UAVs are transmitting a known protocol (WiFi). A priori knowledge of specific features that must be learnt is not always possible, and therein lies the benefit of our deep learning approach that fuses these two phases together and automates the entire process of classification. Authors in [25] design a deep-learning-based UAV-fingerprinting scheme by using modified generative adversarial networks (GANs). However, their objective is to classify different signal protocols, which is not the same as our scenario, where the same make/model UAVs transmit the same protocol.

Multi-classifier approaches have been applied in different wireless problems. Authors in [26] propose a multi-classifier scheme for robust WiFi-based positioning systems. MFMCF [27] propose a fingerprint-based localization system. They construct three different fingerprints of signal strength difference (SSD), hyperbolic location fingerprint (HLF), and

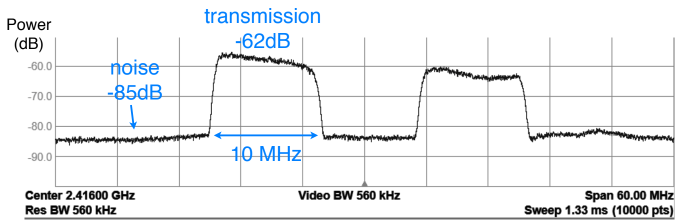


Figure 2: Downlink transmissions of two DJI M100 UAVs.

received signal strength (RSS). MFMCF fuses the fingerprints in a single vector and trains three classifiers of K-nearest neighbours (KNN), support vector machine (SVM), and random forest (RF) for a robust localization system.

To the best of our knowledge, none of the previous works address RF fingerprinting in identical, in-flight UAVs transmitting a proprietary or unknown protocol. We start our investigation by collecting a dataset from COTS DJI M100 UAVs under realistic UAV hovering motion that greatly impacts the wireless channel. Since we cannot equalize the signal, we must rely only on fingerprinting using raw I/Q samples.

III. Dataset, Training/Testing Pipeline

In this section, we provide brief insights on the proprietary waveform used by the DJI M100 UAVs [28], describe the experimental setup and dataset, detail the training pipeline, and analyze different ways to report the test accuracy results for UAV classification as well as new UAV detection.

A. COTS UAV Signal Analysis

1) Uplink: DJI employs *frequency-hopping spread spectrum* (FHSS) for the remote controller (RC) to UAV link. FHSS switches channels following a pseudo-random sequence known at both ends of the link. DJI UAVs typically work on the 5 GHz band (5.725-5.825 GHz) or the ISM band (2.401-2.481 GHz). The FHSS characteristics vary among different models.

2) Downlink: UAVs communicate periodically with the RC in order to report telemetry data or battery level. They may relay video stream data that requires high throughput and low latency links. Based on application requirements, video transmissions may vary in bandwidth and transmission periodicity. The M100 UAVs use the Lightbridge protocol, developed by DJI specifically for long range (up to 5km), robust aerial communication in the 2.4 GHz band. Lightbridge has 8 selectable channels in the ISM band (2.401-2.481 GHz), with a separation of 2 MHz between carriers. Channel selection can be done manually, or it can be left to the radio to determine the channel with the least interference [29].

In Figure 2, we visualize spectrum usage from two concurrent DJI M100 transmissions, as captured by a Tektronix RSA507A spectrum analyzer. We see that each UAV selects a different transmission band to avoid interference. Moreover, the UAV accesses the medium at a fixed rate of $\sim$50 Hz.

B. UAV Dataset

In an RF anechoic chamber, we collect signals from 7 identical DJI M100 UAVs as transmitters. An Ettus USRP X310 [30] equipped with an UBX 160 USRP daughterboard is used to capture signals at the receiver side. We fly the UAVs one at a time at different distances of 6, 9, 12, and 15 feet from the receiver, while they transmit. The receiver collects I/Q samples only in the downlink 10 MHz channel where the UAV is transmitting, as described in Section III-A2.

At each distance, we collect I/Q samples for $\sim$2 seconds, pause for $\sim$10 seconds, and then repeat this process 3 more times. The $\sim$10-second intervals of time partition the overall received signals into 4 non-overlapping *burst*s, each containing $\sim$140 interleaved short periods of data and noise. A high-level overview of the sequence collected at the receiver side for each UAV at a given distance is shown in Figure 3. To complete the dataset, the procedure shown in Figure 3 is collected from all the 7 UAVs, flying at 4 different distances from the receiver. The average calculated SNRs are approximately 33, 31, 28, and 26 dB for distances of 6, 9, 12, and 15, respectively.

To prepare the sequences for our deep learning framework, we extract the portions containing data and separate them from interleaved noise periods to form $\sim$140 sequences per burst. From here on, we refer to these non-overlapping data sequences as *example*s. With 7 UAVs, each having 4 distances, each distance having 4 bursts, and each burst having $\sim$140 examples, we have more than 13k examples with average length of $\sim$92k I/Q samples in the dataset. This complete UAV dataset is released for future investigations[1].

C. Training and Testing the Neural Network

We form our training, validation, and test sets based on the strategies explained in the next subsection. When the sets are formed, we calculate mean μ and standard deviation σ of the training set, in a preprocessing step, and use them later for normalizing the training and test batches. We train the neural network in a per-*slice* basis. Slices are overlapping portions of examples and contain consecutive raw I/Q samples. We choose the slice size as $l = 200$ and feed the training set to the neural network as batches of slices. The training batches are prepared by `Data Generator`, a special class in `Keras` library [31]. Every time a batch is to be formed, a random set of training examples are loaded to the memory and from them, a number of random slices are selected. This random selection of examples and slices, is the equivalent of shuffling the training set, without loading all of them to the memory. It helps the neural network see the training data in a different order in every epoch, which contributes to training more robust models. For each batch, a number of slices equal to "batch size" are selected from different examples to form a batch. Therefore, each batch is a *tensor* with dimensions $(\text{batch size}, l, 2)$, with I and Q coming in two separate channels in the last dimension. Each batch is normalized with respect to the previously recorded mean μ and standard deviation σ of the training set, using (1) before being fed to the neural network.

$$X_{normalized} = \frac{X - \mu}{\sigma} \tag{1}$$

[1] `http://genesys-lab.org/mldatasets`

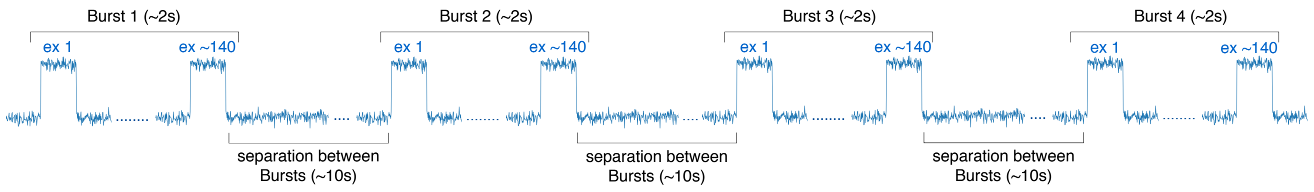


Figure 3: Overview of I/Q samples collected from each UAV for a given distance.

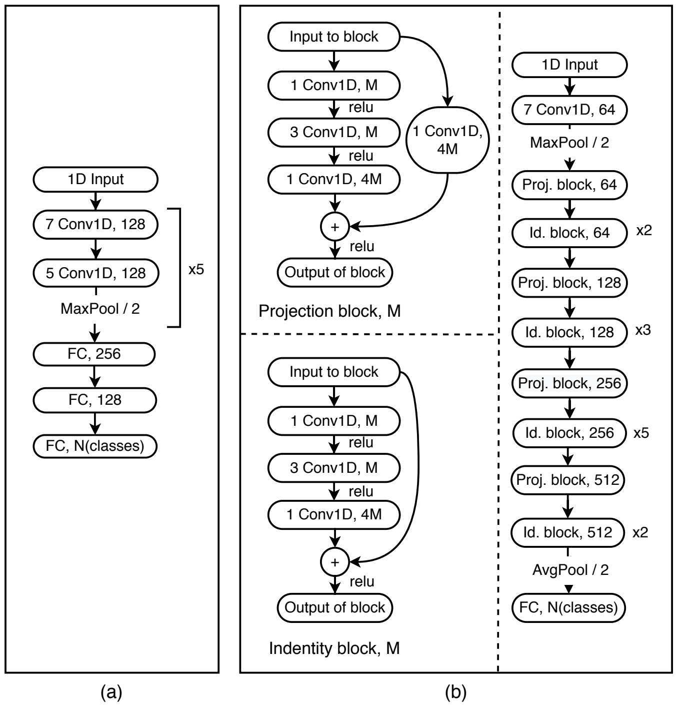


Figure 4: Neural Network architectures for (a) AlexNet1D with ∼1.1M parameters and (b) ResNet1D with ∼16M parameters.

For the neural networks, we use 1D modified versions of AlexNet [32] (abbreviated as *AlexNet1D*) with ∼1.1 million parameters, and ResNet50 [33] (abbreviated as *ResNet1D*) with ∼16 million parameters. These architectures were previously successful in RF fingerprinting and modulation classification [14], [34]. Our version of *AlexNet1D* is a forward convolutional neural network (CNN) with 5 blocks stacked together. Each block consists of (i) a 1D convolutional layer with 128 filters of size of 7, followed by (ii) a 1D convolutional layer with 128 filters of size of 5, followed by (iii) a MaxPooling layer. The 5 blocks are followed by 2 fully connected (FC) layers of sizes 256 and 128, respectively. The final layer is a FC-softmax layer with the same size as the number of classes. Our version of *ResNet1D* is a combination of *Projection block*s and *Identity block*s stacked together. More details about both AlexNet1D and ResNet1D are visually depicted in Figure 4.

As this is a multi-class classification problem where among multiple UAVs, a single UAV is to be selected as the transmitter, we use *categorical cross entropy* loss. We train the neural network using Adam optimizer [35] with learning rate = 0.0001. At the end of each epoch, we test the network on the validation set. We stop training when the validation accuracy does not improve for 3 consecutive epochs. When the training is done, we test the trained network with the test set. Similar to the training process, we test the neural network on a per-slice basis. In the test phase, each example with length LL, is sliced with a stride=1. Consequently, each example is fed to the neural network as a tensor with dimensions $(LL-l+1, l, 2)$. Test batches are also normalized with respect to mean μ and standard deviation σ of the training set, as in (1).

When we feed in the test batches, the outputs of the neural network are batches of probability vectors, each obtained from an input slice. We obtain a prediction for a given slice by performing an $argmax$ on its corresponding probability vector. To achieve a prediction for a given example, we combine probability vectors of the slices in that example. We do this through 1) Probability Sum and 2) Majority Vote [36] that are briefly summarized below:

1. Probability Sum: We feed all the slices from a given example to the neural network, and at the output, we sum all the probability vectors to get a *Probability Sum* vector. The predicted class for the example is obtained from the index of the element having maximum value in the Probability Sum vector.

2. Majority Vote: We classify all the slices in an example and we choose the class that the majority of slices vote for as the predicted class for that example.

We note there are two ways to report accuracy [14]:

A. Slice accuracy: we divide the number of correctly predicted slices by the total number of slices in the test set.

B. Example accuracy: we divide the number of correctly predicted examples by the total number of examples in the test set.

In the rest of this paper, we focus on example accuracy, since it captures the neural network prediction for a complete transmission (example).

D. Preliminary Experiments on UAV Classification

To empirically show the effect of imperfect hovering on the received UAV fingerprint, we introduce two scenarios of training and testing the neural network. We use AlexNet1D architecture, unless specified otherwise.

1) **Train on all bursts, test on all bursts**: For each UAV, each distance and each burst, we shuffle the examples and partition the sequences by 60%, 20% and 20% for training, validation and test, respectively. We observe a test accuracy of 97%. In this scenario, other examples before/after the unseen test example may be present in the training set. As a result, the neural network performs well, although this procedure cannot be applied for real-time UAV classification.

2) **Train on bursts 1, 2, and 3 Test on burst 4**: For each UAV, each distance, and bursts 1, 2, and 3, we shuffle examples and choose 90% and 10% for training and validation, respectively. We test the trained network on the unseen burst 4. In this scenario of training on first examples and testing on last ones, we get 49% accuracy with AlexNet1D. A deeper architecture like ResNet1D also yields marginal improvement with 50% accuracy. This drop in accuracy compared to the former scenario shows the effect of slight UAV movements while hovering. When the transmitting UAV hovers, the received signals are impacted over time due to the continuous channel variations. Since the channel effect is non-negligible, both AlexNet1D and ResNet1D are unable to classify UAVs from unseen (future) bursts with high accuracy.

E. Detecting a New UAV

If a trained network is tested with an example from a new UAV, it classifies the example as one of the known classes, inevitably. To discriminate this prediction from a "correct" old device prediction, we use the statistics obtained from a test set of known old devices –as also used for standard WiFi in [17]– to label each unknown test example as "old" or "new".

1) Overview of the Approach: Our algorithm in [17] was originally used for new device detection in datasets where the training and test sets are non-overlapping, but share a common distribution. To use the algorithm for new UAV detection, after the network is trained on the legitimate (old) UAVs, an intermediate test set containing signals from only these trained (in-library) devices with known true labels are fed to it. Assume the test set has M examples each indexed with m={0,1,..., $M-1$}. The example m is classified using *Majority Vote* method as the predicted old class $C^{(m)}$, and two thresholds are recorded:
1. Probability threshold $\theta_P(u)$**:** For each specific UAV u, we gather the probability vectors of all the slices classified as $C^{(m)}$, across all examples, in a set P_u. Then a statistic mapping function, $\chi(A)$, is applied to set P_u, to obtain the probability threshold for device u, represented as $\theta_P(u)$. This process is repeated for all the old UAVs to achieve separate probability thresholds for each UAV.
2. Ratio threshold $\theta_R(u)$**:** For each specific UAV u, we gather the ratio of slices classified as $C^{(m)}$ in all the examples, in a set R_u. Then, the same statistic mapping $\chi(A)$ is applied to generate the ratio threshold for UAV u, represented as $\theta_R(u)$. Again, this process is repeated for all the old UAVs.

At this point, two thresholds are calculated for each old UAV. We now form a final unseen test set of old and new UAVs, with B examples in total, each being indexed with b={0,1,...,$B-1$}. If slices from a given example b, whether new or old, are fed to the neural network, the example will be classified inevitably as one of the old UAVs. The predicted class for example b, noted as $\hat{y}^{(b)}$ is called the "best guess". In order to decide whether example b is from an old or a new device, we calculate two metrics:
Metric 1. Transmission prediction probability $\tilde{p}^{(b)}$**:** We collect the probability vectors from all the slices in example b classified as the best guess in a set $\tilde{P}^{(b)}$. The set $\tilde{P}^{(b)}$ is mapped to the metric transmission prediction probability $\tilde{p}^{(b)}$, using the statistical mapping function χ.
Metric 2. Estimated correct slice ratio $\tilde{r}^{(b)}$**:** We divide the number of best guess slices by the total number of slices in example b, for the metric estimated correct slice ratio $\tilde{r}^{(b)}$.

Finally, example b is labeled as new or old using (2).

$$y_{ex}^{(b)} = \begin{cases} new & \left(\tilde{r}^{(b)} < \theta_R(\hat{y}^{(b)})\right) \; and \; \left(\tilde{p}^{(b)} < \theta_P(\hat{y}^{(b)})\right) \\ old & otherwise \end{cases}$$
$$b = 0, 1, 2, \ldots, B-1 \tag{2}$$

2) Preliminary Experiments on New UAV Detection: For detecting new UAVs, we follow the steps explained in Section III-E1. For the statistics mapping function χ, we use (3), which showed the best performance among various options explored in [17].

$$\chi(A) = avg(A) - std(A) \tag{3}$$

with avg and std being the average and standard deviation functions, respectively.

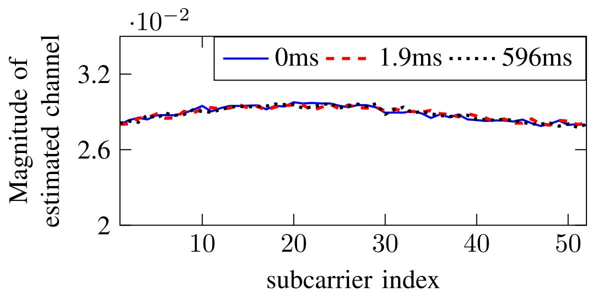
Among the 7 UAVs, without loss of generality, we choose UAV4 as the new device and other 6 UAVs as old devices. To simulate the real-world situation where the new UAV signal appears in the future burst 4, we form our training/validation sets using old UAV examples in bursts 1, 2, and 3. We shuffle the examples and use 90% of them as training set and 10% as validation set. We train a single AlexNet1D on the training set, and use the validation set as the old device intermediate test set to record thresholds. We form the new device test set using examples from UAV4 in burst 4, so that similar to the real case scenario, the new UAV is also from an unseen burst. In this case, the new UAV detection accuracy is defined as the number of examples with $y_{ex}^{(b)}$ labeled as "new", divided by the total number examples in the new device test set. Our results show a new UAV detection accuracy of 68% with AlexNet1D, meaning a single neural network is not able to detect new UAVs from an unseen burst, with high accuracy. However, as we will show later, this accuracy can be boosted up to 99% using this same method, when combined with our multi-classifier approach.

IV. Proposed Robust Multi-classifier Approach

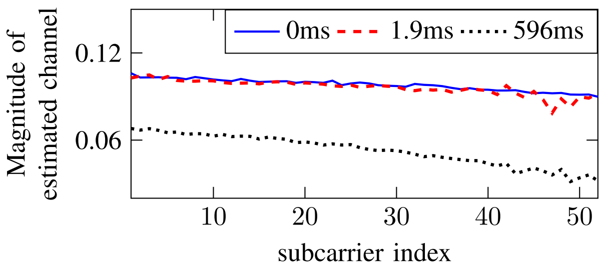
In this section, we introduce our multi-classifier approach with a score-based two-level aggregation method that returns a fused decision for each test example. Additionally, we describe a data augmentation method to increase robustness during training. For a robust detection of new hovering UAVs, we combine the new device detection algorithm with the multi-classifier scheme.

A. Multi-classifier Scheme

As we experimentally showed in Section III-D, UAV hovering changes the wireless channel significantly over time. To gain deeper insights on how channel conditions vary in the case of hovering UAVs, as compared to a static radio case,



(a) Static Tx



(b) Hovering Tx

Figure 5: Change of channel over time in two different cases.

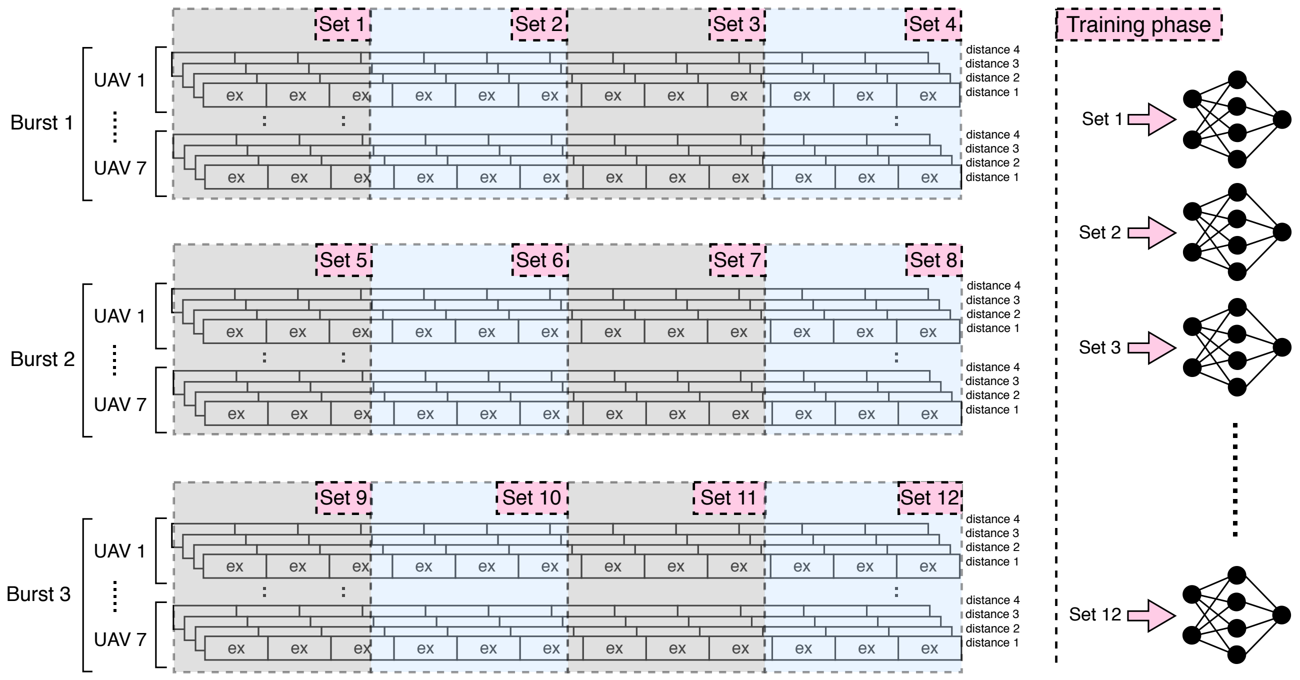


Figure 6: Forming training sets for identical but independent neural networks in the multi-classifier scheme.

we conduct additional experiments. Since the UAVs used in this paper transmit Lightbridge proprietary waveforms, we use software defined radios (SDRs) transmitting standard WiFi to study the estimated channel in two different cases. We generate a waveform with standard protocol of IEEE WiFi 802.11a with 5 MHz bandwidth, using MATLAB `WLAN toolbox`. We transmit the waveform over the air using an Ettus USRP X310 radio as the transmitter hardware, in two different situations described below:

1) **Static Tx:** We place the transmitter statically on the ground.
2) **Hovering Tx:** We mount the transmitter on an M100 UAV, fly the UAV and transmit signals as the UAV hovers in its place.

For both cases, we use an Ettus USRP X310 radio placed on the ground, as the downlink receiver, to collect I/Q samples and estimate the channel. With FFT size 64, if we eliminate the guardband subcarriers, we have 52 channel coefficients for each WiFi packet. We calculate the magnitude of channel coefficients for each packet at three different instants of 0, 1.9 ms, and 596 ms in Figure 5. It is observed that in case of a static transmitter, the channel does not change much during 596 ms. However, in case of a hovering UAV, channel changes drastically from the starting instant 0 to the 596 ms mark.

Thus, when a hovering UAV is transmitting, the rapid changes in wireless channel distort the transmitted fingerprint with rapidly changing transforms. This causes different snapshots of the received signal to have different distributions of I/Q samples. Since data distribution varies over time, instead of a single neural network trained on all data parts, we independently train multiple neural networks each on non-overlapping portions of the dataset. As an illustrative example, Figure 6 shows a specific case of training 12 independent neural networks with different portions of bursts 1, 2, and 3. In this case, each burst is equally partitioned into 4 *set*s to form 12 (=3×4) non-overlapping training sets. We test the trained networks on the examples from the unseen burst 4 by feeding slices from each example to all the trained neural networks and combine their predictions using an aggregation method described next.

B. Aggregation Method

1. Aggregating predictions at the output of each neural network: As explained in Section III-C, in the test phase, two previous methods of Probability Sum and Majority Vote yield ∼50% accuracy for the hovering UAV dataset (more in Section V).

In the proposed method for the first level of aggregation, we further divide each (relatively long) example of length ∼92k samples, to K=10 equal length sub-examples. We slice each sub-example with index k (k={0,1,..., K-1}) as explained in Section III-C, and feed it to the trained neural network in a per-slice basis. If each sub-example with length L is sliced with slice size l, the whole sub-example yields $(L-l+1)$ slices. Consequently, at the output of the neural network, we have $L-l+1$ probability vectors $p_{slice}^{(k,i)}$ for k^{th} sub-example, with i being the slice index (i={0,1,..., L-1}). We make predictions for each sub-example using Probability Sum method. Here, the sum of probability vectors $p_{slice}^{(k,i)}$ for all slices in the sub-example k yields the Probability Sum vector $V_{sub-ex}^{(k)}$, as in (4).

$$V_{sub-ex}^{(k)} = \sum_{i=0}^{L-l} p_{slice}^{(k,i)} \quad , \quad k = 0,1,2,\dots,K\text{-}1 \tag{4}$$

The predicted class $C_{sub-ex}^{(k)}$ for the sub-example with index k, using Probability Sum method is shown in (5).

$$C_{sub-ex}^{(k)} = argmax(V_{sub-ex}^{(k)}) \quad , \quad k = 0,1,2,\dots,K\text{-}1 \tag{5}$$

At this point, each sub-example with index k, yields $L-l+1$ slices, among which $M_{sub-ex}^{(k)} \le (L-l+1)$ are classified as class $C_{sub-ex}^{(k)}$, as in (6).

$$\begin{aligned} M_{sub-ex}^{(k)} &= \sum_{i=0}^{L-l} \gamma(slice^{(k,i)}) \\ \gamma(slice^{(k,i)}) &= \begin{cases} 1 & \text{if } argmax(p_{slice}^{(k,i)}) = C_{sub-ex}^{(k)} \\ 0 & otherwise \end{cases} \end{aligned} \tag{6}$$

For each sub-example with index k, we define $S_{sub-ex}^{(k)}$ as the maximum value of vector $V_{sub-ex}^{(k)}$ divided by total number of slices in the sub-example, which is an indicator of the average prediction probability for the sub-example (7).

$$S_{sub-ex}^{(k)} = \frac{max(V_{sub-ex}^{(k)})}{(L-l+1)} \quad , \quad k = 0,1,2,\dots,K\text{-}1 \tag{7}$$

For each sub-example with index k, we calculate a score, as shown in (8), based on which we do the first level of aggregation.

$$score_{sub-ex}^{(k)} = S_{sub-ex}^{(k)} \times M_{sub-ex}^{(k)} \ , \ k = 0,\dots,K\text{-}1 \tag{8}$$

In the last step, we take a vote among sub-examples: We choose the maximum among the list of $score_{sub-ex}^{(k)}$ as the score of the whole example, $score_{ex}$, as shown in (9). We select the corresponding sub-example as the winning sub-example, and its predicted class C_{sub-ex}, as the predicted class C_{ex} for the whole example, as in (10).

$$score_{ex} = max(\bigcup_{k=0}^{K-1} score_{sub-ex}^{(k)}) \tag{9}$$

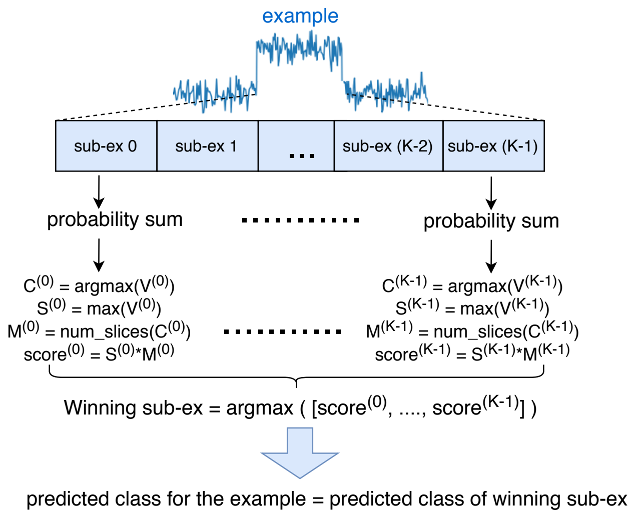


Figure 7: Proposed method of sub-example voting for the first level of aggregation.

$$C_{ex} = C_{sub-ex}^{\left(argmax(\bigcup_{k=0}^{K-1} score_{sub-ex}^{(k)})\right)} \tag{10}$$

In summary, we make a prediction for an example at the output of one neural network, by comparing scores of its sub-examples and selecting a winning sub-example. The score for each sub-example relies on the average prediction probability and the number of slices classified same as the sub-example. Selecting winning sub-examples helps us better identify the wrongly predicted sub-examples and suppress their votes. As the sub-examples vote with their scores, to determine a winning sub-example, we refer to this method as *sub-ex voting*. An overview of this scheme is depicted in Figure 7.

2. Aggregating the predictions from multiple NNs: Previously, we explained how we make a decision for an example at the output of a neural network, as the first level of aggregation. Here we explain how to combine results from multiple neural networks in the second level of aggregation.

In the second level of aggregation, we need the $score_{ex}$ and C_{ex} recorded from all the neural networks. Assume we have J trained neural networks, classifying the same example in parallel. After the first level of aggregation, at the output of the neural networks, we have a list of J scores, $score_{ex}$ and J predicted classes C_{ex} (11):

$$\begin{aligned} score_{ex}\_list &= \bigcup_{j=0}^{J-1} score_{ex}^{(j)} \\ C_{ex}\_list &= \bigcup_{j=0}^{J-1} C_{ex}^{(j)} \end{aligned} \tag{11}$$

We follow the same logic of using *score*s to take a vote between neural networks, as we did for sub-examples, and select the classifier with the highest score, as the winning classifier. The class predicted by that classifier is selected as the predicted class C for the input example, as shown in (12).

$$C = C_{ex}\_list(argmax(score_{ex}\_list)) \tag{12}$$

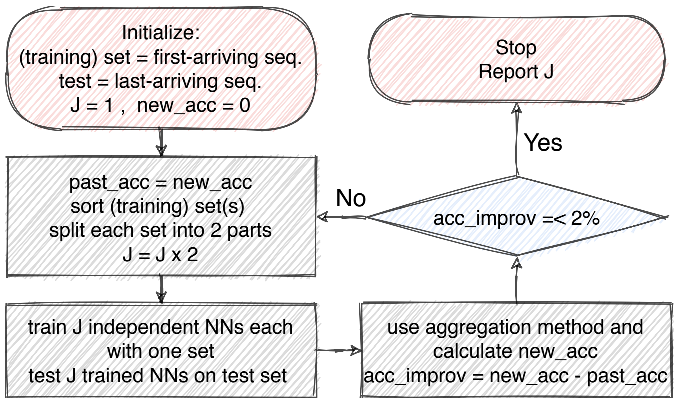

Initialize:
(training) set = first-arriving seq.
test = last-arriving seq.
J = 1 , new_acc = 0
past_acc = new_acc
sort (training) set(s)
split each set into 2 parts
J = J x 2
train J independent NNs each with one set
test J trained NNs on test set
use aggregation method and calculate new_acc
acc_improv = new_acc - past_acc
acc_improv =< 2%
No
Yes
Stop
Report J

Figure 8: The steps of determining the size of multi-NN scheme.

For calculating the accuracy of the multi-classifier scheme, we follow the equations (5)–(12) for all the examples in the test set, and ultimately record combined predicted classes C for each example. Classification accuracy of the multi-classifier is calculated by dividing the number of correctly predicted examples by the total number of test examples.

C. Choosing the Size of the Multi-classifier Scheme

We propose a step-by-step dataset partitioning process to find the number of neural networks that compose the multi-classifier scheme. Without losing generality, for our experiments we use AlexNet1D architecture as each individual classifier.

We start by using first-arriving I/Q sequences as training set (called as *set*) and last-arriving as an intermediate test set. We initialize the number of neural networks in the multi-classifier scheme (J) equal to 1. The goal is to determine J by monitoring the accuracy improvement after each training/testing process. To do this, we follow the steps below:

1) We sort the examples in the *set*(s) in temporal order, and split them into two parts. Now each part becomes a new set, and we update J by doubling it.
2) We train J independent neural networks on 90% of each set. As explained in Section III-C we choose 10% of each set for validation to decide when to stop training.
3) After the neural networks are fully trained, we test them on the unseen intermediate test set. We use the proposed two-level aggregation method (explained in Section IV-B) to calculate the multi-classifier accuracy.
4) Next, we check to see if the accuracy improvement is less than or equal to 2%. If yes, we stop the partitioning process and report J as the number of neural networks required in the multi-classifier scheme. If no, we go to step 1 and repeat the steps.

Figure 8 demonstrates an overview of the dataset splitting process for determining the number of neural networks in the multi-classifier scheme. Each loop that corresponds to going through steps 1–4 is considered a *round*. Since, in each round the former set is halved and the number of classifiers doubles, the algorithm reports J after $\log_2 J$ rounds.

The stopping criterion of 2% can be increased or decreased depending on the user's need. Increasing this criteria causes an earlier stopping point, which results in lower J and lowers final accuracy. Decreasing it, however, potentially increases final accuracy at the expense of larger number of classifiers, resulting in longer training/testing time in the multi-NN scheme.

D. Data Augmentation (DA)

Data augmentation (DA) is a means to expand the training set by modifying the original training samples in a principled manner [37], [38]. While collecting more training data that contains all the variations is expensive, DA can artificially create those variations in the original training set. Moreover, our DA method obviates the need of storing a large and varied training set on disk, since the variation injection happens on the flight in the training pipeline.

In our method of DA, first we normalize the training batch X according to mean μ, and standard deviation σ of the complete training set using (1). Next, the normalized batch passes through the DA block before being fed to the neural network (See Figure 9). The block contains a multi-tap complex FIR filter that is convolved with the data batch passing through it. The convolution happens in "same" mode, which means each slice is zero padded before passing through the filter, and the dimensions of the data batch do not change after filtering. The filter taps are independent and chosen from the same distribution. For the purpose of this paper, we choose an 11-tap FIR filter and we draw its coefficients from complex Gaussian distribution with mean μ_h=0 and standard deviation σ_h=0.125. However, FIR parameters such as number of taps, mean, and standard deviation can be varied for different datasets to optimize the method and achieve the best results.

In each epoch, when each data batch is loaded by `Keras Data Generator` [31], a new set of filter coefficients are drawn from the distribution. In this way, the augmentation block provides extensive variety to the training set over epochs. The resulting trained model is less likely to over-fit, and it is more robust compared to a model obtained from classical training without DA.

For testing a model that is trained with DA, in order to maintain the same scale for the test data as the training data, after the conventional normalization in (1), we need to filter the test data, too. For this purpose, we use FIR filters with the same parameters (i.e., number of taps, mean, and standard deviation) as the FIR parameters in the training phase. To remove the effect of random FIR choice in the test phase, instead of just one filter per batch, we use a number of different filters per test slice. Since the classification decision is made on an ensemble of resulting outputs, we call this method, *ensemble FIR*. As shown in Figure 10, per test slice in the test batch, we draw *ensembleFactor*=10 different sets of FIR taps from the same distribution. We pass the slice through these 10 different filters and stack them together, in the filtered batch. In this case, if the original batch contains $L-l+1$ slices, the filtered batch will have $10 \times (L-l+1)$ slices. Consequently, the neural network yields $10 \times (L-l+1)$ probability vectors. Same as before, we use Probability Sum method to aggregate

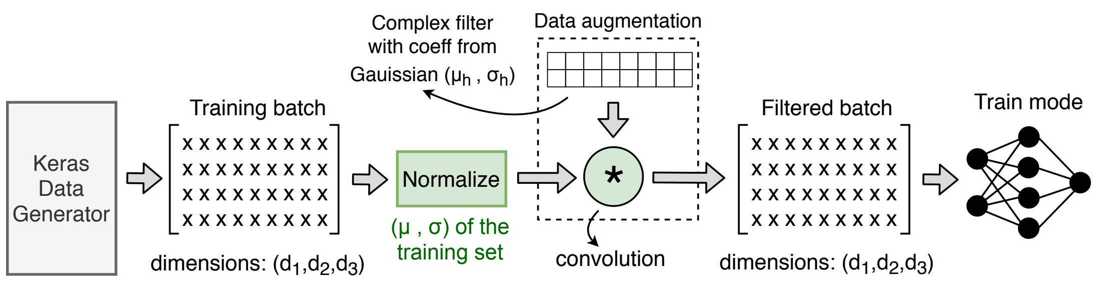


Figure 9: Data augmentation training phase using complex filter being convolved with each data batch.

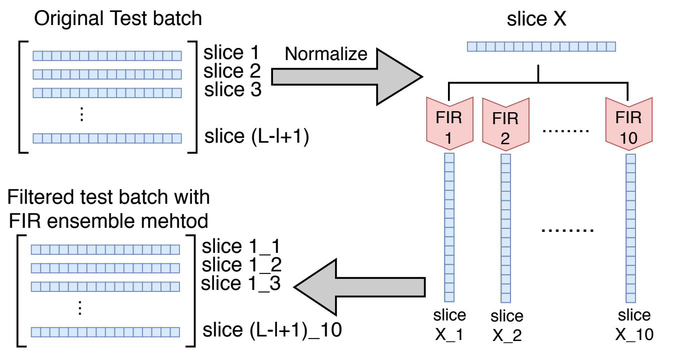


Figure 10: Ensemble FIR method with *ensembleFactor*=10 is used in the test phase of data augmentation scheme.

$p_{slice}^{(k,i)}$ and obtain a prediction for each sub-example. In this case, probability sum vector in (4) changes to (13).

$$V_{aug}^{(k)} = \sum_{i=0}^{(10\times(L-l+1)-1)} p_{slice}^{(k,i)} \ , \ k = 0, \dots, K\text{-}1 \qquad (13)$$

Recall that in our multi-classifier scheme, we use the sets in Figure 6 to train J neural networks. The new training sets are considerably smaller and less varied than the original training set that contains all of bursts 1, 2, and 3. In such situations, DA is a good candidate to prevent trained neural networks from over-fitting.

E. Improving New UAV Detection using Multi-Classifiers

Here, we improve the new device detection algorithm discussed in Section III-E1, by combining outputs from all neural networks in the multi-classifier scheme. We assume we have J neural networks trained and tested on old UAVs, with 2 thresholds being recorded per neural network, and per UAV, as explained in Section III-E1. We also keep the assumption of having K sub-examples in each example. In this case, for each example, we have J lists as in (14) each containing K labels of "new" or "old" represented as $y_{sub-ex}^{(j,k)}$.

$$\text{old-new-list}^{(j)} = \bigcup_{k=0}^{K-1} y_{sub-ex}^{(j,k)} \quad , \quad j = 0, 1, ..., J-1 \quad (14)$$

The goal of aggregation is to combine these $J \times K$ values in a way to achieve a single y_{ex} that determines whether the example is from an old UAV or a new one. Since the labels can only get 2 values in new UAV detection (either "new" or "old"), instead of the rather complex score-based aggregation method (discussed in Section IV-B), we use a simple Majority Vote at both levels of aggregation.

At the first level of aggregation, we identify an example as a new UAV example, if the majority of its sub-examples (more than $\frac{K}{2}$) vote for it to be "new". In (15) $y_{ex}^{(j)}$ is the "old" or "new" label for each example at the output of new device detection algorithm.

$$y_{ex}^{(j)} = \begin{cases} new & \text{old-new-list}^{(j)}.count(\text{“}new\text{”}) > floor(\frac{K}{2}) \\ old & otherwise \end{cases} \qquad (15)$$

At the second level of aggregation, we create a list of J members of all the labels $y_{ex}^{(j)}$ for one specific example, as in (16).

$$\text{old-new-list}^{(total)} = \bigcup_{j=0}^{J-1} y_{ex}^{(j)} \qquad (16)$$

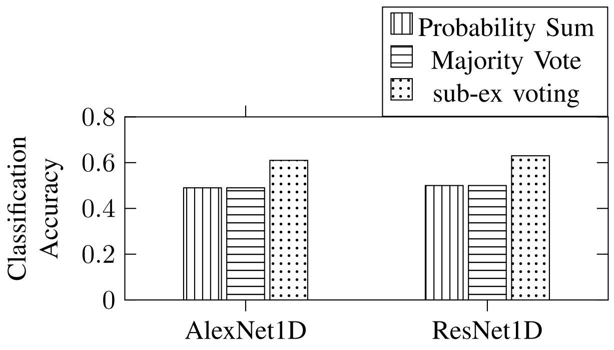
We take another Majority Vote this time across J labels to determine the final ensemble prediction y_{ex} for each example, as in (17).

$$y_{ex} = \begin{cases} new & \text{if old-new-list}^{(total)}.count(\text{“}new\text{”}) > floor(\frac{J}{2}) \\ old & otherwise \end{cases} \qquad (17)$$

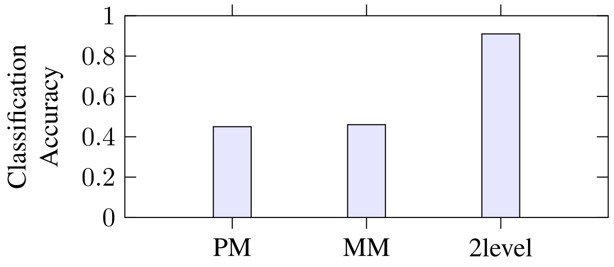
In the next section, we present numerical results for all the proposed methods and compare them against baseline results.

V. Performance Evaluation

We use our UAV dataset collected from 7 hovering UAVs (Section III-B) to show numerical results for the proposed schemes. First, we show the contribution of our first level of aggregation (sub-example voting) on individual neural networks of AlexNet1D and ResNet1D. Second, we run the algorithm explained in Section IV-C on the UAV dataset, and determine the number of neural networks in the multi-classifier scheme. Third, we show the results of the proposed multi-classifier scheme with two levels of aggregation. Fourth, to show the contribution of DA, we retrain the individual neural networks in the multi-classifier scheme using DA, and report their decisions using the two-level aggregation. Last, we show the contribution of multi-classifier scheme on detecting new UAVs with high accuracy.



(a) Single NN



(b) Multi-NN

Figure 11: Comparison in classification accuracy in Single-NN and Multi-NN schemes using different aggregation methods. In Multi-NN, "PM" is short for Probability Sum+Majority Vote, "MM" is short for Majority Vote+Majority Vote and "2level" indicates our proposed 2-level score-based aggregation.

Individual accuracies	Multi-NN accuracy
33%	76%
47%	

(a) round 1: 2 NNs

Individual accuracies	Multi-NN accuracy
27%	87%
31%	
45%	
40%	

(b) round 2: 4 NNs

Individual accuracies	Multi-NN accuracy
22%	89%
30%	
32%	
21%	
45%	
42%	
38%	
40%	

(c) round 3: 8 NNs

Table I: Intermediate results for the process of determining the size of multi-NN scheme.

As mentioned in Section III-C, in all the experiments we use slice size of $l = 200$, and categorical cross entropy loss function, to train the neural networks.

A. Sub-example (sub-ex) Voting

As explained in Section IV-B, the proposed sub-ex voting is an aggregation method for determining the prediction for an example within one neural network. To compare the sub-ex voting method with previously existing aggregation methods for a single neural network, we train two different architectures of AlexNet1D and ResNet1D on bursts 1,2 and 3 and we test them on burst 4. For combining the probability vectors at the output of each neural network, we compare three methods of 1) Probability Sum, 2) Majority Vote, and the proposed 3) sub-ex voting. Figure 11a shows that the results of Probability Sum and Majority Vote are in the same range of $\sim$50% for this dataset. Compared to these methods, our proposed sub-ex voting, improves the accuracy to 61% and 63% for a single AlexNet1D and ResNet1D, respectively.

B. Determining the Size of Multi-NN Scheme (J)

To use the method described in Section IV-C for our UAV dataset, we leave aside burst 4 which is our final test set, to prevent information leakage from this burst into the process of determining J. Bursts 1 and 2 which are the first arriving bursts serve as initial training set, and burst 3 which is collected after them, serves as the intermediate test set. In round 1, we create 2 *set*s, containing complete burst 1 and burst 2, respectively. We train 2 separate neural networks on the two sets and test them on the unseen burst 3. We use the two level aggregation method to calculate the final multi-classifier accuracy. Individual and multi-classifier accuracies for this round are shown in Table Ia. In round 2, we create 4 sets by dividing each previous set into two parts. The 4 sets contain first half of burst 1, second half of burst 1, first half of burst 2, and second half of burst 2, respectively. Consequently, we train 4 neural network on these sets and test them on burst 3. Individual and multi-classifier accuracies for round 2 are shown in Table Ib. In round 3, we create 8 sets again by dividing each previous set into two parts. Therefore, 8 neural networks are trained on the 8 sets ranging from the first quarter of burst 1 to the fourth quarter of burst 2. The test results on the unseen burst 3 in round 3 are shown in Table Ic. The splitting process stops when accuracy improvement diminishes to 2% after 3 rounds and the final J is calculated as 8. At this point, each of bursts 1 and 2 are partitioned into 4 sets.

For the final training/testing, we bring burst 3 into the training scheme, so that we have more sets and larger number of classifiers. Same as bursts 1 and 2, burst 3 is also partitioned into 4 sets, which yields 12 sets in total. The final outcome of dataset partitioning flow is shown in Figure 6, where we train $J = 12$ independent but identical classifiers on different portions of bursts 1, 2, and 3. We finally test all of the networks on the unseen burst 4.

C. Multi-classifier with Two Levels of Aggregation

As explained in Section V-B, for the multi-classifier scheme, we train 12 identical AlexNet1Ds (less complex network in Figure 4a) in parallel. At test time, we feed each example to all the 12 trained neural networks, and use three methods to combine their results:

- As the first level of aggregation, we combine results of sub-examples, using Probability Sum. At the second level, we use Majority Vote among the 12 neural networks in the multi-classifier scheme.
- We use Majority Vote in both levels of aggregation, for combining both sub-example results and multi-classifier results.
- We use the proposed two-level aggregation method to combine the results.

Scheme	Single AlexNet1D	Single ResNet1D	Multi(12) AlexNet1D
No. of parameters	~1.1M	~16M	~13.2M
Training Time (h)	20.5	44	34
Test Time (h)	2.5	14	30
Total Time (h)	23	58	64
Final Accuracy	49%	50%	91%

Table II: The trade-off between complexity of the schemes and final accuracy.

Figure 11b shows the numerical results for these three cases. The combination of Probability Sum and Majority Vote is shown as "PM" (45% accuracy), Majority Vote at both levels as "MM" (46% accuracy), and the two-level aggregation as "2level" (91% accuracy). The two-level aggregation improves the accuracy for multi-classifier scheme by more than 100%. Moreover, our score-based multi-classifier improves the accuracy of single classifiers using Probability Sum or Majority Vote (Figure 11a). This improvement is 85% and 82% for AlexNet1D and ResNet1D, respectively.

Interestingly, the accuracy improvement in multi-classifier approach, compared to single ResNet1D, comes at no cost of increase in model size. There are $\sim$16M and $\sim$13.2M (=12×1.1) parameters, in ResNet1D and our multi-classifier scheme, respectively. In other words, the multi-classifier scheme with 12 AlexNet1Ds still has fewer parameters than a single ResNet1D, even though it yields 82% improvement in accuracy of ResNet1D. This accuracy boost, however, comes at a cost of longer training/testing process. As shown in Table II, the training and test time increases from 23 and 58 hours for single AlexNet1D and ResNet1D, respectively, to 64 hours for the multi-classifier scheme.

The benefit of our method comes from learning different data sequence distributions collected over time in parallel. At test time, in both levels of aggregation, we suppress the least certain predictions and keep only the most certain one.

D. Data Augmentation (DA)

To show the contribution of DA towards improving the accuracy of individual neural networks, we independently retrain each of the 12 classifiers on sets shown in Figure 6, with the DA block included in the training pipeline. We train the neural networks for 50 epochs to ensure sufficient variations of training data is provided to the network. In the test phase, we use the FIR ensemble method to test the classifiers on the unseen burst 4, and calculate individual accuracies, as well as the multi-classifier accuracy using the two-level aggregation method. The individual accuracies without and with augmentation block, and their corresponding multi-classifier accuracies can be seen in Table III. It is observed that DA improves the overall accuracy from 91% to 95%.

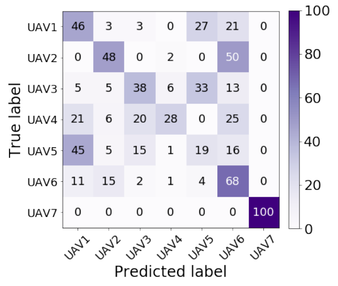
Figure 12 further shows the confusion matrices for four different cases of (a) a single AlexNet1D being trained on bursts 1, 2, 3 and tested on burst 4 using Probability Sum for per-example accuracy (49% accuracy), (b) same case with DA in the pipeline (56% accuracy), (c) multi-classifier scheme with two levels of aggregation (91% accuracy), and (d) same case with DA in the pipeline (95% accuracy). We can see in the latter case, diagonal dark cells have formed, which indicate correct predictions.

In order to show the effect of DA on over-fitting of the neural network, we do another experiment. We train a network with set1 in Figure 6 as an example (NN1). At the end of each training epoch, we test the model on the unseen burst 4, in order to keep track of loss and slice accuracy of the test set over epochs. We do this process without and with DA block. Figures 13a and 13b show the gap between training and test loss reduces from 6.54 to 1.77, when DA is used, which shows a reduction in over-fitting. An increase in the test loss over epochs that we observe in Figure 13a and 13b is not necessarily equivalent to a decrease in test accuracy. This increasing trend could specially be observed in our case of training on set1 and testing on burst 4, where training and test sets are collected under different channels and different distributions. As it can be seen in Figure 13c and 13d, test accuracies are actually improving over epochs, in both cases of without and with DA. The important point is as seen in Figure 13c, without DA training accuracy quickly increases, however test accuracy does not increase with it. This means the model is doing well on the training data, however, it cannot well generalized to the unseen test set, and hence, over-fitting is happening. This adverse situation can be ameliorated by augmenting the training data as seen in Figure 13d. In this case, the model is more robustly trained with DA, which leads to an increase in slice accuracy from 28% to 33%. As shown in Table III, the example accuracy corresponding to this case (NN1) has even a larger increase from 30% to 40% with DA.

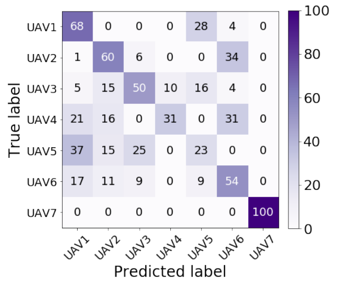
E. New UAV Detection

To implement the new UAV detection method along with multi-classifier scheme, as explained in Section IV-E, we choose UAV4 as the new UAV, and other 6 UAVs as the old ones (same as Section III-E). We exclude signals of UAV4 from sets shown in Figure 6, and use 90% of each set as training set, and 10% as validation set. We train $J = 12$ networks on the old UAV training sets and test them on the old UAV validation sets to record thresholds, as explained in Section III-E1. We form the new UAV test set using examples from UAV4 in burst 4, so that the new UAV signal is also from an unseen burst. After comparing the ratios with the thresholds, we obtain individual accuracies for new UAV detection, shown in Table III. As shown in this table, the multi-classifier accuracy for new UAV detection equals 99%.

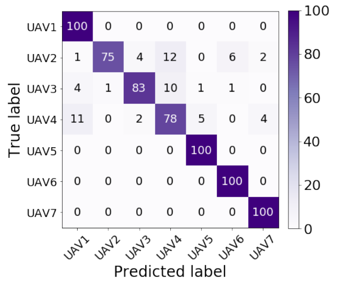
To show how the new UAV detection algorithm performs when an old UAV is fed to it, we do another experiment. We choose UAV6 in burst 4 as an old device in the unseen burst, and we feed its signals to the 12 neural networks trained in this subsection. We repeat the step of comparing ratios and thresholds for each neural network to decide whether the input comes from a new UAV or an old one. Then we use the aggregation method to calculate the multi-classifier accuracy. It should be noted that the old UAV is from the unseen burst 4, which adds to the difficulty of the task. The results shown in Table III illustrate that 11% of examples from the old UAV are



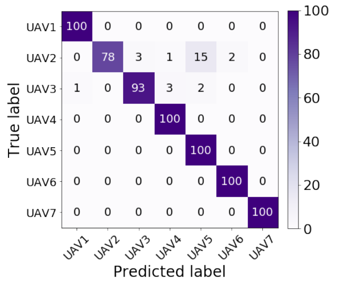
(a) Single NN



(b) Single NN + DA



(c) Multi-NN

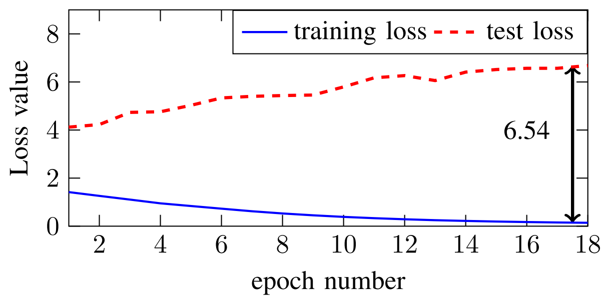


(d) Multi-NN + DA

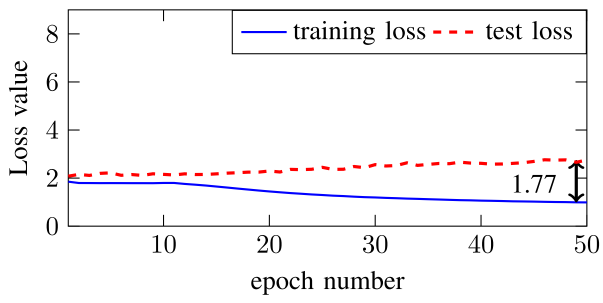
Figure 12: Confusion matrices of test on the unseen burst 4 for four different training schemes.

Neural Network Number	Accuracy Without DA	Accuracy of Multi-NN	Accuracy With DA	Accuracy DA + Multi-NN	New UAV detection individual	New UAV detection Multi-NN	Old UAV detected as new-indiv.	Old UAV detected as new-Multi.
NN 1	30%		40%		56%		25%	
NN 2	39%		36%		64%		21%	
NN 3	34%		38%		84%		27%	
NN 4	32%		35%		64%		22%	
NN 5	26%		40%		96%		33%	
NN 6	35%	91%	42%	95%	96%	99%	28%	11%
NN 7	31%		37%		76%		17%	
NN 8	35%		35%		91%		23%	
NN 9	45%		42%		89%		20%	
NN 10	35%		38%		92%		12%	
NN 11	31%		37%		88%		21%	
NN 12	33%		40%		81%		16%	

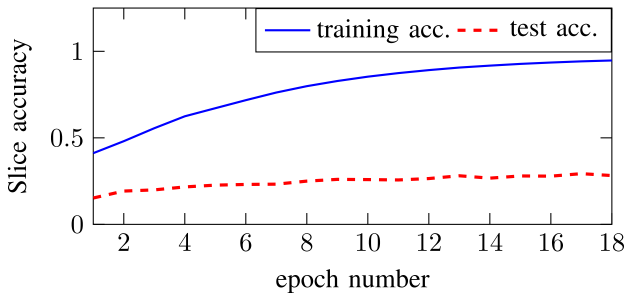
Table III: Individual and multi-NN classification accuracy without and with data augmentation (DA) and new UAV detection.



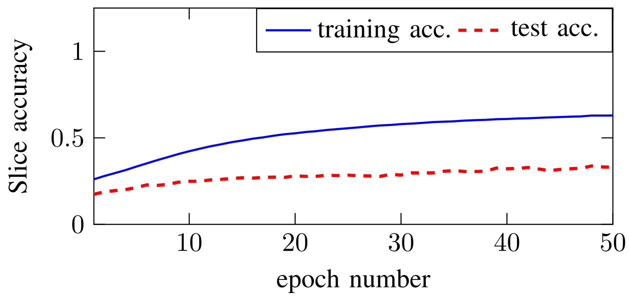
(a) Loss without data augmentation (DA)



(b) Loss with data augmentation (DA)



(c) Accuracy without data augmentation (DA)



(d) Accuracy with data augmentation (DA)

Figure 13: Loss and slice accuracy of training and test sets per epoch for NN1 without and with data augmentation.

Architecture	Prep.	Single NN	Multi-NN
UAV classif.-AlexNet1D	-	49%	91%
UAV classif.-AlexNet1D	DA	56%	95%
UAV classif.-ResNet1D	-	50%	-
New UAV-AlexNet1D	-	68%	99%

Table IV: Comparing accuracy in single NN and multi-NN schemes for UAV classification and new UAV detection.

incorrectly categorized as new UAV examples. Even though the true negative rate of our new UAV detection algorithm is 89%, the high 99% rate of true positives can effectively detect signals from outlier (new) UAVs.

Table IV shows the improvement of accuracy in multi-classifier scheme compared to single classifier. In all cases, training is done on bursts 1, 2, and 3 and test is done on the unseen burst 4. We see that classification accuracy improves from 50% for the best case of single classifier with previously existing methods to 95% with our multi-classifier scheme using two levels of aggregation and data augmentation. Moreover, as shown in Table IV, the 99% new UAV detection accuracy is 45% improvement over the single classifier result of 68% that we reported in Section III-E.

VI. Conclusion

In this paper, we addressed the problem of classifying hovering UAVs that emit proprietary/unknown waveforms using RF fingerprinting with deep neural networks. We empirically showed the adverse effect of imperfect hovering on classification accuracy for 7 UAVs, which initially resulted in a low 49% accuracy using conventional single architecture methods. To tackle this problem, we proposed a multi-classifier scheme to separately learn from different dataset portions. To combine test results at the output of the neural networks, we proposed a novel two-level score-based aggregation method, that returned an overall accuracy of 91%. We used data augmentation to improve individual accuracies of the neural networks in the multi-classifier scheme, which ultimately boosted the classification accuracy upto 95%. Furthermore, our multi-classifier scheme yielded a new UAV detection accuracy of 99%, giving a high confidence that this approach will work in real-world applications.

Acknowledgement

This work is supported by the NSF grant CCF-1937500.

References

[1] GlobeNewswire, "Commercial Drone Market." https://www.globenewswire.com/news-release/2018/02/28/1401040/0/en/Commercial-Drone-Market-to-hit-17bn-by-2024-Global-Market-Insights-Inc.html, 2020.

[2] M. Suresh and D. Ghose, "UAV grouping and coordination tactics for ground attack missions," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 1, pp. 673–692, 2012.

[3] G. Secinti, P. B. Darian, B. Canberk, and K. R. Chowdhury, "SDNs in the sky: Robust end-to-end connectivity for aerial vehicular networks," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 16–21, 2018.

[4] S. Kianoush, A. Vizziello, and P. Gamba, "Energy-efficient and mobile-aided cooperative localization in cognitive radio networks," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 5, pp. 3450–3461, 2015.

[5] K. Sankhe, M. Belgiovine, F. Zhou, S. Riyaz, S. Ioannidis, and K. Chowdhury, "Oracle: Optimized radio classification through convolutional neural networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 370–378, IEEE, 2019.

[6] A. Al-Shawabka, F. Restuccia, S. D'Oro, T. Jian, B. C. Rendon, N. Soltani, J. Dy, S. Ioannidis, K. Chowdhury, and T. Melodia, "Exposing the Fingerprint: Dissecting the Impact of the Wireless Channel on Radio Fingerprinting," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 646–655, IEEE, 2020.

[7] B. W. Ramsey, M. A. Temple, and B. E. Mullins, "PHY foundation for multi-factor ZigBee node authentication," in *2012 IEEE Global Communications Conference (GLOBECOM)*, pp. 795–800, IEEE, 2012.

[8] C. M. Rondeau, M. Temple, and J. Betances, "Dimensional Reduction Analysis for Constellation-Based DNA Fingerprinting to Improve Industrial IoT Wireless Security," in *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.

[9] T. J. Bihl, T. J. Paciencia, K. W. Bauer, and M. A. Temple, "Cyber-Physical Security with RF Fingerprint Classification through Distance Measure Extensions of Generalized Relevance Learning Vector Quantization," *Security and Communication Networks*, vol. 2020, 2020.

[10] G. Reus-Muns, D. Jaisinghani, K. Sankhe, and K. R. Chowdhury, "Trust in 5G Open RANs through Machine Learning: RF Fingerprinting on the POWDER PAWR Platform," in *2020 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2020.

[11] M. Ezuma, F. Erden, C. K. Anjinappa, O. Ozdemir, and I. Guvenc, "Micro-UAV detection and classification from RF fingerprints using machine learning techniques," in *2019 IEEE Aerospace Conference*, pp. 1–13, IEEE, 2019.

[12] G. Reus-Muns, M. Diddi, H. Singh, and K. R. Chowdhury, "Dynamic Channel Selection in UAVs through Constellations in the Sky," in *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[13] M. Ezuma, F. Erden, C. K. Anjinappa, O. Ozdemir, and I. Guvenc, "Detection and Classification of UAVs Using RF Fingerprints in the Presence of Wi-Fi and Bluetooth Interference," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 60–76, 2019.

[14] 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.

[15] F. Restuccia, S. D'Oro, A. Al-Shawabka, M. Belgiovine, L. Angioloni, S. Ioannidis, K. Chowdhury, and T. Melodia, "DeepRadioID: Real-time channel-resilient optimization of deep learning-based radio fingerprinting algorithms," in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pp. 51–60, 2019.

[16] S. Mohanti, N. Soltani, K. Sankhe, D. Jaisinghani, M. Di Felice, and K. Chowdhury, "AirID: Injecting a Custom RF Fingerprint for enhanced UAV Identification using Deep Learning," in *IEEE GLOBECOM*, 2020.

[17] A. Gritsenko, Z. Wang, T. Jian, J. Dy, K. Chowdhury, and S. Ioannidis, "Finding a 'New' Needle in the Haystack: Unseen Radio Detection in Large Populations Using Deep Learning," in *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 1–10, IEEE, 2019.

[18] B. Taha and A. Shoufan, "Machine Learning-Based Drone Detection and Classification: State-of-the-Art in Research," *IEEE Access*, vol. 7, pp. 138669–138682, 2019.

[19] T. Müller, "Robust drone detection for day/night counter-UAV with static VIS and SWIR cameras," in *Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR VIII*, vol. 10190, p. 1019018, International Society for Optics and Photonics, 2017.

[20] T. P. Breckon, S. E. Barnes, M. L. Eichner, and K. Wahren, "Autonomous real-time vehicle detection from a medium-level UAV," in *Proc. 24th International Conference on Unmanned Air Vehicle Systems*, pp. 29–1, Citeseer, 2009.

[21] P. Nguyen, M. Ravindranatha, A. Nguyen, R. Han, and T. Vu, "Investigating cost-effective RF-based detection of drones," in *Proceedings of the 2nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pp. 17–22, 2016.

[22] H. Li, G. Johnson, M. Jennings, and Y. Dong, "Drone profiling through wireless fingerprinting," in *2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pp. 858–863, IEEE, 2017.

[23] Z. Shi, M. Huang, C. Zhao, L. Huang, X. Du, and Y. Zhao, "Detection of LSSUAV using hash fingerprint based SVDD," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–5, IEEE, 2017.

[24] I. Bisio, C. Garibotto, F. Lavagetto, A. Sciarrone, and S. Zappatore, "Unauthorized amateur UAV detection based on WiFi statistical fingerprint analysis," *IEEE Communications Magazine*, vol. 56, no. 4, pp. 106–111, 2018.
[25] C. Zhao, C. Chen, Z. Cai, M. Shi, X. Du, and M. Guizani, "Classification of small UAVs based on auxiliary classifier Wasserstein GANs," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 206–212, IEEE, 2018.
[26] J. Shin, S. H. Jung, G. Yoon, and D. Han, "A Multi-classifier Approach for WiFi-based Positioning System," in *Electrical Engineering and Applied Computing*, pp. 135–147, Springer, 2011.
[27] Y. Yuan, X. Liu, Z. Liu, and Z. Xu, "MFMCF: A Novel Indoor Location Method Combining Multiple Fingerprints and Multiple Classifiers," in *2019 3rd International Symposium on Autonomous Systems (ISAS)*, pp. 216–221, IEEE, 2019.
[28] MATRICE 100, "QUADCOPTER FOR DEVELOPERS." https://www.dji.com/matrice100, 2020.
[29] Matrice 100 DJI, "Airlink guide." https://developer.dji.com/mobile-sdk/documentation/introduction/component-guide-airlink.html, 2018.
[30] Ettus Research, "X310 Radios." https://www.ettus.com/all-products/x310-kit/, 2020.
[31] F. Chollet *et al.*, "Keras," 2015.
[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.
[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
[34] N. Soltani, K. Sankhe, S. Ioannidis, D. Jaisinghani, and K. Chowdhury, "Spectrum Awareness at the Edge: Modulation Classification using Smartphones," in *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 1–10, IEEE, 2019.
[35] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of machine learning research*, vol. 12, no. Jul, pp. 2121–2159, 2011.
[36] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Multi-classification approaches for classifying mobile app traffic," *Journal of Network and Computer Applications*, vol. 103, pp. 131–145, 2018.
[37] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, 2019.
[38] N. Soltani, K. Sankhe, J. Dy, S. Ioannidis, and K. Chowdhury, "More is better: Data augmentation for channel-resilient rf fingerprinting," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 66–72, 2020.

Nasim Soltani is currently a Ph.D. candidate at the department of Electrical and Computer Engineering of Northeastern University, Boston. She is pursuing her degree in wireless communications under the supervision of Professor Chowdhury. She is interested in designing intelligent algorithms for physical layer tasks in the wireless systems. Her current research focuses on deep learning techniques for robust RF fingerprinting, demodulation, and modulation detection.

Guillem Reus-Muns received a B.Sc. degree in telecommunications engineering from the Polytechnic University of Catalonia (UPC-BarcelonaTech). He joined Northeastern University, USA, in 2018, where he got his M.Sc. in electrical and computer engineering and is currently working towards his Ph.D. His current research interests involve cellular networks, machine learning for wireless communications, networked robotics, and spectrum access.

Batool Salehi is currently pursuing a Ph.D. degree in computer engineering at Northeastern University under the supervision of Prof. K. Chowdhury. She received her M.S. (2019) in Electrical Engineering from University of Tehran, Iran. Her current research focuses on mmWave beamforming, Internet of Things, and the application of machine learning in the domain of wireless communication.

Jennifer Dy is a Professor at the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, where she first joined the faculty in 2002. Her research spans both fundamental research in machine learning and their application to biomedical imaging, health, science and engineering, with research contributions in unsupervised learning, dimensionality reduction, feature selection, learning from uncertain experts, active learning, Bayesian models, and deep representations.

Stratis Ioannidis is an Associate Professor in the Electrical and Computer Engineering Department of Northeastern University, in Boston, MA, where he also holds a courtesy appointment with the Khoury College of Computer Sciences. Prior to joining Northeastern, he was a research scientist at the Technicolor research centers in Paris, France, and Palo Alto, CA, as well as at Yahoo Labs in Sunnyvale, CA. His research interests span machine learning, distributed systems, networking, optimization, and privacy.

Kaushik Chowdhury is a Professor at Northeastern University, Boston, MA. He is presently a co-director of the Platforms for Advanced Wireless Research (PAWR) project office. His current research interests involve systems aspects of networked robotics, machine learning for agile spectrum sensing/access, wireless energy transfer, and large-scale experimental deployment of emerging wireless technologies.