It's a Bird, It's a Plane, It's "That" UAV: RF Fingerprinting During Flight

Jerry Gu, Nasim Soltani, M. Yousof Naderi, Kaushik R. Chowdhury
Dept. of Electrical and Computer Engineering, Northeastern University, Boston, MA, USA, 02115
{gu.je, soltani.n}@northeastern.edu, {naderi, krc}@ece.neu.edu

2021 55th Asilomar Conference on Signals, Systems, and Computers | 978-1-6654-5828-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IEEECONF53345.2021.9723260

***Abstract*—UAVs are being rapidly deployed in many surveillance-related and monitoring applications worldwide. Thus, identifying a known UAV in a larger pool of devices is important. This task is often complicated in dense urban environments or low-level flight deployment cases where traditional radar-based detection becomes difficult. In this work, we describe deep convolutional neural network architectures and preprocessing steps suitable for capturing RF signals from in-flight UAVs for detection of the type of the UAV. Our objective is to leverage subtle discriminative features that may be embedded in the signal transmissions through RF fingerprinting for identifying (i) if the test signal comes from an unseen UAV (with respect to the training set), and (ii) if it is from a seen UAV, then we identify the label used during training time. Unlike static data collections, the mobile scenario is more challenging due to the rapid fluctuations in the wireless channel and Doppler effects which impair successful classification. We study the efficacy of our approach under different distances, flight/mobility patterns, interference conditions to emulate real-world situations with high fidelity. Our experimental dataset contains signals from seven different make/models, collected within an RF anechoic chamber.**

***Index Terms*—UAV RF Fingerprinting, Convolutional Neural Networks, New UAV Detection**

I. Introduction

The demand for unmanned aerial vehicles (UAVs) is at an all-time high, and this market segment is expected to keep growing [1]. With various military and civilian applications, the global UAV market size is expected to expand from $14 billion USD in 2018 to $43 billion in 2024 [2]. The recent COVID-19 pandemic may have boosted UAV sales, as consumers seek out socially-distanced activities and companies are exploring autonomous methods for transporting goods and services [3]. However, this unprecedented rise in UAV sales also bring many concerns of public safety and personal privacy, particularly for small commercial-off-the-shelf (s-COTS) UAVs, which have already been used for armed attacks, unlawful surveillance, and cyber attacks [4]. Therefore, in order to combat these threats, there is a growing investment in research towards UAV detection and classification.

Several methods for UAV detection and classification has already been proposed. The use of visual recognition, infrared (IR) recognition, acoustic recognition, and radar probing have met with limited success with modality-specific shortcomings, such as limited detection range and sensitivity to environmental noise. Use of radio frequency (RF)-based techniques, also known as RF fingerprinting, is a candidate solution to some of these problems due to its energy efficiency, low-cost hardware installation, and ability to deal with noise via passive listening, high gain antennas, and de-noising techniques, respectively [5]. Our previous works use RF fingerprinting to detect and classify proprietary waveforms from DJI Matrice 100 UAVs [6] as well as other DJI UAV models [7]. A key limitation of our prior works, as well as many of the other works in this topic, is that the dataset used for classification of UAVs only consists of transmitted signals from UAVs in fixed positions, i.e., stationary on the ground, or hovering at a fixed position and altitude in the air. In reality, UAVs, particularly when used with malicious intent, are rarely in only one constant position; rather, they are constantly flying from one place to another. Thus, motion adds additional complications such as Doppler shift and rapid fluctuations in the wireless channel, which may negatively impact classification of UAV models using their transmitted RF signals. To the best of our knowledge, this is the first paper to address the specific impact of flying and motion on UAV classification with RF fingerprinting.

Our approach, while leveraging the class of well-known neural networks known as convolutional neural networks (CNNs), contains a number of novel aspects. For the UAV dataset, we collect transmitted waveforms from seven COTS UAVs within an RF anechoic chamber to isolate the UAV signals from any background noise. The signals are collected from UAVs in two motion-based categories: stationary and flying (either taking off, flying, or landing). The transmissions, which are composed of streams of raw in-phase and quadrature (I/Q) samples, are further filtered and preprocessed into smaller data transmissions. Then, we use two distinct CNN architectures, one which has previously shown to be widely successful in RF fingerprinting and image processing tasks [8], and a novel 'light' version of the previous architecture. Our goal is to both classify seven COTS UAV models and demonstrate the effects of motion on classification. Finally, given that new UAVs are always being developed, we implement a method for new UAV detection, that is, detection of UAV types that our CNN models have not been trained on. We do this by comparing the statistics from the UAVs that the architectures are trained on with the statistics of the incoming signals, using limited additional computational resources.

The remainder of this paper is organized as follows. In Section II, we elaborate on the collected dataset, including the UAV models used and the data collection and filtering process. We describe in Section III the UAV classification and new UAV

detection methods, and evaluate our methods in Section IV. Finally, we give our conclusions for this work in Section V.

II. Background

A. UAV Models

We collect RF signals from seven UAV models within an RF anechoic chamber. These models, along with some of their basic characteristics, are presented in Table I. For the full list of specifications, please see [9]–[15].

Model	Operating Frequency	Dimensions (Unfolded)	Weight	Battery Capacity
Autel EVO	2.4-2.4835 GHz	38 x 30.5 x 10 cm (15 x 12 x 3.9 in.)	863 g (1.90 lbs)	4300 mAh
Autel EVO 2	2.4-2.4835 GHz	42.4 x 35.4 x 11 cm (16.7 x 13.9 x 4.3 in.)	1150 g (2.5 lbs)	7100 mAh
DJI Mavic 2 Pro	2.4-2.4835 GHz; 5.725-5.850 GHz	32.2 x 24.2 x 8.4 cm (12.7 x 9.5 x 3.3 in.)	907 g (2 lbs)	3850 mAh
DJI Mavic Air 2	2.4-2.4835 GHz; 5.725-5.850 GHz	18.3 x 25.3 x 7.7 cm (7.2 x 10 x 3 in.)	570 g (1.3 lbs)	3500 mAh
Parrot ANAFI	2.4-5.8 GHz	17.5 x 24 x 6.5 cm (6.9 x 9.4 x 2.6 in.)	320 g (0.7 lbs)	2700 mAh
Skydio 2	2.4-2.483 GHz; 5.18-5.24 GHz; 5.725-5.85 GHz	22.3 x 27.3 x 7.4 cm (8.7 x 10.7 x 2.9 in.)	775 g (1.7 lbs)	4280 mAh
Skydio R1	2.4-2.483 GHz; 5.18-5.24 GHz	33 x 40.6 x 5 cm (13 x 16 x 2 in.)	998 g (2.2 lbs)	16 min (Capacity not given)

TABLE I: Summary of UAV models used.

B. Collection Setup

1) Hardware and Software: In our setup, we use two types of receivers (Rxs) as RF sensors: (i) one CRFS RFeye Portable Recorder Real-time Spectrum Analyzer (RTSA), and (ii) two Ettus X310 Universal Software Radio Peripherals (USRPs). The RTSA is connected to a ETS-Lindgren 3181 Broadband Mini-Bicon antenna, and the X310s each use two VERT2450 dual-band omni-directional antennas.

The RTSA is connected to a laptop containing CRFS Deepview, which is manually set to consecutively record a 100 MHz bandwidth (BW) at each of 2.4, 5.2, and 5.8 GHz, depending on the known frequencies transmitted by the UAV being tested. For the X310s, both USRPs are connected to a laptop running GNURadio. A custom script in GNURadio is implemented such that up to two of the aforementioned frequencies are collected at once. The primary purpose of the RTSA dataset is to visualize the data, and the X310 dataset is formatted and is used for further analysis in this work. The two datasets, though recorded simultaneously, are not synchronized.

2) Setup and Procedure: The hardware and laptops are placed on a raised platform within the RF anechoic chamber with the UAV under testing placed >20 ft. away such that there is line-of-sight (LOS) between the UAV being tested and the antennas. The collection setup is shown in Fig. 1. In order to collect the dataset, we designate a UAV pilot who controls the UAV remotely, and a data collector, who operates the sensors concurrently.

C. Dataset and Filtering

For the dataset, we specify four unique mobility-based scenarios for the UAVs, and we record the UAV transmissions for each scenario:

- **Stationary:** In this scenario, both the UAV and controller are turned on and placed >20 ft. from the custom antenna.

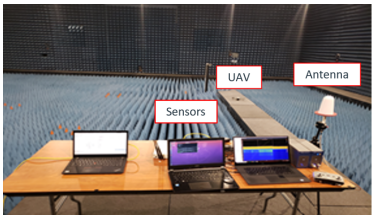

Fig. 1: Experimental setup in the RF anechoic chamber.

The UAV operator checks to see if the devices are connected; that is, if the controller can detect the UAV, and if the UAV is able to fly. The UAV is grounded in place without any propellers turned on.

- **Takeoff:** In this scenario, the UAV is on ground before recording starts. Once the sensor operator begins recording, the UAV operator immediately gives a three-second countdown to the sensor operator before issuing the command for the UAV to *take off*, i.e., begin flying upwards. After taking off, the UAV operator keeps the UAV hovering at a set height in the air for the duration of the recording.
- **Flying:** In this scenario, the UAV is hovering mid-air before the recording begins. The UAV operator may fly the UAV in any direction during the recording, as long as the UAV is 20-40 ft. from the antennas, unless otherwise specified.
- **Landing:** In this scenario, the UAV begins in a hovering, mid-air position before recording. Once the sensor operator begins recording, the UAV operator immediately gives a three-second countdown to the sensor operator before inputting the command for the UAV to *land*, i.e., descend downwards to the ground, turning off its propellers after it has landed. The UAV remains on ground for the rest of the duration of the recording.

The latter three scenarios are grouped together into one category, *motion*. For each scenario, we start by collecting ∼10s (∼5M I/Q samples) for each UAV and every scenario, at every known transmission frequency band.

Once the dataset is collected, we move to selectively filter the data such that the dataset only contains only the video *downlink* signal from each UAV to its corresponding controller, as confirmed by viewing the spectrum when only the UAV is turned on. First, we select a 10 MHz BW in which the downlink signal per UAV is known to be transmitting, and cut out the rest of the signal. Then, we split the filtered signal into chunks of 10,000 samples and label each file for easier file management. We use a root mean squared (RMS) threshold, we filter out any noise, cutting out large gaps of the signal in the time domain. Finally,we collect 5 GB worth of signals from each UAV, consisting of an approximately equal number of samples for stationary and motion scenarios, to ensure balance in NN training.

III. Methodology

In this section, we describe the architectures of the CNNs we used for RF fingerprinting, as well as the training and test

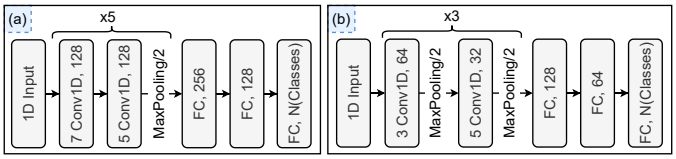

Fig. 2: Architectures of the (a) *AlexNet1D*, and (b) *AlexNet1D-lite* models.

pipelines for UAV model classification. Then, we describe our methodology for detecting when a transmission comes from a new UAV model that the CNN has not been trained on.

A. Convolutional Neural Network Architectures

For the goal of RF fingerprinting using our flying UAV dataset, we use two CNN architectures, inspired by the AlexNet model [8] and implemented in Python using Keras [16] libraries with TensorFlow backend. The first model, which we call the *AlexNet1D* model, is a forward CNN with ∼1.1M parameters. It consists of five stacked groups of convolutional layers, followed by two fully-connected (FC) layers of sizes 256 and 128, respectively, and ending with a Softmax layer with the same size as the number of UAV models (classes). Each stacked group contains two convolutional layers each with 128 filters, with convolution size of 7 and 5, respectively, followed by a MaxPooling layer. The second architecture, which we call the *AlexNet1D-lite* model, is a smaller model based on the *AlexNet1D* model with ∼69k parameters that consists of three stacked groups, a fully-connected layer of size 128, another fully-connected layer of size 64, and a last fully-connected layer with the same size as the number of UAV models once more. The three groups each contain a convolutional layer with 64 filters and 3 taps, a MaxPooling layer, another convolutional layer with 32 filters and 5 taps, and a final MaxPooling layer. The architectures are shown in Fig. 2.

B. Training and Testing Pipelines

Preprocessing. In order to prepare the entire dataset for usage in either CNN model, we first perform a preprocessing step. We randomize and partition the dataset into distinct training, validation, and testing groups, consisting of 70%, 10%, and 20% of the total dataset, respectively. From there, we assign each transmission in the training set a 'true' label that reflects which UAV the transmission came from. Once the entire training set is labeled, we calculate the mean μ and standard deviation σ over the entire training set for normalization purposes during training, validation, and testing.

Training Pipeline. We train and test our CNNs on a per-*slice* basis. Each slice is a sequence of consecutive I/Q samples with length *slice_size*, in each transmission. We feed the training data to the CNNs in the form of tensors with dimensions of (*batch_size*, *slice_size*, 2). The first dimension "*batch_size*" is chosen as a power of 2 and contributes to GPU parallelization. During training, a number as large as *batch_size* transmissions are randomly selected from the training set and loaded in the memory. From each transmission, one slice with length slice_size with random position is selected. Therefore, batch_size slices are stacked together, and the real and imaginary parts are separated to construct the last dimension of the tensor. After the tensors are formed, each tensor X is normalized with respect to mean (μ) and standard deviation (σ) of the training set, as shown in (1), which were calculated during the preprocessing step. The normalization step yields the normalized batch $X_{\text{norm.}}$, that is fed into the CNN, along with the one hot representation of true labels in the form of (*batch_size*, # Classes).

$$X_{\text{norm.}} = \frac{X - \mu}{\sigma} \tag{1}$$

We use the Adam optimizer with a learning rate of 0.0001 and a categorical cross entropy loss function to train the CNNs. At the end of each training epoch, we test the CNN on the validation set. We stop training when the validation accuracy does not improve for 3 consecutive epochs.

Testing Pipeline. During testing, we load test set transmissions into the architecture one by one and slice each transmission into consecutive slices with stride = 1. We stack the slices together to form the test batch, which is fed into the trained CNNs. The output of CNN is a *probability vector* for each test slice. The probability vector is a vector with a length equal to the number of classes and its elements show the probability of each class being the predicted class by the CNN. The predicted class is the index of the largest element in the probability vector and is calculated by passing the probability vector through an $argmax$ function. We report two types of accuracy: (i) slice accuracy, and (ii) transmission accuracy. To calculate slice accuracy, we classify all the slices in all the transmissions in the test set, and divide the number of correctly predicted slices by the number of total slices in the test set. To calculate transmission accuracy, we sum all the probability vectors gained from each transmission to achieve a *probability sum* vector. Then, we choose the index of the largest element in the probability sum vector as the predicted class for each transmission. We divide the number of correctly predicted transmissions by the total number of transmissions in the testing set to achieve transmission accuracy.

C. New UAV Detection

In the proposed classifier CNN, the number of neurons in the last layer are the same as the number of UAVs (classes) in the training set. We call the UAVs, whose signals are seen by the CNN during training, the 'in-library' UAVs. If any input slice, either from an in-library or 'out-of-library' (new) UAV, is fed to the trained CNN, it inevitably triggers a last layer neuron that corresponds to one of the in-library UAVs. Therefore, we need a method to distinguish between a signal that is either from one of the in-library UAVs or from a new UAV that will inevitably be classified as an in-library UAV. To do this, we extend the new class detection algorithm that is proposed in [17] for new device detection of static transmitters to our flying UAV dataset.

To explain this method, we assume a test set consisting of transmissions from a specific UAV in our dataset as the new UAV, and a training set consisting of transmissions from

rest of the UAVs as in-library UAVs. We train the CNN on the training set. When the CNN is fully trained, we test the network on the same training set consisting of transmissions from in-library UAVs, and calculate two types of thresholds:

Threshold 1. Probability Threshold. For each UAV, we obtain a set of correctly classified probability vectors in all the transmissions belonging to that UAV. In the set belonging to each UAV, we record the maximum value in each vector. We perform a statistic function (whose options are shown in Table II) on all the maximum values that belong to each UAV, to obtain the Probability Threshold for each UAV.

Threshold 2: Ratio Threshold. For each UAV, we go through the transmissions belonging to that UAV. We record the number of correctly classified slices divided by total number of slices in each transmission. Next, we perform the same statistic function that we use for Threshold 1, to obtain the Ratio Threshold for each UAV.

After the thresholds are calculated, we test the trained network on transmissions with unknown labels that we call the test set. As explained before, each test transmission is predicted to belong to one of the in-library UAVs. We call this prediction the 'best-guess'. In order to evaluate the validity of the best-guess, we calculate two metrics for each test transmission:

Metric 1: Prediction Probability. We obtain a set of maximum values in each probability vector, belonging to each test transmission. We perform the same statistic function that we used for calculating thresholds on this set to obtain the Prediction Probability for that test transmission.

Correct Slice Ratio. We divide the number of slices classified as best-guess by the total number of slices in that transmission to obtain the Correct Slice Ratio for that test transmission.

Notation	*avg*	*min*	*var1*	*var2*
Function	μ	$min(X)$	$\mu - \sigma$	$\mu - 2 * \sigma$

TABLE II: Statistic functions used to calculate Threshold 1, Threshold 2, and Metric 1.

IV. Performance Evaluation

In this section, first, we show UAV classification results where the CNNs are trained and tested on separate partitions of the dataset of seven UAVs. Next, we show new UAV detection results where the CNN is trained on six of the UAVs. The new device detection algorithm distinguishes the seventh (new) UAV from the rest.

A. UAV Classification

We use our dataset (Section II) to explore the effects of each architecture, mobility scenarios, and individual UAVs, on the NN pipelines in order to understand the impact of each. As described in Sections II and III, we partition our dataset consisting of seven UAVs in four mobility-based scenarios into training (70%), validation (10%), and test (20%) sets. We train both the AlexNet1D and AlexNet1D-lite architectures with the training sets for a maximum of 30 epochs and use a *batch_size* of 512 and a *slice_size* of 256. When the architectures are fully trained, we test each architecture with the test set and obtain the slice and transmission accuracies. To show the impact of the

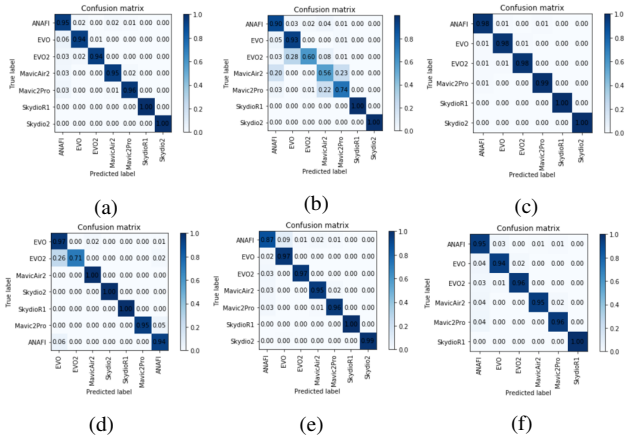

Fig. 3: Testing different premises: (a) AlexNet1D model, (b) AlexNet1D-lite model, (c) AlexNet1D model, training on stationary scenarios, testing on motion scenarios, (d) AlexNet1D model, training on motion scenarios, testing on stationary scenarios, (e) AlexNet1D model, No DJI Mavic Air 2 UAV, (f) AlexNet1D model, No Skydio 2 UAV

Premise	**Slice Accuracy (%)**	**Transmission Accuracy (%)**
a) AlexNet1D	94.52	96.28
b) AlexNet1D-lite	92.02	93.64
c) Training on stationary, testing on motion	73.74	80.22
d) Training on motion, testing on stationary	88.29	96.04
e) No DJI Mavic Air 2	97.66	98.85
f) No Skydio 2	94.20	96.11

TABLE III: Slice and transmission accuracies for each premise.

aforementioned effects, we construct specific premises listed in Fig. 3 and Tab. III and show the slice and transmission accuracies for each premise.

To further explain these results, we group together sets of two premises at a time to contrast their results. In the first group, we test the effects of the NN architecture with (a) the AlexNet1D architecture and (b) the AlexNet1D-lite architecture. Overall, the AlexNet1D model performed slightly better with a (slice accuracy, transmission accuracy) of (94.52%, 96.28%) across all scenarios and UAV models. This demonstrates that although the more complex AlexNet1D model yields a higher accuracy, the AlexNet1D-lite model, with 94% fewer parameters, was able to perform adequately (92.02%, 93.64%), losing only up to 3% classification accuracy in both slice and transmission accuracy.

In the second group, we test the effects of mobility scenarios on classification, which is one of the main focuses of this paper. By (c) training on stationary scenarios only and testing on all motion scenarios and then (d) training on all motion scenarios and only testing on stationary scenarios with the AlexNet1D model, we demonstrate that UAV mobility has an impact on classification accuracy with (c) having (73.74%, 80.22%) and (d) having (88.29%, 96.04%). These $\sim$16% decreases is explained by the robustness of the signals in the variety of motion scenarios; even though the number of samples for stationary and all motion scenarios are approximately equal,

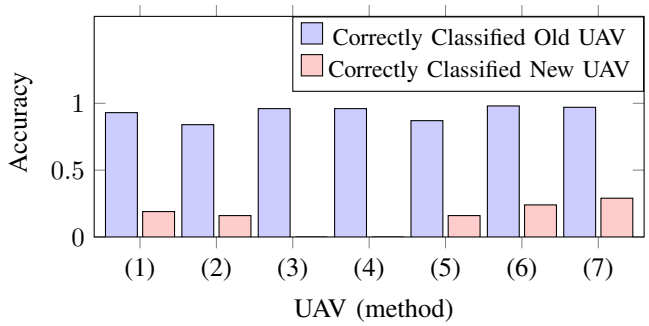

Fig. 4: New UAV detection results for: (1) DJI Mavic Air 2, var1 method; (2) Skydio 2, var1 method; (3) Skydio 2, avg method; (4) Skydio 2, min method; (5) Skydio 2, var2 method; (6) Skydio 2, var1 method; (7) Skydio 2, var1 threshold with var2 metrics

the motion scenarios contain a larger variety, and therefore influence the μ and σ for each UAV to a greater degree.

For the last group of two premises, we check to see if individual UAV models bias the classification results in preparation for new UAV detection. Therefore, we remove only the (f) DJI Mavic Air 2 and only the (g) Skydio 2 from the original dataset of seven UAVs, obtaining accuracies of (f) (97.66%, 98.85%) and (g) (94.20%, 96.11%). As the accuracies differ by less than 3% for both slice and transmission accuracy, we can say without loss of generalization that the individual UAVs do not have a dramatic influence on the overall classification accuracy.

B. New UAV Detection

We test the new UAV detection algorithm with our dataset and display the results in Fig. 4. Again, without loss of generality, we test the algorithm by using the DJI Mavic Air 2 in (1) and the Skydio 2 in (2)-(7) as the new UAV, and by using the variety of statistics explained in Sec. III. On average, the algorithm had a 93% accuracy rate in detecting any given UAV input as coming from an old, 'in-library' UAV, but only had a 15% accuracy rate in detecting a new UAV input as new, with the highest accuracy being 29% for method (7). This large discrepancy is explained by the similarity of the calculated thresholds between the old and new UAVs, which would also explain the high accuracy of correctly classifying old UAVs.

There are several ways to potentially increase the new UAV detection accuracy. Further analysis of methods such as slice or prediction aggregation on new UAV detection would be warranted, as would a novel weighting scheme for the aggregation of predictions. As proposed by [6], a multi-classifier scheme to verify the results of our single NN network may also improve accuracy. We will consider these methods in our future work.

V. Conclusion

This paper addresses the effect of motion on UAV RF fingerprinting by training two separate CNN architectures on seven UAV models in a variety of motion-based scenarios. We show that classifying UAVs in scenarios where the UAV is only stationary can result in up to a 16% decrease in accuracy when training a NN architecture on UAVs in only stationary scenarios and testing on UAVs in non-stationary motion scenarios. Furthermore, we test a new UAV detection algorithm on our collected dataset, which is able to detect 'in-library' UAVs with an average of 93% accuracy, but is only able to detect new UAVs with an average of 15% accuracy. Finally, we proposed possible future directions to increase robustness of our methods and improve results, including slice and/or prediction aggregation, weighted predictions, and the use of a multi-classifier scheme.

VI. Acknowledgements

This work was supported by the NSF award CNS 1923789.

References

[1] P. Butterworth-Hayes, "Commercial UAV market to reach USD15.62 Billion by 2026 - Polaris Market Research," Sep 2020. [Online]. Available: https://www.unmannedairspace.info/latest-news-and-information/commercial-uav-market-to-reach-usd15-62-billion-by-2026-polaris-market-research/

[2] L. Schroth, "Drone Market 2019-2024: 5 Things to Know," Oct 2020. [Online]. Available: https://droneii.com/the-drone-market-2019-2024-5-things-you-need-to-know

[3] F. Greenwood, "Assessing the impact of drones in the global covid response," Jul 2021. [Online]. Available: https://www.brookings.edu/techstream/assessing-the-impact-of-drones-in-the-global-covid-response/.

[4] J.-P. Yaacoub, H. Noura, O. Salman, and A. Chehab, "Security analysis of drones systems: Attacks, limitations, and recommendations," *Internet of Things*, vol. 11, p. 100218, 2020.

[5] M. Ezuma, F. Erden, C. K. Anjinappa, O. Ozdemir, and I. Guvenc, "Micro-UAV Detection and Classification from RF Fingerprints Using Machine Learning Techniques," *2019 IEEE Aerospace Conference*, Mar 2019.

[6] N. Soltani, G. Reus-Muns, B. Salehi, J. Dy, S. Ioannidis, and K. Chowdhury, "RF Fingerprinting Unmanned Aerial Vehicles With Non-Standard Transmitter Waveforms," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, p. 15518–15531, Nov 2020.

[7] G. Reus-Muns and K. Chowdhury, "Classifying UAVs With Proprietary Waveforms via Preamble Feature Extraction and Federated Learning," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 7, p. 6279–6290, Jul 2021.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, p. 84–90, 2017.

[9] "EVO." [Online]. Available: https://auteldrones.com/products/evo

[10] "EVO II specification." [Online]. Available: https://auteldrones.com/pages/evo-ii-specification

[11] "Mavic Air 2 - Specifications - DJI." [Online]. Available: https://www.dji.com/mavic-air-2/specs

[12] "Mavic 2 - Product Information - DJI." [Online]. Available: https://www.dji.com/mavic-2/info

[13] "Operator Manual-Skydio 2." [Online]. Available: https://support.skydio.com/hc/en-us/articles/360036102154-Access-Skydio-2-Manuals-and-Guides

[14] "Skydio R1 Overview." [Online]. Available: https://www.aniwaa.com/product/drones/skydio-r1/

[15] "Technical Specifications." [Online]. Available: https://www.parrot.com/us/drones/anafi/technical-specifications

[16] K. Team, "Simple. flexible. powerful." [Online]. Available: https://keras.io/

[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," *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, Nov 2019.