Deep Learning for Identification of RF Emitters Based on Transmitter Artifacts

Greg W. Zdor, Jordan Barker and Sean McCarty
Georgia Institute of Technology
{gzdor3, jbarker63, smccarty30}@gatech.edu

Abstract

One of the key performance indicators for 5G wireless systems defined by the International Telecommunication Union is the ability to support up to a million Internet of Things (IoT) devices per square kilometer [1]. This is a challenging goal both from a cellular network provider and IoT device manufacturer perspective given the need to minimize IoT device Size, Weight, and Power (SWAP) [2].

Therefore, the need to enable the secure operation of large scale IoT device networks has motivated the development of radio frequency fingerprinting (RFF) solutions. RFF is defined as the identification (classification) of a transmitter (TX) based off its emitted signal, and properties unique to that given transmitter and transmission (e.g., TX hardware artifacts). In this report, each project team member summarizes their development of an RFF deep learning solution. This includes a discussion of our observation that RFF algorithm generalization depends on accounting for RF channel variability. Results show deep learning learns RFF data well but has difficulty generalizing to untrained radio frequency (RF) propagation channels.

1. Introduction / Background / Motivation

The goal of our project is to evaluate several different deep learning architectures to address the need for robust IoT device authentication via RF fingerprinting. A 2019 IEEE journal article "IoT Device Security Using RF Fingerprinting" cautioned that "the computational complexity of cryptographic protocols and scalability problems make almost all cryptography-based authentication protocols impractical for IoT" [3]. Also, "Ghost-in-ZigBee: Energy Depletion Attack on ZigBee-Based Wireless Networks" discusses the ramifications of an adversarial attack that aims to reduce a ZigBee device's battery life via "luring a node to do superfluous security-related computations" [4], pointing to the need for improved IoT device authentication methods.

The U.S. Department of Homeland Security Cybersecurity and Infrastructure "The Internet of Things: Impact on Public Safety Communications" March 2019 white paper identified several anticipated IoT benefits included simplifying the response to traffic accidents. However, capitalizing on these benefits requires addressing the need for robust IoT device authentication, for which RFF holds promise, as evidenced by the Defense Advanced Research Projects Agency (DARPA)'s investment in its Radio Frequency Machine Learning Systems (RFMLS) program [5] [6].

1.1. ORACLE RF Fingerprinting Dataset

The dataset that we selected for our project was released in conjunction with Northeastern University GENESYS Laboratory's 2018 IEEE INFOCOM journal article "ORACLE: Optimized Radio clAssification through Convolutional Neural nEtworks" [7].

This dataset includes twenty million samples each for sixteen Ettus Universal Software Defined Radios (USRP) recorded 802.11a (Wi-Fi) protocol signals. The raw data for each radio is stored as complex valued numbers. This data format is typically described as in-phase (real) and quadrature phase (imaginary) data, or IQ data. Orthogonal Frequency Division Multiplexing (OFDM) is the underlying modulation scheme for this dataset. Modulation is the process of modifying one or more of an RF signal's characteristics (i.e., phase, frequency, amplitude) to convey information. OFDM transmits information on multiple frequencies in parallel at a slower rate. The benefit of this approach is that this modulation scheme is more resilient to multipath interference. Multipath interference is a mobile communications channel impairment that occurs due to multiple copies of a signal arriving at a receiver that are caused by reflections off nearby objects.

2. Approach

Northeastern University's approach for applying deep learning to their dataset was based on a neural network design that operated on 128 samples of I / Q pairs. Following their example, we treated the in phase and quadrature components of the raw RF signal data as two real-valued channels, resulting in a single training sample consisting of a 2x128 tensor, with dimensions interpreted as [number of features, time samples].

Each member of our team implemented their deep learning algorithms using PyTorch Lightning. Our rationale for this approach is that it abstracts a significant fraction of the details associated with model training and evaluation. The benefit of these features is that this enables a deep learning algorithm researcher to focus on algorithm development.

Our baseline model is a replication of the ORACLE model within the PyTorch framework. This model consists of two 1D convolution layers at the beginning that have 50 filters each and a kernel size of seven. The first convolution expands the two input channels to 50 channels while decreasing the length from 128 to 122. The output length L_{out} of each convolution operation is given by (1):

$$L_{out} = \frac{L_{in} - K + 2*P}{S} + 1 \qquad (1)$$

Where L_{in} is the input sequence length to the convolution layer, K is kernel size, S is kernel stride, and P is padding. ORACLE's convolution layers are followed by three fully connected layers. Rectified Linear Units (ReLUs) are used between each of the layers and dropout is applied at a rate of 50% between each of the dense layers. A softmax classifier is used at the end of the network to output prediction probabilities. This baseline model contained 1,525,212 parameters.

Implementing the data loader and augmentations typically is a significant fraction of the effort required to implement a deep learning algorithm. The data loader approach used by Sean McCarty and Greg Zdor was based on the 2014 IEEE journal article "MMap: Fast billion-scale graph computation on a PC via memory mapping" that described collaborative Georgia Institute of Technology and Korea Advanced Institute of Science and Technology (KAIST) research [8]. The data loader returned batches of IQ samples with an average amplitude of 1, ensuring consistent input value ranges into the model. Beyond this, the data loader work included implementing two RF signal augmentation algorithms. However, Jordan Barker did not incorporate this data loader / data augmentations due to software integration issues on his computer that proved too time consuming to resolve.

Data augmentation for RF signal data requires a different approach than what is typically applied to image data. For this project we evaluated two different data augmentations. First, we added random complex Gaussian noise and shifted the phase of the I/Q signal data blocks. Our rationale for this initial set of data augmentations is based on Sean McCarty's 26-year experience as a professional electrical engineer. Second, Sean McCarty implemented an RF channel effects data augmentation that applied a random Center Frequency Offset (CFO) and a multipath channel model to the training data. He expected that this revised data augmentation would improve poor test dataset generalization performance based on research described in a 2021 IEEE journal article "Wireless Fingerprinting via Deep Learning: The Impact of Confounding Factors" [9]. In the next three subsections of our project report each team member describes their deep learning algorithm approach.

2.1. Greg Zdor's Approach

Greg Zdor's approach began with motivation from the baseline ORACLE researcher's CNN network, coupled with insights from O'Shea et. al's "Over the Air Deep Learning Based Radio Signal Classification" paper [10]. O'Shea's work points to the success of 1D convolutional layers as feature extractors for modulation classification, a task like RFF in input data type and similarly plagued with the unique difficulty of RF channel effects. All data used was the run1 for train and validation and run2 for test, from the 14 ft. and 20 ft. measurements.

Following the design principles from discussion in, Greg Zdor constructed a 1D CNN using filter size of 3 and max pooling kernel size of 2 [10] . The first layer consisted of 128 filters, followed by a batch normalization layer, ReLU activation, and lastly max pooling. Layer two was identical except for using 64 filters. Two dense layers of hidden sizes 1024 and 256 followed by a softmax output completed the initial network.

Initial training results showed this model learned the training set well, which confirmed initial design decisions. To maximize this model's performance, hyper parameter tuning in the form of a variable number of convolution and dense layers architecture was developed. Specifically, Ray Tune, a distributed compute and hyper parameter tuning package, was utilized to develop a search space of model and training parameters to search across.

Key parameters in this search space included number of convolution layers, varying 1 to 5, number of dense layers, varying 2 to 5, number of convolution filter, dense layer hidden sizes, learning rate, momentum, and a dropout layer was added after each dense layer, with varying dropout rates ranging from 0.1 to 0.7. Model architecture tuning then consisted of running 250 training iterations or trials, where for each, Ray Tune sampled the search space and drew a set of architecture and training parameters, instantiated, and trained the model. Validation loss was set as the metric to optimize, and Ray Tune's built-in implementation of the Hyper Opt Search parameter space optimization algorithm was used as the search space optimizer [11]. Tuning was executed on a NVIDIA DGX with 8 Tesla V100-SXM2 GPUs, which enabled parallel running of tuning iterations using Ray Tune's built-in implementation of the Async Hyper Band Scheduler algorithm, a highly performant scheduler algorithm [12]. Developing this training framework and integrating PyTorch Lightning with Ray Tune proved an extensive task which took the remainder of Greg Zdor's project time. The final tuning approach can be visualized below as passing a trainable model plus search space parameters and dataset into Ray Tune, which then schedules, runs, and optimizes the input algorithm parameters.

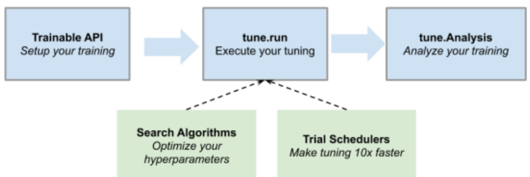

Figure 1. Ray Tune hyper parameter tuning flow diagram. Image source.

2.2. Jordan Barker's Approach

Using the ORACLE architecture as a starting point, several modifications were explored and implemented to improve upon its design. The first is changing the 1D convolution layers such that the first convolution outputs 32 filters and the second convolution outputs 64 layers. Increasing the number of convolution filters improved performance empirically.

Using the ORACLE architecture as a starting point, several modifications were explored and implemented to improve upon its design.

The first is changing the 1D convolution layers such that the first convolution outputs 32 filters and the second convolution outputs 64 layers. Increasing the number of convolution filters improved performance empirically. Kernel sizes of 3, 5, 7, and 9 were considered for this model. Kernel size of 3 is a common choice which has been shown to be faster and give better performance in some contexts [13]. For this use case, Jordan's analysis is consistent with the ORACLE results that suggests that a kernel size of 7 yields the best performance. Batch normalization was added between layers to improve training. Leaky ReLUs were used in place of ORACLE's ReLUs to give a small gradient when the unit is negative. Lastly, the size of the dense layers was increased so that the model had more space to learn features.

PyTorch contains several pretrained models that have a proven track record on the task of image classification. EfficientNet-B0 and ResNet-50 were chosen since they offered a good trade-off between accuracy and parameter size [14] [15].

Several changes were required starting with the input data to make them work for our case. The input data is reshaped from 2x128 to shape 2x1x128. This gives it the same number of dimensions as an image, except the models expect colored images that contain three channels in the second dimension. A 2D convolution with three kernels of size one was added at the beginning of the network to accomplish this data transformation. Next, a 2D convolution with three kernels of size one outputs a new tensor of shape 3x1x128. After the convolution operation, average pooling is performed over the tensor to get a new tensor of shape 3x128x128. This tensor is then fed through the existing model architecture, and the last linear layer was modified from having 1000 outputs to use 16 outputs for each of our device classes.

Ideally, we would be able to freeze the pretrained weights and avoid recomputing them while just training the new input layers and output layer. In practice, the model was unable to learn effectively with the pretrained weights so ultimately everything needed to be retrained.

2.3. Sean McCarty's Approach

Sean McCarty's Spring 2023 Data & Visual Analytics group project involved the fusion of several algorithms to detect misinformation. His contribution to this project was implementing a Residual network to analyze image encoded text. He based his solution on the April 2022 "Implementing ResNet18 for Image Classification" Kaggle tutorial authored by Srindhi. His rationale for choosing this approach was two-fold. First, historically residual networks have achieved good ImageNet benchmark performance. Second, several UCLA researchers recently published RF fingerprinting research based on residual networks [16].

Sean McCarty also implemented a network that combined residual blocks and a Long-Term Short-Term (LSTM) residual network. His rationale for this approach is that feature maps generated by a residual block can be interpreted as embeddings typically processed by an LSTM block. However, this approach requires transpositions of the LSTM input and output. Also, a recent Association of Computing Machinery (ACM) article "CLD-Net: A Network Combining CNN and LSTM for Internet Encrypted Traffic Classification" describes the application of this combined deep learning architecture to cybersecurity [17].

3. Experiments and Results

The ORACLE dataset includes measurements across eleven distances for sixteen devices. For our project we trained our deep learning algorithms using 80% of the data from the 14 and 20 ft. distance run1 measurements. Appendix A provides plots of example training data for both ranges of consideration, along with 2 ft. and 56 ft. Note in the selected ranges, the signal is noticeably above the noise floor, thus at a positive signal to noise ratio (SNR). We tested our algorithms' ability to generalize using the corresponding run2 measurements. Unfortunately, we did not observe good generalization performance. However, that is a typical problem encountered by RFF algorithm developers. A potential next step to address this issue would likely involve transitioning to a complex-valued deep learning architecture paired with more robust channel effects modeling. For example, this approach is discussed in "ChaRRNets: Channel Robust Representation Networks for RF Fingerprinting" that was recently published by several Expedition Technology researchers [18] [19].

All our neural networks use multiclass accuracy as the performance evaluation metric, defined as:

$$Accuracy = \frac{1}{N}\sum_i^N 1(y_i = \hat{y}_i) \qquad (2)$$

Where y_i is the predicted label, y_i hat is the truth label and N is number of samples in a batch. Given RF fingerprinting is a classification task, categorical cross entropy was used as the loss function to optimize, and it may be defined as:

$$Categorical\ Cross\ Entropy = -\sum_i^N \log(y_i) \qquad (3)$$

Where y_i is the softmax probability for the truth label index.

We measured the success of each algorithm based on its validation and test data performance. This included evaluating our algorithm's accuracy and loss learning curves, classification reports, and confusion matrices. Our analysis suggests that we succeeded in implementing algorithms that obtained good training and validation dataset performance. However, our analysis suggests additional effort is required to ensure our algorithms are more robust to RF channel variations.

3.1. Greg Zdor's Experiments & Results

All training leveraged the 14 ft. and 20 ft. dataset using random complex Gaussian noise and phase shift I/Q data augmentation with a batch size of 512. For the initial baseline model, it was trained for 25 epochs, cross entropy loss was the loss function, and stochastic gradient descent (SGD) at its default learning rate and momentum settings was the optimizer. Model converged to 92 % training accuracy, 91 % validation accuracy, yet when evaluated on the test set yielded only 19 % test accuracy.

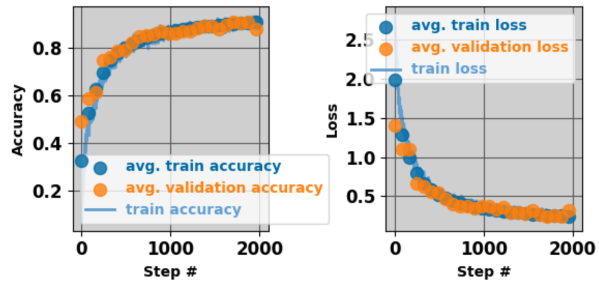

Figure 2. Initial CNN training curves.

Both training and validation loss and accuracies track each other closely, and do not diverge noticeably, indicative of model generalizing to the training set. The poor test performance is explained by the test set consisting of channel effects that are not present in the train data.

The second phase of Greg Zdor's experimentation was using Ray Tune to hyper parameter the above-described network. Training consisted of 250 tuning trials, each training up to 25 epochs. Since the Cornell-developed Asynchronous Successive Halving Algorithm scheduler terminates training instances after a given grace period, this grace period was set strictly to only 1 epoch, to facilitate only training models where validation loss continually decreased [12].

As Figure 3 shows, the top performing model did not achieve an increase in validation accuracy, making it to 91%, however, *how long* it took the model to get there was dramatically reduced, only training to 2 epochs before early stopping kicked in, compared to the default parameter values requiring 25 epochs to converge to 92 validation accuracy.

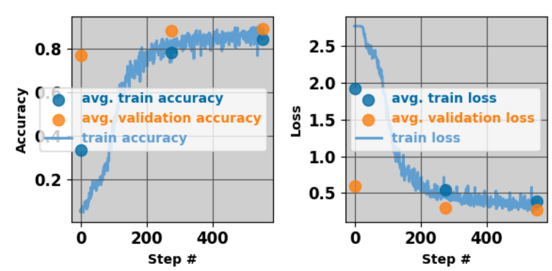

Figure 3. Top CNN hyperparameter tuning.

```
Layer (type)                  Output Shape         Param #
================================================================
BatchNorm1d-1                 [-1, 2, 128]               4
Conv1d-2                      [-1, 67, 126]            469
MaxPool1d-3                   [-1, 67, 124]              0
Conv1d-4                      [-1, 81, 122]         16,362
MaxPool1d-5                   [-1, 81, 120]              0
Linear-6                      [-1, 428]          4,160,588
Dropout-7                     [-1, 428]                  0
Linear-8                      [-1, 366]            157,014
Dropout-9                     [-1, 366]                  0
Linear-10                     [-1, 234]             85,878
Dropout-11                    [-1, 234]                  0
Linear-12                     [-1, 16]               3,760
================================================================
Total params: 4,424,075
```

Figure 4. Top Performing Architecture

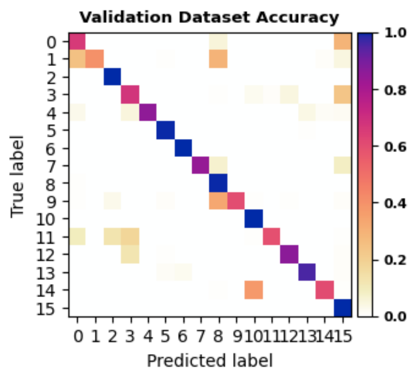

Figure 5. Hyperparameter Tuning

Notable differences from the baseline CNN and the hyperparameter tuning selected version (HPTS) include the first convolution layer has only 67 filters instead of the default 128, and instead of having only 2 dense layers, 4 dense layers proved most performant. Figure 5 illustrates that the top HPTS model's accuracies per class and confirms observed 91% average validation accuracy observed.

Beyond model and training hyperparameters, data parameters, namely the window size l or number of time-samples per input vector to the model, was varied. The starting initial architecture was used in both cases, the first using l = 128, and second at l = 1280. The l = 1280 samples showed a slight increase in train and validation accuracy at 96% and 93% respectively, but evaluation yielded only 18% test accuracy, again pointing to the lack of generalization across RF channels. O'Shea et al found a roughly 3% increase in performance for each doubling of input size, starting at l = *16* up to l = 512, above performance plateaus [10]. However, O'Shea's dataset included simulated multipath channel affects, allowing model generalization across channels, a feature not present in the ORACLE dataset [10].

Training at l = 1280 took only 3 epochs before early stopping kicked in, while the learning rate and batch size was constant in both cases at 1e-3 and 512 respectively. The 10x larger input vector size acted as increasing the batch size by 10x or reducing the number of steps (a step is a single forward and backwards pass on a minibatch) per epoch by 10x. One reason for the slightly higher validation accuracy at l = *1280* could be at a 10x larger batch size, each minibatch is more representative of the whole dataset, enabling less variability in and faster convergence for the learning curves. While a larger learning rate may help a larger batch size converge faster, exploring that relationship was beyond the scope of this experiment.

3.2. Jordan Barker's Experiments & Results

Several combinations of hyperparameters were tested including dropout percentage, learning rate, batch size, and regularization amount. Figure 6 shows the results of these tests on the CNN classifier.

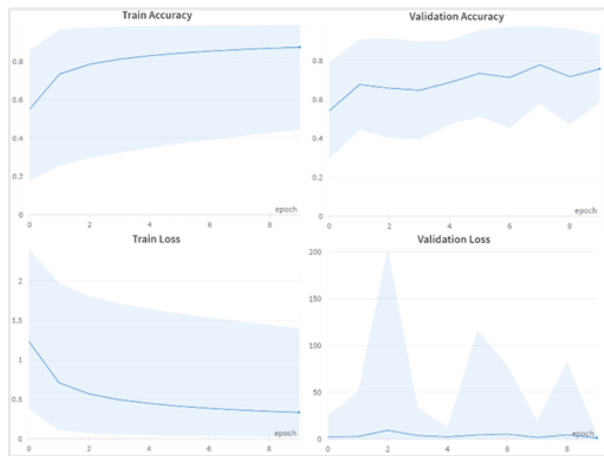

Figure 6. CNN Hyperparameter Optimization

Many of the configurations were found to overfit by achieving a high training set performance yet failing to generalize to the validation set. The final model chosen was one that improved both metrics slowly and demonstrated stable learning. Once trained, this model achieved a 95% train and validation accuracy.

The transfer learning models, EfficientNet-B0 and ResNet-50, demonstrated superior performance over the CNN classifier on the validation dataset, with improvements of approximately 3% and 4% respectively. This performance boost can likely be attributed to their larger sizes. In comparison to the CNN classifier, EfficientNet-B0 is roughly four times larger, while ResNet-50 is a substantial 23 times larger. However, it is worth noting that the growth in model size does not correspond proportionally to the observed increase in performance.

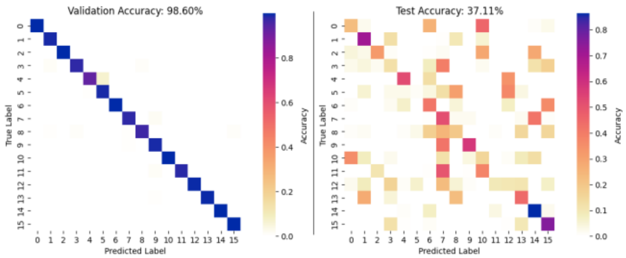

Figure 7. ResNet-50's test and validation accuracy.

3.3. Sean McCarty's Experiments & Results

Sean McCarty's initial algorithm experiments focused on analyzing data from the 2 ft. and 8 ft. measurements. Based on team discussions he transitioned to analyzing data from the 14 ft. and 20 ft. measurements. The best performance in terms of test set generalization (~32% accuracy) was for a combined LSTM / CNN neural network that was trained using random complex Gaussian noise and phase shift I/Q data augmentation.

We present the learning curves for this model in Figure 8. These results suggest this model is not overfitting on the training data since there is close training and validation dataset accuracy & loss agreement.

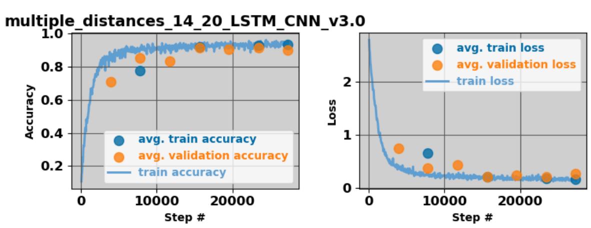

Figure 8. LSTM + CNN learning curves (I/Q data augmentation).

We show the corresponding validation dataset confusion matrix in Figure 9. This result is consistent with the average 90% accuracy estimated by the Python scikit-learn library's classification report function. However, we also illustrate in Figure 9 that additional effort is required to ensure that our models are insensitive to RF channel variability [20].

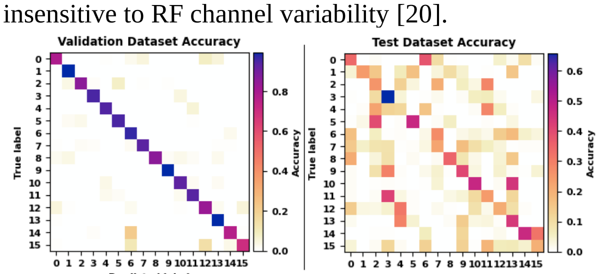

Figure 9. LSTM + CNN validation and test dataset accuracy (I/Q data augmentation).

4. Conclusion and Results Discussion

Results showed iterating upon our neural network designs in the form hyper parameter tuning via Ray Tune, transfer learning and adding temporal learning layers like an LSTM, all yielded higher validation accuracy than the ORACLE baseline model architecture. In this way the project was a success, yet still to be resolved is the problem of generalizing to unseen RF channels – a task likely solvable given more time and the implementation of a Rician distribution-based fading channel model in the input data loader augmenter.

Metric	ORACLE Baseline	Greg Zdor	Jordan Barker	Sean McCarty
Train accuracy	0.77	0.96	0.98	0.94
Validate accuracy	0.89	0.93	0.98	0.90
Test accuracy	0.27	0.27	0.38	0.32
Train loss	0.61	0.13	0.06	0.16
Validate loss	0.31	0.16	0.04	0.27

Table 1. Performance comparison of best performing models for the 3 approaches.

Each of our models incorporated convolutional layers due to their ability to detect local patterns as well as spatial hierarchies. They are also invariant to translation which we hoped would help them generalize to this classification problem, but there remains much to be improved. Recent research has shown that the transformer architecture can be adapted to image classification problems using the Vision Transformer architecture to achieve state of the art performance. Future work could explore adapting this architecture to the task of RF Fingerprinting to improve the generalization to unseen RF multipath channels problem.

References

[1] Afif Osseiran, Stefan Parkvall, Patrik Persson, Ali Zaidi, Sverker Magnusson and Kumar Balachandran. "5G wireless access: an overview". Ericsson White Paper 1/28423-FGB1010937, April 2020.

[2] Curtis Watson, PhD, Kelvin Woods, PhD, and DJ Shyy, PhD, "TW: 6G and Artificial Intelligence & Machine Learning", The MITRE Corporation, Bedford, MA, USA. 2021.

[3] Douae Nouichi, Mohamed Abdelsalam, Qassim Nasir, and Sohail Abbas. "IoT Devices Security Using RF Fingerprinting," 2019 Advances in Science and Engineering Technology International Conferences (ASET), Dubai, United Arab Emirates, 2019, pp. 1-7.

[4] Xianghui Cao, Devu Manikantan Shila, Yu Cheng, Zequ Yang, Yang Zhou and Jiming Chen. "Ghost-in-ZigBee: Energy Depletion Attack on ZigBee-Based Wireless

Networks," in IEEE Internet of Things Journal, vol. 3, no. 5, pp. 816-829, Oct. 2016.

[5] U.S. Department of Homeland Security Cybersecurity and Infrastructure and Security Agency. “The Internal of Things: Impact on Public Safety Communications”, March 2019. https://www.cisa.gov/sites/default/files/2023-02/CISA%20IoT%20White%20Paper_3.6.19%20-%20FINAL.pdf

[6] John, Davies, “DARPA RFMLS Program Summary” https://www.darpa.mil/program/radio-frequency-machine-learning-systems

[7] Kunal Sankhe, Mauro Belgiovine, Fan Zhou, Shamnaz Riyaz, Stratis Ioannidis, and Kaushik Chowdhury. 2019. “ORACLE: Optimized Radio clAssification through Convolutional neural nEtworks. In IEEE INFOCOM 2019 - IEEE Conference on Computer Communications”. IEEE Press, 370–378.

[8] Zhiyuan Lin, Minsuk Kahng, Kaeser Md. Sabrin, Duen Horng Polo Chau, Ho Lee, and U Kang, "MMap: Fast billion-scale graph computation on a PC via memory mapping," 2014 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 2014, pp. 159-164.

[9] Metehan Cekic, Soorya Gopalakrishnan, and Upamanyu Madhow. “Wireless Fingerprinting via Deep Learning: The Impact of Confounding Factors," 2021 55th Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 2021, pp. 677-684.

[10] Timothy James O’Shea, Tamoghna Roy, and T. Charles Clancy, "Over-the-Air Deep Learning Based Radio Signal Classification," in IEEE Journal of Selected Topics in Signal Processing, vol. 12, no. 1, pp. 168-179, Feb. 2018.

[11] James Bergstra, Remi Bardenet, Yoshua Bengio, Balazs Kegl. “Algorithms for Hyper-Parameter Optimization”. https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

[12] Liam Li and Kevin Jamieson and Afshin Rostamizadeh and Ekaterina Gonina and Jonathan Ben-tzur and Moritz Hardt and Benjamin Recht and Ameet Talwalkar, “A System for Massively Parallel Hyperparameter Tuning,” Third Conference on Systems and Machine Learning, 2020.

[13] Zhang, Aston and Lipton, Zachary C. and Li, Mu and Smola, Alexander J. “Dive into Deep Learning”, arXiv preprint arXiv:2106.11342, 2021.

[14] Mingxing Tan and Quoc V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”, arXiv. 1905.11946. 2020.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778.

[16] Samer Hanna, Samurdhi Karunaratne, and Danijela Cabric, "Open Set Wireless Transmitter Authorization: Deep Learning Approaches and Dataset Considerations," in IEEE Transactions on Cognitive Communications and Networking, vol. 7, no. 1, pp. 59-72, March 2021.

[17] Hu, Xinyi and Gu, Chunxiang and Wei, Fushan and Chen, Chi-Hua, “CLD-Net: A Network Combining CNN and LSTM for Internet Encrypted Traffic Classification”, Sec. and Commun. Netw, January 2021.

[18] Carter N. Brown and Enrico Mattei and Andrew Draganov, “ChaRRNets: Channel Robust Representation Networks for RF Fingerprinting”, arXiv, 2021.

[19] Enrico Mattei and Greg Harrison, “Machine learning for wireless spectrum awareness”, Expedition Technology, Inc. 2018.https://www.exptechinc.com/wp-content/uploads/2018/10/mattei_harrison_ml_for_wireless_spectrum_awareness.pdf

[20] Kunal Sankhe, Mauro Belgiovine, Fan Zhou, Luca Angioloni, Frank Restuccia, Salvatore D’Oro, Tommaso Melodia, Stratis Ioannidis, and Kaushik Chowdhury. "No Radio Left Behind: Radio Fingerprinting Through Deep Learning of Physical-Layer Hardware Impairments," in IEEE Transactions on Cognitive Communications and Networking, vol. 6, no. 1, pp. 165-178, March 2020.

5. Appendix

During the preparation of our project proposal each team member performed a literature search that facilitated the development of our approach. This included summarizing potential datasets that the team reviewed and down selected during one of its periodic meetings. Also, all team members implemented their own PyTorch Lightning model definitions, trainers, and evaluation classes. We summarize individual team member contributions in Table 2.

Table 2 Team Member Contributions

Team Member	Contributions
Greg W. Zdor	• Initial data exploration • Implemented Ray Tune hyperparameter tuning • Implemented CNN and Residual Networks & experimented with larger network input sequence sizes • Wrote Appendix section and part of Approach and Dataset sections for final report
Jordan Barker	• Proposed working with the ORACLE dataset • Implemented ORACLE baseline model • Implemented transfer learning models • Implemented vanilla data loader
Sean McCarty	• Implemented data loader and data augmentation Python code. • Implemented a Residual CNN & LSTN + Residual CNN networks • Wrote final report introduction section

PyTorch Lightning was used as the foundation for our dataset, model definitions, trainer, and evaluation classes, due to its benefits of abstracting away boilerplate and enabling standardization across experiments. Code was developed and stored in GitHub locations that are identified in Table 3 and Table 4. There is a README.md at Greg Zdor's repository with instructions on installing Python dependencies, activating conda environment, running training, evaluation, and more.

Table 3 Project Code Repositories

Student	GitHub Link
Jordan Barker, Sean McCarty	https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/
Greg Zdor	https://github.com/gzdor/Radio_Frequency_Fingerprinting_802.11

Table 4 Key Python Software Modules

Student	Python Module	GitHub Link
Greg Zdor	Neural net definitions	https://github.com/gzdor/Radio_Frequency_Fingerprinting_802.11/blob/main/src/pkgs/ml/neural_net_definitions.py
	Training	https://github.com/gzdor/Radio_Frequency_Fingerprinting_802.11/blob/main/src/tools/ml_train/model_trainer.py
	Evaluation	https://github.com/gzdor/Radio_Frequency_Fingerprinting_802.11/blob/main/src/tools/ml_evaluation/model_evaluation.ipynb
Jordan Barker	Neural net definitions	https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/blob/main/Scripts/JJB_networks.py
	Training	https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/blob/main/Scripts/JJB_train.py
	Evaluation	https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/blob/main/Notebooks/JJB_model_eval.ipynb
Sean McCarty	Memory Mapper	https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/blob/main/Scripts/memory_mapper.py
	PyTorch Lightning Dataset Module	https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/blob/main/Scripts/sigmf_dataset.py
	Neural Network Definitions	• https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/blob/main/Scripts/residual_block.py • https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/blob/main/Scripts/residual_rff_network.py • https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/blob/main/Scripts/lstm_residual_rffnet.py
	Training	https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/blob/main/Notebooks/SPM_Train_LSTM_CNN_RFF_Model.ipynb
	Evaluation	https://github.gatech.edu/jbarker63/CS-7643-Deep-Learning-Group-Project/blob/main/Notebooks/SPM_Eval_LSTM_CNN_RFF_Model.ipynb

5.1. ORACLE Dataset Exploratory Data Analysis

We summarize the ORACLE RF fingerprinting dataset in Table 5 including the associated paper link, sampling rate, center frequency, and data format.

Table 5 ORACLE RF fingerprinting dataset

Dataset Producers	**Northeastern University Institute for the Wireless Internet of Things** **https://genesys-lab.org/**
Dataset Overview	**https://genesys-lab.org/oracle (Download links on this page)**
Associated Paper Link	**https://ieeexplore.ieee.org/document/8737463**
Data Size	• **Sixteen classes (16 different USRP X310 Software Defined Radios)** • **Twenty million samples / class**
Sampling Rate	**Five Million Samples per Second (MSPS)**
Center Frequency	**2.45 GHz**
Data Format	• **Raw data in binary file format (read as np.complex128)** • **Metadata (labels) in standard Signal Metadata (SigMF) format** **https://github.com/sigmf/SigMF**

Figure 10 and Figure 11 illustrate training samples from run1 with signal strength varying with range. For our analysis, we used samples from 14- & 20-feet Tx / Rx distance measurements for training, validation, and testing. These ranges were selected for several reasons, using the entirety of the dataset was too large given student compute resources, additionally, training samples were selected where the signal was still above the noise floor yet still at realistic ranges of a dozen or more feet.

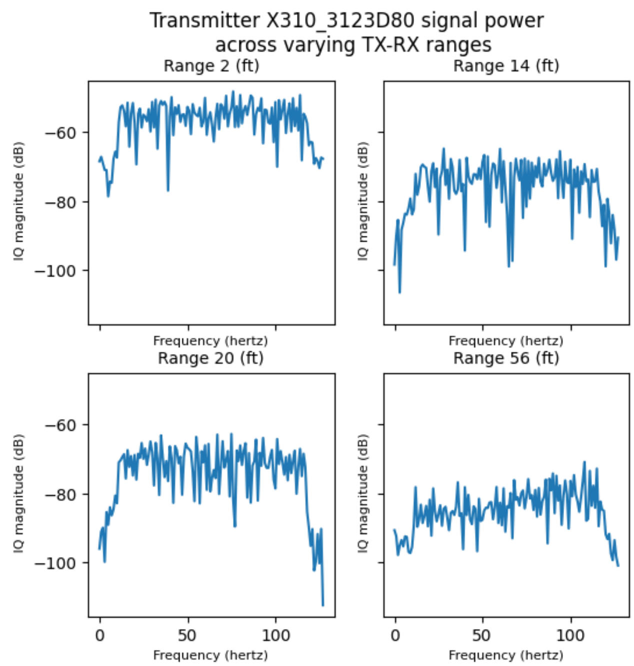

Figure 10 Frequency Domain Signal Power as a Function of Tx / Rx Distance

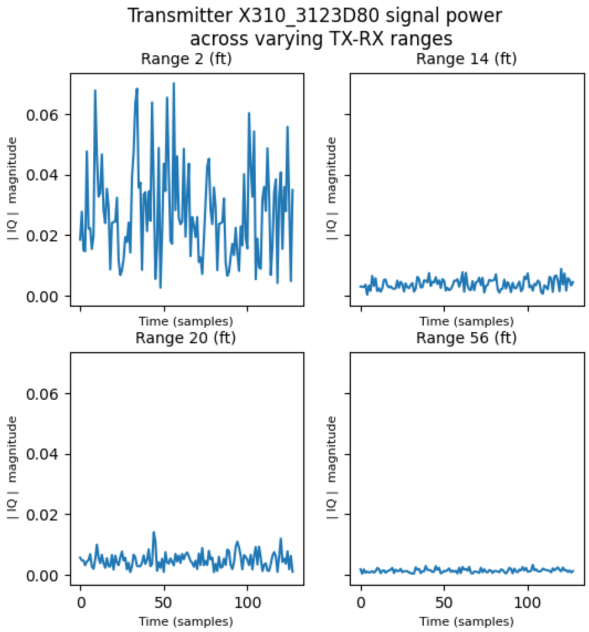

Figure 11 Time Domain Signal Magnitude as a Function of Tx / Rx Distance

We show in Table how signal attenuation varies as a function of Tx / Rx distance. In this table free space path loss is described using a logarithmic scale in decibels [dB]. For example, a 10 dB increase in free space path loss corresponds to an order of magnitude decrease in received power.

Table 6 ORACLE Dataset Signal Attenuation

Range (feet)	Free Space Path Loss (dB)
2 (minimum range in dataset)	35.9
14	52.8
20	55.9
62 (maximum range in dataset	65.7

Free space path loss (FSPL) in Table 6 above is the relationship quantifying the attenuation of a radio signal's energy across some range *D* at a given transmission frequency *f* or wavelength λ and may expressed in decibels (dB) as:

$$FSPL\ (dB) = \log_{10}\left(\frac{4\,\pi\,D}{\lambda}\right)^2 \qquad (4)$$

While FSPL assumes as obstacle free, line-of-sight (LOS) path between TX and RX in free space, usually air, and real-world conditions inherently include some channel conditions that add channel impairments to the signal. For the ORACLE dataset, researchers state there was a clear LOS path across air, therefore, looking at FSPL attenuation across varying ranges provides a close model of the signal attenuation in this dataset [7]. Using a recording center frequency of 2.45 GHz, the same that the ORACLE dataset was recorded at, is how the Table 6 values were calculated.