Applied Machine Learning, Fall 2021
Course Project

Team: Rich Seibert - rs2585, Larry Xu - qx36, Roberto Bruni - rb668

Abstract:

In this project we built a machine learning model capable of classifying various wireless signals. Unlike other papers that have been written on this subject, we apply real world signals that we found in the RF (radio frequency) space and that were collected off an SDR (software defined radio). Other papers often simulated data of simple and distinct modulation, which have no noise or other signals present in the data. In the real world, the RF space is often very crowded in certain bands and has many complexities, such as RF noise and interference such as multipathing. Our results show that we were able to classify signals with a 92.5% accuracy using a CNN.

Introduction:

With more and more wireless devices entering the market, the RF spectrum is becoming increasingly congested. An important issue in this field is knowing what and where devices are transmitting for various reasons. An example of this is in the airline/defence industry, where knowing what signals are transmitting near a certain location (such as an airport or military base) can identify problematic or adversarial devices such as drones.

Wireless signal detection is traditionally done with handcrafted algorithms in the signal processing and communications community. These algorithms often take advantage of well known properties of signals, such as center frequency of the transmission, bandwidth, and specific modulation. For example, for a certain FSK (frequency shift keying) signal coming from a RC (radio control) airplane, you could find that its signal is transmitted in a range from 910-920MHz with possible center frequencies every 1MHz, a bandwidth of 15KHz, and has a FSK sync pattern of 10111011. The traditional detection algorithm would probably do some sort of subband tuning and SNR (signal to noise ratio) search at each one of those frequencies, and if the SNR is above a threshold it would run a FSK demodulation and try to match the sync pattern. Although this is simple and fairly efficient for FSK signals, for more complex signals such as OFDM (orthogonal frequency-division multiplexing) this task becomes much more difficult.

What we did in this project was essentially automate this task using machine learning. Instead of having someone manually look at a signal and make a detection algorithm for it, you could feed a recording of a signal into the model to train on, and it would be able to classify it.

Background:

RF signals have many physical characteristics, bandwidth, center frequency, burst duration, etc. These features can often be seen by looking at the frequency and time domain. In addition to the parameters listed above, signals can have a vast amount of different modulation techniques and standards depending on requirements, such as range, data rate, power, interference resistance, etc. There are dozens, if not hundreds of these different techniques, and possibly thousands of different implementations of each one. Some of the most common modulation schemes seen today are AM/FM (amplitude/frequency modulation, used by radio stations), FSK (frequency shift keying, used by hobby RC controllers and other simple devices like car keys and garage openers), and OFDM (orthogonal frequency-division multiplexing - one of the most popular modulation methods used today: 3G, 4G, 5G, WiFi, and many others are all OFDM). The differences in these parameters can be utilized by an algorithm or machine learning model to uniquely identify signals.

Method:

We applied the following supervised learning models: SVM, KNN, random forests, and CNN. We found that the performance of CNNs were superior to the SVM, KNN and random forest models and therefore did most of our tuning for the CNN.

To collect the data, we used a RTL-SDR device which is a cheap software defined radio able to collect signals from 1 MHz to 1.2 GHz. We recorded various signals we found transmitting during the duration of the project. We collected 104 recordings each with a duration of 4 seconds, with a sample rate of 1 Msps and bandwidth of 1 MHz. Therefore, the number of data points we had for each collection was about 4 million samples. Each of these collections was cut up into many sections of around 4K (4096 samples, we experimented with different sizes) and labeled each of these 4K segments with the corresponding signal type. Each sample in the 4K array became a feature. Each 4K array became a data point with a label for the model, so in total we had around 100,000 (4e6 * 104 / 4K) data points for the model to work with.

There were 10 signal types (classes) in our data set, with at least 6 collections of the same signal type, and with at least 2 different center frequencies. This was to make sure we were collecting signals coming from different transmitters and that the model wasn't just classifying signals by looking at other things in the collection other than the signal of interest. This would be analogous to a situation where a classifier is made for identifying pictures of dogs and cats, but in all the dog training data there's an object in the background that the model is picking up on and using to classify dogs.

The 10 signal classes we trained the models on were: Tone, FM, 3G, ATCS, fast burst, SCADA, NBFM, IDEN, DDS, 4G [See appendix for more information on these signals]. We labeled these signals by looking at the spectrograms and cross referencing the signal frequency, bandwidth, and modulation type with a community run signal database called Signal ID Guide [5].

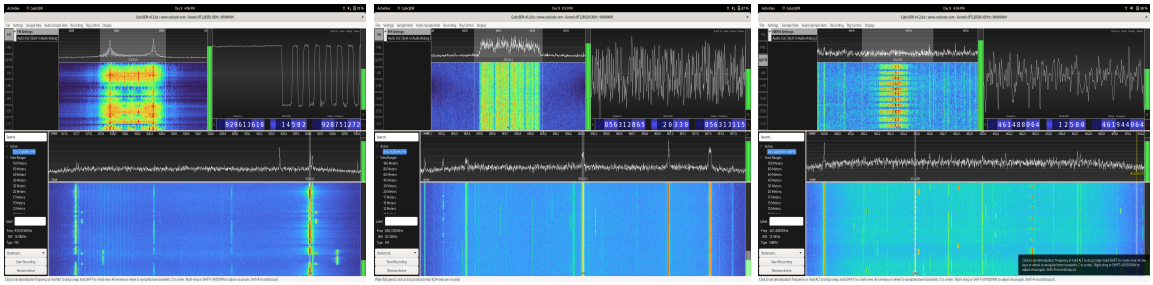


Figure 1: Images of CubicSDR software interface showing the following signals from left to right: SCADA, DDS, Fast Burst. The interface shows the waterfall spectrogram in the bottom half, in the top left the spectrogram with a y-cut, and the top right shows the demodulation

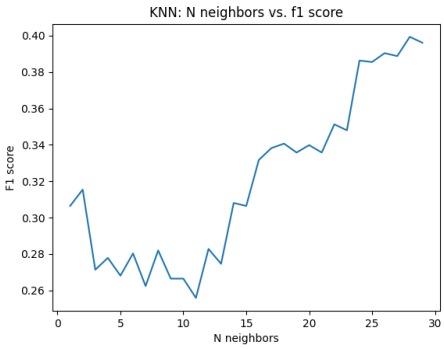
We essentially modeled the problem as an image classification problem, but instead of pixel intensities, you have IQ data (in-phase and quadrature) which is complex data. So instead of values for red, green, and blue, you have values for the real and imaginary components of the signals at each sample. We trained this model using the raw IQ data from the SDR only performing minimal preprocessing, which was taking the magnitude time domain data and normalizing it from 0 to 1 and completely ignoring the phase. We found that using the magnitude data seemed to give better results than the phase. We did not sub-band tune to the bandwidth of the signal, as we did not think that would be a fair assessment of the model's effectiveness.

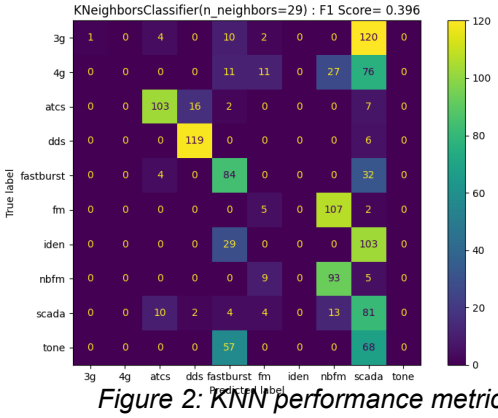
Experimental analysis:

As stated previously, we started off with a few simple models. The first method we tried in the milestone was SVM and we improved upon that model here.

During our experiments we tried these models with various input parameters. After we found the best input parameters, we generated a confusion matrix and classification report with said parameters. We trained them with a cut data set, only using 20 collections, 2 for each signal, and then cut that down 80%, so only about 4% of the total data set. This was because training with the full data set took too long. We evaluated the results by looking at F1 scores and confusion matrices. We felt that in real life scenarios, it would probably depend on the application of this project as to what metrics, like accuracy vs F1 score, would be more important. We chose to use precision, recall, and F1 score in this case because we cared about false negatives and false positives. If it were implemented in a real world situation, it would essentially be useless if there were false negatives or false positives. Saying this, because we had a very well balanced data set, the accuracy scores were very close to the F1 scores regardless. For our train test split, we used 70% of the data for training, 20% for dev set, and 10% for testing and we shuffled. Even though 30% is usually too much for the test/dev set, we were running with a small data set for these so we thought the split made sense here.

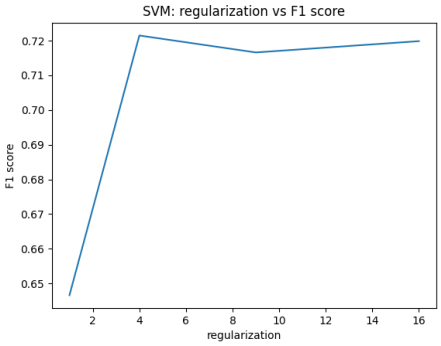
The following images are the results for the KNN model with varying n-neighbors (30 was the best), SVM with varying regularization (found a regularization of 4 with kernel RBF to be the best), and then random forests with varying tree size (tree size of 50, depth of 7 was the best):

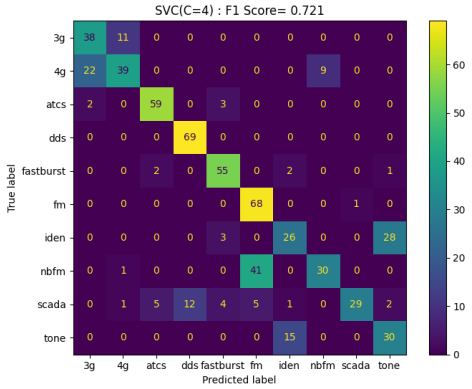




	precision	recall	f1-score	support
3g	1.00	0.01	0.01	137
4g	0.00	0.00	0.00	125
atcs	0.85	0.80	0.83	128
dds	0.87	0.95	0.91	125
fastburst	0.43	0.70	0.53	120
fm	0.16	0.04	0.07	114
iden	0.00	0.00	0.00	132
nbfm	0.39	0.87	0.54	107
scada	0.16	0.71	0.26	114
tone	0.00	0.00	0.00	125
accuracy			0.40	1227
macro avg	0.39	0.41	0.31	1227
weighted avg	0.39	0.40	0.31	1227

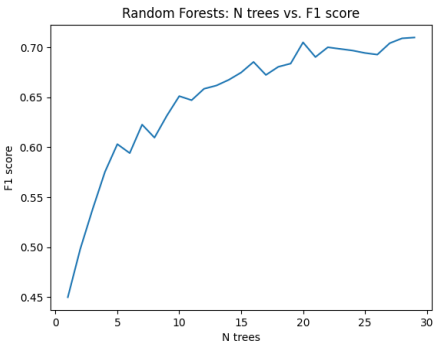
Figure 2: KNN performance metrics

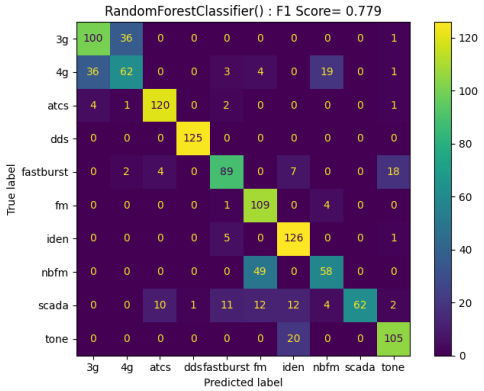




	precision	recall	f1-score	support
3g	0.61	0.78	0.68	49
4g	0.75	0.56	0.64	70
atcs	0.89	0.92	0.91	64
dds	0.85	1.00	0.92	69
fastburst	0.85	0.92	0.88	60
fm	0.60	0.99	0.74	69
iden	0.59	0.46	0.51	57
nbfm	0.77	0.42	0.54	72
scada	0.97	0.49	0.65	59
tone	0.49	0.67	0.57	45
accuracy			0.72	614
macro avg	0.74	0.72	0.70	614
weighted avg	0.75	0.72	0.71	614

Figure 3: SVM performance metrics





	precision	recall	f1-score	support
3g	0.71	0.73	0.72	137
4g	0.61	0.50	0.55	125
atcs	0.90	0.94	0.92	128
dds	0.99	1.00	1.00	125
fastburst	0.80	0.74	0.77	120
fm	0.63	0.96	0.76	114
iden	0.76	0.95	0.85	132
nbfm	0.68	0.54	0.60	107
scada	1.00	0.54	0.70	114
tone	0.81	0.84	0.83	125
accuracy			0.78	1227
macro avg	0.79	0.77	0.77	1227
weighted avg	0.79	0.78	0.77	1227

Figure 4: Random forests performance metrics

After Performing the tests above, we decided to try CNNs considering their renowned performance on image classification problems. We did label encoding and then one-hot-encoding on the data labels. For the train test split we used 80% for training and 10% for our dev and test set each and we shuffled. We first started with 2 3x3 convolution layers with ReLu activation, and 2 max pooling layers with a pool size of 2x2. After that, we had a flatten layer and two dense layers using ReLu and softmax activation. For training the model, we used

10 epochs and a small batch size of 8 due to memory limitations. For the first set of tests, we decided to use a very small subset of the available data, about 1%, to train faster.

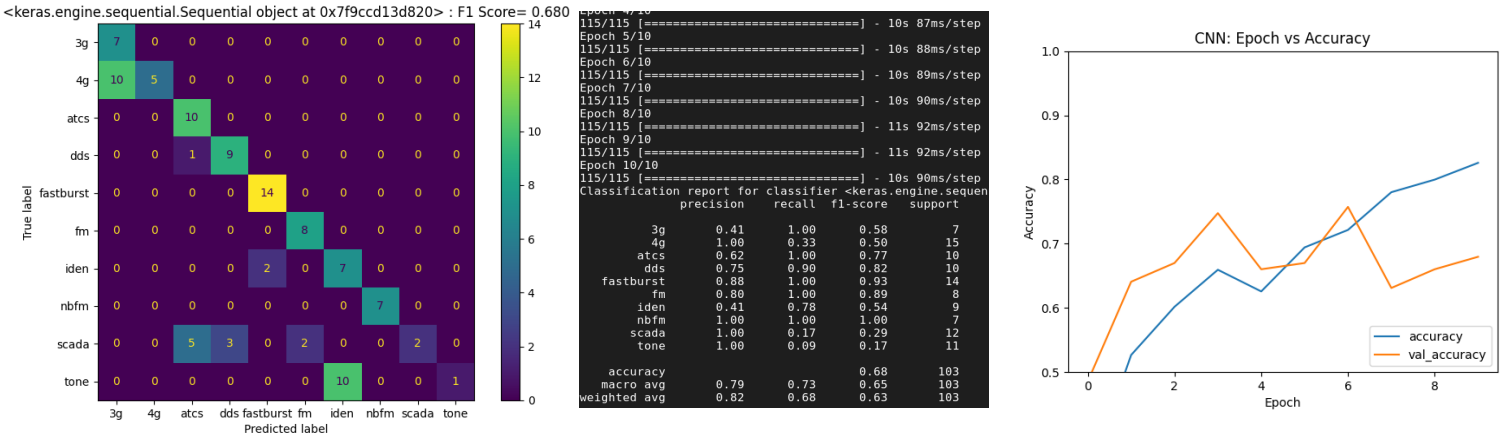


Figure 5: CNN performance metrics

For such a small amount of data,the CNN model did quite well. As we can see towards the end of the training, the validation accuracy and our dev set accuracy begin in split (we used accuracy here because it's easier to calculate with the keras API and there was virtually no difference with our data set because it was balanced) . So we can see we have a high variance issue here and we are overfitting the training data. To fix this, we applied regularization using a l1_l2 kernel regularizer with l1=1e-5 and l2=1e-4. After testing with regularization, we also added a dropout set with a rate of 0.5.

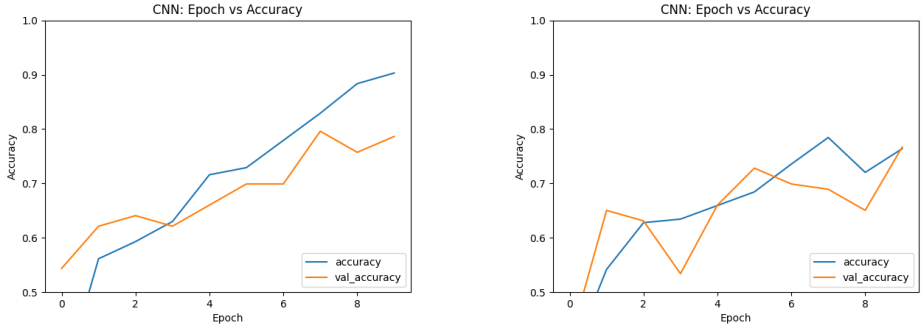
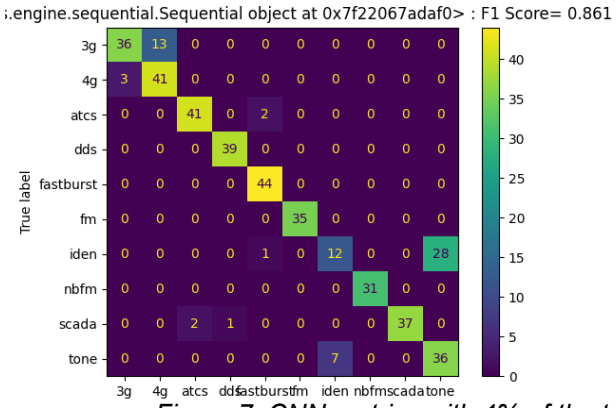


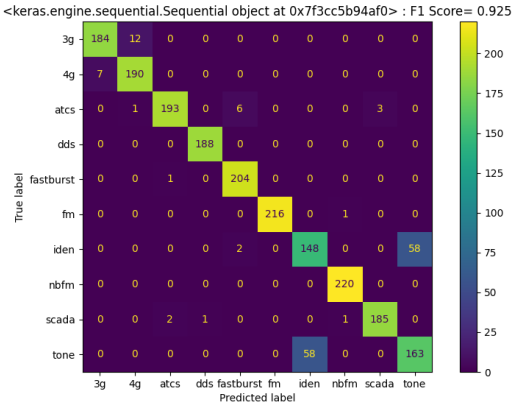
Figure 6: Epoch vs accuracy plots with regularization (left) and regularization+dropout (right)

We can see that after adding both regularization and dropout, our training results are closely matching our dev set results. For our final test, we ran the CNN on the full cut data set that we were using with the previously tested models, or about 4% of the total data. Using more data along with the regularization and dropout increased our F1 score significantly from 0.68 to 0.86. Two classes but we can see in the confusion matrix that 'iden' and 'tone' are being misclassified. We tried to solve this issue by using more data, going from using about 4% of the total dataset to 20%.



	precision	recall	f1-score	support
3g	0.92	0.73	0.82	49
4g	0.76	0.93	0.84	44
atcs	0.95	0.95	0.95	43
dds	0.97	1.00	0.99	39
fastburst	0.94	1.00	0.97	44
fm	1.00	1.00	1.00	35
iden	0.63	0.29	0.40	41
nbfm	1.00	1.00	1.00	31
scada	1.00	0.93	0.96	40
tone	0.56	0.84	0.67	43
accuracy			0.86	409
macro avg	0.87	0.87	0.86	409
weighted avg	0.87	0.86	0.85	409

Figure 7: CNN metrics with 4% of the total data set, iden and tone are being misclassified



	precision	recall	f1-score	support
3g	0.96	0.94	0.95	196
4g	0.94	0.96	0.95	197
atcs	0.98	0.95	0.97	203
dds	0.99	1.00	1.00	188
fastburst	0.96	1.00	0.98	205
fm	1.00	1.00	1.00	217
iden	0.72	0.71	0.71	208
nbfm	0.99	1.00	1.00	220
scada	0.98	0.98	0.98	189
tone	0.74	0.74	0.74	221
accuracy			0.93	2044
macro avg	0.93	0.93	0.93	2044
weighted avg	0.93	0.93	0.92	2044

Figure 8: CNN metrics with 20% of the total data set, 92.5% accuracy

Discussion and Prior Work:

Deepsig.ai [4] is a group that makes products and writes papers on this subject. When researching the data they provide, we found they only have simulated signals. Other papers have the same issues where they use simulated data, such as *Deep Learning for RF Signal Classification in Unknown and Dynamic Spectrum Environments* [6] and *Over the Air Deep Learning Based Radio Signal Classification* [7]. This project takes real world data from a SDR and applies machine learning to it, which has not been done with such a small budget. Although these other papers received slightly higher accuracy, we think our results are extremely good.

Conclusion:

We can see that RF signal classification is definitely a plausible problem to solve with machine learning. Using CNN models we got excellent results with real world data. Some open directions for this could be SNR (signal-to-noise ratio) testing to see how sensitive the model is. In addition, it would be interesting to see how improvements could be made, such as instead of just training on time magnitude data we also train on phase or FFT frequency domain data.

Appendix:

Signal's classified:

Signal Name	Description	Frequency Range	Modulation	Bandwidth
Supervisory Control And Data Acquisition (SCADA)	Supervisory Control And Data Acquisition (SCADA) is a control system architecture that is used in industrial applications for computerized automated systems. Wireless telemetry is used on RTU's to send data to control units for operators to utilize.	413 MHz — 950 MHz	FSK	12 KHz
Integrated Digital Enhanced Network (iDEN)	iDEN is a TDMA-based digital wireless standard developed by Motorola. It is a type of trunked radio with cellular phone benefits.	806 MHz — 869 MHz	QAM, TDMA	18.5 kHz
Tone	Just a tone. Found throughout the spectrum			
4G LTE Network	Long Term Evolution Network. Also known as 4G LTE Data and Evolved Universal Terrestrial Radio Access (E-UTRA). Data service for wireless consumer devices.	700 MHz — 2,200 MHz	OFDM, PSK, QAM	1.4 MHz — 20 MHz
3G WCDMA	WCDMA, known primarily as 3G mobile, is a family of 3G data protocols used to send voice, text and signaling data to smart phones and other wireless devices.	824 MHz — 2,100 MHz	QAM, QPSK, CDMA	4.2 MHz
FM (frequency modulation)	FM radio	Around 100 MHz	FM	10 KHz
NBFM (narrow band frequency modulation)	Other FM radio stations, usually used for weather stations and emergency	Various locations, but not around	NBFM	5KHz

	stations	100 MHz like FM radio is		
Automated Train Control System (ATCS)	Automated Train Control System (ATCS), specifically ATCS Spec. 200, is a standardized communication system for railroads designed to ensure safety by monitoring locations of trains and locomotives, providing analysis and reporting, and automation of track warrants and similar orders.	896.888 MHz — 936.988 MHz	FSK	12.5 KHz
Fast Burst	A bursty (on-off) signal that we did not identify. Mostly likely some kind of FSK signal	461 MHz	?	10 KHz
DDS	Similar to IDEN sigal, no information on what it's used for	851 MHz	FSK?	15 KHz

Resources and citations:

Simulation (not used):

[1] https://dspillustrations.com/pages/posts/misc/python-ofdm-example.html

Datasets (not used, bad data):

[2] https://ieee-dataport.org/open-access/drone-remote-controller-rf-signal-dataset#files

[3] https://opendata.deepsig.io/datasets/2018.01/2018.01.OSC.0001_1024x2M.h5.tar.gz?__hstc=233546881.9c91e0549f9b6bfce6708a49c211c1c9.1614872457734.1614872457734.1614872457734.1&__hssc=233546881.1.1614872457735&__hsfp=1843090487

Papers and RF signal information:

[4] https://www.deepsig.ai/

[5] https://www.sigidwiki.com/wiki/Signal_Identification_Guide

[6] https://arxiv.org/abs/1909.11800

[7] https://arxiv.org/pdf/1712.04578.pdf

[8] https://medium.com/gsi-technology/residual-neural-networks-in-python-1796a57c2d7

[9] https://www.rtl-sdr.com/tag/signal-identification/

Libraries and code:

[10] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html#sklearn.metrics.ConfusionMatrixDisplay

[11] https://scikit-learn.org/stable/modules/model_evaluation.html#the-scoring-parameter-defining-model-evaluation-rules

[12] https://www.tensorflow.org/tutorials/images/cnn

[13] https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

[14] https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/

[15] https://keras.io/api/layers/regularizers/

[16] https://pypi.org/project/pyrtlsdr/#description (wrapper for RTL-SDR so we could collect data)