WiFi Spectrum Analyzer and Traffic Forecasting: Implemented with Adalm-Pluto, GNU Radio and LSTM Networks

Tsourdinis Theodoros
ttsourdinis@e-ce.uth.gr

Chatzistefanidis Ilias
ichatsist@e-ce.uth.gr

I. Project Execution Requirements

- Python2 & Python3 Installed
- Python3 Modules Installed: statistics , numpy , scipy and matplotlib
- On the grc file , change the path of the filename variable block to a specific desired directory.
- Run the total.py by executing: python3 total.py

II. INTRODUCTION

As WiFi access points are constantly increasing, the need to see in real time how busy the spectrum is and to predict changes in the availability of WiFi channels is very important, as this way we can avoid collisions and achieve better throughput over time. This is the purpose of the project, a low-cost WiFi spectrum analyzer, which in the first part shows the availability of WiFi channels in real time, finds the least busy channel and keeps the history of the availability of the channels over time. In the second part it finds the pattern of channel occupation and predicts future changes in the availability of frequencies, using neural networks.

To achieve this we used the following hardware - software tools. A low-cost SDR ADALM-Pluto which played the role of RF Antenna. The GNU Radio software tool for signal processing. The Python programming language (Python2 & Python3) in order to make some scripts for continuous measurements, the creation of plots and the implementation of neural networks. An LSTM Neural Network which, based on previous measurements, finds patterns from busy channels and makes predictions for future measurements.

III. Experimental Setup and Method

In order to receive the signals on the different frequencies of WiFi, we needed an RF antenna that is permanently in reception mode. This need was covered by Adalm -Pluto which via a USB-Ethernet cable was connected to the PC from where we processed the signals. Compared to other SDR devices the Adalm-Pluto is an economical device, with the only bottleneck being that the max sample rate when we repeatedly receive is no more than 4 or 5 MHz.

For signal processing, we connected the SDR device to GNU-Radio Software, which makes full use of the capabilities of Pluto SDR as it has specific blocks that trigger its functions. We specifically used the PlutoSDR Source block which practically allows us to receive signals at a specific RF Bandwidth and Sample Rate. Initially, to test that we receive transmissions, we used the QT GUI Sink and the QT GUI Waterfall Sink blocks provided by GNU-Radio

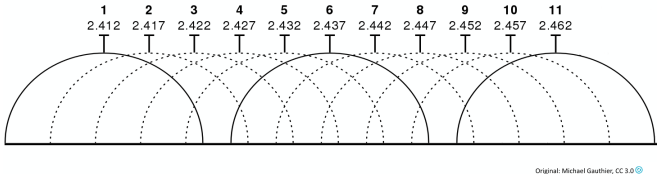


Fig. 1: 2.4 GHz WiFi Channels

and through this graphical interface we saw the signals at all frequencies and caught some peaks which represented the transmissions. In this way we adjusted some parameters that help in better signal reception such as gain and LO Frequency. As we see below in figure 2, we take 1024 values from the stream resulting from the PlutoSDR Source block and take them in the frequency field with the help of the FFT block (Fast Fourier Transform) in order to analyze the signals more easily and drop the noise floor. Then, we normalize the values, as each point of the FFT transform is the result of a sum over a certain time interval of the time-based samples. That's why we divide the FFT output by 1024. After that we convert the I-Q data to logarithmic scale in order to measure the signal strength in decibels. Finally we store the samples in binary files which are then processed through our algorithm.

IV. Our Algorithm

A. Measurement Technique

When creating a Spectrum Analyzer, we need to know exactly the position and the bandwidth of the channels. In our case, we focus on the 2.4 GHz spectrum, where channels have 22 MHz bandwidth and overlap with each other, as shown in figure 1. After much discussion with the teaching assistant we conclude, that obtaining measurements for 12 MHz out of 22, around the center frequency, for every channel could give us a sufficient picture of channel's usage. For this reason, we take three measurements for each channel. One around channel's center frequency, one around a frequency with 5 MHz offset left of the center frequency and the final around a frequency with 5 MHz offset right of the center frequency, as shown in figure 3

Thus, in practice we take measurements with 4 MHz bandwidth for center frequencies that have distance 5 MHz from each other; starting from center frequency 2.407 GHz, then 2.412 GHz, 2.417 GHz, ... ,until 2.467 GHz. In total, we collect measurements for 13 bandwidths to decide the usage of 11 channels.

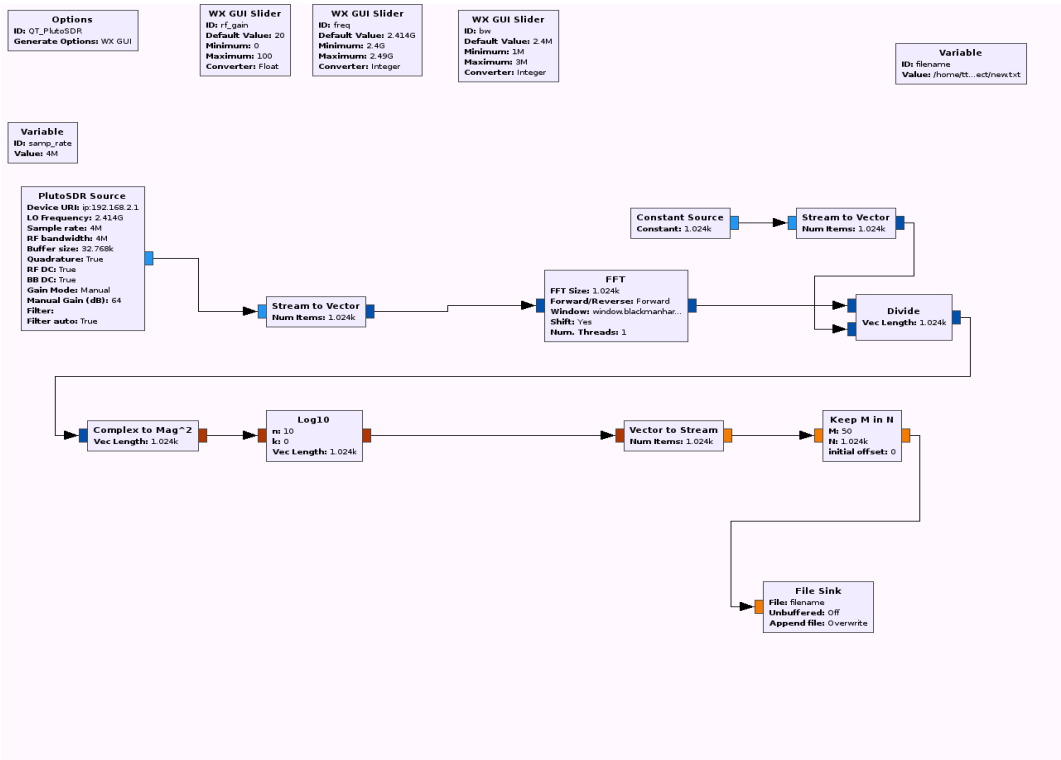


Fig. 2: GNU-Radio Flowgraph

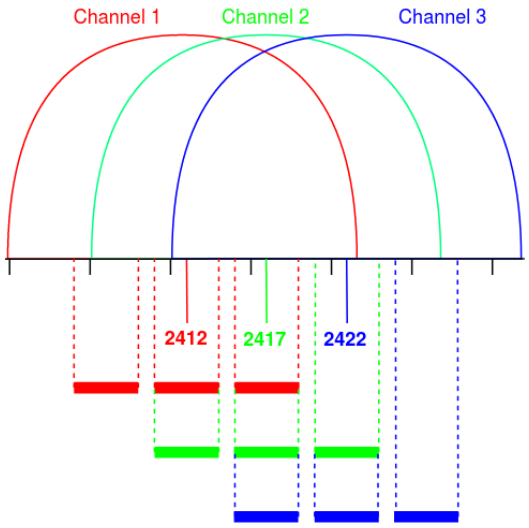


Fig. 3: Three measurements of 4 MHz bandwidth for every channel

B. Python Scripts

We created three python scripts: file_creator.py, datamaker.py and total.py.

1) file_creator.py: In this file we take all measurements for 13 frequency bandwidths, storing them in 13 binary files. After experimentation we decided that each measurement should last approximately 2 seconds, so as to obtain enough information about the energy levels and not to waste much time in a single measurement.

2) datamaker.py: With this script we convert the binary files to decimal and we obtain the information we need. Specifically, the resulting decimal values are floats with range [-60 , 20] approximately. Each value shows the energy levels in decibels of the measured bandwidth.

In the beginning, we plotted their Probability Density Functions (PDFs), many times and for different center frequencies, while observing the Waterfall Sink[1]. In this way, we can decide how the PDF looks like when there is and when there is no transmission. Finally, the pattern of the Energy PDF is shown in figure 4, where in Fig.4a there is not a transmission and in Fig.4b there is.

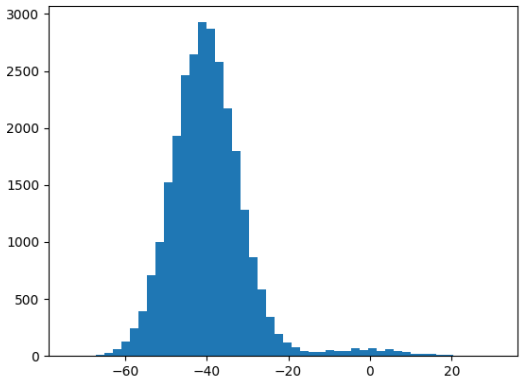
When there are no transmissions the most values are gathered between -60 and -20 dB like a normal distribution with mean $\mu = -40$ dB and there are not values above -20 dB. On the opposite, when there are transmissions we see a second distribution to rise appart from the first that we mentioned. This second distribution looks also like the normal and its values are ranged between -20 and +20 dB; indicating higher energy levels.

In this point, we decided to construct a statistic metric to measure bandwidth usage. We call it *Busy_Percentage* and it is defined as:

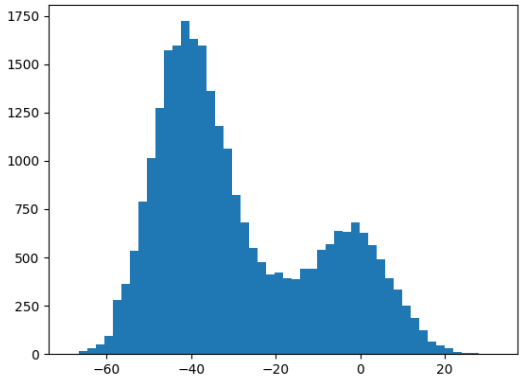
$$Busy\_Percentage = \frac{\#Transmission\_Values}{\#All\_Values} \quad (1)$$

,where *Transmission_Values* are the decimal energy values

[1]A graphical sink to display multiple signals on a waterfall (spectrogram) plot.



(a) *Busy_Percentage* = 3%: No Transmissions.



(b) *Busy_Percentage* = 30%: Transmissions exist.

Fig. 4: Probability Density Function (PDF) of Energy Measurement Values: X-axis = Energy Values and Y-axis = Number of times that a X value appears

measured that are greater than a predefined threshold-value; beyond this value, we consider that there is a transmission.

From our observations about the pattern of the Energy PDF, we ended up that the best value to set as threshold is **-20 dB**. If we observe closely the figure 4a, it is clear that there are not any values above -20 dB, when the *Busy_Percentage* is 3%, meaning that there is no transmission. On the contrary, at Fig.4b when transmissions do exist (*Busy_Percentage* is 30%), we see many values above -20 dB. So, we can rewrite Equation (1), as:

$$Busy\_Percentage = \frac{\#(Values > -20dB)}{\#All\_Values} \quad (2)$$

3) total.py: This is the final wrapper script, which calls file_creator.py internally for T iterations (where T is the number of measurements for each channel) in order to get continuous measurements for each channel. It then calls datamaker.py to manage the binary data by converting it to decimal format and returning the busy percentage for each channel. Then, we plot in real-time the occupation of

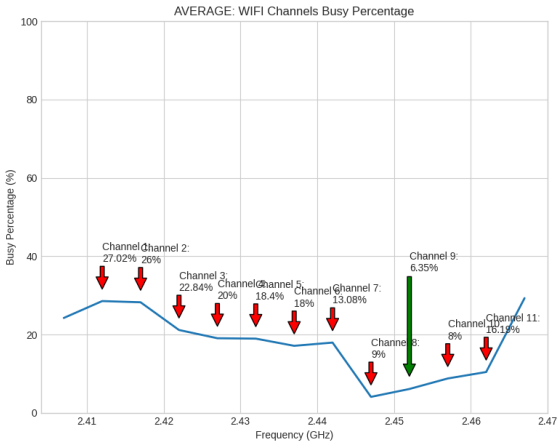


Fig. 5: Average WiFi Spectrum *Busy_Percentage* after 100 iterations: Best-optimal Channel found to be Channel 9

frequencies, having on the x-axis all the frequencies and on the y-axis the busy percentage corresponding to them. The plot is the same as in figure 5, but it refers only to the one current iteration (*CURRENT Plot*) and not in terms of average. On the curves, we mark (with the red color), in which channel the below frequencies correspond to. We also mark the optimal channel, which is the least busy, with the green color. After that, we plot the average occupation of the frequencies (Figure 5), which works with the same logic as the previous plot but in this case we keep the average of busy percentage values (*AVERAGE Plot*). Finally, in a multi-plot (TIME Plot) we have the behavior of each channel over time separately (Figure 6) and we constantly show which is the optimal (the one with the green line). We repeat this procedure for T iterations.

V. Results

To execute our experiment we have to take into account some characteristics of our algorithm:

- the duration of a measurement in a specific 4 MHz bandwidth, at a certain center frequency, is 2 seconds
- we have to measure 13 bandwidths across the WiFi spectrum for 1 iteration,
- there is extra processing time to convert binary to decimal values and to obtain the *Busy_Percentage* metric

Thus, we found out that 1 iteration of our algorithm lasts about 1.5 minutes.

Moreover, we want to collect enough data to be able to know the real spectrum usage and to make an accurate prediction about its future traffic. For these reasons, we decide to execute our experiment for 100 iterations; meaning 2.5 hours measurements. The results are shown in figures 5 and 6.

As we can see in figure 5 at the end of the experiment - after 100 iterations (2.5 hours) - we found out that the lower average *Busy_Percentage* value belongs to channel 9;

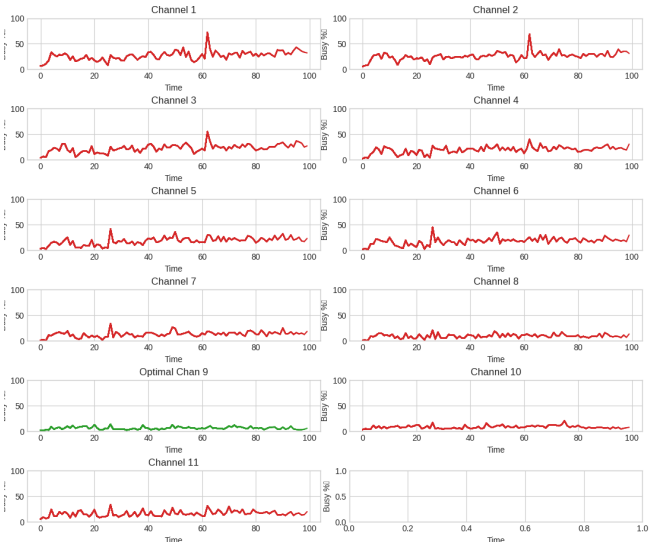


Fig. 6: Channels' *Busy_Percentage* in Time (100 Iterations): The best-optimal channel 9 is with green colour

so it is the optimal. Now, if we observe the figure 6, which shows us the usage for all channels during the whole experiment, we see that all channels do not remain in a constant *Busy_Percentage* value; instead there are fluctuations and bursts of high usage or periods of inactivity. Nevertheless, it is obvious that channel 9 was the most inactive channel across all iterations, when the other channels have constantly higher *Busy_Percentage* values.

It is important to be mentioned that some times during the experiment the *CURRENT Plot* showed that another channel is the optimal for a specific iteration, e.g. channel 8. Nevertheless, the *AVERAGE Plot* always remained at showing the channel 9 as the best channel since the beginning of the experiment.

VI. Future Forecasting

We would like to go a step further and to make our algorithm smart. The target is to find a pattern in the behaviour of WiFi spectrum usage. In this way, we predict which channel will be the optimal in the future and we are able to choose this one avoiding possible interference and collisions. Since this is a Time Series Forecasting, we use Long short-term memory (LSTM) Neural Networks.

A. Prepossessing

To achieve this we collect the experiments' data for all channels from the file "channels_meas.txt" that we created with our algorithm. Then, we have to prepare our data to fit the model's expected input; 3 input dimensions are: samples, time steps, and features.

It is very important to split our data in training and testing data. In this way we will use the first to train our model in order to make predictions for the testing data. Then we will evaluate how accurate the predictions are by comparing them with the real testing data. A common way is to split in 80% training and 20% testing data. This means that the first 2

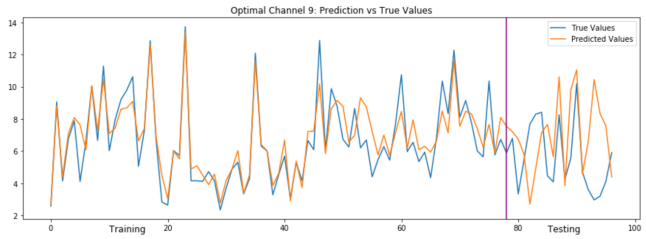


Fig. 7: Optimal Channel 9 *Busy_Percentage* in Time: Training and Testing Prediction vs Real Data

```
Channel 1
Predicted Mean Value of Testing:  29.69
Real Mean Value of Testing:  32.69
Channel 2
Predicted Mean Value of Testing:  27.86
Real Mean Value of Testing:  29.85
Channel 3
Predicted Mean Value of Testing:  26.82
Real Mean Value of Testing:  27.48
Channel 4
Predicted Mean Value of Testing:  23.8
Real Mean Value of Testing:  22.55
Channel 5
Predicted Mean Value of Testing:  24.86
Real Mean Value of Testing:  22.78
Channel 6
Predicted Mean Value of Testing:  18.14
Real Mean Value of Testing:  20.49
Channel 7
Predicted Mean Value of Testing:  14.8
Real Mean Value of Testing:  15.48
Channel 8
Predicted Mean Value of Testing:  10.89
Real Mean Value of Testing:  9.32
Channel 9
Predicted Mean Value of Testing:  6.93
Real Mean Value of Testing:  5.63
Channel 10
Predicted Mean Value of Testing:  7.74
Real Mean Value of Testing:  6.93
Channel 11
Predicted Mean Value of Testing:  15.42
Real Mean Value of Testing:  16.43
```

Fig. 8: Testing *Busy_Percentage* % values: Predicted vs Real for all channels

hours out of the 2.5 hours of the experiment will be used to train the model, finding patterns in the *Busy_Percentage* of every channel. Then the model will be ready to make prediction for the last half hour of the experiment based on what it learned from the previous 2 hours and without knowing anything for the last half one.

B. Predictions

At first, we fit the data of optimal channel 9 to our model and make the prediction as shown in figure 7. We see that the model do finds the pattern in training data and it is also able to make a sufficient prediction for the last half hour (right from purple line border). Specifically, it is predicted that the channel 9 in the last 30 minutes of the experiment (testing

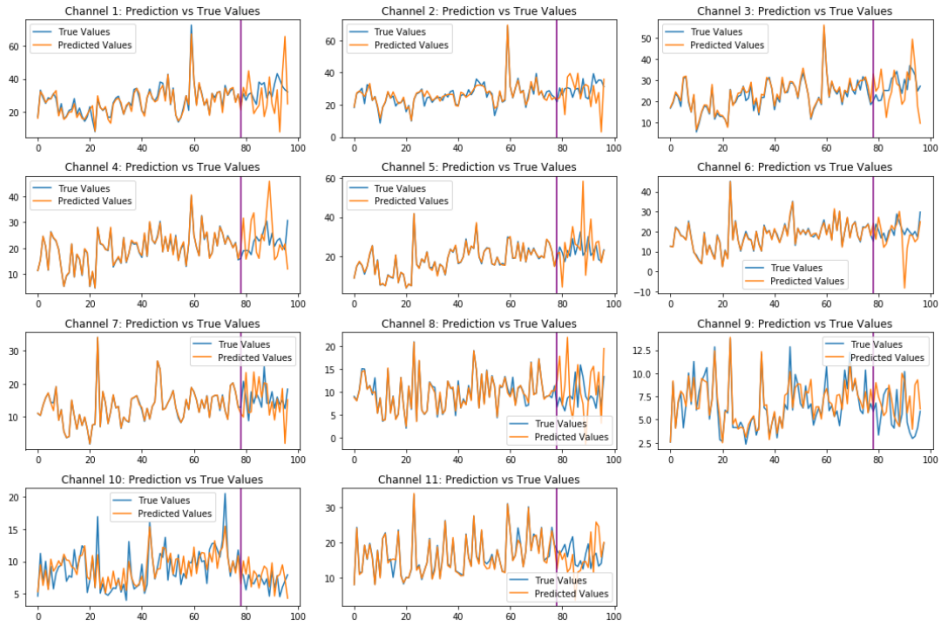


Fig. 9: All Channels *Busy_Percentage* in Time: Training and Testing Prediction vs Real Data

data) will have Average *Busy_Percentage* 6.98 %. The real Average *Busy_Percentage* is 5.63 %, found from real testing data. So our model predicts the value with high accuracy.

Now we make predictions for all channels as shown at figure 9. Specifically, the exact values for all channels are shown at figure 8, where it is clear that our model achieves a high accuracy at predicting the values.

C. Recommendation

At this point our model is able to make a prediction for the whole spectrum's usage and also a recommendation of the optimal channel. Both are for the last 30 minutes of the experiment (testing data), based on what it learned only from the first 2 hours.

In figure 10 we observe how our model predicted the average usage of the channels for the last 30 minutes and also the real spectrum usage. The lowest value is located at channel 9 and so, our model recommends us to use this one. The prediction is very accurate and our testing data show us that the channel 9 is indeed the optimal.

VII. CONCLUSIONS

As we can see from the above figures, our algorithm with the measurements, not only gives a detailed picture of the state of the channels in the band frequencies of 2.4 Ghz, but also gives us enough data that can be fed to a neural network like LSTM we used and lead us to pretty good predictions. So overall, our WiFi Spectrum Analyzer has number of attractive features such as being low cost, easy to configure, gives an accurate picture of the occupation status of the channels and quite good predictions for the status of the channels in the future.

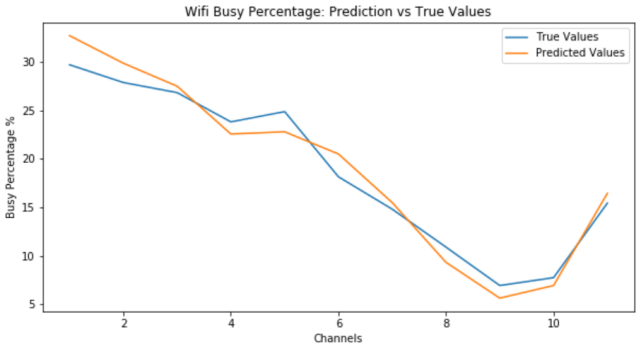


Fig. 10: Average Spectrum (11 Channels) *Busy_Percentage* % Prediction vs Real

VIII. FUTURE WORK

In the future, our project could be upgraded with some extra features such as better training of LSTM neural network in order to have better accuracy by taking more measurements which for example will last for a week. We could also put more information about the state of the channels, such as the max throughput we can have on each channel. In addition we could do a mapping between the SSID of the access points and the channels, so that we know which access points are transmitting to specific channels. Finally, we could integrate the frequencies of the 5 Ghz channels and take corresponding measurements there as well.

REFERENCES

[1] L. Getz, W. Schwartz, K. Smith, "Internet of Things Spectrum Monitoring and Localization" (2019). A Major Qualifying Project Submitted to the Faculty of WORCESTER POLYTECHNIC INSTITUTE

[2] Nelson, Matthew Erik, "Implementation and evaluation of a software defined radio based radiometer" (2016). Graduate Theses and Dissertations. 15053.

[3] A. Ivanov, N. Dandanov, N. Christoff, V. Poulkov. “Modern Spectrum Sensing Techniques for Cognitive Radio Networks: Practical Implementation and Performance Evaluation” (2018)
[4] R. P H, R S S. Thejaswini, K. Venugopal, P. Kumar M, Niveditha J, Pallaviram Sure, “Wideband Spectrum Sensing using Sub-Nyquist Sampling Approaches ”
[5] E. O. Kandaurova, D.S. Chirov, Member, IEEE, “Algorithm and Software for Intelligent Analysis of the Frequency Spectrum for Cognitive Radio Systems”
[6] Christopher Gravelle and Ruolin Zhou, “SDR Demonstration of Signal Classification in Real-Time using Deep Learning”