Identifying Network Protocols with Neural Networks

Emin Alp Arslan
College of Natural Sciences
University of Texas at Austin
eminalp@utexas.edu

Abstract

I experiment with various neural network architectures to build a model that can effectively and efficiently identify network protocols (BLE and Zigbee) and distinguish them from noise given a raw RF signal (in I/Q data form). I find that FCNs (fully convolutional networks) have the best tradeoff between accuracy, performance, and inference on arbitrary number of samples.

1 Background

1.1 Introduction

Many Internet of Things (IoT) devices require communication through low-power network protocols like Bluetooth (BLE) and Zigbee. These protocols are easily overpowered by high-power protocols like WiFi which leave only small windows for the low-power protocols. Because of this, these low-power protocols need to make sure that they don't interfere with each other during these windows to gain maximum throughput.

1.2 Purpose

The goal of this paper is to see if neural networks can replace the traditional signal identification pipeline by providing higher accuracy and/or lower latency predictions. As more and more IoT devices get AI accelerators, neural networks become more and more feasible for this application. In this paper, I test various neural network architectures and techniques applied to this problem.

2 Dataset

I used a MATLAB script to generate synthetic signals of BLE, Zigbee, and Noise. The data generated was raw I/Q data of these classes of signal at various SNRs (signal to noise ratio). I trained the models directly on this raw I/Q data with no pre-processing so that predictions can happen with the lowest latency possible.

3 Training

I trained the neural network models by randomly sampling from the dataset such that the model trained on all three signal types with varying SNRs. I randomly truncated the number of samples (from the original 32 samples) for each signal so that the model would generalize away from a specific number of samples. The loss function I used was cross entropy since the goal is classification.

4 Experiment 1: Architecture

For my first experiment, I tried out various neural network architectures like CNNs, RNNs, and various hybrids and modifications.

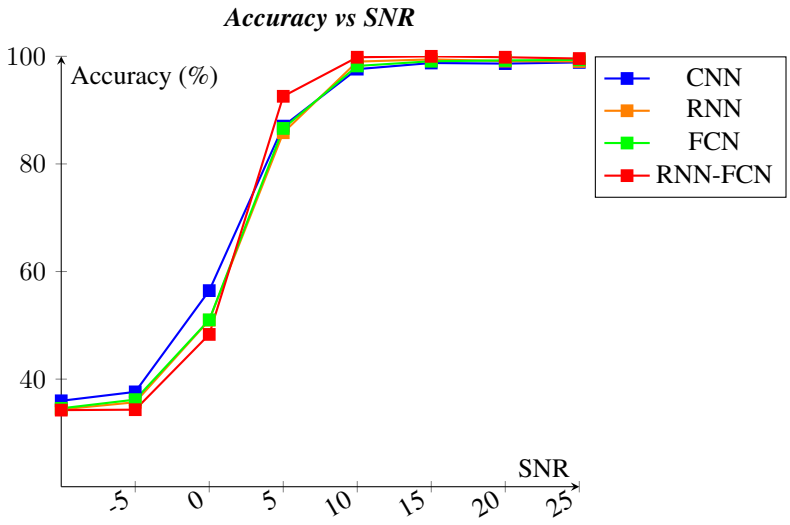

Figure 1: Chart comparing the accuracy of the models with 32 samples of data at different SNRs.

4.1 Justification

Other papers that use neural networks for RF signal classification like Scholl et al. seem to generally use CNNs, while intuitively it seems like such tasks are better suited for RNNs. My hypothesis is that while RNNs might be able to get slightly better performance, CNNs will be almost as performant while being much faster since they handle many samples at once in a single forward pass as opposed to the RNN, which has to perform a forward pass for each sample. Bai et al. seems to indicate that CNNs might even often outperform RNNs for sequence modeling tasks. My goal with this experiment is to test my hypothesis and find the best architecture for my problem.

4.2 Using ChatGPT

For this experiment, I used ChatGPT to come up with various architectures I could use for this problem. I also asked it to come up with pros and cons for the different architectures it proposed. Most of its claims were confirmed by my experiments. For example, it said that RNNs would be more computationally expensive than CNNs, which I confirmed while training the models. It got some things wrong, like stating that CNNs could handle arbitrary input sizes, but fixed its response after being corrected and proposed FCNs as an alternative.

4.3 Results

The results can be seen in Figure 1, which compares the following four neural network architectures' performance when classifying signals with 32 samples of data.

CNN The convolutional neural network was the baseline model I used when evaluating the other models. It had very good accuracy across SNRs, but had the big limit of only being able to handle a set number of samples. In practice, it might be necessary to use the model with different numbers of samples to make a classification.

RNN The recurrent neural network seemed like the best fit for this task. In my case, I used a bidirectional GRU with 3 layers into 3 linear layers. The model got very good results, on par with or

even better than the CNN, but was much slower to train and at inference time. Its biggest advantage though was its ability to make predictions with any number of samples.

FCN The fully convolutional network remedies the main problem of the CNN through robustness to the input size. In the context of this problem, it means that FCNs can handle different number of samples. It does this by replacing the linear layers with convolutional layers with a kernel size of 1 and adding global pooling before the output. The FCN gets the best of both worlds between the CNN and RNN, fast inference and training, very high accuracy, and predictions with varying number of samples. The only issue I had that was unique to the FCN was that training was very inconsistent.

RNN-FCN The RNN-FCN (or GRU-FCN in my case) is, despite the rise of transformers, the state of the art for many time series classification tasks. It combines RNN modules with the FCN getting the advantage of the fast inference and spatial awareness of the FCN with the temporal dependency of the RNN.

4.4 Discussion

While the best accuracy came out of the RNN-FCN (as expected since it can leverage the abilities of both FCNs and RNNs), I decided on the FCN as the best model for this task. It has fast inference and training, can handle aribtrary input size, and has very good accuracy. I learned that each of these architectures have tradeoffs when it comes to solving these types of timeseries classification problems.

5 Experiment 2: Regularization

My second experiment was trying out various regularization techniques to improve my models' accuracy and generalization.

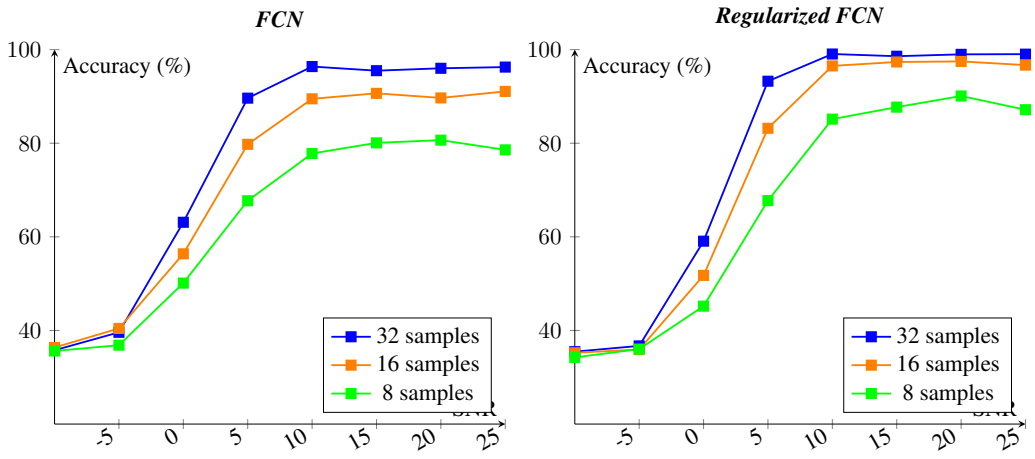

Figure 2: Charts showing the accuracy of the original and regularized FCN models at different number of samples across SNRs.

5.1 Justification

Regularization techniques are often applied to neural networks to make sure that they don't overfit the training data and force it to generalize. As a result, the models should achieve better accuracy when evaluated with testing data.

5.2 Using ChatGPT

I used ChatGPT to come up with regularization techniques I could use for my models. Many of the techniques ended up improving my model substantially.

5.3 Results

Figure 2 shows that improvement the regularized FCN model had over the original FCN model. The following is a list of regularization techniques used.

L2 Norm Also referred to as weight decay, this technique had the smallest effect on the model out of all the techniques. Since the effect was unnoticeable, my hypothesis is that the model's weights never exploded during training due to the other regularization techniques.

Dropout Dropout made very substantial improvements to my models. This technique was probably why weight decay didn't have much of an impact in my models, since it makes sure that the model doesn't over-rely on few parameters. Dropout alone pushed model accuracy at high SNRs from 97% accuracy to 99%+ accuracy.

Batch Normalization I placed batch normalization primarily in the convolutional layers of the models, and it also improved the model accuracy noticeably.

Data Augmentation Truncating the signals to random signal lengths (different number of samples) during training drastically increased the model's accuracy at lower number of samples while only marginally hurting the model at higher number of samples.

5.4 Discussion

The regularization techniques not only made the model more accurate, but more importantly, it improved the model for varying number of samples and generalized it to data outside the training data. I learned that data augmentation can be a very important technique to generalize a model to different use cases, and that it is a very problem-dependent regularization technique.

6 Final Model

The model I ended up selected based on the results of the experiments was the regularized FCN model whose results are shown in Figure 2. The model has 5 convolutional layers, with the last two acting as fully connected linear layers by setting their kernel sizes to 1. The number of channels increases, the decreases right before the output: 1, 16, 64, 128, 64, 3. The first two convolutional layers have kernel sizes of (1, 4), and the third layer has a kernel size of (2, 4). The conv layers are all followed by ReLU activation functions, dropout, then batch norm. The dropout after the fourth conv layer is higher (0.5) as opposed to the previous conv layers (0.2). The last conv layer's output goes through global adaptive average pooling and this return the output.

Putting batch norm after dropout seems to be uncommon, but worked better for me in my tests. I tried using max pooling layers after each conv layer, but they made the model unstable during training. The model would get very different results every time it was trained. Increasing the number of epochs also made the model unstable when max pooling was added. Global max pooling instead of global adaptive average pooling also seemed to work well as the final step before the output.

My goal was to make the model big enough to the point where it could be useful in practice, but not too big such that it would be difficult to run on IoT devices.

References

[1] Scholl, S. (2022). RF signal classification with synthetic training data and its real-world performance. arXiv preprint arXiv:2206.12967.

[2] Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271.

Model Card

Model Details

- Model classifies raw I/Q RF data into three classes: BLE, Zigbee, and noise.
- FCN (fully convolutional network).
- Developed by Emin Alp Arslan in 2023.

Intended Use

- Intended to be used in embedded systems.
- Intended for IoT devices which need to quickly identify network protocols before transmission.

Metrics

- Accuracy over each SNR value.
- Accuracy vs SNR for different number of samples.

Training Data

- Synthetically generated I/Q RF signal data.
- Signals are 32 samples and have varying SNR values.
- Randomly sampled and truncated when training and evaluating.

Evaluation Data

- Some of the data generated by the script are held out as testing dataset.

Caveats and Recommendations

- Only trained with and tested on synthetic data, may not comprehensively cover all real world use cases.

Quantitative Analyses

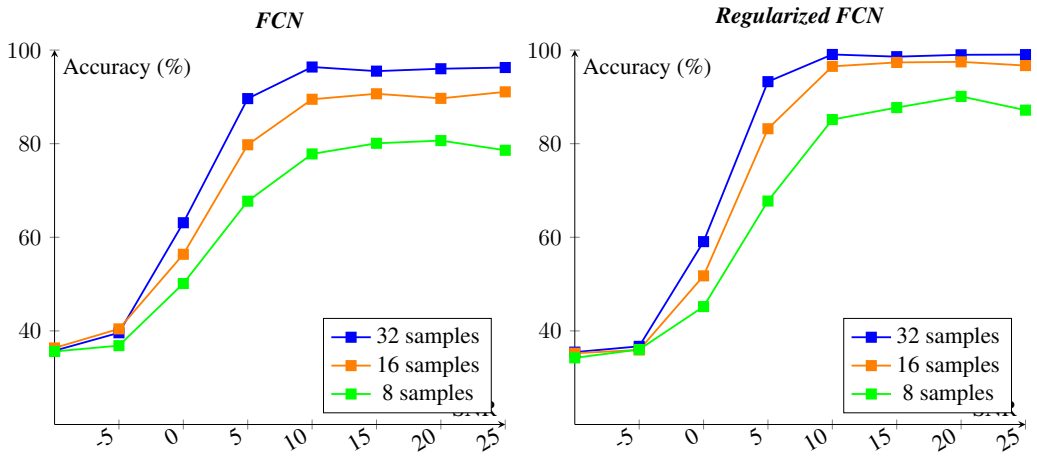

Figure 3: Model card describing the FCN model for identifying network protocols.

Datasheet

Motivation
The dataset was created to enable research into classifying network protocols like BLE and Zigbee using neural network models.

Composition
The data instances are two sequences of numbers each with a length of 32. Each pair of numbers represents a single sample from a given signal. The signal class, along with its SNR value, frequency offset, and signal sensor mode (BLE vs Zigbee) are included at the beginning of the line. The RF data is represented as raw I/Q data, which is the reason for each sample having two numbers. There are 48,000 signals in the file, 1,000 signals for each combination of signal class, sensor mode, and SNR value (ranging from -10 to 25 in intervals of 5 inclusive). The dataset is a random sample of instances of these signals. There is no missing data. Data splits for training, validation, and testing should be randomly sampled from the dataset. The dataset is self-contained and doesn't include confidential, sensitive, or personal information.

Preprocessing
The data is meant to be "raw" data that is not preprocessed in any way other than the labels attached to before the start of each signal.

Uses
This dataset has been used to train a neural network FCN model for identifying the three classes in the dataset from just the raw signal data. The main use of this dataset would be similar tasks of classification. This dataset should only be used for tasks where specifically BLE and Zigbee signals, and nothing else, are needed.

Maintenance
The dataset has not been found to have erratum.

Example signal from the dataset
(index, SNR, sensor mode, freq offset, signal classification, ... 32 samples of i component of signal)

(index, SNR, sensor mode, freq offset, signal classification, ... 32 samples of q component of signal)

(428, 5, 1, 0.000000, BLE, -0.026708, -0.513484, -0.916891, -1.116161, -1.085267, -0.905052, -0.708186, -0.595104, -0.581697, -0.609672, -0.611445, -0.577674, -0.565370, -0.638208, -0.797143, -0.970020, -1.073858, -1.092528, -1.091020, -1.145353, -1.255766, -1.327911, -1.238143, -0.924264, -0.429339, 0.125423, 0.606469, 0.918694, 1.018583, 0.908879, 0.636533, 0.287981)

(428, 5, 1, 0.000000, BLE, 0.948169, 1.212369, 1.319639, 1.237492, 1.007315, 0.687322, 0.310738, -0.106364, -0.527943, -0.878548, -1.069016, -1.053152, -0.867228, -0.620620, -0.443329, -0.427704, -0.595633, -0.896349, -1.228099, -1.468071, -1.515347, -1.347157, -1.055812, -0.824376, -0.828682, -1.114410, -1.538353, -1.829770, -1.745603, -1.220446, -0.411035, 0.391846)