AI for RF Signal Modulation Classification

2024 AI Challenge

BAE Systems, Inc.

Rev 1.2 – J.M. Shima

Background and Motivation

Radio frequency (RF) signals are ubiquitous in today's world. Everything from cell phones, base stations, digital communications towers, GPS, RADAR, IoT, satellite downlinks (TV), and WiFi make up some of the wireless landscape today. Past systems that sought to classify incoming signal types were very much rules based user-in-the-loop schemes, requiring expert knowledge and user intervention to decipher and categorize incoming signals. Recently, advances in deep-learning neural networks showed great promise at automating this previously insurmountable task – providing a path to autonomous collection and analysis w/o user intervention. This type of RF survey is used in many disciplines from in-band RF interference detection, RF intelligence gathering, spectral signal compliance, jammer detection, and spectrum management.

Task

Your task is to develop an automated signal classifier as a payload skill for a next-gen satellite mission. The mission is responsible for surveying Earth from orbit and categorizing RF signals into set classes as it scans predetermined areas. The frequency range of the antenna is 6-18 GHz in which signals are collected. The phantom system we are using will sample RF data, mix the signal to baseband, and store it off. That data will serve as your training set.

You will create an AI solution that ingests these sampled RF waveforms (including any data preprocessing) and return a signal ID for each signal in the training set. Each waveform will be provided with a label of its known modulation type for supervised training.

Provided Data

The waveforms consist of time-series data sampled at a set sampling frequency F_s = 100 MHz. As mentioned above, the sampled data will also be provided at baseband so no other frequency mixing is necessary. The data are also complex-valued in the form of inphase (I) and quadrature-phase (Q) samples. This IQ format is common in sampled comm systems, in which each sample is represented as z = I +jQ, where j = sqrt(-1). This format is more expressive over real-valued signals, as the magnitude and phase is readily computed for each sample. For more on IQ, please refer to the references.

Each training example will consist of 1024 samples of a waveform modulated with a random data payload. The 11 signal classes will be common digital modulation types as follows:

1. **Binary phase-shift keying (BPSK)** – with raised cosine pulse-shaping filters w/ varying excess BW "α" and filter length

2. **Quadrature phase-shift keying (QPSK)** – with raised cosine pulse-shaping filters w/ varying excess BW "α" and filter length

3. **Eight state phase-shift keying (8PSK)** – with raised cosine pulse-shaping filters w/ varying excess BW "α" and filter length

4. **Minimum-shift keying modulation (MSK)** – with frequency spacing set to R/2, where R = bit rate (bits/sec)

5. **Frequency-shift keying modulation (FSK)** – with continuous and discontinuous symbol transitions with frequency spacing between R/2 and R

6. **Four-level pulse-amplitude modulation (PAM4)** – with raised cosine pulse-shaping filters w/ varying excess BW "α" and filter length

7. **Gaussian minimum-shift keying (GMSK)** – with varying initial phase offset and time-bandwidth product

8. **Gaussian frequency-shift keying (GFSK)** – with modulation index = 1 and varying time-bandwidth product

9. **Sixteen-state quadrature amplitude modulation (QAM16)** – with raised cosine pulse-shaping filters w/ varying excess BW "α" and filter length

10. **Sixty-four state quadrature amplitude modulation (QAM64)** – with raised cosine pulse-shaping filters w/ varying excess BW "α" and filter length

11. **One-hundred twenty eight state quadrature amplitude modulation (QAM128)** – with raised cosine pulse-shaping filters w/ varying excess BW "α" and filter length

For those modulations with pulse-shaping, α can range from [0,1]. Each signal example will also have randomized parameters for SNR, total bandwidth, channel fading, sample rate offset, frequency offset (Doppler), and clock offset. This simulates real-world impairments on the RF signals that the model must account for during classification. The SNR and bandwidth for any

signal will be comprised of values between 15, 20, 25 dB and 2 - 20 MHz, respectively. Note that for higher-order modulations, the data bit rate can be very different for large and small signal bandwidths.

Notes and Tips

You can use raw I and/or Q into the network as a time series, or you can preprocess the data (i.e., into frequency domain, PCA, wavelets, etc) first and feed that as the input to the network. According to the given sample rate and signal bandwidths, the waveforms will have anywhere from 5 up to 40 samples per data symbol. Data symbols are the atomic unit of any communications protocol, and the number of data bits per symbol is governed by the modulation scheme. Data symbols are sent at the *symbol rate*, which will be anywhere from 20 Msps to 2.5 Msps. To get the bit rate of the link, multiply the symbol rate by the bits/symbol of the modulation scheme. For example, BPSK has 1 bit/symbol, whereas QPSK has 2 bits/symbol, 8PSK has 3 bits/symbol, and 16QAM has 4 bits/symbol. The more bits/symbol a modulation scheme employs increases its *spectral efficiency*.

Your model will be tested, on a set of data not used during training, to determine its performance during inference. You should split the data into a training and validation set (like 90% and 10%, respectively), where the validation set it used during training to monitor the quality of learning. Again, the validation set is not used for actually training the network but rather for monitoring the training session for cues on network health over training epochs.

Scoring

You will be scored on two factors. First is the total classification accuracy, over all classes, from 0-100%. Furthermore, since this is a solution to be deployed on a satellite, we want the total compute resources to be low enough not to exceed the satellite power budget. As such, the second scoring factor is the number of total *learnable* parameters (N_{params}) used in the model. Note that a lower number of model parameters with the same accuracy will give a better score. This is a design tradeoff. Since preprocessing of the data typically does not use learnable parameters, preprocessing functions are not counted as parameters. Note this is within reason and in the spirit of the problem set (do not abuse). To better value system performance, you must meet the following requirements:

- a total classification accuracy >= 70%
- reasonable pre-processing stages with compute resources (FLOPS) < than the actual AI model block
- Max number of model parameters (learnable + unlearnable) < 2e6

Not meeting these requirements will result in a minimum score given. The scoring will use the following equation, and the team with the ***maximum*** score will be deemed the winner.

$$score = \frac{accuracy_{total}\%}{100} + \frac{10}{\log 10(N_{params})} + \beta - \varepsilon$$

Where: $\beta = \begin{cases} 0.55, & if\ accuracy_{total}\% > 95\% \\ 0, & else \end{cases}$

$$e = \begin{cases} 0.2, & if\ pre\ processing\ (FLOPs) > model\ inference\ (FLOPs) \\ 0, & else \end{cases}$$

It is up to each team to determine estimate of FLOP count for preprocessing and model inference in this case. If you are performing pre-processing steps be sure to inform us during submission.

Submitting Test Data Results

There will be two data sets given to evaluate the performance and generalization ability of your solution during periods of performance. You will be given the raw test data only without the corresponding labels. The first data set will consist of new examples never seen by the training and act as a milestone checkpoint. The second set is the final grading data set of examples. You are responsible for running each set through your model and storing off the resultant class returned by your model. The results for all examples should be stored in a text file, where the class outputs are integers that correspond to the index of the signal type as listed in the 'modulationTypes' vector provided to you. In other words, the class mapping is simply:

Signal class	Class index
BPSK	1
QPSK	2
8PSK	3
16QAM	4
64QAM	5

128QAM	6
PAM4	7
FSK	8
MSK	9
GMSK	10
GFSK	11

File Format:

The text file should have one index per row followed by a line feed/carriage return <LF> <CR>, or a newline '\n'. The first entry of your file should be the total number of learnable parameters in your model. For example, if you model uses 1.2 million learnables, the beginning of the file would look something like:

```
1200000
1
10
4
```

Where the integer class results follow the network size. This txt file will be used to score your model's performance, so double check that the format is correct. There is a sample file

'*test_res_train.txt*' that shows the indexed class outputs for a portion of the training data set. Submission details for the txt files will be given later.

Various and Sundry

1. Learnable parameters in your model are generally known as weights and biases. When you train a network or other model, the parameters that get updated with new information during training are "learnable". The number of learnables is loosely related to the solving power of the network. To check the number of learnable parameters, you can do the following:

   MATLAB: `analyzeNetwork(model)`

   Pytorch: `total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)`

   Keras: `model.count_params()`

2. You may be asked to submit an ONNX model of your network. ONNX is a standardized format for creating network structures and can be read in by most all toolsets.

3. If you attempt to use several networks in an ensemble method, be aware that each network's number of parameters are counted towards the total number of learnables.

4. The training set consists of 27,000 examples of each modulation type over the SNR range, for a total of 297,000 examples.

5. The training data set was originally created in MATLAB. As such, you can use Ball's network version of MATLAB to read in and reformat the data. Or you can use python to read in a .mat file. Note that you will need to be able to read a v7.3 MATLAB .mat file, and the scipy library will be unable to read it in due to the HDF5 header. One option is using a custom library: GitHub - skjerns/mat7.3 is one python library to use. Another is pymatreader · GitLab

6. The variables in the training data will be the complex-sampled signal data (1024 pts), the modulation types in a string array, and the one-hot encoded vectors (length 11) of the modulation truth (labels) that corresponds with each signal entry.

   Note the one-hot encoded vectors of each truth label corresponds to the index (base 1) of the string values in 'modulationTypes'. For example, 'BPSK' is the first entry in this array, which corresponds to a one-hot vector = [1 0 0 0 0 0 0 0 0 0 0]. Likewise, 'QPSK' is the 2nd entry and that gives a one-hot vector = [0 1 0 0 0 0 0 0 0 0 0]. The last entry in 'modulationTypes' is 'GFSK' and this corresponds to the one-hot vector label [0 0 0 0 0 0 0 0 0 0 1].

   Formatting the labels further will depends on your training code library inputs (categorical, ints, etc). Below is an example python script of reading in the training file:

```
import mat73
import numpy as np
from os.path import isfile, join
fname = join('c:/projects/train_data','ai_rf_challenge_train_data.mat')
data = mat73.loadmat(fname)
```

You can also look at the example file '*convert_mat_data.py*' if you want to convert the data from the mat file into numpy arrays and save it off for easier use in Python.

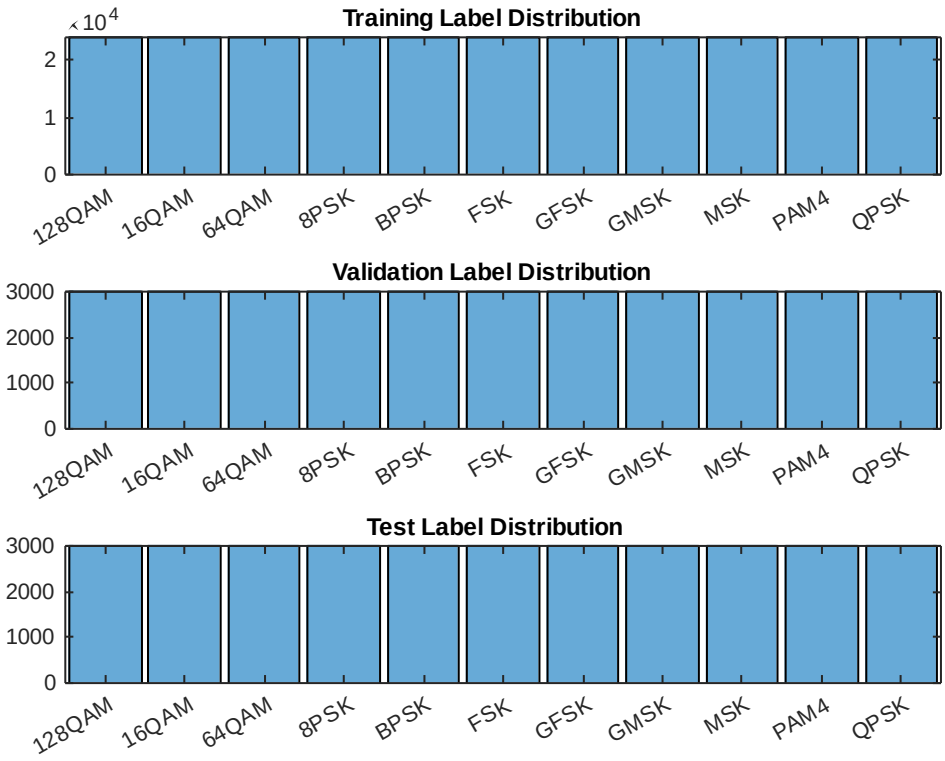
7. You can use any deep learning or ML toolset. You will have to be able to ingest a set of test data to run an inference pass, so creating a generic input data parser will help.

8. An output confusion matrix after running your test inferences should be generated, which will include your total accuracy metric towards your score.

9. In this project, assume the data was labeled by a different group of people working at a satellite ground station. During training be sure you check results to detect whether or not performance degradation is due to normal design tradeoffs or potential data issues.

10. On a Ball desktop computer with an older GTX 1080 GPU, the network trains in about 10-20 minutes (depending on hyperparameters). CPU training can be upwards of 1-2 hrs.

11. The histogram of the data set classes shows a balanced training and test set, as shown below.



Good luck on the challenge!

References

1. Shima, J., "*Quadrature (Complex-Valued) Signal Processing – A Technical Overview",* http://www.dspdude.com, 2009

2. Sklar, B., Digital Communications Fundamentals and Applications, Prentice Hall, 1988.

3. Smith, S., The Scientist and Engineer's Guide to Digital Signal Processing, https://www.dspguide.com/

4. Lec 1 | MIT 6.450 Principles of Digital Communications I, Fall 2006 – YouTube: https://www.youtube.com/watch?v=KXFF8m4uGDc

5. Modulation – Wikipedia https://en.wikipedia.org/wiki/Modulation#Digital_modulation_methods