MODULATION CLASSIFICATION USING CONVOLUTION NEURAL NETWORKS
GROUP 38

Kartik Kulgod⋆*, Young Li*⋆*, and Parker Martin*⋆

⋆Department of Electrical and Computer Engineering
University of California San Diego, La Jolla, CA 92093-0238

ABSTRACT

Modulation Classification is an increasingly relevant design feature in the field of wireless communications. The transmitting radio of modern communication systems may choose one of many types of RF modulation, creating a requirement that a receiving radio be able to identify the modulation scheme in use. Radio performance can be improved by using machine learning to classify the modulation used in incoming signal energy. We attempted to implement existing work [1] by applying Convolutional Neural Networks to the problem of modulation classification. Our model accuracy achieved its peak modulation classification of 83.5% when the signal-to-noise ratio of the input data was +12 dB.

1. INTRODUCTION

Radio Frequency (RF) modulation is the process of communicating a low-frequency baseband signal using a high-frequency carrier frequency. There are both analog and digital modulation schemes, but both are mechanisms for communicating data across a communication channel (such as over the air between a pair of antennas). Which modulation type is used traditionally depends on the characteristics of the communication channel (for example, if it is quickly time-varying), and the constraints of the radio architecture. The modulator and demodulator in the transmitting and receiving radios have architectures that are specific to the modulation type. Radio architectures are typically constrained by size, weight, and power (SWaP), and these metrics motivate the use of only a single modulation scheme in traditional radios.

Modern communication system architectures increasingly allow a radio to use one of several modulation schemes. These Cognitive Radios (CRs) can employ a variety of modulations, allowing the radio to spontaneously move between modulation schemes [2]. A receiving radio often may therefore not know what modulation a transmitting radio is using. Additionally, a radio may wish to use channel sensing to identify the types of signals used by nearby emitters, and will not know the modulation schemes used in those cases.

The simplest receiver design for an unknown modulation scheme involves a brute-force approach: run parallel demodulators for each modulation type of interest, knowing that only the demodulator corresponding to the correct modulation has a chance to be successful. This can use a tremendous amount of power and is therefore undesirable. A more attractive design would involve determining the modulation scheme in use by simply analyzing the incoming RF signal energy. Machine learning can help to achieve this result.

Our goal is to implement existing work [1] by applying Convolutional Neural Networks (CNNs) to the problem of modulation classification. Our algorithm takes a complex time-series of RF signal energy as an input to our CNN, and outputs a one-hot encoded vector indicating a probability distribution to be used to classify the modulation scheme in use as one of several possible modulation types.

2. RELATED WORK

Prior to the use of machine learning for Automatic Modulation Classiciation (AMC), the use of pairwise log likelihood ratio tests was suggested to be effective for modulation classification [3]. Despite positive results in simulation, the implementation would scale poorly to scenarios involving many possible modulations.

The most notable paper related to the use of machine learning for AMC is *Convolutional Radio Modulation Recognition Networks* by O'Shea et. al from 2016 [1]. The authors used a dataset they created synthetically, "introducing modulation, pulse shaping, carried data, and other well characterized transmit parameters identical to a real world signal" [1]. They showed that CNN models outperformed benchmark integrated cyclic-moment based feature classifiers, especially at low signal-to-noise ratios (SNR). Their results are shown in Fig 1.

The focus of our project was to use a CNN architecture and compare our results with [1]. The authors [1] made their RadioML 2016.10A dataset available for free [4], and we used the same dataset for our project.

In 2018, the same authors looked [5] at the addition of

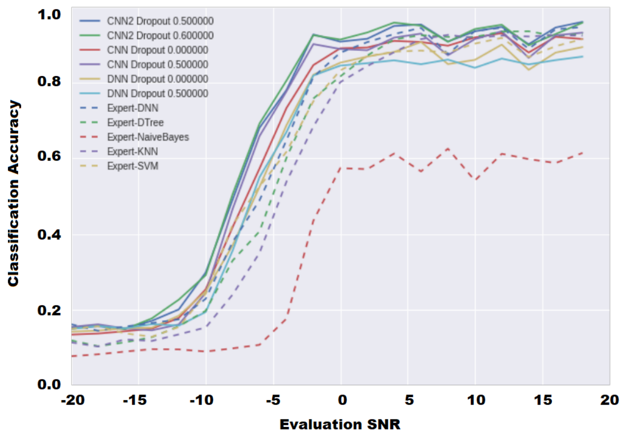


Fig. 1. Model accuracy from [1] as a function of SNR. CNN and CNN2 models outperform deep neural net (DNN) and expert benchmark models, especially at mid-range SNRs.

Analog Modulation Types	Digital Modulation Types		
AM - SSB	BPSK	QAM4	PAM4
AM - DSB	8PSK	QAM16	QAM64
WBFM	CPFSK	GFSK	

Table 1. Modulation types in RadioML 2016.10A dataset [4].

deep learning (DL) approaches and collected actual over-the-air RF signals for their dataset. The authors [5] saw improved performance compared with their previous paper due to the addition of convolutional layers and the use of longer time series (1024 compared with 128 used in 2016 [1]).

In 2018, Tang, et. al found additional performance improvements using auxiliary classifier generative adversarial networks (ACGANs), compared to the benchmark CNN model AlexNet [6].

3. DATASET AND FEATURES

We used the RadioML 2016.10A dataset [4] for our project. This dataset consists of radio signals that were synthetically generated using GNU Radio [7]. It consists of 11 modulation types (3 Analog Modulation and 8 Digital Modulation schemes) which are listed in Table 1, collected over 20 SNR conditions ranging from -20dB to +18dB. For each combination of modulation type and SNR value, the dataset contains 1000 complex radio signal data examples, each with a length of 128 samples, implying the size of the dataset is (20*11*1000, 2, 128). One example complex time series from our dataset is plotted in Fig 2. The dataset is stored in a pickle file, which is a compressed version of a python dictionary. The keys of the dictionary are a python tuple of (modulation type, SNR value) which correspond to a 1000*2*128 numpy [8] array of floating point numbers.

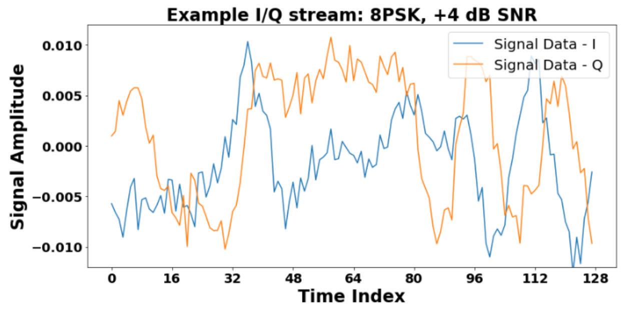


Fig. 2. Example 128-sample complex time series with 8PSK modulation at +4 dB SNR.

2D Convolution	Channels = 256, Kernel size = 1*3
Batch Normalization	Channels = 256
ReLU	
2D Convolution	Channels = 80, Kernel size = 2*3
Batch Normalization	Channels = 80
ReLU	
Flatten	
Linear	Features = 256
ReLU	
Linear	Features = 11

Table 2. Architecture of our initial CNN model.

4. METHODS

4.1. Initial Approach

4.1.1. Initial CNN Architecture

The initial architecture of our neural network that resulted in decreasing loss per epoch is described in Table 2. It is a simple CNN, with 2 convolution layers and 2 linear layers. A rectified linear unit (ReLU) function was used in between the layers. We implemented it using PyTorch [9].

4.1.2. Initial Hyperparameters

We initially trained our model for 50 epochs. We used the Cross Entropy Loss function, which calculates softmax values internally in PyTorch [9]. We used the Stochastic Gradient Descent algorithm for our optimizer and set the initial learning rate to 0.01. The training batch size was kept at 128 data points.

4.1.3. Positive SNR Training vs All SNR Training

We explored two alternative training approaches. Our first approach involved training our neural network using a 50:50 train/test split on the entire dataset. Our second approach involved using a 50:50 train/test split on positive SNR data (0 to +18 dB) but reserving all negative SNR data (-20 to -2 dB) for testing.

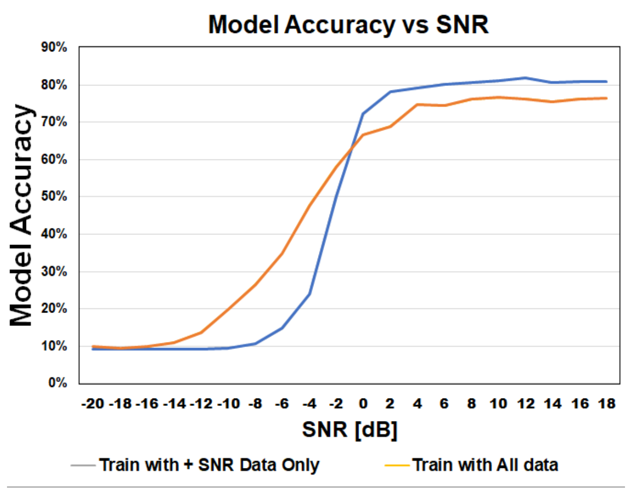


Fig. 3. Relative model performance when omitting negative SNR data from training.

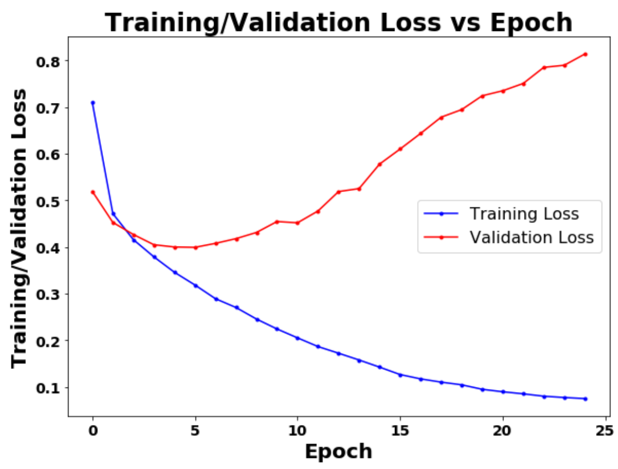


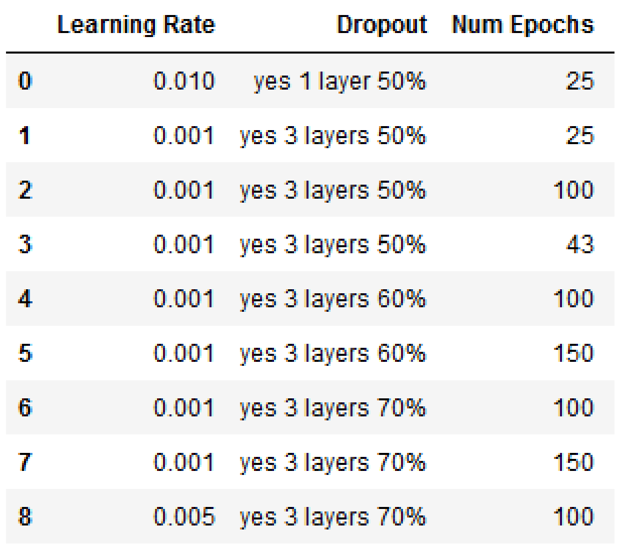
Fig. 4. Training and Validation Loss vs Epoch for initial model, indicating severe overfitting.

Our hypothesis was that our model performance would be hindered by training on data with negative SNR. We suspected that a model using negative SNR data for training would inadvertently "learn" the noise as features of the data, which would cause the model to perform worse.

The performance of our model for both approaches can be seen in Fig 3. The model trained using only positive SNR data has a higher peak accuracy, although it is notable that the model trained on all data performs better for low SNRs.

4.1.4. Overfitting

After implementing a separate train/test split to facilitate validation testing, our training and validation loss curves showed significant overfitting, as shown in Fig 4. We explored the addition of dropout during hyperparameter tuning.



	Learning Rate	Dropout	Num Epochs
0	0.010	yes 1 layer 50%	25
1	0.001	yes 3 layers 50%	25
2	0.001	yes 3 layers 50%	100
3	0.001	yes 3 layers 50%	43
4	0.001	yes 3 layers 60%	100
5	0.001	yes 3 layers 60%	150
6	0.001	yes 3 layers 70%	100
7	0.001	yes 3 layers 70%	150
8	0.005	yes 3 layers 70%	100

Table 3. A table listing a subset of the hyperparameter combinations we tried. Row 2 lists the hyperparameters using in our final model.

4.2. Hyperparameter Tuning

We attempted to improve our model's classification accuracy by tuning the hyperparameters of our model. The hyperparameters we tuned included the learning rate, the number of epochs, the number of dropout layers, and the dropout percentage. We tested many combinations of hyperparameters, a subset of which are shown in Table 3. Our model performed best with the set of hyperparameters listed in row 2 of Table 3.

5. RESULTS

After determining the best set of hyperparameters from tuning, we retrained our model using a 70:30 train/test split for positive SNR data and a 0:100 train/test split for negative SNR data. Our final model accuracy for each SNR is shown in Fig 5. Our model has a classification accuracy that peaks at 83.5% for input data with an SNR of 12 dB. This is comparable to the average accuracy of 87.4% achieved by O'Shea et. al [1].

To better visualize our model's performance on the variety of SNR data and the trends in misclassification, we created a confusion matrix of our model's performance for every SNR. In Fig. 7. a) for -2 dB SNR data, we can almost observe a recognizable diagonal in the confusion matrix signifying that our model can often correctly classify modulations even at slightly negative SNR data. In Fig. 7. b) for +8dB data, we notice the strong diagonal with some small outliers. Our model sometimes mixes up the classifications for QAM16 and QAM64, two very similar digital modulation types. Furthermore, we can highlight that our model sometimes misclassifies WBFM for AM-DSB which is consistent with O'Shea et. al's results [1]. The trends in the confusion matrix in Fig. 7.

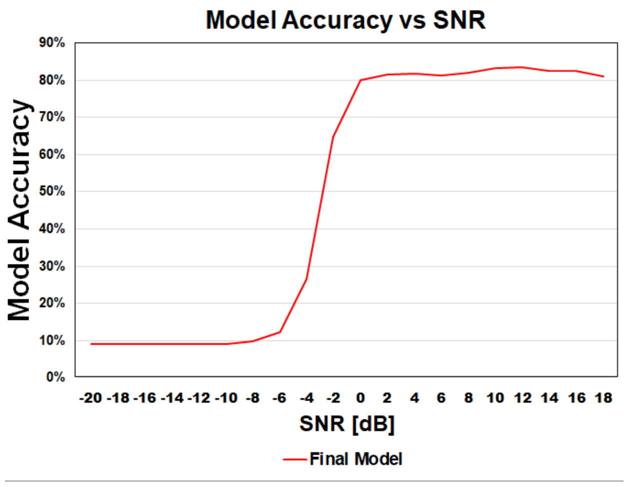


Fig. 5. SNR vs Accuracy for our final CNN model with Dropout (50%) layers added after each Conv2D layer.

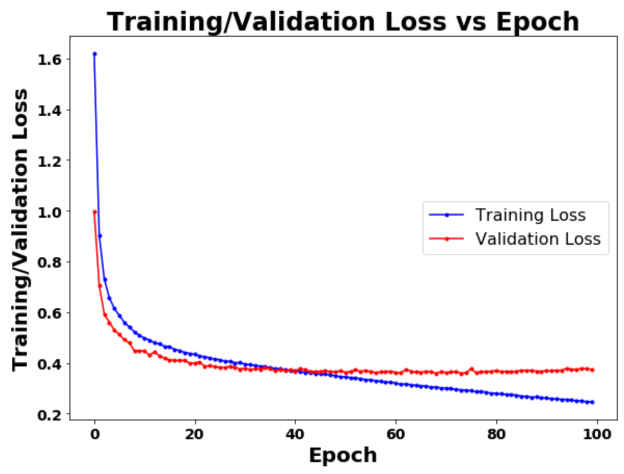


Fig. 6. training loss and Validation loss variation with epochs

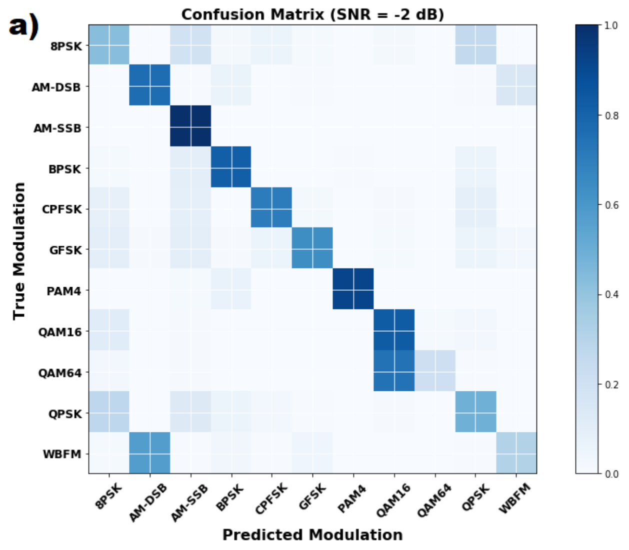
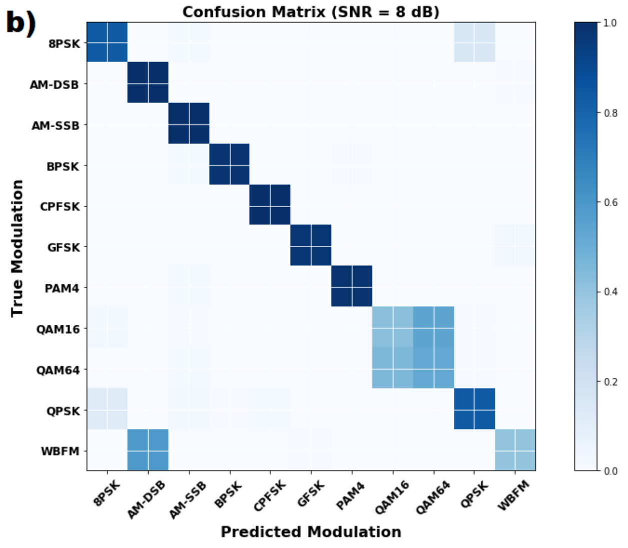



Fig. 7. **a)** Confusion Matrix from our model at -2 dB SNR. **b)** Confusion Matrix from our model at +8 dB SNR.

b) are the same as those in the confusion matrixes we produced for all higher SNRs (10 to 18 dB), which affirms the consistency of our model across high SNR data. On the other hand, for low SNR data (-18 to -6 dB) our confusion matrixes show an almost entirely vertical bar, suggesting no ability to classify the modulation. Notably, we observe that the model from O'Shea et. al [1] was able to correctly classify between QAM16 and QAM64, suggesting this might be the source of our relatively lower classification accuracy.

6. CONCLUSION

We investigated the application of a CNN model for modulation classification, following the work of O'Shea et. al [1]. We started with a simple model, and then tuned our hyperparameters to improve our model performance. We examined the effect of training on only positive SNR data compared with training on data of all SNRs and concluded that training on positive SNR values is beneficial and pragmatic. As we have seen in [3], a non-Machine Learning (ML) approach, is cumbersome, and by exploring a ML approach we have furthered the possibility of ML implementations in Wireless Communications.

In the future, we would like to explore the possibility of using deeper CNN's (such as ResNet33 [5]), different feature maps (channels), using an expansive dataset such as RadioML 2018.01A [4], which is based on real radio signals. In addition we would also like to experiment with other hyperparameters such as the batch size and optimizers (such as Adam, RMSProp, or Adagrad).

7. CONTRIBUTIONS

Kartik: Dataset input and manipulation, initial model training and significant model debugging, dataset, methods, neural network sections of presentation and paper.

Young: Model improvements and hyperparameter tuning, adding validation testing for overfit detection, results and future work sections of presentation and paper.

Parker: CNN model implementation, confusion matrix calculation and generation, abstract, background/introduction, related work, references, and presentation review response sections of presentation and paper.

8. REFERENCES

[1] Tim O'Shea, Johnathan Corgan, and T. Clancy. Convolutional radio modulation recognition networks. In *International Conference on Engineering Applications of Neural Networks*, pages 213–226. Springer, 2016.

[2] S. Haykin. Cognitive radio: brain-empowered wireless communications. *IEEE Journal on Selected Areas in Communications*, 23(2):201–220, 2005.

[3] P. Panagiotou, A. Anastasopoulos, and A. Polydoros. Likelihood ratio tests for modulation classification. In *MILCOM 2000 Proceedings. 21st Century Military Communications. Architectures and Technologies for Information Superiority (Cat. No.00CH37155)*, volume 2, pages 670–674 vol.2, 2000.

[4] Deepsig Inc. Rf datasets for machine learning. `https://www.deepsig.ai/datasets`. Accessed: 2020-04-21.

[5] T. J. O'Shea, T. Roy, and T. C. Clancy. Over-the-air deep learning based radio signal classification. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):168–179, 2018.

[6] Bin Tang, Ya Tu, Shaoyue Zhang, and Yun Lin. Digital signal modulation classification with data augmentation using generative adversarial nets in cognitive radio networks. *IEEE Access*, PP:1–1, 03 2018.

[7] GNU Radio Website. `http://www.gnuradio.org`, accessed February 2012.

[8] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[9] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

9. REVIEW COMMENT RESPONSES

9.1. Responses to Comments From Group 4

Strengths:

- Very clear explanations on defining the problem and models.
- Good explanation on model architectures.
- Interesting hypothesis on omitting worse performed negative SNR data for training to focus on high accuracy data.

Thank you.

Some Advice:

- It would be better to only write shorter words instead to very detailed explanation on each slide. The word dense on each slide is a little bit too high and some of the words with graph is too small in terms of a presentation, which is blurred in the videos.

We will take that into consideration during future presentations.

- Omitting worse performed negative SNR data for better model is impressive, but could you justify further on what will happened to your model on overall performance if not doing so?

We addressed this during the presentation. When training on negative SNR data the model may "learn" noise thinking that it is a feature of the data, which harms the model accuracy.

- Each slide takes longer than 1 minutes. It would be better to divide some of those single slide contents into a couple to different one and limit overall time of presentation + code review a little bit. It could help the audience to focus on which part you are talking about.

We will take that into consideration during future presentations.

- You split the prediction accuracy of model with different SNR, that is good, but I wish to find your overall training/validation accuracy graph for each epoch as a trend. The given image in the 11 th page of ppt shows the result in papers, it would be also nice that you could add your result into it for comparison.

We are confused by your feedback. The image on slide 11 from [1] shows prediction accuracy as a function of SNR. We showed the same plot from our own model on slide 10, and it is on page 3 of our report. Plotting overall accuracy is not especially useful, because it depends significantly on the SNR of the input signal. We showed our loss per epoch for both training and validation to illustrate model training and the presence of any overfitting on slide 10, and it is on page 3 of our report.

- Mentioning further work with improvement is good, but it is confusing that you mentioned the work you are trying to reproduce as the very initial one. My impression is that a couple of people had made some models, and you choose the oldest baseline model to reproduce. Maybe you could also find some even older papers that also focus on this problem and shows the improvements of your paper for comparison.

We indicated explicitly during our presentation that the paper we were emulating was the first paper on this subject.

- Maybe you could try with different architecture rather than a straightforward CNN model in order to train the data and get higher accuracy on those low or negative SNR data, for example adding deeper layers. It would be much better to discover some your new improvements than simply using original CNN from the authors.

We explored a wide range of hyperparameters as indicated in Table 3 and described on Page 3, including adding extra layers, in an attempt to improve our model.

- When you mentioned that author use 50 60% of dropout layers but you are not, you could go deeper to justify your reason, and the pros and cons on your choice over the authors.

At the time that we made our video, exploring dropout was still in future work. We have since implemented dropout, which reduced overfitting as expected, and this is shown on Page 4.

9.2. Responses to Comments From Group 63

The topic is about finding better solutions to intelligently detect modulation schemes. This was proposed via an automatic modulation classification (AMC). They used the RadioML_2016.10A dataset which consisted of 220000 synthetically generated radio samples ranging from -20dB to 18dB. There are 11 modulation types, which were used as the classes for classification. There existed a single CNN architecture, with a daily impressive accuracy in the end.

- Maybe experiment with the hyperparameters in your CNN. You seem to have only tried one architecture, although it did seem to perform well.

We explored a wide range of hyperparameters as indicated in Table 3 and described on Page 3 in an attempt to improve our model.

- Talk about how you came up with the architecture, and whether you borrowed any aspect of it from another paper.

We were exploring the approach done by another paper [1], and explained this during our video. However, our exploration of CNN architectures (number of layers, feature maps, etc.) was done by trial-and-error during the process of hyperparameter tuning.

- Why did you choose to have a train test split of 50-50?

That was simply where we started. After making our video, we ended up using a train/test split of 70:30 on positive SNR data, and a train/test split of 0:100 on negative SNR data, as a result of hyperparameter tuning.

- The observation and results section was extremely well written. The explanation of overfitting and then tuning it to account for it via dropout layers makes a lot of sense. It was easy to follow.

Thank you.

- For the results section, maybe include some sort of chart / confusion matrix that can clearly show which signals were misclassified.

Definitely. This was discussed during the presentation as being part of future work, and is included on page 4 of our paper.

- The walk through the code is very clear. That part is extremely well done. Each contributor said a little about the structure, which made it easy to follow. Maybe next time a short demonstration with minimal epochs can be used.

Thank you.

9.3. Responses to Comments From Group 76

The background and the motivation for the project has been explained really well including the breakdown of the dataset that they are using. The group has developed a CNN network. The experiment of training only on positive SNR is interesting and having worked on a similar experiment, we can verify that training only on the positive SNR value does improve the performance. The plots and figures have been explained clearly and even the negative results have been explained and not ignored. They also showed the live demo of the code execution and their code looks clean and well commented.

Thank you.

Some suggestions and doubts:

- Although they did explain why ML can help, they could also cite some old papers when ML was not used and what techniques did they use back then for modulation classification.

Great idea. Added a discussion in Related Work on a pre-ML approach to modulation classification, on Page 1.

- The group mentioned that they have used sigmoid at the output layer. As far as we know, sigmoid is used for binary classification whereas softmax is used for multiple classes' classification. It could be a typo.

It was a typo. We used softmax.

- The train test split of 50:50 could be changed to 80:20 or 70:30 because it's considered better to have a bigger training size.

After making our video, we ended up using a train/test split of 70:30 on positive SNR data, and a train/test split of 0:100 on negative SNR data, as a result of hyperparameter tuning.

- They can also play around with optimizers (Adam, Adagrad), learning rate and even the batch size. Maybe, they can tweak the learning rate and modify it with epochs if they observe plateauing of validation loss.

We explored a wide range of hyperparameters, as indicated in Table 3, in an attempt to improve our model. Some of these we described in our Conclusion for future work if we had more time.

- Can also experiment with max pooling layer.

We explored a wide range of hyperparameters, as indicated in Table 3, in an attempt to improve our model.

- Maybe, if time permits, also try out a different model/ layer like LSTM to better recognize the temporal patterns in the data.

We would love to explore this, but used our remaining time to tune the hyperparameters of our model, construct confusion matrices, explore dropout, and write our paper.