Comparing Convolutional and Transformer Models for Signal Classification

Hussein Adi - Coursework Sample

Introduction and Background

Using neural networks for signal analysis may initially appear counterintuitive. The properties of signals, such as frequency and amplitude, can be directly computed. Yet advanced signals store data via modulation, such as by a series of waves along a set of frequencies. This introduces a complex task; the data from the signal cannot be retrieved without the correct modulation technique, which is not always known. For example, sensitive environments, such as for IoT or 5G networks, may have unknown signals that cause interference. Neural networks can be used to learn the characteristics of such signals and identify them. Another significant use case is in healthcare, such as using NNs to identify and classify abnormal heart activity [4].

This project aims to compare two models for the task of RF signal classification: a convolutional model and a transformer-based model, in order to determine whether the less-conventional transformer architecture can perform better than a CNN. Convolutional neural networks are more commonly used in signal classification tasks, such as in Mozilla's DeepSpeech model. We will build and train both a conventional CNN model as well as a transformer-based model, with the goal of designing the transformer model to perform better than the CNN of similar complexity.

Objectives

The primary objective is to determine whether a transformer-based neural network can perform better on signal classification tasks than a more conventional CNN model. We selected a CNN for our conventional model for two main reasons: its suitability for finding patterns in gridded data, and for its proven ability in signal classification tasks such as voice recognition and heart arrhythmia [4]. The incentive for attempting to achieve the same or better performance with transformers is to utilize their ability to capture different aspects of signals; modelling long-term trends could be crucial in classifying signals with slower modes or classifying signals whose identifiable features are spread along its sample - hence the need for memory and attention.

Validation and Performance

Our main methods of comparing and validating the performance between models will be their performance on validation sets during training and the held-out test set after training. The dataset consists of 172,800 labelled rows of 18 different signal types; 25% of the data will be left out solely for test purposes. Some of the signals in the dataset are similar. For example, the main difference between PSK31 and PSK63 is only the rate in which they transmit data. They both use Phase-Shift Keying (PSK) for modulation and their signals are expected to be similar. To identify how well our models distinguish between similar signals, as well as to identify which signals are the most confusing, we will construct a confusion matrix after classifying the test set to further assess performance.

Methods

The project will use the following radio signals dataset at panoradio-sdr.de/radio-signal-classification-dataset/. This is a synthetic dataset created specifically for machine learning tasks. The signal data is stored as a single `.npy` file which contains a 2-dimensional numpy array. This array consists of 172,800 vectors of different signals. Each vector has 2048 In-phase Quadrature (IQ) samples, which are complex number representations of the signal. Each signal has a frequency of approximately 6000 ± 250 Hz. The signals have various signal-to-noise ratios, ranging from -10 to +25. These noise ratios combined with frequency offsets introduces noise to the dataset, intended for the models to generalize better. The numpy array itself does not contain labels, but a `.csv` file is also included in the dataset which contains the corresponding labels.

Table 1: List of signal modes in the panoradio dataset.

ID	Mode Name
0	Morse Code
1	Phase Shift Keying PSK31
2	Phase Shift Keying PSK63
3	QPSK31
4	Radio Teletype RTTY 45/170
5	Radio Teletype RTTY 50/170
6	Radio Teletype RTTY 100/180
7	Olivia 8/250
8	Olivia 16/500
9	Olivia 16/1000
10	Olivia 32/1000
11	DominoEx
12	Multi-Tone 63
13	Navigational Telex Navtex
14	Single-Sideband (upper)
15	Single-Sideband (lower)
16	AM Broadcast
17	Radiofax

The table above lists the signal classes in our dataset. Most of them are radio signals in the high-frequency band meant to transmit text, with the classical example being morse code, but notably it includes modes such as radiofax and AM. The data being transmitted is not fixed; for example, the text modes encode a variety of messages, the AM signals represent a range of speech and music from different genres.

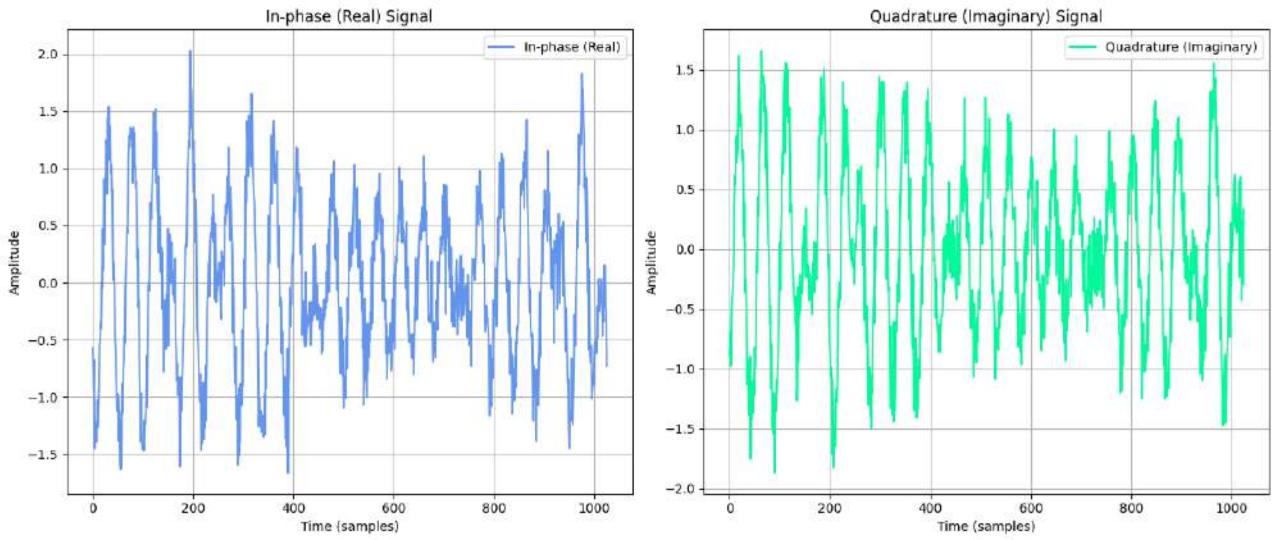


Figure 1: Visualization of a PSK63 signal over 1024 samples.

Figures 1 and 2 illustrate examples of two different signals from the dataset: Phase Shift Keying (PSK) and Radio Teletype (RTTY). RTTY has a more complicated encoding, hence its more complex signal, although PSK transmits at a faster rate.

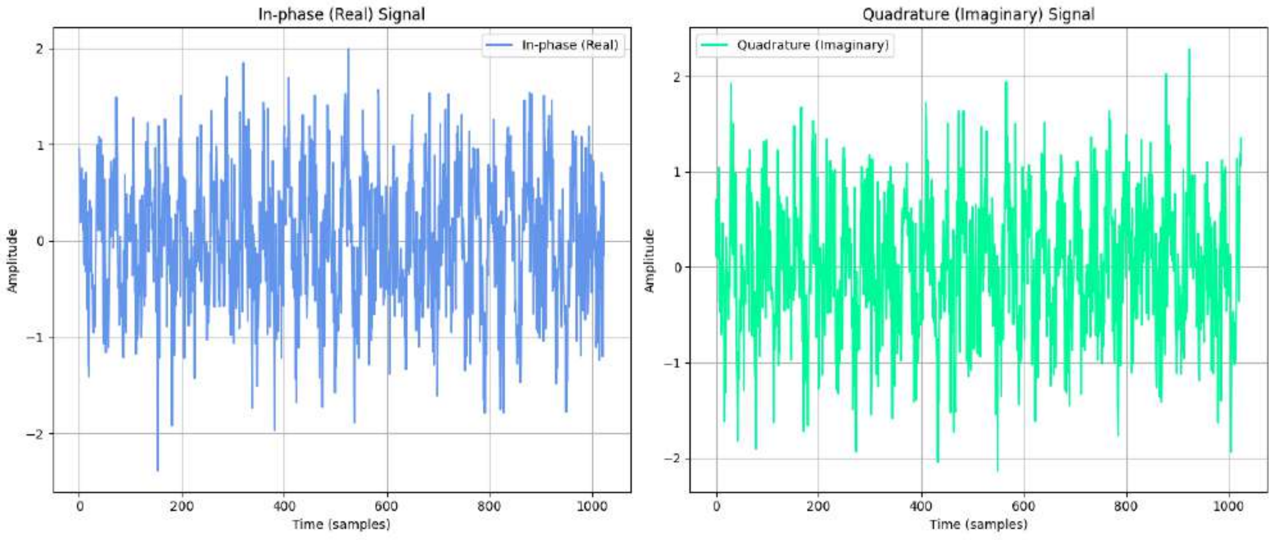


Figure 2: Visualization of a RTTY 45/170 signal over 1024 samples.

Technical Overview

The implemented code consists of classes in Python. The code depends on numerous machine learning and data libraries, mainly `TensorFlow.Keras`, `scikit-learn`, `numpy`, and `pandas`. The signal data is pre-processed by a custom data generator class. It loads the data and labels into memory, converting them to numpy arrays. It then splits the samples into real and imaginary features, and one-hot encodes the labels. Finally, the dataset is shuffled and then split into training, validation, and test sets.

Later, this generator class was modified to augment the data by adding amplitude and phase features to each sample, increasing the number of features from two to four. The effect this had on the models is explained in the results section of this report.

Convolutional Neural Network

The CNN model is contained in its own class. Using the Keras sequential API, we feed the data as a 1D array into a convolutional layer with ReLU activation. Batch normalization was added between layers to normalize the inputs from layer to layer. A dropout rate of 0.2 was included to account for unwanted ”noisy” features, but at the same time we wanted to avoid heavy dropouts since our models would need to capture many features given the different signal classes. The code below shows how this was implemented in the source code.

```
model = Sequential([
         Conv1D(filters=32, kernel_size=5, activation='relu', input_shape=input_shape),
         BatchNormalization(),
         MaxPooling1D(pool_size=2),
         Conv1D(filters=64, kernel_size=5, activation='relu'),
         BatchNormalization(),
         Conv1D(filters=128, kernel_size=5, activation='relu'),
         BatchNormalization(),
         GlobalAveragePooling1D(),
         Dense(64, activation='relu'),
         Dropout(0.2),
         Dense(num_classes, activation='softmax')])
```

Transformer-based Network

The transformer model is also built as a Python class. The transformer layer and the rest of the model were constructed with the Keras functional API. Initially, we attempted to build this using Keras's in-built `TransformerEncoder` layer, but elected to create our own custom layer to have more control over fine-tuning and the layers within the transformer. The code below displays how this was implemented. `num_heads` is the number of heads in our transformer layer, with each head used to "pay attention" to different aspects of the signal. `key_dim` is the size of the vector representing the Q-K-V values.

```
def transformer_encoder(self, inputs, num_heads, ff_dim, dropout_rate):

    attention = MultiHeadAttention(num_heads=num_heads, key_dim=ff_dim)(inputs, inputs)

    attention = Dropout(dropout_rate)(attention)
    attention = LayerNormalization(epsilon=1e-6)(inputs + attention)

    feedforward = Dense(ff_dim, activation="relu")(attention)
    ff = Dropout(dropout_rate)(feedforward)
    ff = Dense(inputs.shape[-1])(ff)
    ff = Dropout(dropout_rate)(ff)
    ff = BatchNormalization()(ff)
    ff = LayerNormalization(epsilon=1e-6)(attention + ff)

    return ff

def build_model_A(self, input_shape, num_classes):
    inputs = Input(shape=input_shape)
    x = self.transformer_encoder(inputs, num_heads=2, ff_dim=64, dropout_rate=0.3)
    x = GlobalAveragePooling1D()(x)
    x = Dense(64, activation="relu")(x)
    outputs = Dense(num_classes, activation="softmax", dtype="float32")(x)
    model = Model(inputs, outputs)
    return model
```

For both models, `categorical_crossentropy` was used for loss calculations, as we are classifying multiple classes, and used the `adam` optimizer. Models were trained over 10 epochs.

```
self.model.compile(optimizer="adam",
                   loss="categorical_crossentropy",
                   metrics=["accuracy"])
```

Results

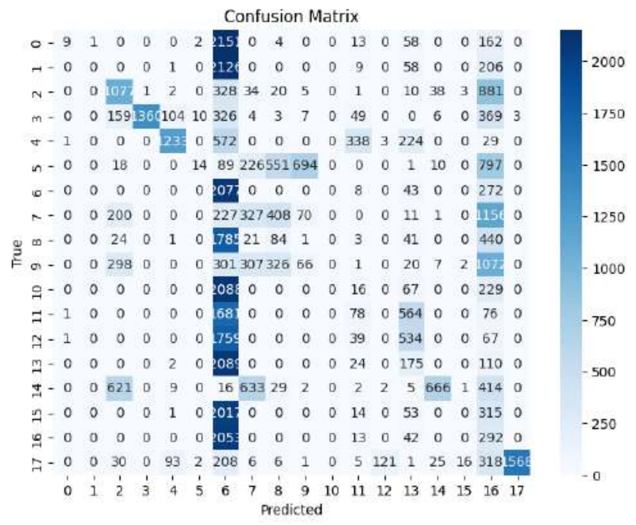
Initial Tests

Initial tests on both the CNN and transformer model proved to be unsatisfactory, and somewhat inconclusive given their poor performance. During these initial tests, the only pre-processing of the data involved was splitting each complex sample into its real and imaginary parts and one-hot encoding the labels. The CNN model consistently performed better than the transformer-based model, despite efforts to achieve similar training times, but overall, both models performed poorly. Note that, given 18 signals classes, a random guess for has an expected accuracy of 5.56%.

Model Architecture	Final Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
CNN	42.19	20.57	15.41
Transformer	9.80	9.56	9.60

Table 1: Training, validation, and test accuracies of the base CNN and transformer models.

The table above shows that, on unseen data, the CNN is only performing twice as well as random guesses. Despite dropouts, the model heavily overfits, and exhibits very low performance on the validation data. The transformer model proved difficult to tune and performed poorly. Changing regularization in the dense layer and learning rates resulted in little effect. The transformer was a poor performer on all sets. During training, it only increased its training accuracy from 7.08% to 9.8%.


(a) Confusion matrix for the initial CNN model

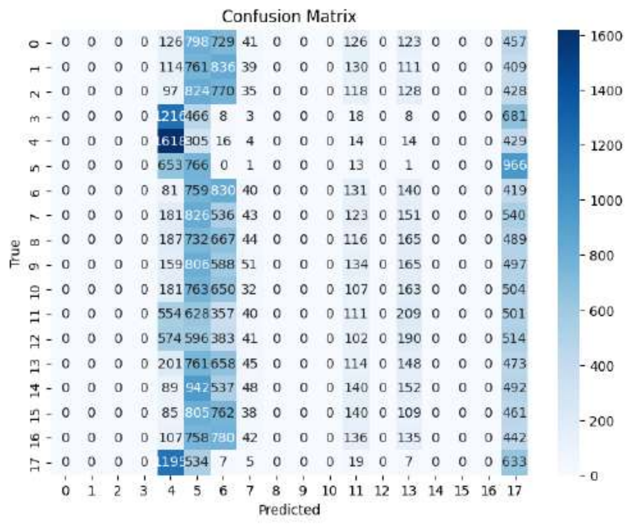

(b) Confusion matrix for the initial transformer model

Figure 3: Confusion matrices for CNN and transformer models.

The figure above displays the resulting confusion matrices resulting from predicting unseen test data, and provides us some insight as to what is causing this poor performance. Ideally, most of the values should be in the top-left to bottom-right diagonal, or at least the largest number in each column should fall on this diagonal. This is the case for some signals; PSK31, PSK63, and QPSK63 are predicted with high accuracy. QPSK31 in column 3 is predicted with almost perfect accuracy: in only one occaision did the model predict QPSK31 falsely. The QPSK31 row, however, shows that there were many cases where other signals were predicted falsely, when the true signal was QPSK31. The model is accurate in predicting radiofax; whilst also exhibiting horizontal "spread" of predictions, there are only 3 test cases out of 2000 in the test set where the model predicted 17 and it was not. The most striking features of this matrix is its overwhelming tendency to predict RTTY 100/180 (6) over other types; this signal most likely shares common features with most other signals. On non-augmented data, the transformer model performed poorly, even more so than the CNN model. This was in spite of our attempts to tune hyperparameters without adding a significant number of layers as this resulted in huge increases in training times. The transformer is less biased towards predicting RTTY 100/180 but overall has extremely poor performance. For many signals, despite thousands of examples in the test set, it does not predict them at all.

Intermediary Improvements

Given the low performance of both models, we considered augmenting our dataset. Each individual signal has 2048 samples over time, and there are thousands of each signal type in the data set. As such, insufficient training data by volume was ruled out as a primary reason for this performance. We did not know how to reproduce the data in order to augment it, and therefore it was not an option. Additionally, this data was set up to be ideal for classification tasks. We suspected that the problem could be the number of features of the model; only two features explicitly change over time. We considered that this may not be sufficient, and as such we chose to augment the data. The augmentation method was straightforward: using the separated real and imaginary channels of the IQ signal, the amplitude and phase of the signal for that sample were calculated using the formulas below.

$$Amplitude = \sqrt{Real^2 + Imaginary^2}$$

$$Phase(\phi) = \tan^{-1}\left(\frac{Imaginary}{Real}\right)$$

These were added separately as new features, increasing the number of the features from 2 to 4. Additional augmentation measures were considered, such as phase difference (the difference between the phase of successive samples), or other augmentations involving the rate of the change of the signal, but we chose not implement additional data augmentations and instead focus on improving our model architecture.

Building on the initial models as shown in the methods section, we improved their architecture. We recognized that the random signal-to-noise ratio in the dataset may cause the models model features that were simply noise, so for the CNN model, we increased dropout from 0.2 to 0.5. Additionally, we modified our pooling from global average pooling to max pooling. For the transformer, we observed slightly better performance with lower dropouts, so we decreased it from 0.3 to 0.2. We also added normalization between layers, and continued to fine-tune our Q-K-V dimension and number of heads. Adjusting the learning rate did not reliably increase performance, so we left this as default for both the CNN and transformer.

Quantitative Results

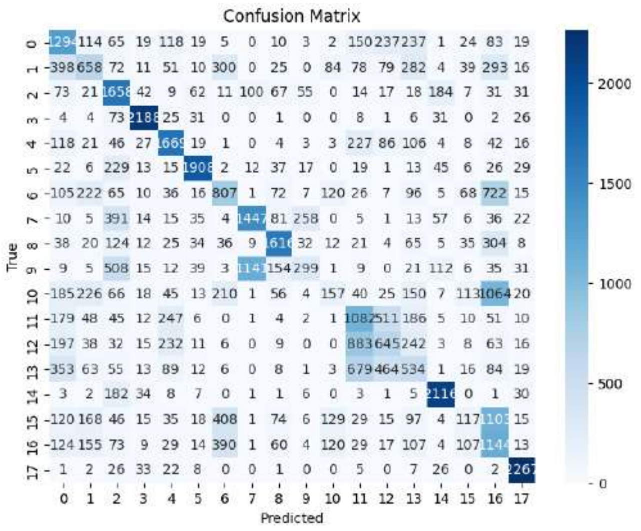
Model Architecture	Final Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
CNN	37.27	49.72	50.01
Transformer	16.78	16.11	16.42

Table 2: Updated model accuracies after training new models on augmented data.

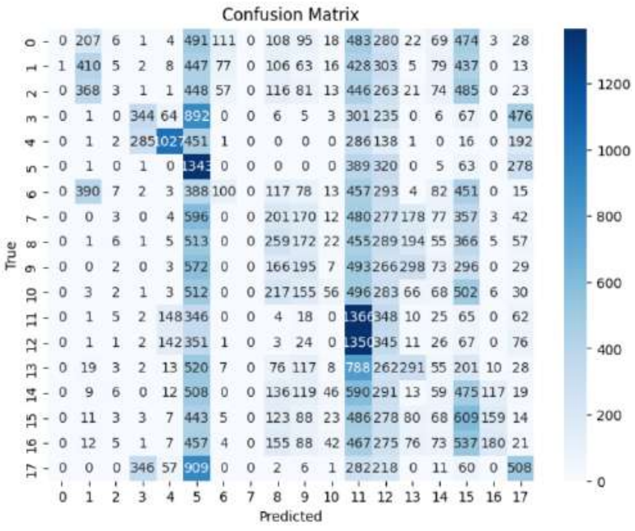
Augmenting the data and fine-tuning the model parameters, as well as normalizing before activation, resulted in significantly improved performance. The data augmentation alone increased the accuracy on the test set to 41% for the CNN and 14% for the transformer model - this had a greater effect on improving the overall accuracy than increasing the size of the models and adding normalization between layers.

Qualitative Analysis

The confusion matrices indicate where the models continue to struggle. The CNN, while improving considerable, still struggles with low predictions for Olivia 32/1000, most often confusing it with AM (16). A small, 3x3 square is apparent where the model frequently confuses DominoEx, Multi-Tone, and Navtex with one another (11, 12, 13). However, a diagonal is clearly visible and this is a significant improvement from the sparse predictions in the earlier matrix. We also observe a decrease in accuracy for radiofax. Our initial model was exclusively accurate in labelling radiofax signals in the test set. The updated model has a few hundred predictions for radiofax across all other signals, but still exhibits strong accuracy overall when predicting radiofax.



(a) Confusion matrix for the updated CNN



(b) Confusion matrix for the updated transformer model

Figure 4: Confusion matrices for updated CNN and transformer models trained on augmented datasets

We observe a weak diagonal in the confusion matrix for the updated transformer model. Much of the same characteristics persist, with minor improvements. We can see the same vertical "striations" of predictions for certain signals, indicating the model predicts these the most often. The major improvement is, rather than 0 predictions for numerous signals, the model is now at the very least predicting most signals, except for morse code and Olivia 8/250. It is best at labelling RTTY 45/170 and radiofax.

Whilst not included directly as a measure of accuracy, our final transformer model saved as a `.keras` file was larger than CNN model by over 1.5 times (140kb to 256 kb), and took on average twice as long to train. These may be attributable to the architecture of the models and the way they are stored, but factoring this into the analysis could indicate the additional complexity required for the transformer model to achieve accuracies similar to that of the CNN. It is difficult to directly compare the two.

Limitations and Implications

Technical Limitations

This experiment was limited by the level of complexity in both models, which in turn was limited by memory and GPU constraints; this, however, was more limiting for the transformer model. Access to a GPU allowed us to repeatedly tune the CNN over various head sizes, layer numbers, and layer sizes, while keeping the training for each epoch under 5 minutes, allowing the model to train over ten epochs in less than an hour. This was not the case for the transformer model. Whilst we loosely attempted to build the two models with the same complexity based on dense layers and training times, we could not achieve transformer accuracies comparable to CNN accuracies without allowing the model more time to train. Moreover, the size of the dataset (5.5GB) meant that preprocessing could take upwards of a minute and was computationally intensive, requiring multiple temporary copies of the dataset, which consistently used all available RAM on the host machine. This meant that fine-tuning and updating the transformer model was significantly more tedious and we could not spend as much time as we would have liked to optimize it, which contributed to its poor performance.

Regarding the dataset, it would be better to use real instead of synthetically generated signals. For experimental purposes, synthetic data may be fine for testing which architecture is appropriate, but such a model is unlikely to be useful in any real-world scenario. Future experiments would benefit greatly from large, high-quality RF data. Additionally, we felt that the assortment of signals, although beneficial for generalizing, does not reflect real-world environments; the signals in the dataset are primarily amateur radio modes for transmitting text, along with a few other types.

Design Limitations

Whilst the limited computing power and insufficient complexity in the models were a technical limitation, a major design limitation was not augmenting the data enough. Augmenting the data resulted in significantly improved performance for our CNN. Rather than offloading the task of recognizing properties such as amplitude, phase, and baud rate, augmenting the data would allow the models to track these variations reliably, and these are basic augmentations.

When initially designing the experiment, we recognized that in order to learn to classify many classes, particularly similar ones, we would have to feed the class-wise results of the confusion matrices back into the original models. Roughly, this would involve finding the weights associated with such classes in the model, conducting Principal Component Analysis (PCA) in order to identify the most relevant ones, and fine-tune them for each class, or adding a bias to the neurons capturing the features of the most confusing signals, most notably Olivia 32/1000. We were unable to implement such a feature.

Finally, the lack of domain-related knowledge to apply signal transformations during preprocessing in order to augment the data was also a potential limitation. Augmenting the data via signal transformations should have been planned from the outset.

Future Work and Implications

Based on the limitations described earlier, and by drawing on the differences between this experiment and similar professional work, we can identify limitations and suggest the following improvements for future work:

- **Complex Models -** Similar work using neural networks for signal classification tasks involves highly complex-multilayer models. One study involving classifying sound waves, fed into an RNN as sound waves, used a multilayered recurrent neural network with a 512-dimension hidden layer and was trained over 50 epochs; a research paper on classifying amateur signals trained their model over 200 epochs [2] [3]. Evidently, any real work should involve significantly more complex models than what we were able to implement.
- **Real and Domain-specific Environment -** Professional approaches to signal classification typically target a specific set of signals, or in a specific environment that produces such signals. A similar experiment in classifying a mixture of digital and analogue radio signals was able to achieve 93.5% accuracy on low-noise signals, but with a smaller classification task of 11 signals [2].
- **Ensemble & Alternative Approach -** If the purpose of this experiment is to identify and test less-conventional methods for signal classification, then future similar work could build on more creative methods for the same task. One such way is to convert the signal data into an image, and then feed it into an image classification model; such an approach has already been implemented with success by researchers [1]. This, of course, would necessitate training multimodal models and would apply domain-specific knowledge. Research and experiments into using ensemble models, such as combining CNNs and LSTM, have been successful [2].

The outcome of this project has few implications, given the poor performance of the models. However, it has shown that, given a simple IQ signal, CNNs will significantly outperform transformer-based models of the same complexity due to their propensity to model grid-like data well. Given the improvements in accuracy after adding two features, we would recommend feature engineering to signal data before feeding it into a neural network. For a more robust experiment on classification, a transformer with larger dimensions and heads should be used to truly determine whether the low accuracy is attributable to hardware and time limitations as opposed to the inability of attention mechanisms to rapidly fluctuating signals.

Conclusion

We conclude that convolutional neural networks are more suited to the task of radio signal classification when compared to a similar transformer-based neural network, particularly with limited or ordinary hardware (e.g., no GPU). Despite the limitations of our experiment, we also conclude that the successful application of transformers for signal classification tasks requires additional feature engineering that could augment the data to allow the heads of the transformer to explicitly model characteristics of RF signals. Combined with increased complexity, such a project would provide a more comprehensive assessment of a transformer's capability for signal classification.

References

[1] Shichuan Chen et al. "Radio–Image Transformer: Bridging Radio Modulation Classification and ImageNet Classification". In: *Electronics* 9.10 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9101646. URL: https://www.mdpi.com/2079-9292/9/10/1646.

[2] Ziad Elkhatib et al. "Radio Modulation Classification Optimization Using Combinatorial Deep Learning Technique". In: *IEEE Access* 12 (2024), pp. 17552–17570. DOI: 10.1109/ACCESS.2024.3357628.

[3] Hiromitsu Nishizaki and K. Makino. "Signal Classification Using Deep Learning". In: July 2019, pp. 1–4. DOI: 10.1109/SENSORSNANO44414.2019.8940077.

[4] Xianlu Pan, Ying Wang, and Yu Qi. "Artificial Neural Network Model and Its Application in Signal Processing". In: *Asian Journal of Advanced Research and Reports* 17 (Jan. 2023), pp. 1–8. DOI: 10.9734/AJARR/2023/v17i1459.