CS230

Robust Atrial Fibrillation Detection Using Deep Neural Networks

Kelly Brennan & Noah Youkilis

kbrenn@stanford.edu & nbyouk@stanford.edu

Abstract

Automated atrial fibrillation (AF) detection aims to identify AF, the most common sustained arrhythmia, from electrocardiogram readings. In this work, we seek a robust implementation of such an algorithm, which would allow for efficient, autonomous population-level screening. We therefore employ deep learning to investigate two different automated AF detection algorithms, drawing inspiration from existing successful networks in the literature: a convolutional neural network and a recurrent neural network. Experimentation with network architecture and hyperparameter tuning illuminates benefits and drawbacks to each of these algorithms; however, the CNN-based architecture ultimately proves itself to be more accurate than the RNN with a faster evaluation (forward propagation) time.

1 Introduction

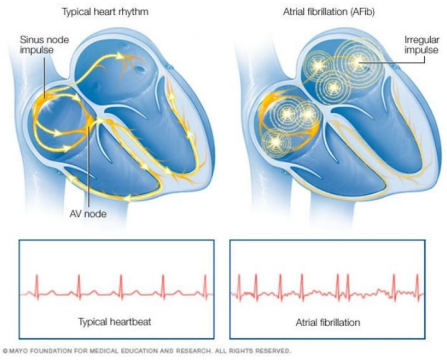

Figure 1: Examples of AF and regular rhythms.

Atrial fibrillation (AF or A-Fib) is the most common heart arrhythmia that often goes undetected, and even if it is detected, managing the condition may be challenging. AF is an irregular and often very rapid heart rhythm (arrhythmia) that can lead to blood clots in the heart [Benjamin et al., 1998]. The condition increases the risk of stroke, heart failure, and other heart-related complications [Mayo Clinic, b]. A pretrained neural network which classifies AF from ECG segments would have great utility in aiding the diagnosis of AF in that it could be deployed in hospitals, doctors' offices, at-home care, or wearable medical devices to quickly and efficiently screen or monitor a greater portion of the population.

In this work, we train both a recurrent neural network (RNN) and a convolutional neural network (CNN) to classify atrial fibrillation from electrocardiogram (ECG) segments. The RNN operates directly on an ECG segment, while the CNN first computes a Short-Time Fourier Transform (STFT). Both approaches are shown to be quite successful; however, after analysis summarized in this paper, we conclude that the CNN prevails. To this end, we start by conducting a literature review, after which we show our dataset along with preprocessing steps we take. Next, we discuss the model architectures employed, followed by results obtained from both algorithms. Finally, we conclude the paper, motivating why we believe a CNN is ultimately a better choice for the task of automated AF detection.

CS230: Deep Learning, Spring 2022, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

2 Related Work

A variety of algorithms have been introduced and explored in the literature for the purpose of AF detection, many of which claim impressive results. For the sake of brevity, we list here a few demonstrative case studies. Examples of successful CNN-based network architectures can be found in [Tutuko et al., 2021, Zihlmann et al., 2017]. An end-to-end approach was even explored in [Hannun et al., 2019]. Lastly, an RNN featuring a bidirectional LSTM was featured in [Faust et al., 2018a]. Notably, this approach does not operate directly on ECGs, but rather on RR intervals, or the time between successive R peaks of an ECG. Such an approach enables the use of an RNN; as we will show below, passing entire ECG sequences (in our particular case, up to 30,000 values) into an RNN is quite costly. However, operating only on RR intervals (1) precludes the direct use of an ECG signal, and (2) operates on only one feature of an ECG, rather than extracting a multitude of features, as is possible with other implementations.

Additionally, a few approaches have proposed using contrastive learning as a preprocessing step [Kiyasseh et al., 2020, Gopal et al., 2021]. Most AF detection frameworks operate with a single (2-lead) ECG signal as the input. However, available ECG data often contains multiple signals, obtained by placing multiple leads on a single patient. These signals all have the same label (AF or normal sinus rhythm, NSR). Therefore, the use of contrastive learning here helps to generalize the neural network to be able to detect AF regardless of differing lead placements, or *vectors*, on a patient. While we wholeheartedly agree with this analysis, and indeed suggest it as future work, it is beyond the scope of this paper. Instead, we treat each signal as a completely separate sample, duplicating labels as necessary.

3 Dataset and Processing

Our original dataset is from PhysioNet [Moody, 1999, Physionet.org, 2022] and includes 63 hours of ECG recording data from 63 subjects at a sample rate of 500 Hz, distributed roughly as 1/3 AF and 2/3 NSR. We further process this data by splitting each hour-long segment into 1 minute chucks. A-Fib is defined as any episode lasting longer than 30 seconds, according to the American Heart Rhythm Society [Mayo Clinic, a], so we label the data as 1 if AF is present for more than 30 seconds and 0 otherwise (the dataset contains *beat* annotations, which we translate into sample-level annotations). Finally, we randomly hold 10 patients aside as a test set, in order to not spread samples corresponding to the same patient across both train and test sets, as this would not give us a good sense of the generalizability of any resulting network. The remaining 53 patients constitute our training data.

While originally training our neural networks, we encountered issues suggestive of a lack of data. With only 53 subjects in our training dataset, we concluded that although we have more than enough *samples*, our dataset does not contain enough diversity or positive AF examples. We added data from the China Physiological Signal Challenge (CPSC) 2018 [Ng et al., 2018] to our training set, which features the same 500 Hz sample rate as our original data and a roughly even distribution. This data contains samples ranging from 6 seconds to 60 seconds in length, so zero-padding is required for data compatibility. Because our goal was to perform well on our original dataset, i.e. full 60 second samples, we added this additional data purely to our training set.

In summary, our training dataset includes over 5,000 samples of ECG from both the PhysioNet-MIT and CPSC databases, while our test dataset includes data from 10 subjects from the PhysioNet-MIT database, with distributions listed in Table 1. Note the disparity in distributions between the training and test dataset; this is due to our adding the balanced CPSC data to the imbalanced PhysioNet-MIT data. All data conversion and processing code is available in our GitHub repository, `https://github.com/nbyouk/cs230`.

Table 1: Data distribution

Dataset	# subjects	# samples	Percentage samples positive
Training	53 MIT + 2006 CPSC	5073	41%
Test	10 MIT	600	25%

3.1 Pre-processing and Zero Padding

Because the RNN doesn't assume a specific data size or require feature extraction, raw ECG input data may be used without preprocessing. However, the CNN requires all the input data to be of the same size, necessitating zero-padding to account for the CPSC data.

3.1.1 Raw ECG

Sample input ECG signals are visualized in Fig. 2 (left) for both channels, including an example of zero-padding. An AF and a NSR sample are shown from each dataset. Note that the CPSC data contains more than 2 channels: we selected the first two channels for the purpose of this work, but the use of more channels–particularly when coupled with contrastive learning–would certainly result in a more generalizable network.

3.1.2 Fourier Transform for CNN

To extract frequency information of the signal and understand how it changes over time, we apply a short-time Fourier transform (STFT) before passing it into our CNN. Instead of taking the discrete Fourier transform (DFT) of the whole signal in one go, this method splits the signal into small, overlapping pieces and takes the DFT of those pieces in a sliding window fashion. We plot this as a spectrogram in Fig. 2 (right) to visualize the frequency, power, and time aspects of the signal. This STFT transform completes all preprocessing required for the CNN.

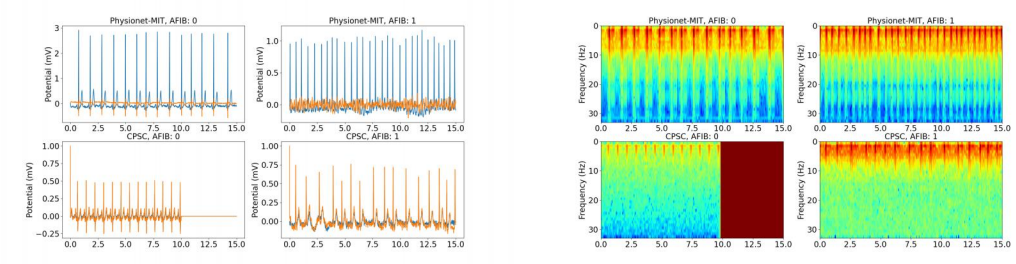

Figure 2: (Left) Sample raw ECG data, plotted for 15 seconds. Note that the CPSC ECG signals closely match each other; these channels measure a similar vector. (Right) Log-spectrograms of the ECG signals. Each is a 2D graph with the third dimension represented by colors. Dark blues correspond to low amplitude and brighter colors up through red correspond to progressively stronger amplitudes.

4 Methods

The training framework for this project is developed in PyTorch. We train both a CNN and a RNN. In both cases, we use the standard weighted Binary Cross Entropy loss (1), setting the weight parameter $\omega = 3$ to account for the imbalanced data distribution in our test set. This also serves to promote better recall–in our case, a good choice, due to the greater consequences associated with a false negative than a false positive.

$$loss = -[\omega y_n * log(\sigma(x_n)) + (1 - y_n) * log(1 - \sigma(x_n)] \tag{1}$$

4.1 CNN

We constructed a CNN consisting of 5 convolution/max pool layers, followed by two dense layers. This CNN, visualized in 3, takes as input a 936 x 33 matrix, the output of the STFT discussed above. It performs mostly same-convolutions across rows and columns; however, it focuses on max-pooling across rows, due to the aspect ratio of its volumes. A similar strategy can be found in Zihlmann et al. [2017], Tutuko et al. [2021]; however, we use our own specific architecture which differs from both of these implementations. As an aside, in Tutuko et al. [2021], a wavelet transform is used instead, and experimentation with adding some RNN layers to the CNN is also performed. However, similar results to ours are achieved, so we deemed these additions unnecessary. We also include a dropout layer for regularization purposes. The specifics for this CNN are outlined in Table 4 in the Appendix.

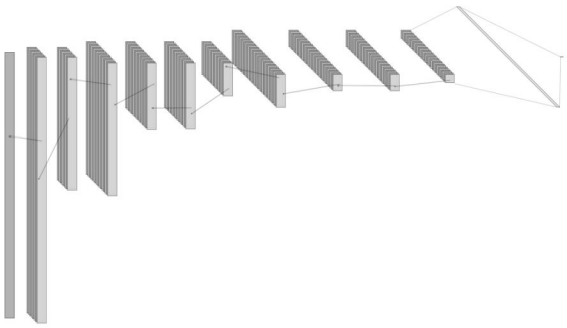

Figure 3: Convolutional neural network architecture.

4.2 RNN

For our RNN, we mainly take inspiration from Faust et al. [2018b]. As discussed earlier, however, this approach operates on RR intervals, while we aim to construct an RNN which operates directly on an ECG, so our approach is in this sense novel. We start with an embedding matrix with learnable weights. To enable the embedded layer, we bin (discretize) the data into 100 groups. Upon experimentation, we found that passing sequences of length 30,000 into an RNN is unacceptably slow; to help mitigate this, we also down-sample our ECG data by taking only every 10th value. Despite our down-sampling, the RNN still runs much slower than the CNN, as we discuss in Section 5. We then pass the output of the embedding layer to the core of our RNN, a bi-directional LSTM layer. We follow the LSTM layer with a dropout layer with a dropout probability of 0.1, followed by a dense layer to make our prediction. A summary of our RNN can be found in 5 in the Appendix.

4.3 Hyperparameter tuning

We settled on our final network architectures and chose all values of hyperparameters heuristically. We make no claim as to the optimality of any individual hyperparameter; the goal of this work is to experiment with various architectures to get a feel for what would be appropriate as final architectures. Nevertheless, we summarize some choices here. In particular, a batch size of 32 was chosen as a compromise between compute power and a desire for generalizability, with a corresponding learning rate in each architecture of 1×10^{-4}. We settled on our final architectures (kernel sizes, number of hidden layers, etc.) by experimenting with different values and observing performance on our test set. We perform no rigorous hyperparameter optimization and instead leave such optimizations as future work.

5 Results/Discussion

As shown below, our CNN model achieves 96.5% accuracy and 90.4% recall on our test set, while our RNN model achieves 86.0% accuracy and 68.2% recall; loss and accuracy evolution during training is presented in Fig. 4. Note that because there is so much data repetition (see Section 3), both networks perform quite well after only one epoch. In both cases, we take the best weights achieved in terms of accuracy on the test set as our final model; this does not correspond to the final loss/accuracy achieved (in particular, see the sharp increase in the loss at the end of training for the RNN). Finally, note that the RNN reports metrics per epoch starting at the end of the first epoch, and thus the plots provided are more jagged and start at a lower loss and a higher accuracy than the CNN, which reports intra-epoch training metrics.

The CNN clearly outperforms the RNN for this scenario. The performance summary 2 and confusion matrices 3 provide further specific details on the performance of each model. Lastly, the CNN is a much leaner model, requiring about 5 minutes per batch to train, as opposed to the about 21 minutes per batch required to train the RNN.

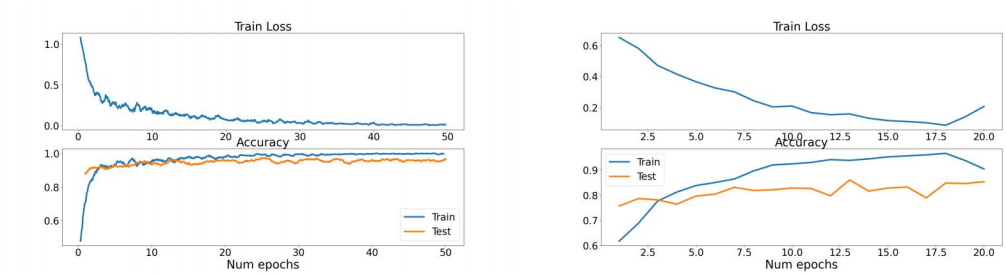

Figure 4: Optimization of our CNN (left) and RNN (right). Training loss is shown on top, while accuracy is shown on bottom, both plotted against epoch number.

Table 2: Performance summary of our models

Metrics (Macro)	Loss Type	CNN	RNN
Accuracy	Train	0.994	0.938
	Dev	0.972	0.860
Precision	Train	0.989	0.920
	Dev	0.961	0.752
Recall	Train	0.998	0.941
	Dev	0.921	0.649
F1 Score	Train	0.993	0.927
	Dev	0.936	0.681

Table 3: Confusion matrices using best weights achieved by each model.

CNN Prediction outcome

actual value	**p**	**n**	**total**
p′	143	11	154
n′	6	440	446
total	149	451	

RNN Prediction outcome

actual value	**p**	**n**	**total**
p′	103	52	155
n′	32	413	445
total	135	465	

6 Conclusions/Future Work

This work shows that a fairly straightforward implementation of both a CNN and a RNN can produce good classification results of atrial fibrillation. In our experimentation, the CNN explored in this paper produces better classification results with a faster run-time than the RNN, leading us to conclude that it is the better choice for this particular task. Future work in this area includes implementing contrastive learning with the multiple ECG signals as discussed in Section 2 to pre-train the model and increase generalizability. Furthermore, for the RNN, various feature extraction techniques (such as a peak detector for computing R-R intervals) could be used to pre-process the data before feeding it to the RNN, which would increase not only its viability but hopefully its accuracy, allowing it to compete with the CNN. A further step would be to expand these neural network architectures to classify other arrhythmia types, changing perhaps from a sigmoid binary classification output to either a softmax output or a vector sigmoid output (the latter allowing for multiple classification), making use of transfer learning.

Code

All code is contained in our `https://github.com/nbyouk/cs230`, including all data processing, training, weights, and results. This codebase is based on code within the publicly available repository `https://github.com/cs230-stanford/cs230-code-examples`.

Contributions

Both authors worked collaboratively on much this project. Greatest individual contributions are summarized below. Credit also goes to Joseph Sullivan (Kestra Medical Technologies), who evaluated all of the annotations for the PhysioNet-MIT dataset.

Kelly Brennan (`kbrenn@stanford.edu`) handled data preprocessing for the CPSC dataset, conducted much of the literature review, ran the models for testing and evaluation, and attempted an implementation of contrastive learning.

Noah Youkilis (`nbyouk@stanford.edu`) handled data preprocessing for the PhysioNet-MIT dataset, and was responsible for the implementation of both neural networks.

Appendix 1: CNN implementation details

Table 4: CNN architecture

Layer	Input dimension	# channels	Kernel size	Output volume
Input	936 x 33	1	-	-
Conv1	936 x 33	8	(5,5) same	936 x 33
MaxPool1	936 x 33	8	(2,1)	468 x 33
Conv2	468 x 33	16	(3,3) same	468 x 33
MaxPool2	468 x 33	16	(2,1)	234 x 33
Conv3	234 x 33	16	(3,3) *padding=(0,1)	232 x 33
MaxPool3	232 x 33	16	(2,1)	116 x 33
Conv4	116 x 33	32	(3,3) same	116 x 33
MaxPool4	116 x 33	32	(2,1)	58 x 33
Conv5	58 x 33	32	(3,3) same	58 x 33
MaxPool5	58 x 33	32	(2,1)	29 x 33
Flatten	29 x 33	-	-	30,624
ReLU	30,624	-	-	30,624
Dropout (0.5)	30,624	-	-	30,624
Dense	30,624	-	-	1000
ReLU	1000	-	-	1000
Dense	1000	-	-	1
Sigmoid output	1	-	-	1

Appendix 2: RNN implementation details

Table 5: RNN architecture

Layer	Input dimension	Output dimension
Input	length (max 3000)	-
Binning	length	length
Embedding	length	length x 100
Bi-directional LSTM	length x 100	length x 400
MaxPool1D	length x 400	400
Dropout	400	400
Dense	400	1
Sigmoid output	1	1

References

E. J. Benjamin, P. A. Wolf, R. B. D'Agostino, H. Silbershatz, W. B. Kannel, and D. Levy. Impact of atrial fibrillation on the risk of death: the framingham heart study. *Circulation*, 98(10):946–952, 1998.

O. Faust, A. Shenfield, M. Kareem, T. R. San, and H. Fujita. Automated detection of atrial fibrillation using long short-term memory network with rr interval signals. *Computers in Biology and Medicine*, 102:327–335, 2018a.

O. Faust, A. Shenfield, M. Kareem, T. R. San, H. Fujita, and U. R. Acharya. Automated detection of atrial fibrillation using long short-term memory network with rr interval signals. *Computers in biology and medicine*, 102:327–335, 2018b.

B. Gopal, R. W. Han, G. Raghupathi, A. Y. Ng, G. H. Tison, and P. Rajpurkar. 3kg: Contrastive learning of 12-lead electrocardiograms using physiologically-inspired augmentations. 2021. doi: 10.48550/ARXIV.2106.04452. URL `https://arxiv.org/abs/2106.04452`.

A. Hannun, P. Rajpurkar, M. Haghpanahi, G. Tison, C. Bourn, M. Turakhia, and A. Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature Medicine*, 25:65–69, 2019.

D. Kiyasseh, T. Zhu, and D. A. Clifton. CLOCS: contrastive learning of cardiac signals. *CoRR*, abs/2005.13249, 2020. URL `https://arxiv.org/abs/2005.13249`.

Mayo Clinic. 2017 hrs expert consensus statement on catheter and surgical ablation of atrial fibrillation. section 2: Definition, a. `https://www.heartrhythmjournal.com/article/S1547-5271(17)30590-8/fulltext#secsectitle0020`.

Mayo Clinic. Atrial fibrillation: causes and symptoms, b. `https://www.mayoclinic.org/diseases-conditions/atrial-fibrillation/symptoms-causes/syc-20350624`.

G. Moody. Mit-bih normal sinus rhythm database. mit-bih normal sinus rhythm database v1.0.0, 1999. `https://www.physionet.org/content/nsrdb/1.0.0/`.

E. Y. K. Ng, F. Liu, C. Liu, L. Zhao, X. Zhang, X. Wu, X. Xu, Y. Liu, C. Ma, S. Wei, Z. He, and J. Li. An open access database for evaluating the algorithms of electrocardiogram rhythm and morphology abnormality detection, 2018.

Physionet.org. Mit-bih atrial fibrillation database v1.0.0, 2022. `https://physionet.org/content/afdb/1.0.0/`.

B. Tutuko, S. Nurmaini, A. E. Tondas, M. N. Rachmatullah, A. Darmawahyuni, R. Esafri, F. Firdaus, and A. I. Sapitri. Afibnet: an implementation of atrial fibrillation detection with convolutional neural network. *BMC Medical Informatics and Decision Making*, 21(1):1–17, 2021.

M. Zihlmann, D. Perekrestenko, and M. Tschannen. Convolutional recurrent neural networks for electrocardiogram classification, 2017. URL `https://arxiv.org/abs/1710.06122`.