Assignment 1 Report

Zeyu zhang, Student ID:518021910953

Introduction

Atrial fibrillation (AF) is an abnormal heart rhythm characterized by the rapid and irregular beating of the atrial chambers of the heart.[1]

It is the most common type of cardiac arrhythmia, occurring in 1%–2% of the general population and around 9% of the elderly, and is associated with significant mortality and morbidity of many heart diseases.[2]

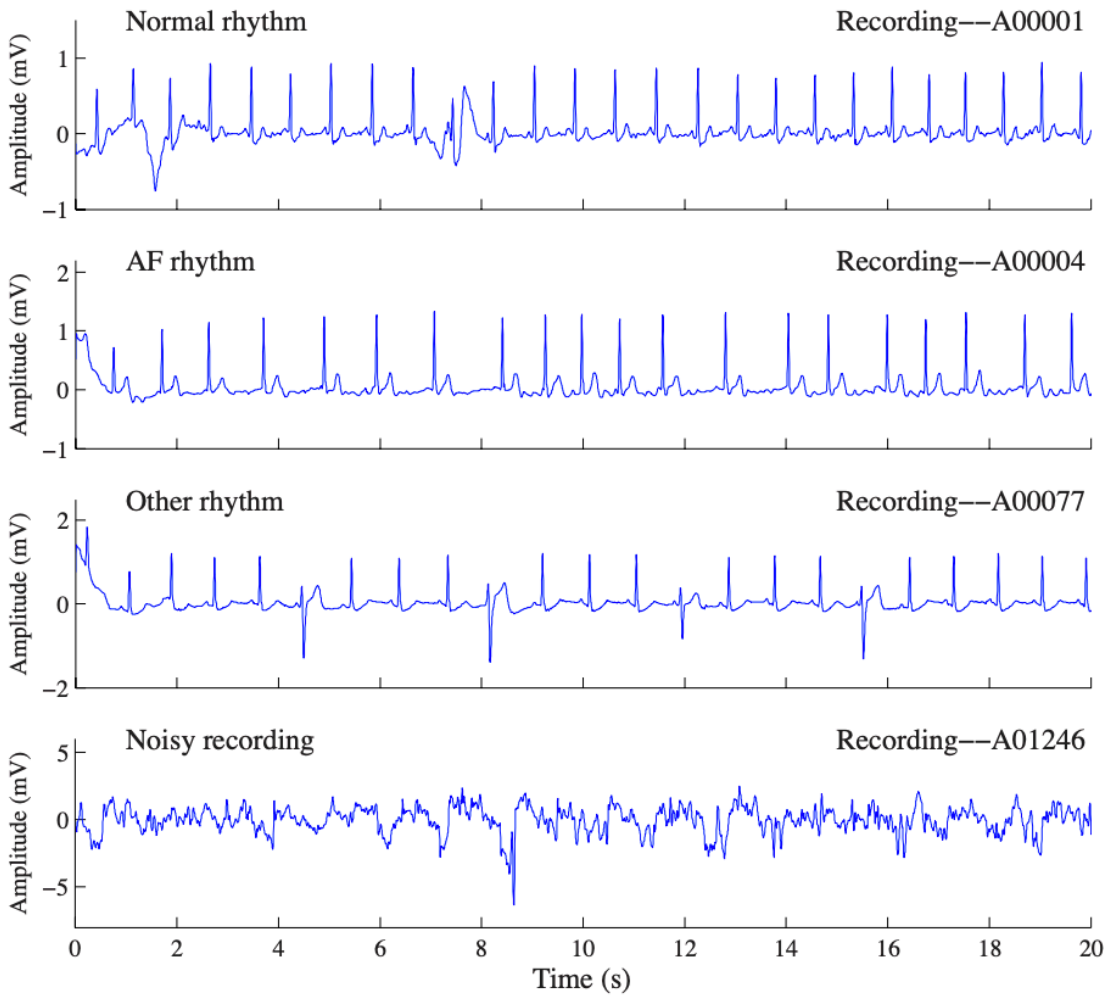
The main method of AF detection is investigating its ECG graph. Common AF detectors are based on the analysis of the absence of P waves or the presence of fibrillatory f waves in the TQ interval. There have been various studies on AF detection, but their practical use are still limited. These methods are mainly focused on classification of 2 catogories: noraml and AF, and the input ECG signal often should be clean and of the same length. This has inspired the PhysioNet/CinC Challenge 2017 (2017), which aimed to encourage the development of algorithms to classify single short ECG lead recordings of variable lengths as as normal, AF,other rhythm,or noisy recordings.

Deep learning, in recent years, has emerged as an effective tool for data analysis. The use of artificial neural networks in deep learning have drastically improved sequential data processing tasks such as speech recognition, language translation and text to speech software, due to the powerful feature learning abilities of neural networks for understanding complex datasets. Most state-of-the-art neural networks perform predictions from raw data inputs, taking efficiency in data analyses to a higher level and bypassing the need of expert knowledge. The purely data-driven nature of these algorithms allows their performance accuracy to increase accordingly with increasing amounts of data. Which In this study, the deep learning methods enabled us to classify these ECG graphs without complex feature-extraction techniques. In the following passage, I will also compare classifiers with deep neural network structrues with an SVM-based classifier.

Method

Dataset

The ECG classification challenge was a sequential classification task where a single label was required for each individual input signal. The training dataset is the PhysioNet/CiC Challenge 2017 dataset. It is consisted of 8,528 single lead ECG recording ranging from 9 to 60 seconds in length with a sampling rate of 300 Hz. The Traning set have 4 classes of the ECG signals: AF, Normal, Other and Noisy.[3]



The validation set was a small subset from Training set, which is manually excluded from the traning process, and it will not participate in any modification works on hyperparameters during training. More details on the data set are shown in table below:

Type	# recording	Time length (s)				
		Mean	SD	Max	Median	Min
Normal	5154	31.9	10.0	61.0	30	9.0
AF	771	31.6	12.5	60	30	10.0
Other rhythm	2557	34.1	11.8	60.9	30	9.1
Noisy	46	27.1	9.0	60	30	10.2
Total	**8528**	**32.5**	**10.9**	**61.0**	**30**	**9.0**

SVM Method

Feature extraction is an important step in SVM-based classifiers, and it is extremely demanding of corresponding professional skills. For example ,Na Liu proposed a SVM-based classifier with an input feature of 33-Dimesions. The feature extraction process used various advanced signal processing techniques, including Discrete Wavelet Transformation (DWT) to denoise, R peak detection and paticular features derived by statistical properties of RRIs and P waves of ECG.[2] Combined with the complex feature extraction methods, it gains a F-1 Score of 0.80 in the Validation set, competing with the state-of-the-art deep learning methods.

But for non-professionals, the feature extraction methods are way to challenging to understand or even propose one of their own. To give SVM a try, we used feature extraction methods by first apply bandpass filtering, with a a processed signal of sharper peaks, then we proposed a simple heartbeat interval detection method and then we generate a histogram of these intervals,reflecting its statistical feature. We set bars to 20 and used the normalized histogram as SVM classifier's final input vector.

Neural Network Method

Data Preparing

The ECG recordings in PhysioNet/CinC Challenge 2017 dataset are not of the same length. To start traning, we first need to generate a training set of the same shape so that can be inputed into the network. A natural approach is do zero padding to generate trainable data of same length. By using zero padding, challenges has that a RNN network's performance would be unpredictable by accepting part-zero sequnces. This can be solved by using a Masking Layer in the network in order to discard dimensions of the output feature vector resulting from zero padded processed values.

But according to the description of the dataset, exact max length of all recordings are not clear(We can only know it 's aroud 18000), and the length of recordings at least vary from 2700 to 18000 (9-60s).

I used a "Window-Moving" technique to generate same-length datasets. First we define a $window\_size$ as trimming parts of recordings from the whole recording. The generated clip would be of length equal to $window\_size$. And then we define a $stride$ as Window Stepping forward, for each stride walked by a clip of currently windowed sequence will be generated. By using this method, we can easily get same-lengthed data, while maintaining most of its sequential adjacency informations compared to just split the original recording to small parts. And we also avoid zero-padding in sequences with drastically differing lengths.

Network Architecture

The first-place method of PhysioNet/CinC Challenge 2017 challenge is based on a deep neural network, which consisted of varois CNN, LSTM and residual blocks. I realized a smaller network called CRNN from scratch, though, consisting of conv blocks, a LSTM block and fully connected block. The main idea is first using CNN to extract as much features of the input as possible, with retaining a reletively long input for following LSTM blocks. And then I use LSTM, trying to extract the inputs' sequential informations and generating final feature map, which will be sent to a

dense block with output as a 4-D vector, since it is a classification problem. As time constraints, I only be able to implement a simple version of CRNN, shown in table below:

Name	Input	Output	Type
CNN	1,$window\_size$	1,$out\_channels$	Conv1d
RNN	1,$out\_channels$	1,$out\_channels$	LSTM
Dense	256	4	Fully Connected

$window\_size$ and $out\_channels$ are user definable, which can be easily modified in a configuration file alongside with all 21 configurable parameters.

After completing this network ,I found it happens to coincide with Mohamed Limam [4] 's methods, is a much simpler version of their CRNN, their AF score is over 0.85 on a subset of training set, which proves the potential of these CRNN architechures.

RCNN + SVM

The fully connected block generates a 4-dimentional feature as the output of RCNN network. A straightforward approach is just pick the largest value's place as the output of 0,1,2,3, corresponding to the four classes of AF, Normal, Other and Noisy. Another approach is to calculate the distance between output and standard one-hot representation of the 4 classes: [1 0 0 0], [0 1 0 0], [0 0 1 0] and [0 0 0 1]. From the other perspective, we can consider the output of RCNN network as an input of a classification problem. SVM is a powerful classification method and it would show superiority to these two "Naïve" methods. So I attached a SVM classifier as the backend of CRNN network to enhance the results even further.

Training

For training our models we split our data into training and test set. We use 90% for training and 10 % for test. The test is composed of 10% of each class. We set $window\_size = 3000$, $stride = 500$, and after loading, a set of 111970 sequential data with length of 3000 (10s). These data will be divided to a training set of 100773 data and a test set of 11197 data.

```
Loading files...
100%|████████████████████████████████████████| 8227/8227 [00:07<00:00, 1134.96it/s]
Generating dataset...
Orignial:  8227  After:  (111970, 3000) (111970,)
Traing set shapes(x,y): (100773, 1, 3000) (100773,)
Test set shapes(x,y): (11197, 1, 3000) (11197,)
Training strat, begin with Lr= 0.0001
```

I used `Adam` as its optimizer since its usually brings faster convergence than typical `SGD` optimizers. And I also introduced learning rate scheduler mechanism, with `torch.optim.lr_scheduler.ReduceLROnPlateau` as its default method. The parameters of learning rate, weight decay, LR Reduce Factor, LR Reduce Patience can all be defined by user.

At each epoch, the network's loss can be monitored with a certain frequency, here I set it to 20 iterations. It can be modified in `cfg.py` with an entry `print_freq` .

At the end of each epoch, it will perform a test on test set, providing just-in-time training informations.

```
Epoch  1 , Iter  20 :  Loss= 1.3774726629257201   Lr= 0.0001
Epoch  1 , Iter  40 :  Loss= 1.3603540301322936   Lr= 0.0001
Epoch  1 , Iter  60 :  Loss= 1.3605573117733   Lr= 0.0001
Epoch  1 , Iter  80 :  Loss= 1.3588933169841766   Lr= 0.0001
[[-0.09394822  0.04092106 -0.2733979  -0.379251  ]
 [ 0.31369942  0.00788502 -0.11895818 -0.56694233]
 [ 0.485838   -0.11933212  0.12974128 -0.46381775]
 ...
 [ 0.08268423  0.21104428 -0.02012718 -0.20737356]
 [ 0.01229059 -0.14628568  0.02533188 -0.41579407]
 [-0.15781882  0.28353223  0.27094546 -0.13055038]]
pred and gt Shapes: (11197,) (11197,)
[1 1 0 ... 1 2 1] [1 0 0 ... 1 2 1]
{'AF': {'precision': 0.10604994723883222, 'recall': 0.617827868852459, 'f1-score': 0.1810267187030922, 'su
pport': 976}, 'Normal': {'precision': 0.6219444444444444, 'recall': 0.33924242424242423, 'f1-score': 0.439
01960784313726, 'support': 6600}, 'Others': {'precision': 0.31212615551930395, 'recall': 0.167689161554192
22, 'f1-score': 0.21816799695933103, 'support': 3423}, 'Noisy': {'precision': 0.05555555555555555, 'recall
': 0.020202020202020204, 'f1-score': 0.02962962962962963, 'support': 198}, 'accuracy': 0.3054389568634456,
 'macro avg': {'precision': 0.273919025689534, 'recall': 0.28624036871277386, 'f1-score': 0.21696098828379
756, 'support': 11197}, 'weighted avg': {'precision': 0.472246665372958, 'recall': 0.3054389568634456, 'f1
-score': 0.3417761194496187, 'support': 11197}}
F-1 Score:  0.21696098828379756
Epoch  2 , Iter  20 :  Loss= 1.352461302280426   Lr= 0.0001
```

Result

Evaluation Methods

I use F_1 scores as the evaluation metric. It can be defined by:

	Normal	AF	Other	Noisy	Total
Normal	Nn	Na	No	Np	$\sum N$
AF	An	Aa	Ao	Ap	$\sum A$
Other	On	Oa	Oo	Op	$\sum O$
Noisy	Pn	Pa	Po	Pp	$\sum P$
Total	$\sum n$	$\sum a$	$\sum o$	$\sum p$	

$$
\begin{aligned}
F_{1n} &= \frac{2 \times N_n}{\sum N + \sum n} \\
F_{1a} &= \frac{2 \times A_a}{\sum A + \sum a} \\
F_{1o} &= \frac{2 \times O_o}{\sum O + \sum o} \\
F_{1p} &= \frac{2 \times P_p}{\sum P + \sum p}
\end{aligned} \tag{1}
$$

And finally, the overall F_1 score:

$$
F_1 = \frac{F_1 n + F_1 a + F_1 o + F_1 p}{4} \tag{2}
$$

For reference, the Acc score can be calculated as:

$$
Acc = \frac{N_n + A_a + O_0 + P_p}{\sum N + \sum A + \sum O + \sum P} \tag{3}
$$

Results And Comparison

According to real-time calculated F_1 scores during training process, I picked one model checkpoint file, and load that into a model to get inference. I used the independent Validation set, so it is a completely hidden test.

Comparing various classifier backends:

Methods	F_1	Acc	F_1a	F_1n	F_1o	F_1p
RCNN	0.243	0.473	0.035	0.651	0.00	0.00
RCNN+SVM backend	**0.301**	**0.351**	**0.149**	**0.468**	**0.305**	**0.277**
RCNN+Random Forest	0.205	0.496	0.00	0.669	0.155	0.00
RCNN+Decision Tree(one-hot encoded)	0.225	0.364	0.185	0.547	0.137	0.028
RCNN+Naive Bayes	0.252	0.384	0.181	0.53	0.27	0.026

We can see SVM is the superior classification method when it comes to F_1 scores. But for the nature of the distribution of these 4 classes, we need to tune the model a little bit. Due to the even distribution of the 4 classes of samples, the noisy and AF samples tend to be neglected if all classes have the same weight. A simple solution is to set weight to 'balance', which the weight of each classed will be determined by its distribution in training set. We can also asssign penalty factor C, with default of 1. After several trials, I found when C=1.2, weight='balance', the SVM's performance is optimal.

Finetuning SVM classifier parameters:

Methods	F_1	Acc	F_1a	F_1n	F_1o	F_1p
Default	0.220	0.357	0.078	0.507	0.323	0.000
weight='balance'	0.298	0.347	0.147	0.465	0.303	0.277
class_weight={0:3,1:0.6,2:1.2,3:6}	0.284	0.370	0.058	0.505	0.323	0.245
class_weight={0:3.5,1:0.7,2:1.4,3:3}	0.234	0.360	0.101	0.510	0.323	0.000
C=2, weight='balance'	0.248	0.363	0.157	0.509	0.326	0.000
C=1.2, weight='balance'	**0.301**	**0.351**	**0.149**	**0.468**	**0.305**	**0.277**

Discussion

In this study, a simple RCNN network was constructed and trained, combined with a SVM-based classifier backend, carried out the AF ECG classification task. In a strict hidden test of 300 test samples, the model gets a overall F_1 score of 0.301, and accuracy of 0.351. Various other classification methods are also tested.

The result is far from promising, but the idea is right on track.though this simple model has not learnt much information from dataset, the basic CRNN architecture have been proven effective with deeper constructions. More of my work has been done in the framework, though. A complete and scalable code architecture decouples training, data loading, evaluating, configuration and network design. Classifier backend can be easily defined and changed since by modifying `ClsBackend` class while its interface changed, so other evaluation code would not need to be rewrited.

Acknowledgments

In this BI054 Artifitial Intelligence and Medical Enginnering course, the thoery of deep learning unveiled for me and I have learned a lot. Many thanks for Professor Wang and TA Ho.

Some Thoughts

With extremly limited amount of time for completing this work (about a day and a half, including writing report), I reviewed what we have learned, read several papers and spent a lot of time finding documentations and sample codes online. It's sad that the model is not very effective, though I have read the souce code of the #1 challenger of PhySioNet/CiC 2017 challenge and planning to implement his ResNet96 deep neural network next. But reaching the deadline, I have only be able to use this "Demo" network to gather results.

References

1. Wikipedia. Atrial fibrillation, https://en.wikipedia.org/wiki/Atrial_fibrillation ↩

2. Na Liu $et\ al.$ 2018 Physiol. Meas. 39 064004 ↩ ↩

3. AF Classification from a Short Single Lead ECG Recording - The PhysioNet Computing in Cardiology Challenge 2017 -https://physionet.org/content/challenge-2017/1.0.0/ ↩

4. Mohamed Limam $et\ al.$ 2020. Atrial Fibrillation Detection and ECG Classification based on Convolutional Recurrent Neural Network ↩