A Novel Approach to Classify Electrocardiogram Signals Using Deep Neural Networks

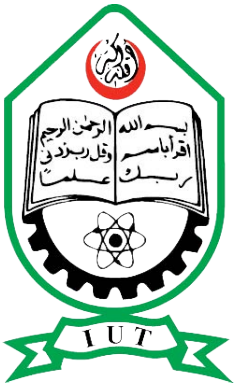
Authors

Tasnim Ahmed
Student ID: 154407

Ariq Rahman
Student ID: 154404

Supervisor

Tareque Mohmud Chowdhury
Assistant Professor



Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)
Organization of the Islamic Cooperation (OIC)
Gazipur, Bangladesh

November, 2019

Declaration of Candidates

This is to certify that the work presented in this thesis is the outcome of the analysis and investigation carried out by the candidates under the supervision of Tareque Mohmud Chowdhury in the Department of Computer Science and Engineering (CSE), IUT, Dhaka, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

- - - - - - - - - - - - - - -
(Signature of the Candidate)

Tasnim Ahmed
Student ID: 154407
Academic Year: 2018-19
November, 2019

- - - - - - - - - - - - - - -
(Signature of the Candidate)

Ariq Rahman
Student ID: 154404
Academic Year: 2018-19
November, 2019

- - - - - - - - - - - - - - -
(Signature of the Supervisor)

Tareque Mohmud Chowdhury
Assistant Professor
Department of Computer Science & Engineering (CSE)
Islamic University of Technology (IUT)
November, 2019

Abstract

Atrial fibrillation (AF) is an abnormal heart rhythm that takes place when electrical impulses fire off from multiple places in the atria (the top chambers of the heart) in a disorganized way. This causes the atria to twitch and results in an irregular heartbeat or pulse. Atrial fibrillation is a major cause of stroke. Atrial Fibrillation is usually screened manually with the help of Electrocardiodiagram (ECG) reading. Manually reading ECG is usually a tedious and time-consuming task, which is laden with human errors. Therefore, an automated process is quintessential. However, discerning anomaly in heart function using an efficient automated process has been a challenging task for quite some time. In this paper we propose two intricate Neural Network architectures for the classification amongst four types of heart condition-Normal, Atrial Fibrillation, Noisy Sinus Rhythm and Alternative Rhythm, using a dataset from PhysioNet/2017 challenge. Volunteers in PhysioNet/2017 challenge dataset came from diverse background and had a wide window of variation in their physical attributes, making the dataset sufficiently reliable. Also, the size of this dataset exceeded any other before it on this topic, which further adds to the comprehensiveness of this dataset.Initially, a preprocessing is done on the dataset to make it more robust and push the accuracy to its edge. We then trained a Deep Neural Network, which combined feature extraction layers of CNN with long-short term memory (LSTM). Our method reached a summit accuracy of 91.19%.The second model just applied a CNN model and the resulting output reached an accuracy of 84.3%

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Cardiovascular disease (CVD) has been one of the most pernicious Non-communicable maladies in the world. Commensurately, more people have succumbed from Cardiovascular disease that any other[3].On the scales of statistic, approximately half (17.5 million) of the Non-communicable disease was the result of CVD in 2012 amongst 56 million deaths recorded globally at that time. The occurrence of CVDs are prognosticated to hit 22.2 million by 2030 [3] and increasingly major healthcare has been set aside for this cause only. By 2030 around US$20 trillion are expected to be spent for CVD[1].

1.2 Problem Statement

Traditionally, CVD has been manually diagnosed from ECG signals and aside from being taxing and time-consuming task, it is also susceptible to human error. Recent endeavours have tried to computerize the process of detecting CVD from using ECG signals and some of them are mentioned below in the related work section. Majority of the work majority is being focused on Myocardial infarction and similar type of CVD. We propose mechanisms to computerize the process using new set of data that is based on Physionet/Cinc dataset . We try to classify four types of heart conditions mentioned later using CNN and CNN + LSTM.

1.3 Research Challanges

Attaining perfect computerized process is laden with challenges. These challenges occlude the way to receiving a pristine set of out comes. Some arise from the laggings of even the present state of the art technology and some arise from the inherent problem with the procedure itself. Mitigating these laggings will create way for the process to evolve further .Some of them are mentioned below:

1. **Features:** Different features are needed to ascertain different different charectericstics of the ECG signal. A collection of features reflect the kind of heart condition the person has .For eg T wave ST elevation etc

2. **Noise:** Noise can be a nuisance. The presence of noise deforms the signal. Where the shape of the signal is quintessential for determining the heart condition. It is of upmost necessity that all form of noises should be removed.However, removing noise is not an easy task . Multiple state of the art filters are being used and even after that , the filters are incapable of bringing perfection.

3. **Scarcity of data:** Scarcity of data is an inherent challenge. Atomization of malady detection using deep neural network is a relatively new field and the same goes for heart codition . There are only few datasets available for the public. Therefore , it is very difficult to cross check the results with other similar ECG dataset and hence puts a mark on the validity and reliability of the outcome.

4. **Age, Gender, Sex, etc.:** Age,gender race varies across a wide spectrum . These variable makes the grounds for comparison more arbitrary. For eg a 13 year olds ecg may be different from the 60 year olds for the same heart condition. Also, as the dataset is collected from a particular region on the globe, an model based on the dataset tends to be skewed

5. **Computational power:** Large amount of data needs to be scrutinized before right set of parameters for the deep network is obtained. This takes huge computational cost. In some cases the computational cost may supersede the benefit

6. **Storage space:** Large amount of space is needed to store the voluminous amount of data. This incurs an extra cost .This cost may surpass the benefit from the goal.

1.4 Thesis Objectives

Already the challenges of the project has been discussed .The objective now would be to choose the right set of parameters to better computerize the whole project . This includes either choosing the right set of features, working with a prolific dataset , mitigating the effect of gender , age and culture variation, making sure that the output can be reached using right set of parameters and cutting down on the space and time complexity.

1.5 Thesis Contributions

We worked on a relatively new dataset. This dataset is large in number and contains data from a wide variation of people including gender. We also did some data sampling and data augmentation. We were able to reach promising accuracy of 91.1% that was better than any other previously.

1.6 Organization Of The Thesis

The rest of the thesis will be organized as follows: in Chapter 2 we present the literature review of existing methods and their performance as well as limitations for the detection process. In Chapter 3, we propose our dataset and type of resampling used.

In Chapter 4,we discuss the types of prepossessing used. Our proposed method with various challenges are shown. Finally, in Chapter 5, we describe the proposed architecture. and shows the future scopes for further developing the proposed method.

Chapter 2

Literature Review

This section reviews the state-of-the-art methods for the considered problems.

In 2017,

2.1 CNN Implemention in 2s and 5s Networks

Acharya.et.al[1] devised a mechanism to detect coronary artery disease (CAD) using deep convolutional neural network (CNN) that consists of four convolutional layers, four max-pooling layers, and three fully-connected layers. The novelty of their work lies in that it evades the taxing task of choosing the right set of parameters as CNN automatically learns from it. The neural network formulation is applied on two disparate samplings, Net1 (2s) and Net2 (5s) , on the Fantsia database(for Normal) and St.Petersburg Institute of Cardiology (for CAD).The CNN is able to distinguish between Normal and abnormal with an accuracy of 94.95% for Net 1(2s) and accuracy of 95.11% for Net2(5s). Model architecture is shown in Fig.2.1.

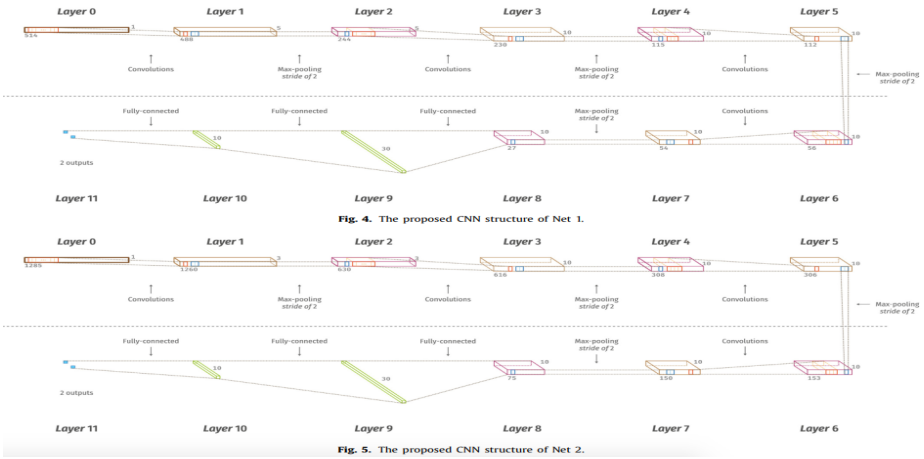


Figure 2.1: Proposed CNN Network - Net 1 and 2 by [1]

2.2 CNN with/without Noise

In 2017, Acharya.et.al[3] proposed another mechanism for the detection of MI using the PTB database.This paper avoided the burden of feature selection by using convolutional neural network (CNN).The CNN consists of four convolutional network , four max-pooling layers and three fully connected layers.The dataset was processed into two fractions, one pertaining noise and another doesn't and both were equally trained in CNN. The results showed an accuracy of 93.53% and 95.22% with and without noise respectively, showing that noise dosen't affect the output.

2.3 KNN, MLP Implementation

In 2014, Safdarian et.al[4], presented a paper for the detection and localization of MI using features that were extracted from the PhysioNet database.These features included the T wave integral (area under the T wave) , which delineated the T wave structure and the total integral(area under one cycle). These features were then fed and trained through a plethora of classifiers such as Artificial neural network (ANN), Probabilistic Neural Network (PNN),Kth Nearest Neighbor (KNN), Multilayer Perceptron (MLP) and Naïve Bayes classifier. Their work touched an accuracy of 76% for detection of MI and 96% for localization.

2.4 KNN Implementation

In 2010, Arif et.al[5], published a paper for the detection and localization of MI patients using features such a T wave amplitude. Q wave and ST level deviation.This paper used the PTB database using 20160 ECG beats. These features were used to formulate a Kth Nearest Neighbour for detection and localization. Moreover, to wan this exorbitant size of data Arif-Fayyaz pruning algorithm was being used. For MI detection the sensitivity and specificity were shown to be 99.9% and after pruning the sensitivity and specificity was plummeted down to 97% and 99.6% respectively.

2.5 Atrial Fibrilation

One of the CVDs is Atrial fibrillation(AF), which is also the central focus of our paper.AF is an anomaly in heart rhythm that is characterized by rapid and irregular beating of atria. Often it starts with short periods of abnormal beating that intensifies over time. AF is a class of cardiac arrhythmia that increases likely hood of a heart attack and is usually the most common form of CVD ; thus drawing our attention to classify this condition. Recently, a new dataset has been provided by the PhysioNet/Cinc challenge 2017. PhysioNet/Cinc challenge 2017 contains an amalgamation of four short ECG recordings marked as normal sinus rhythm, AF, other types of rhythm and noise. A major drawback of the previous datasets was that its size was small. Also, the variation of the attributes amongst entities was skewed. This new dataset, PhysioNet/Cinc challenge 2017, contains entities from diverse background and attributes. Although the accuracy obtained from examples on the previous dataset were skyrocketing, still the results from the new dataset are reliable and robust due to the large range of variation between samples. Some of the works done with this dataset has been mentioned later.

Atrial Fibrillation (A Fib) is one of the most common abnormal heart rhythms, particularly once a person reaches the age of 65 years, (earlier in many patients). It is a major health problem. The heart is a pump, and in order to function efficiently, it is regulated by an internal pacemaker that regulates the beat of the heart. Normally the electrical impulse starts in the top of the heart, travels downward from the upper chambers (the atria) to the lower chambers (the ventricles) and results in a regular rhythmic contraction of the chambers.

With atrial fibrillation, control of electrical activity in the upper chambers becomes disorganized and the atria fibrillate (quiver or twitch quickly)

causing an irregular rhythm. Stroke is the greatest risk for the patient with atrial fibrillation. Some patients are not aware of atrial fibrillation and in them, the risk remains unrecognized. In these patients, the first indication of AFib may be admission to a hospital with a stroke. Other patients may be aware of an irregular heartbeat and the sensation may be uncomfortable. If the heartbeat is very fast for a long period of time, it can also lead to heart failure.

Therefore a major goal of the Heart Rhythm Society is create awareness of atrial fibrillation in order to prevent stroke and heart failure. Atrial fibrillation may be a primary electrical abnormality of the heart, be associated with underlying heart problems including problems with heart valves, coronary arteries, heart muscle, congestive heart failure or be related to problems with the thyroid gland or other disorders of metabolism.

2.6 CNN on Physionet/CinC

Martin et.al[6] proposed a paper that instituted two neural network architecture for the identification of Atrial fibrillation (AF) from PhysioNet/CinC Challenge 2017 dataset. One was a Convolutional Neural Network (CNN). The second architecture combined the convolutional network for feature extraction with long-short-term memory. Data augmentation was done on the ECG data, which was one of the cardinal features of their work. Two main data augmentation techniques were used: dropout bursts and random sampling. This augmentation primarily served to mitigate the overfitting of the neural network. The second architecture obtained a promising F1 score of 82.1%.

2.7 Morphology Based Feature on Physionet/CinC

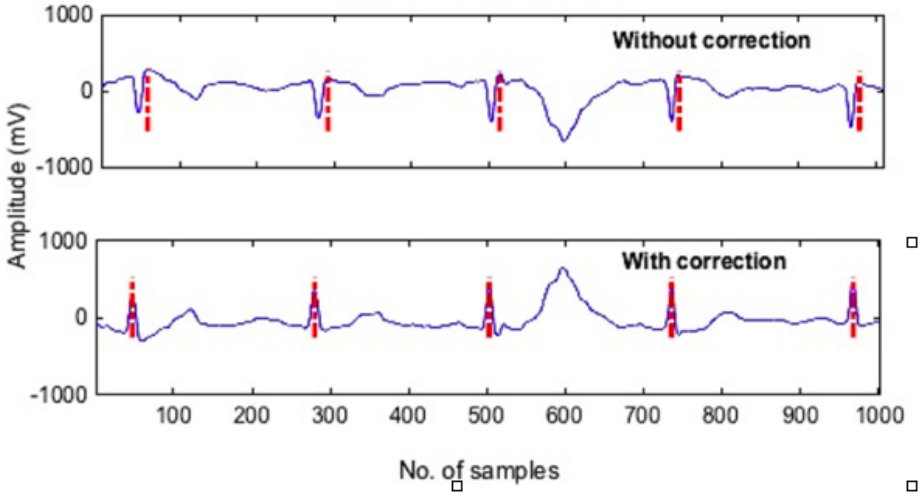
In Dionisije.et.al[7], performed similar work for the detection of heart rhythm was done with the PhysioNet database that contained an amalgamation of Normal sinus rhythm, normal sinus rhythm and alternative rhythm.The proposed work retrieved different morphology-based features of ECG signals that were then trained in multiclass classifier besides a random forest classifier. A F1 score of 80% was obtained hidden test set of challenges.

2.8 RNN on Physionet/CinC

Fernando.et.al[8] also classified four types of aforementioned classes from the Physionet Challenge of 2017.They juxtaposed feature-based classifier, implementing the WFDB Toolbox of Matlab , with a Residual Network ResNet also known as RNN.The feature-based classifier secured F1 score of 72% on the training set and 79% on the hidden set.CNN,on the other hand, secured 83% on the test set.

2.9 Morphological Feature with GA on Physionet/CinC

Ruhi.et.al[2] did similar classification on the same challenge dataset for the classification of normal sinus rhythm, AF noisy, or other rhythms. Morphological, time and frequency domain features were extracted and the feature space dimension was curbed using Genetic Algorithm (GA). Finally, the data was trained in a random forest classifier that reached a tenfold cross-validation accuracy of 82.7%. Detailed results are depicted in Fig.2.2.



Rhythm Type	F_1- Score	
	Training dataset	Test dataset
Normal	0.89	0.91
Atrial fibrillation	0.76	0.74
Other	0.72	0.70
Average	*0.79*	*0.78*

Figure 2.2: Detailed results by Ruhi et al. [2]

We propose, in our paper, two deep neural network based architectures for the classification of four types of heart condition mentioned above. The first architecture combines the feature extraction of convolution layers with long-short term memory. Before that, a data augmentation method is applied to the dataset to make it more robust and increase accuracy. Our results were promising and we reached an overall accuracy of 91.19%.The second one applies a general convolutional model with resulted in an output of 84.3%.

2.10 An Intro to CNN

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole

network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.So what changes? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

2.10.1 How CNN Process Inputs?

Neural Networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the "output layer" and in classification settings it represents the class scores.
Regular Neural Nets don't scale well to full images. In CIFAR-10, images are only of size 32x32x3 (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, the input images in CIFAR-10 are an input volume of activations, and the volume has dimensions 32x32x3 (width, height, depth respectively). As we will soon see, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions 1x1x10, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension.

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet architecture.

Example Architecture: We will go into more details, but a simple ConvNet for CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC].

2.10.2 Conv Layer Function

Convolutional layer(Conv Layer) will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters. RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]). Pool layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12]. FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume. In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting.

Lets first discuss what the Conv layer computes without brain/neuron analogies. The Conv layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward

pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of filters in each Conv layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

The brain view: If you like the brain/neuron analogies, every entry in the 3D output volume can also be interpreted as an output of a neuron that looks at only a small region in the input and shares parameters with all neurons to the left and right spatially (since these numbers all result from applying the same filter).

Local Connectivity: When dealing with high-dimensional inputs such as images, as we saw above it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (equivalently this is the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space (along width and height), but always full along the entire depth of the input volume.

Now considering a small example, suppose that the input volume has size [32x32x3], (e.g. an RGB CIFAR-10 image). If the receptive field (or the filter size) is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights (and +1 bias parameter). Notice that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.

2.10.3 CNN Computation Example

Suppose an input volume had size [16x16x20]. Then using an example receptive field size of 3x3, every neuron in the Conv Layer would now have a total of 3*3*20 = 180 connections to the input volume. Notice that, again, the connectivity is local in space (e.g. 3x3), but full along the input depth

(20).

Spatial arrangement:We have explained the connectivity of each neuron in the Conv Layer to the input volume, but we haven't yet discussed how many neurons there are in the output volume or how they are arranged. Three hyperparameters control the size of the output volume: the depth, stride and zero-padding. We discuss these next:

First, the depth of the output volume is a hyperparameter: it corresponds to the number of filters we would like to use, each learning to look for something different in the input. For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a depth column (some people also prefer the term fibre). Second, we must specify the stride with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially. As we will soon see, sometimes it will be convenient to pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes (most commonly as we'll see soon we will use it to exactly preserve the spatial size of the input volume so the input and output width and height are the same). Use of zero-padding. In the example above on left, note that the input dimension was 5 and the output dimension was equal: also 5. This worked out so because our receptive fields were 3 and we used zero padding of 1. If there was no zero-padding used, then the output volume would have had spatial dimension of only 3, because that it is how many neurons would have "fit" across the original input. In general, setting zero padding to be P=(F−1)/2 when the stride is S=1 ensures that the input volume and output volume will have the same size spatially. It is very common to use zero-padding in this way and we will discuss the full reasons when we talk more about ConvNet architectures.

Note again that the spatial arrangement hyperparameters have mutual constraints. For example, when the input has size W=10, no zero-padding is used P=0, and the filter size is F=3, then it would be impossible to use stride S=2, since (W−F+2P)/S+1=(10−3+0)/2+1=4.5, i.e. not an integer, indicating that the neurons don't "fit" neatly and symmetrically across the input. Therefore, this setting of the hyperparameters is considered to be invalid, and a ConvNet library could throw an exception or zero pad the rest to make it fit, or crop the input to make it fit, or something. As we

will see, sizing the ConvNets appropriately so that all the dimensions "work out" can be a real headache, which the use of zero-padding and some design guidelines will significantly alleviate.

2.10.4 Conv Layer Example

A Real-world example of convolutional Layer will be discussed in lines following this. The Krizhevsky et al. architecture that won the ImageNet challenge in 2012 accepted images of size [227x227x3]. On the first Convolutional Layer, it used neurons with receptive field size F=11, stride S=4 and no zero padding P=0. Since (227 - 11)/4 + 1 = 55, and since the Conv layer had a depth of K=96, the Conv layer output volume had size [55x55x96]. Each of the 55*55*96 neurons in this volume was connected to a region of size [11x11x3] in the input volume. Moreover, all 96 neurons in each depth column are connected to the same [11x11x3] region of the input, but of course with different weights. As a fun aside, if you read the actual paper it claims that the input images were 224x224, which is surely incorrect because (224 - 11)/4 + 1 is quite clearly not an integer. This has confused many people in the history of ConvNets and little is known about what happened. My own best guess is that Alex used zero-padding of 3 extra pixels that he does not mention in the paper.

Parameter Sharing: Parameter sharing scheme is used in Convolutional Layers to control the number of parameters. Using the real-world example above, we see that there are 55*55*96 = 290,400 neurons in the first Conv Layer, and each has 11*11*3 = 363 weights and 1 bias. Together, this adds up to 290400 * 364 = 105,705,600 parameters on the first layer of the ConvNet alone. Clearly, this number is very high.

It turns out that we can dramatically reduce the number of parameters by making one reasonable assumption: That if one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2). In other words, denoting a single 2-dimensional slice of depth as a depth slice (e.g. a volume of size [55x55x96] has 96 depth slices, each of size [55x55]), we are going to constrain the neurons in each depth slice to use the same weights and bias. With this parameter sharing scheme, the first Conv Layer in our example would now have only 96 unique set of weights (one for each depth slice), for a total of 96*11*11*3 = 34,848 unique weights, or 34,944 parameters (+96 biases). Alternatively, all 55*55 neurons in each depth slice will now be using the same parameters. In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

If all neurons in a single depth slice are using the same weight vector, then the forward pass of the Conv layer can in each depth slice be computed as a convolution of the neuron's weights with the input volume (Hence the name: Convolutional Layer). This is why it is common to refer to the sets of weights as a filter (or a kernel), that is convolved with the input.

2.10.5 Backpropagation in CNN

The backward pass for a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters). This is easy to derive in the 1-dimensional case.
As a topic, several papers use 1x1 convolutions, as first investigated by Network in Network. Some people are at first confused to see 1x1 convolutions especially when they come from signal processing background. Normally signals are 2-dimensional so 1x1 convolutions do not make sense (it's just pointwise scaling). However, in ConvNets this is not the case because one must remember that we operate over 3-dimensional volumes, and that the filters always extend through the full depth of the input volume. For example, if the input is [32x32x3] then doing 1x1 convolutions would effectively be doing 3-dimensional dot products (since the input depth is 3 channels).

A recent development is to introduce one more hyperparameter to the Conv layer called the dilation. So far we've only discussed CONV filters that are contiguous. However, it's possible to have filters that have spaces between each cell, called dilation. As an example, in one dimension a filter w of size 3 would compute over input x the following: w[0]*x[0] + w[1]*x[1] + w[2]*x[2]. This is dilation of 0. For dilation 1 the filter would instead compute w[0]*x[0] + w[1]*x[2] + w[2]*x[4]; In other words there is a gap of 1 between the applications. This can be very useful in some settings to use in conjunction with 0-dilated filters because it allows you to merge spatial information across the inputs much more agressively with fewer layers. For example, if you stack two 3x3 CONV layers on top of each other then you can convince yourself that the neurons on the 2nd layer are a function of a 5x5 patch of the input (we would say that the effective receptive field of these neurons is 5x5). If we use dilated convolutions then this effective receptive field would grow much quicker.

2.10.6 Pooling

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce

the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75 percent of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2x2 region in some depth slice). The depth dimension remains unchanged.

In addition to max pooling, the pooling units can also perform other functions, such as average pooling or even L2-norm pooling. Average pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice.

Many types of normalization layers have been proposed for use in ConvNet architectures, sometimes with the intentions of implementing inhibition schemes observed in the biological brain. However, these layers have since fallen out of favor because in practice their contribution has been shown to be minimal, if any.

2.10.7 Fully-connected Layer:

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

2.11 Long Short Term Memory(LSTM):

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber (1997), and were refined and popularized by many people in following work.1 They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!
All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

The key to LSTMs is the cell state.The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"An LSTM has three of these gates, to protect and control the cell state.

LSTM solved the problem of recurrent neural network.Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.

During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.

So in recurrent neural networks, layers that get a small gradient update stops learning. Those are usually the earlier layers. So because these layers don't learn, RNN's can forget what it seen in longer sequences, thus having a short-term memory.

The tanh activation is used to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1.
When vectors are flowing through a neural network, it undergoes many transformations due to various math operations. So imagine a value that continues to be multiplied by let's say 3. You can see how some values can explode and become astronomical, causing other values to seem insignificant.

A tanh function ensures that the values stay between -1 and 1, thus regulating the output of the neural network. You can see how the same values from above remain between the boundaries allowed by the tanh function.

So that's an RNN. It has very few operations internally but works pretty well given the right circumstances (like short sequences). RNN's uses a lot less computational resources than it's evolved variants, LSTM's and GRU's.

The LSTM has got many gates. First, we have the forget gate. This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.Secondlly,To update the cell state, we have the input gate. First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.Now ,we should have enough information to calculate the cell state. First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state.Last we have the output gate. The output gate decides what the next hidden state should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

Chapter 3

Datasets and Pre-processing

We performed comparisons on two types of datasets. The first one is comparatively new and is enriched with data of entities from various background. The second one is a bit archaic and has limited amount of data.

3.1 Physionet Dataset

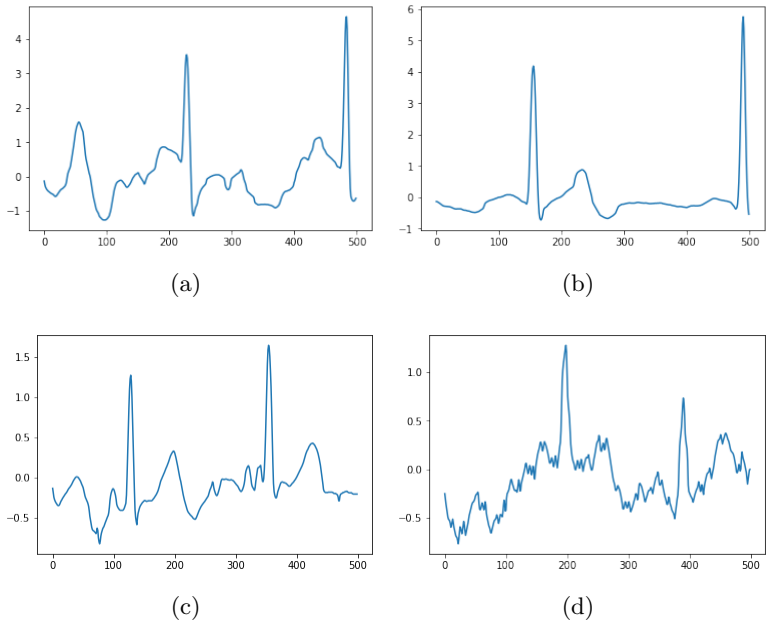
This dataset is obtained from Computing in Cardiology Challenge 2017 which was organised by Physionet decribed in [9]. An aggregation of 8,528 short single lead ECG recordings were provided by AlivCor in this dataset. ECG recordings were sampled as 300 Hz and they have been band pass filtered by the AliveCor device. All data are provided in MATLAB V4 WFDB-compliant format (each including a *.mat file containing the ECG and a *.hea file containing the waveform information). These samples have a minimum duration of 9.0 s and a maximum duration of 61.0 s. There are in total 4 different classes in this dataset. They are - normal sinus rhythm, atrial fibrillation (AF), an alternative rhythm, or is too noisy to be classified. More details on the dataset is shown in Table 3.1. Labels for each classes were provided by AlivCor.

Table 3.1: Class Information in Dataset

Type	Sample Count	Mean Time Length (Seconds)
Normal	5154	31.9
AF	771	31.6
Other Rhythms	2557	34.1
Noisy	46	27.1

3.1.1 Physionet Dataset Samples

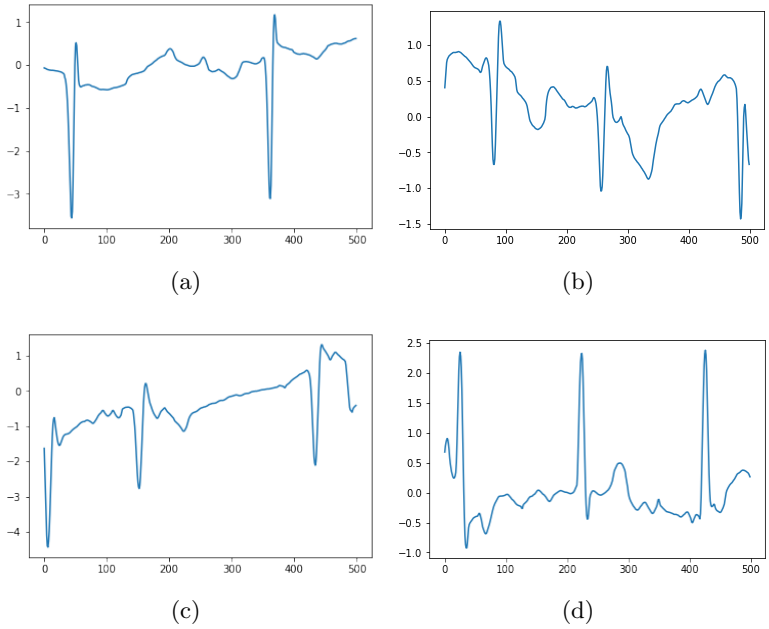
Few samples of each of the described four categories in our dataset is depicted below. All of these signals have been trimmed (up to 500 data points in each sample) for plotting. Fig. 3.1 shows the normal ECG signals. Fig. 3.2 shows the AF rhythms. Fig. 3.3 shows other types of abnormal rhythms. Fig. 3.4 consists of ECG signals that are too noisy to recognize.



(a) (b)

(c) (d)

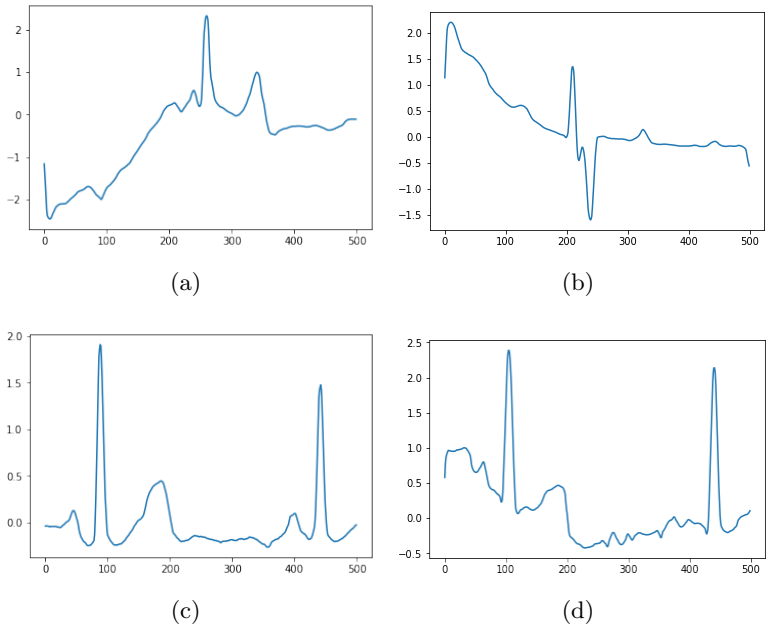
Figure 3.1: Normal ECG Rhythms



(a) (b)

(c) (d)

Figure 3.2: AF Rhythms



(a) (b)

(c) (d)

Figure 3.3: Other types of abnormal Rhythms

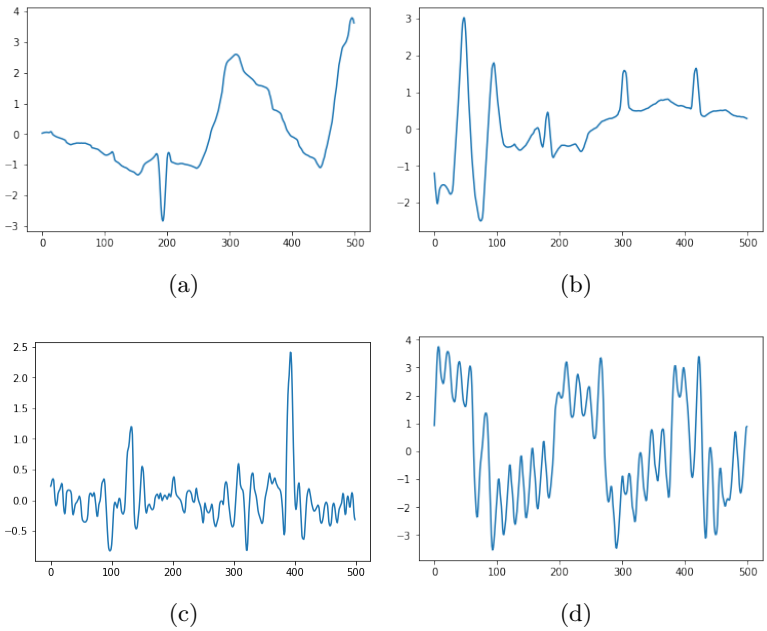


Figure 3.4: Too Noisy Rhythms

3.2 PTB Dataset

The ECGs in this collection were obtained using a non-commercial, PTB prototype recorder with specifications like 16 input channels, (14 for ECGs, 1 for respiration, 1 for line voltage), innput voltage of ±16 mV, compensated offset voltage up to ± 300 mV,input resistance of 100 Ω (DC),resolution of 16 bit with 0.5 μV/LSB (2000 A/D units per mV),bandwidth of 0 - 1 kHz (synchronous sampling of all channels),noise voltage of max 10 μV (pp) and finally 3 μV RMS with input short processing were there .

The database contains 549 records from 290 subjects (aged 17 to 87, mean 57.2; 209 men, mean age 55.5, and 81 women, mean age 61.6; ages were not recorded for 1 female and 14 male subjects). Each subject is represented by one to five records. There are no subjects numbered 124, 132, 134, or 161. Each record includes 15 simultaneously measured signals: the conventional 12 leads (i, ii, iii, avr, avl, avf, v1, v2, v3, v4, v5, v6) together with the three Frank lead ECGs (vx, vy, vz). Each signal is digitized at 1000 samples per second, with 16 bit resolution over a range of ± 16.384 mV. On special request to the contributors of the database, recordings may be available at sampling rates up to 10 KHz. Within the header (.hea) file of most of these ECG records is a detailed clinical summary, including age, gender, diagnosis, and where applicable, data on medical history, medication and in-

terventions, coronary artery pathology, ventriculography, echocardiography, and hemodynamics. The clinical summary is not available for 22 subjects. The diagnostic classes of the remaining 268 subjects are summarized below fromt the :

Diagnostic class	Number of subjects
Myocardial infarction	148
Cardiomyopathy/Heart failure	18
Bundle branch block	15
Dysrhythmia	14
Myocardial hypertrophy	7
Valvular heart disease	6
Myocarditis	4
Miscellaneous	4
Healthy controls	52

Figure 3.5: PTB dataset classes

3.3 PTB Dataset Vs Physionet Dataset

Both these seemingly antithetical storage off data tends to serve the same functionality - trying to provide adequate formatted data for training machine learning models. However, their distinction tends to become clear in matters of the quality of data, the latter taking input from a more diverse demographic context. Some other important comparative analysis regarding the datasets are shown below in the table:

Table 3.2: A comparative analysis of PTB and Physionet Dataset

PTB dataset	**Physionet 2017 Cinc**
It contains 549 recordings	It contains 8528 short recordings
maximum sampling rate was 10KHZ	It was sampled at 300Hz
MATLAB V4 WFDB- compliant format was the default format for the data	MATLAB V4 WFDB- compliant format was the default format for the data
maximum duration was 80.0s and minimum duration was 19.0s for the signals	maximum duration was 61.0s minimum duration was 9s for the signals
There were 9 different classes of heart conditions	There were in all 4 different classes of heart conditions
The Ecg recordings were provided by the National Metrology Institute of Germany	The Ecg recordings were provided by a company by the name of AlivCor

3.4 Pre-Processing of ECG Signals

This section describes the pre-processing of the ECG samples which helped us to get a better and robust result. This was done in two phases - **Sample Reshaping, Dataset Augmentation**.

3.5 Sample Reshaping

In this step we made all the samples equal in shape. The shape of the minimum length sample in the dataset is 1 x 2714 and maximum length sample is 1 x 18286. The method we used here is that - when a sample is smaller than the size of the largest sample in the dataset, then it is concatenated at the end of itself (completely or partially). This recursive process continues until it becomes equal to the shape of the largest sample. A sample and its reshaped form is depicted in Fig. 3.6 and in Fig. 3.7 respectively.

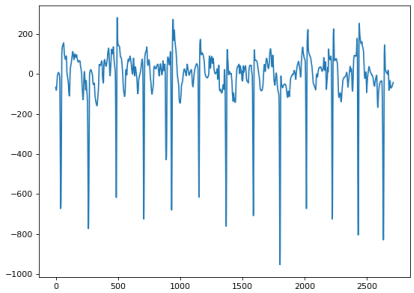


Figure 3.6: Smallest sample in Dataset

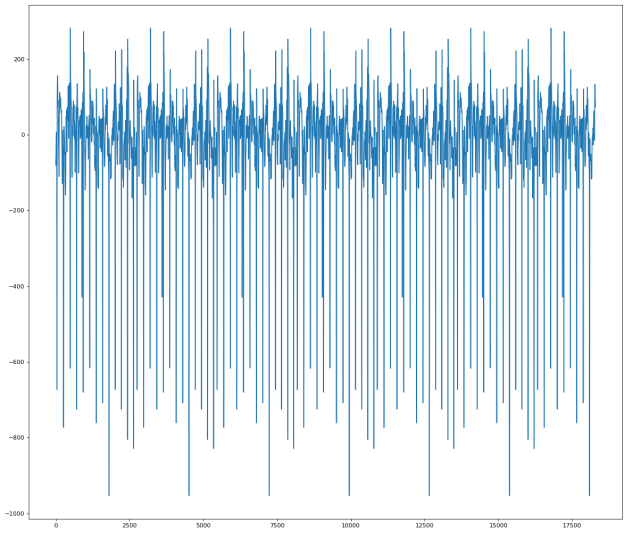


Figure 3.7: Reshaped form of the sample depicted in Fig. 3.6

In Fig. 3.6 we can see that, this sample has a shape of 1 x 2714. Then it is reshaped and the new shape is now 1x18286 shown in Fig. 3.7. After completing sample reshaping step, all of the samples now have same shape which is 1 x 18286.

3.6 Dataset Augmentation

Number of parameters to be trained in our proposed method which is described in Section 4 is relatively larger to the number of samples in out dataset. Due to this phenomenon, we observed over-fitting in our training without data augmentation. Data augmentation not only reduced over-fitting, it also helped us to get a better accuracy results with a dataset consisting of classes with imbalanced frequencies. Using data augmentation, number of samples belonging to each classes were made equal and now, each of these four classes consists of 4542 samples. Now the whole dataset consists of 18168 samples. The augmentation of these time series data is done based on the method described by T. T. Um et al. [10] and instead of using static parameters we used probabilistic parameters for augmenting samples. Here, we performed two types of data augmentation, they are described below.

- **Jittering:** We added noise with each samples, where sigma is a probabilistic variable with a value ranging from 0.5 to 1.5. Sigma is the

standard deviation of noise added with each sample.

- **Scaling:** Scaling is simple multiplying each data points with a value. Let this value be alpha. And for this example our value ranges from 0.2 to 1.4 in a probabilistic manner.

So, our final dataset has a shape of 18168 x 18286 x 1.

Chapter 4

Proposed Method

In this section, we describe our method architecture and training parameters.

4.1 Architecture

Convolutional Neural Network, shortly known as CNN, is a feed-forward neural network that is recently gaining popularity in the field of signal processing, like in the manipulation ECG singals, for their efficiency and portability. CNN is influenced by the biological process in that the connectivity of the neuron resembles the organization of the visual cortex. These are simply the regularized version of a multilayer perceptron. CNN consists of three layers generally: convolutional, pooling and fully connected layer. Convolutional neural network is used to figure out the relation between the inputs. The convolution layer makes convolution on the input and passes it to the next layer. The next layer is the pooling layer that is used to truncate the dimension and streamline the underlying computation. A fully connected layer is used to connect every neuron in one layer to every neuron in another layer and its function is to classify the input into a set number of classes. Recurrent Neural Network (RNN) is a form of Neural Network, which unlike the feed-forward neural network are circular. RNN is particularly used for time series data where there is a lag of unknown duration between the events in the time series and hence are good for the classification, processing and predicting this kind of time series data. Long short term Memory particularly has been designed to keep in a record of long term dependencies with the help of memory cells. An LSTM is made out of an input gate, an output gate and forget gate. LSTM keeps in the hold of the output from the previous layer, or segments in time, and augments it with the new input to produce the corresponding output. The properties of CNN and LSTM makes it suitable for them to be used as some promising tool for the inquiry and the classification of heart condition using the ECG

data. The following sections have been devoted to detailing the description of our proposed work.

4.1.1 CNN + LSTM

Our first proposed method contains an amalgamation of LSTM and CNN. CNN is initially used to pluck out the hidden pattern in the ECG dataset. On the other hand, LSTM is used to capture the amplitude temporal pattern in the ECG dataset. Fig. 4.1 shows the overall layout of the neural network architecture. Our proposed method contains an amalgamation of LSTM and CNN. CNN is initially used to pluck out the hidden pattern in the ECG dataset. On the other hand, as the ECG dataset contains an ordered sequence of entities, LSTM is being used to capture the amplitude-temporal pattern in the ECG dataset. Finally, the multilayer perceptron (MLP) is used to classify the mixed input data into four distinguishable aforementioned heart condition. Sequentially, our model consists of four CNN layers and two LSTM layers. Fig. 4.1 shows the overall layout of the neural network architecture.

The input is morphed into a dimension of 18286 x 1 blocks , and is fed first into a 1-dimensional convolutional neural network. The third dimension, representing the number of samples, is variable. The output contains 128 output of size 18232 x 1 . This output is then trimmed, using the maxpooling layer into dimension of 1823 x 1. The second layer , similarly , takes an input of 1823 x 128 from the maxpooling layer and produces 128 outputs of 1799 x 1. The same process repeats for the two more CNN layers, according to the figure, before our output is being passed to the LSTM.

Our designed LSTM is used to integrate features detected in the consecutive overlapping data blocks. With the help of the internal cycle, it is able to keep record of the preceding steps. As shown in the figure, our first LSTM model generates a subtotal of 128 outputs which have size of 66 x 1 and after the second LSTM , we obtain a single output of size 64 x 1.

Finally, our output is tied with a fully connected MLP , tuned using a softtmax layer, to produce a final output of size 4 x 1, distinguishing amongst four aforementioned types of heart condition. Our model had a total 8,44,164 with similar number of trainable parameters.

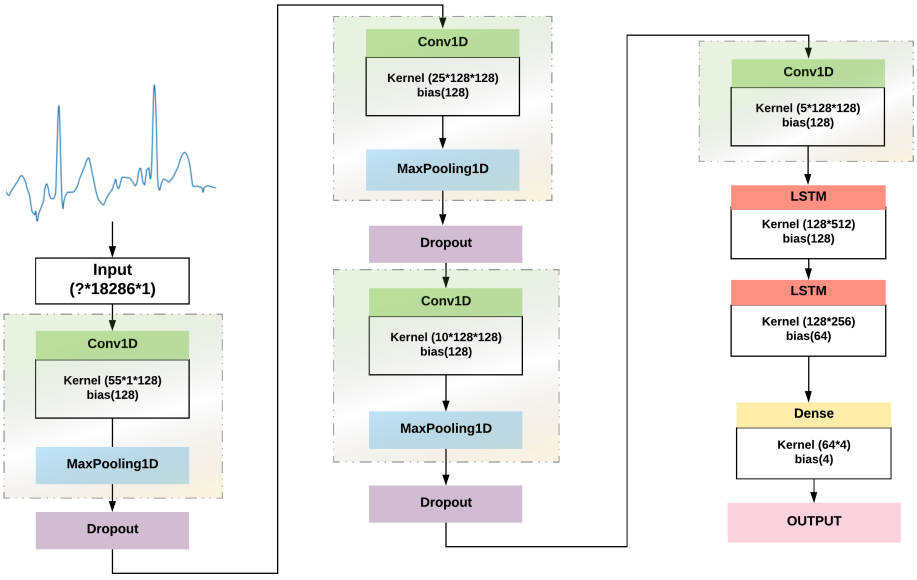


Figure 4.1: CNN+LSTM Architecture

4.1.2 CNN

Here we implement a convolutional neural network. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli. A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better. A kernel is a matrix with the dimensions [h2 * w2 * d1], which is one yellow cuboid of the multiple cuboid (kernels) stacked on top of each other. For each kernel, we have its respective bias, which is a scalar quantity.And then, we have a output for this layer, the green matrix in Fig 2, which has dimensions [h3 * w3 * d2]. For each position of the kernel on the image, each number on the kernel gets multiplied with the corresponding number on the input matrix (blue matrix) and then they all

are summed up for the value in the corresponding position in the output matrix (green matrix). Now, coming to the functionality of the pooling layer in detail. The main purpose of a pooling layer is to reduce the number of parameters of the input tensor and thus - Helps reduce overfitting - Extract representative features from the input tensor - Reduces computation and thus aids efficiency The input to a pooling layer is a tensor. In case of Max Pooling, a kernel is moved across the matrix and for each position the max value is taken and put in the corresponding position of the output matrix. In case of Average Pooling, a kernel of size n*n is moved across the matrix and for each position the average is taken of all the values and put in the corresponding position of the output matrix. This is repeated for each channel in the input tensor. And so we get the output tensor.So, a thing to note is, Pooling downsamples the image in its height and width but the number of channels(depth) stays the same. Now comes Fully connected layer. Fully Connected Layer is simply, feed forward neural networks. Fully Connected Layers form the last few layers in the network. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer. In our model from figure 9 below, we have a input of size 18286x1.The first layer sweeps with a kernel of suitable size 50x1x128 and bias of 128.The output is the used to train another convolutional layer of kernel 1x128x128 and bias 128 .Then a max-pooling of 1D is applied.Then the output is again passed through two other convolutional layer of size 20x128x128 and 1x128x128 .Finally Global Average pooling 1D is applied and the input is finally sieved through three dense layer before producing a output of bias 4 representing the four heart conditions.

Fig. 4.2 shows the overall layout of the neural network architecture.

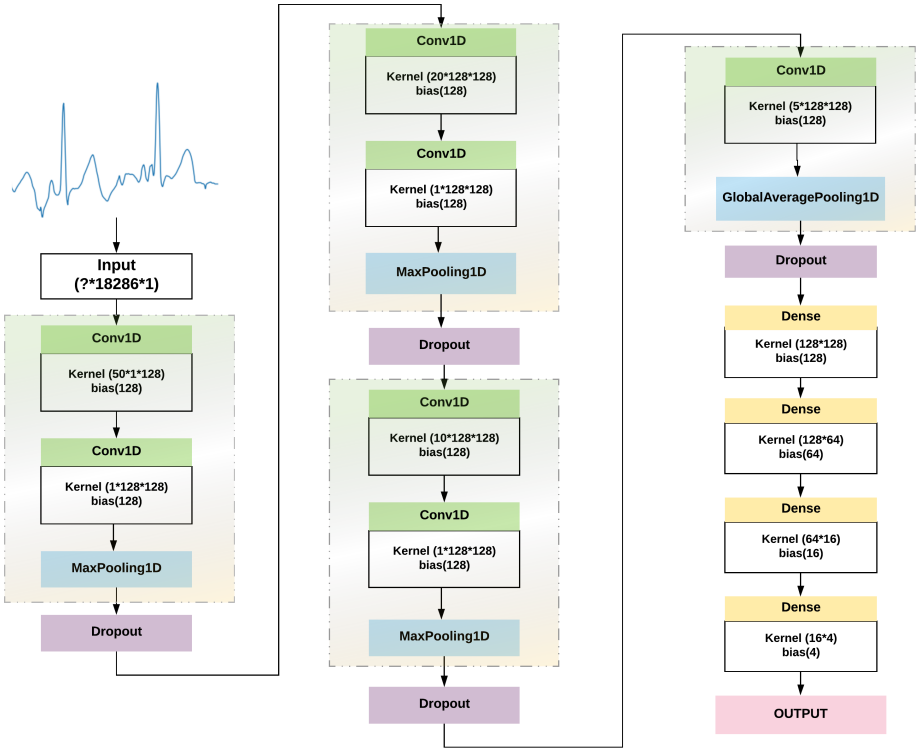


Figure 4.2: CNN Architecture

4.2 Training

For training our model, we used a server with a 2x Intel Xeon Silver 4116 2.1Ghz processor and a RAM of 512 GB DDR4. Integrated HD Graphics were used as the Graphics Processing Unit (GPU). The whole training process was conducted under a python 3.7 environment with Tensorflow and Keras libraries with other necessary libraries. With a batch size of 32, the CNN model was trained for 200 epochs and CNN + LSTM model was trained for 50 epochs. For both of the models Adam optimizer with a learning rate 0.01 was used. The model with the best validation accuracy was saved and it was obtained at 191 epoch for CNN model and at 22 epoch for CNN + LSTM model. The main contributions to our training method are given below:

4.2.1 Weight Initialization

We have implemented weight initialization for the CNN + LSTM model described in Section 4.1.1. Usually, weight values in TensorFlow or Keras are initialized with 0 or random values. If it is initialized with 0, then the

derivative concerning loss function will be W[1] for every w. So, weights will be the same for all win every iteration. Using random values have two problems, if it is very high, it will take a long time to be minimized. If it is too low, then vanishing gradient problem will occur for activation problems like Sigmoid. To overcome this problem, we replaced two LSTM layers with Fully Connected Dense layers and trained the model. Then we saved the weights in a separate file. Now, we initialize the weights from the saved values and again replaced dense layers with LSTM and trained the model.

4.2.2 Adaptive Learning Rate

Instead of fixed learning rate of 0.01, we used a adaptive learning rate. An adaptive learning rate can be implemented in many ways like Time decay or Step decay. We choose Step decay. After every epoch the learning rate was decreased. Algorithm 4.2.3 to calculate the learning rate of the current epoch is given in Algorithm 4.2.3.

4.2.3 Algorithm for Step Decay Learning Rate

Algorithm 1 Calculating Adaptive Learning Rate

initial_lrate = 0.01
drop = 0.05
epochs_drop = 10.0
lrate = initial_lrate * power(drop, floor((1+epoch)/epochs_drop))

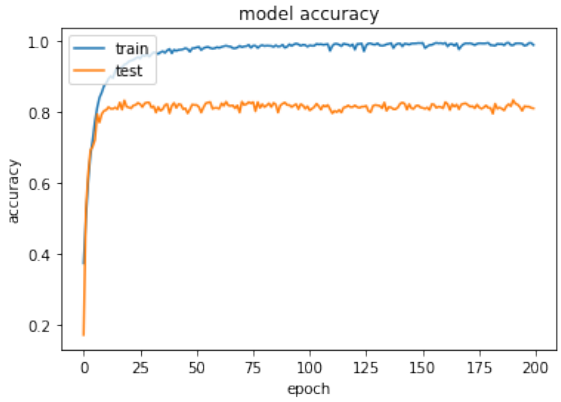


Figure 4.3: Change of model accuracy over epochs (CNN)

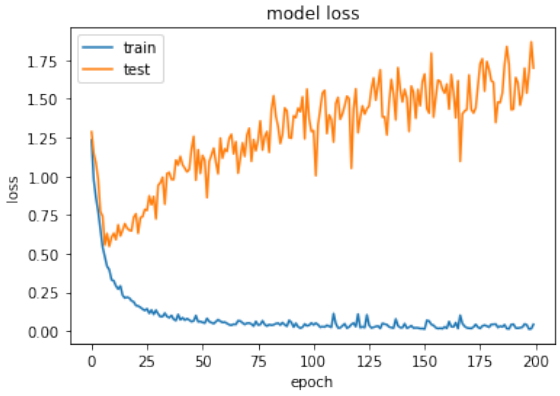


Figure 4.4: Change of loss over epochs (CNN)

Change of model accuracy and loss changed for our CNN + LSTM model over epochs is depicted in Fig. 4.5 and Fig. 4.6 respectively.

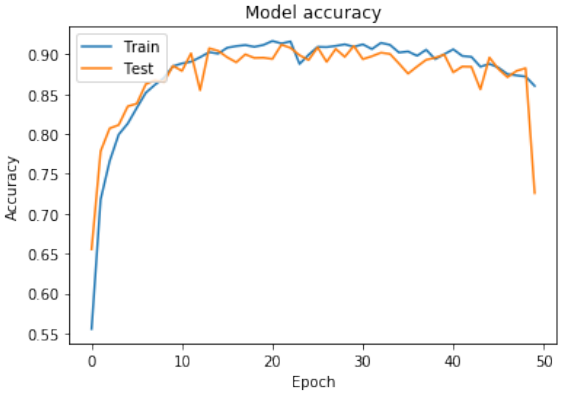


Figure 4.5: Change of model accuracy over epochs (CNN + LSTM)

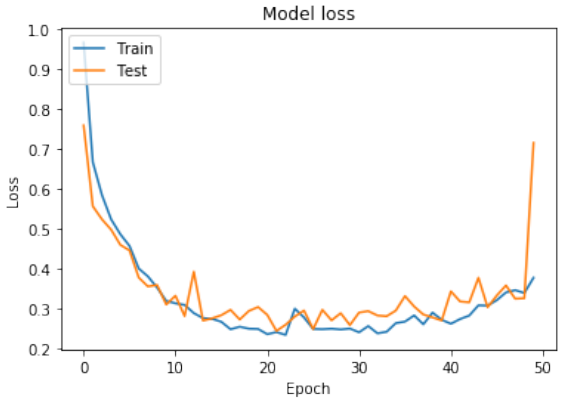


Figure 4.6: Change of loss over epochs (CNN + LSTM)

Adam was set as optimizer and categorical_crossentropy was used as loss function in both cases. The model with the best validation accuracy was saved and it was obtained at 191 epoch for CNN model and at 22 epoch for CNN + LSTM model.

Chapter 5

Performance Evaluation

In this section, we expound on the computer efficiency that we obtained and have reached the promulgated benchmarks.. Table 5.1 lists a comparison of the output of our model alongside similar proposals from other authors mentioned in the beginning .

Table 5.1: Comparison of Proposed method to Other Authors' with same database

Paper Name	Proposed Method	Dataset	No. of Celasss	Accuracy
Martin et.al	CNN+LSTM	Physionet2017 8528 recordings	2	82.1%
Dionisije.et.al	multiclass classifier +random fores	Physionet2017 8528 recording	4	80%
Ruhi et.al	Random Forest	Physionet2017 8528 recordings	2	82.7%
Fernando.et.al	CNN	Physionet 2017 8528 recordings	4	83%
Our method (1)	CNN	Physionet2017 8528 recordings	4	84.3%
Our method (2)	CNN+LSTM	Physionet2017 8528 recordings	4	91.19%

Table 5.2: Results from other authors using dfferent database

Paper Name	Proposed Method	Dataset	No. of Classes	Accuracy
Safaradin et.al	PNN+KNN	Pysionet PTB 549 record	2	76%
Acharya.et.al	CNN	Fantasia database 1285 samples(5s) 514 samples(2s)	2	94.95% for Net(2s) 95.11%for Net(5s)
Acharya et.al	CNN single network	Pysionet PTB 549 record	2	94.95%
Arif et.al	KNN+pruning	Physionet PTB 549 record	2	94.7%
Our method(1)	CNN	Physionet PTB 549 record	2	86%
Our method(2)	CNN + LSTM	Physionet PTB 549 record	2	97%

From Table 5.1, we can observe that the dataset we have used has the largest number of samples and we have also augmented the dataset in such a way so that all kinds of classes have the same number of samples. This not only increased the number of samples but also reduced over-fitting. Because for such a huge number of trainable parameters described in Section 4, the number of samples should be of a good amount to get rid of over-fitting. This makes our model more robust. And our model has achieved a very good accuracy comparing to other authors'.

Our results shows that the output accuracy tends to be deviate within an approximate margin of 3% using the Convolutional Neural Network model and within a margin of approximately 6% when a CNN + LSTM model is being implemented. There may be several plausible reasons pertaining to these slight digressions. Some of these causes relegating the output accuracy's in in different boundaries are hypothesized in the following lines and paragraphs.

Firstly, the different nearly non-mutually exclusive sets of human entities having different gender, race, age in the two data-sets may contribute to the difference in output. It has been proven that the peaks and troughs of ECG signals are dependent on ethnicity. There may also be a causal link to the other aforementioned qualities. Therefore, it may not come as surprise that our model , which is heavily dependent on plucks out pattern , will respond differently to data-sets containing different patters of peaks and trough . This may be the cause of difference in accuracy. Had it been not for the fact that the data-sets were different with respect to the attributes of their en-

tities, we would have seen the output accuracy congregate on the same value.

Secondly, coming to a more sticking point: why does the accuracy of second data-set, PTB, run below the first one. To formulate a cogent reason, we have to again scrutinize their respective datasets. In the first one, i.e the first data-set, the dataset is more voluminous , containing sufficient and sheer amount of data. What does that indicate? This indicates chances of less over-fitting. A data-set containing limited amount of data tends to adapt the model to highly adulate in favor for its output. This leads to over-fitting. An over-fitted data tends to show high accuracy compared to a normal one. This may look tempting at the first sight , however, the adversity to that is the trained model when being run on a new, unseen set of data , performs poorly, even though its accuracy sky-rocketed amidst the training dataset. On the contrary, the second model in brimmed with data from a diverse background. Any possibility, therefore, of getting over-fitted is being ameliorated at the very beginning.

Another interesting fact to notice is that the accuracy for the CNN model , both for the PTB data-set and the Physionet data-set, tends to be low compared to the CNN + LSTM model. The inherent architectural nature of the respective models account for this dichotomy. CNN model is better for image and has proven its merits, giving acuuracy greater tham 50%. On the other hand, LSTM is geared to work well on time series data, which our whole implementation is predicated upon. When a CNN model is being fused with a LSTM model it outperforms in results given by the CNN model alone. The minute dependencies between corresponding intervals is delineated meticulously by the LSTM model conjoined with the CNN model.

Chapter 6

Conclusion and Future Works

We developed a neural network architecture, CNN +LSTM, for the detection classification of ECG signal from the PhysioNet/2017 challenge into four types. Our dataset is relatively new and is enriched with entities that have a diverse background, making our dataset robust. We proposed an augmentation scheme in our paper that was able to push the accuracy to the epitome. We made a comparison of our results with other similar classification mechanisms from different authors on a similar line of research. Our results show promising outcome with an accuracy of 91.19%. Future studies may take into account of different features such as the T wave, QRS complex e.t.c and their corresponding relation with the heart conditions, hence provide new insight into features reflecting specific heart maladies. Further, our model can also be integrated into software for mobile devices that are geared to detect ECG signals.

Chapter 7

Appendix

7.1 Code for Library and Input data

```
from sklearn.metrics import confusion_matrix, accuracy_score
from keras.callbacks import ModelCheckpoint
#from biosppy.signals import ecg
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import MinMaxScaler, RobustScaler
import pandas as pd
import scipy.io as sio
from os import listdir
from os.path import isfile, join
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation,
Dropout, Conv2D, MaxPooling2D, Flatten, LSTM, Conv1D,
GlobalAveragePooling1D, MaxPooling1D
from keras import regularizers
import matplotlib.pyplot as plt
```

```
import pickle
with open('Tasnim/train_18168_19092019_ECG_augmented.pickle', 'rb')
as f:
    X_train_aug, Y_train_aug = pickle.load(f)
with open('Tasnim/validation_853_19092019_ECG_augmented.pickle', 'rb')
as f:
    X_val, Y_val = pickle.load(f)
```

```
X = np.concatenate((X_train_aug, X_val), axis=0)
Y = np.concatenate((Y_train_aug, Y_val), axis=0)

from sklearn.model_selection import train_test_split
X_train,X_val,Y_train,Y_val = train_test_split(X, Y, test_size=0.2,
shuffle=True)
X_train.shape,Y_train.shape,X_val.shape,Y_val.shape
```

7.2 Code for CNN Model Training

```
number_of_classes = 4
# def create_model():
model = Sequential()
model.add(Conv1D(128, 50, activation='relu', input_shape=(18286, 1)))
model.add(MaxPooling1D(10))
model.add(Dropout(0.2))
model.add(Conv1D(128, 20, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Dropout(0.2))
model.add(Conv1D(128, 10, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Dropout(0.2))
model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalAveragePooling1D())
# model.add(Flatten())
model.add(Dense(128, kernel_initializer='normal', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, kernel_initializer='normal', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(16, kernel_initializer='normal', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(number_of_classes, kernel_initializer='normal',
activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
checkpointer = ModelCheckpoint(filepath='Conv_models_cnn_aug_2/
Best_model.h5', monitor='val_acc', verbose=1, save_best_only=True)
hist = model.fit(X_train_aug, Y_train_aug,
validation_data=(X_val, Y_val), batch_size=64, epochs=200, verbose=2, shu
pd.DataFrame(hist.history).to_csv(path_or_buf='Conv_models_cnn_
aug_2/History.csv')
predictions = model.predict(X_val)
```

```
score = accuracy_score(change(Y_val), change(predictions))
print('Last epoch\'s validation score is ', score)
df = pd.DataFrame(change(predictions))
df.to_csv(path_or_buf='Conv_models_cnn_aug_2/Preds_' +
str(format(score, '.4f')) + '.csv', index=None, header=None)
pd.DataFrame(confusion_matrix(change(Y_val),
change(predictions))).to_csv(path_or_buf='Conv_models_cnn_aug_2
/Result_Conf' + str(format(score, '.4f')) + '.csv', index=None,
header=None)
```

7.3 Code for CNN + LSTM Model Training

```
model = Sequential()
model.add(Conv1D(128, 55, activation='relu', input_shape=(18286, 1)))
model.add(MaxPooling1D(10))
model.add(Dropout(0.1))
model.add(Conv1D(128, 25, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Dropout(0.1))
model.add(Conv1D(128, 10, activation='relu'))
model.add(MaxPooling1D(5))
model.add(Dropout(0.1))
model.add(Conv1D(128, 5, activation='relu'))

model.add(LSTM(128, activation='tanh', return_sequences=True))
model.add(LSTM(64, activation='tanh', return_sequences=False))
model.add(Dense(4, kernel_initializer='normal', activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])


filepath="Conv_models_aug_latest_2/CNN+LSTM_weights-improvement-
{epoch:02d}-{val_acc:.2f}.h5"
checkpoint = keras.callbacks.ModelCheckpoint(filepath,
monitor='val_acc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
history = model.fit(X_train, Y_train,
          batch_size=32,
          epochs=50,
          verbose=2,
          callbacks=callbacks_list,
          shuffle = True,
          validation_data=(X_val, Y_val))
```

```
model.save('Conv_models_aug_latest_2/CNN+LSTM_tasnim+07_19_aug')
model.save_weights('Conv_models_aug_latest_2/
weights_CNN+LSTM_W_2007_aug.h5')
```

Bibliography

[1] U Rajendra Acharya, Hamido Fujita, Oh Shu Lih, Muhammad Adam, Jen Hong Tan, and Chua Kuang Chua. Automated detection of coronary artery disease using different durations of ecg segments with convolutional neural network. *Knowledge-Based Systems*, 132:62–71, 2017.

[2] Ruhi Mahajan, Rishikesan Kamaleswaran, John Andrew Howe, and Oguz Akbilgic. Cardiac rhythm classification from a short single lead ecg recording via random forest. In *2017 Computing in Cardiology (CinC)*, pages 1–4. IEEE, 2017.

[3] U Rajendra Acharya, Hamido Fujita, Shu Lih Oh, Yuki Hagiwara, Jen Hong Tan, and Muhammad Adam. Application of deep convolutional neural network for automated detection of myocardial infarction using ecg signals. *Information Sciences*, 415:190–198, 2017.

[4] Naser Safdarian, Nader Jafarnia Dabanloo, and Gholamreza Attarodi. A new pattern recognition method for detection and localization of myocardial infarction using t-wave integral and total integral as extracted features from one cycle of ecg signal. *Journal of Biomedical Science and Engineering*, 7(10):818, 2014.

[5] Muhammad Arif, Ijaz A Malagore, and Fayyaz A Afsar. Detection and localization of myocardial infarction using k-nearest neighbor classifier. *Journal of medical systems*, 36(1):279–289, 2012.

[6] Martin Zihlmann, Dmytro Perekrestenko, and Michael Tschannen. Convolutional recurrent neural networks for electrocardiogram classification. In *2017 Computing in Cardiology (CinC)*, pages 1–4. IEEE, 2017.

[7] Dionisije Sopic, Elisabetta De Giovanni, Amir Aminifar, and David Atienza. Hierarchical cardiac-rhythm classification based on electrocardiogram morphology. In *2017 Computing in Cardiology (CinC)*, pages 1–4. IEEE, 2017.

[8] Fernando Andreotti, Oliver Carr, Marco AF Pimentel, Adam Mahdi, and Maarten De Vos. Comparing feature-based classifiers and convolutional neural networks to detect arrhythmia from short segments of ecg. In *2017 Computing in Cardiology (CinC)*, pages 1–4. IEEE, 2017.

[9] Gari D Clifford, Chengyu Liu, Benjamin Moody, H Lehman Li-wei, Ikaro Silva, Qiao Li, AE Johnson, and Roger G Mark. Af classification from a short single lead ecg recording: the physionet/computing in cardiology challenge 2017. In *2017 Computing in Cardiology (CinC)*, pages 1–4. IEEE, 2017.

[10] Terry T. Um, Franz M. J. Pfister, Daniel Pichler, Satoshi Endo, Muriel Lang, Sandra Hirche, Urban Fietzek, and Dana Kulić. Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, ICMI 2017, pages 216–220, New York, NY, USA, 2017. ACM.