SEPTEMBER 2020

M.Sc. in Electrical Electronics Engineering

KASIM SHOBAK

REPUBLIC OF TURKEY

GAZİANTEP UNIVERSITY

GRADUATE SCHOOL OF NATURAL & APPLIED SCIENCES

SPARSE REPRESENTATION BASED ECG HEARTBEAT CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

M.Sc. THESIS

IN

ELECTRICAL AND ELCTRONICS ENGINEERING

BY

KASIM SHOBAK

SEPTEMBER 2020

SPARSE REPRESENTATION BASED ECG HEARTBEAT CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

M.Sc. Thesis

in

Electrical and Electronics Engineering

Gaziantep University

Supervisor

Assoc. Prof. Dr. Sema KOÇ KAYHAN

by

Kasim SHOBAK

September 2020

©2020 [Kasim SHOBAK]

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Kasim SHOBAK

ABSTRACT

SPARSE REPRESENTATION BASED ECG HEARTBEAT CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

SHOBAK, Kasim
M.Sc. in Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Sema KOÇ KAYHAN
September 2020
48 pages

The cardiovascular disorders can be monitored using electrocardiogram (ECG or EKG). Since the interpretation of arrhythmias disorders is difficult by human, it is of great importance to develop automated models to help physicians classify arrhythmias in a real-time and accurate manner. The classification of ECGs signals is challenging as there are similarities among ECG heartbeats within inter-patient and intra-patient recordings, in addition to the noise resulted from ECGs' apparatus and settings.

This paper presents a new model, which uses sparse representations of ECG signals and classifies them using a Convolutional Neural Network. An orthogonal matching pursuit method is implemented to produce appropriate sparse-represented signals based on an overcomplete Gabor dictionary. The sparsified (SR-ed) inputs are fed into the proposed network which consists of 5 successive double 1-D convolutional layers to classify 5 ECG categories. In contrast to traditional algorithms, this method does not require preprocessing such as feature extraction or selection.

The MITBIH Arrhythmia database is used to evaluate the performance of the developed model. The model is tested using non-SR-ed, SR-ed and compressed SR-ed (CSR-ed) data for both imbalanced and balanced classes datasets. The experimental results show that the model accuracies of the SR-ed data reach up to 97.44% and 97.32% for the imbalanced and balanced datasets respectively. While for the non-SR-ed ones, the accuracies reach up to 98.84% and 98.69% for the imbalanced and balanced datasets respectively. Finally, as for the CSR-ed inputs, the accuracies reach up to 93.96% and 83.38% for the imbalanced and balanced datasets respectively.

Key Terms: Sparse Representation, Orthogonal Matching Pursuit, Overcomplete Gabor Dictionary, Convolutional Neural Networks, ECG, Arrhythmia

ÖZET

TIBBI SİNYAL SINIFLANDIRMASI İÇİN SEYREK TEMSİL TABANLI ÇEVRİMSEL SİNİR AĞLARININ KULLANILMASI

SHOBAK, Kasim
Yüksek Lisans Tezi, Elektrik-Elektronik Mühendisliği
Tez Yöneticisi: Doç. Dr. Sema KOÇ KAYHAN
Eylül 2020
48 sayfa

Kardiyovasküler bozukluklar, elektrokardiyogram (EKG veya ECG) kullanılarak izlenebilir. Kalp rahatsızlıklarının yorumlanması insanlar için zor olduğundan, doktorların aritmi varyasyonlarını gerçek zamanlı ve doğru bir şekilde sınıflandırmalarına yardımcı olacak otomatik modeller geliştirmek büyük önem taşımaktadır. EKG cihazlarından ve ayarlarından kaynaklanan gürültüye ek olarak hasta kayıtlardaki EKG kalp atışları arasında benzerlikler olduğundan EKG sinyallerinin sınıflandırılması zordur.

Bu çalışmada, tamamlanmış bir Gabor sözlüğü üzerinden EKG sinyallerinin seyrek temsilini kullanan ve bunları Evrişimli Sinir Ağı kullanarak sınıflandıran yeni bir model sunumaktadır. Aşırı tamamlanmış Gabor sözlüğü aracılığıyla uygun seyrek vektörler üretmek için Dik Eşleştirmeme Takip Algoritması kullanılır. Üretilen seyrek girdiler, 5 farklı EKG kalp atışı kategorisini otomatik olarak sınıflandırmak için, önerilen beş ardışık çift 1-D evrişimli katmandan oluşan ağı, beslemek için kullanılır. Geleneksel algoritmaların aksine, bu yöntem, özellik çıkarma veya seçme gibi ön işleme gerektirmez.

Geliştirilen modelin performansını değerlendirmek için MITBIH Aritmi veritabanı kullanılmıştır. Önerilen model, hem seyrek hem de seyrek olmayan girdiler kullanılarak test edilmiştir. Model, dengesiz sınıfların etkisini azaltmak amacıyla sınıfları artırılmış ve dengelenmiş veriler üzerinde de test edilmiştir. Deneysel sonuçlar, dağıtılmış girdiler için model doğruluğunun dengesiz ve dengeli veriler için sırasıyla% 97.44 ve% 97.32'ye ulaştığını göstermektedir. Seyrek olmayanlar için ise bu performans dengesiz ve dengeli veriler için sırasıyla% 98,84 ve% 98,69 olarak gerçekleşmiştir.

Anahtar Kelimeler: Seyrek Temsil, Dik Eşleşme Takip Algoritması, Aşırı Gabor Sözlüğü, Konvolüsyonel Sinir Ağları, EKG, Aritmi

''Dedicated to my family''

ACKNOWLEDGMENTS

Undertaking this master has been a truly life-changing experience for me and it would not have been possible to do without the support and guidance that I received from many people.

I have been very pleased to have this great opportunity to further my education in Gaziantep University, Turkey. I deeply thank the faculty, staff, students, and friends for their support and encouragement.

I would like to express my sincere gratitude and thanks to my advisor Assoc. Prof. Dr. Sema KOÇ KAYHAN, for the continuous support of my master study and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my master study.

Finally, I must express my very profound gratitude to my beloved family for providing me with unfailing support and continuous encouragement throughout my life. This accomplishment would not have been possible without them. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIEATIOS

CNN	Convolutional Neural Network
OMP	Orthogonal Matching Pursuit
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
GD	Gabor Dictionary
SR	Sparse Representation
PNSR	Peak Noise Signal Ratio
DWT	Discrete Wavelet Transform
CK	Convolutional Kernel
AF	Activation Function
FCL	Fully Connected Layer
CL	Convolutional Layer
SVM	Support Vector Machine
CSR	Compressed Sparse Representation
A_{cc}	Accuracy
P_P	Precision
S_E	Sensitivity
F_S	F1-score
LSTSVM	Least Square Twin SVM
PSO	Particle Swarm Optimization
WPE	Wavelet Packet Entropy
RR	The time between the R peaks of two heartbeats
RF	Random Forests
SMOTE	Synthetic Minority Over-Sampling Technique

CHAPTER I: INTRODUCTION

1.1. Motivation of Study

Artificial Intelligence (AI) refers to systems or machines that mimic the human intelligence in implementing objectives based on the information collected from problems environments.

AI utilizes machine and deep learning to enhance human being capabilities in different domains. Many experiments have been achieved to develop AI models aiming to assist medical doctors in different medical works. Among the medical problems that AI has enhanced is the medical images classification such as ECGs or EEGs classifications.

Electrocardiogram is also a noninvasive diagnostic that is it provides heartbeat recordings through which doctors can identify crucial heart rhythms that can cause severe conditions or even death. The classical visual interpretation of ECGs can be slow and prone to error. Therefore, within recent years, a lot of studies have been focusing on developing automatic classification systems of arrhythmias. Researchers have deployed Machine Learning (ML) (Zhou & Li, 2016; Sahoo, Sabut, Kanungo, & Behera, 2017; Raj & Ray, 2018) and Deep Learning (DL) (Pandey & Janghel, 2019; Acharya, et al., 2017; Kachuee, Fazeli, & Sarrafzadeh, 2018; Zubair, J, & C, 2016) algorithms to build automated ECG classification models.

CHAPTER II: LITERATURE REVIEW

2.1. Electrocardiogram

An electrocardiogram, which is known as ECG or EKG, is a test by which the electrical activity of heartbeats is measured. With every beat, an electrical wave (or impulse) travels across the heart. This wave forces the muscle to pump blood to the body from the heart. The timing of the lower and top chambers can be shown in a normal ECG heartbeat.

There are three waves caused by the electrical activity of the heart. The first wave which is called “P wave” and can be made by the upper chambers or left and right arteria, following a flatten line when the electrical wave travels to the lower chambers. The ventricles or the left and right chambers create the second wave which is called “QRS complex”. The “T wave”, which is the final wave, represents the return to relaxing state for the ventricles or the electrical recovery.

An ECG provides two substantial types of information. First, though the measurement of time intervals spent on the ECG, a doctor is able to determine the duration needed by the electrical wave to travel through the heart. Knowing how long the electrical wave takes to pass from a given part of the heart to another one, shows whether the electrical activity is slow or normal, irregular or fast. Second, though the measurement of the electrical activity’s amount travelling through the muscle of the heart, a cardiologist can find out whether any part of the heart is overworked or too large.

It can be used to investigate symptoms of a possible heart problem, such as chest pain, palpitations (suddenly noticeable heartbeats), dizziness and shortness of breath.

An ECG can be utilized to diagnose symptoms and signs of a potential heart problem, like palpitations (suddenly remarkable heartbeats), shortness of breath, dizziness, or chest pain.

Through ECGs we can detect:

Heart attacks – the case when the blood supply to the heart is blocked suddenly.

- Cardiomyopathy – the case when the walls of the heart become enlarged or thickened.
- Coronary heart disease – where the blood supply of the heart is interrupted or blocked by a growth of fatty substances.

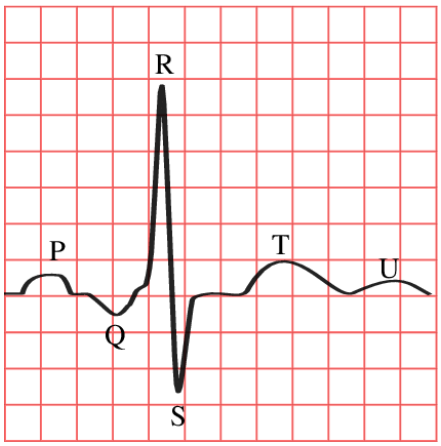

Figure 2.1: Waves of ECG Lead II

- Arrhythmias– where the heartbeats are too quick, too slow, or irregular. Our study focuses on this type, as the dataset used here including data points of this disease.

In an ECG, there are 12 leads used to represent the heart's electrical activity which are recorded from four limb electrodes attached to surface of the following limbs: the right arm (RA), the left arm (LA), the left leg (LL), and the right leg (RL).

Among the 12 leads, we need to know here about the Lead II by which the ECG dataset, utilized in this work, was recorded as we will see in the next section. Lead II represents the inferior view of the body and calculated by analyzing activity between the RA and LL electrodes. Figure 1 shows the shape of the wave recorded by Lead II.

2.1.1. ECG Dataset

The MITBIH data provided by PhysioNet (Mark & G., May-June 2001; PhysioNet, 2001), which is recommended by the American Association of Medical Instrumentation (AAMI) to be used in such studies, is used to gauge the performance of the proposed model. The dataset includes 48 half-hour ECG recordings obtained from 47 patients. These recordings were sampled at the sampling rate of 360 Hz with half an hour duration. This data includes recordings captured from two ECG leads. The data of the lead II is only considered in this study as the same as most of the studies reported in the literature.

The total number of these recordings is 109,446 and none of them are excluded in this study as implemented in some studies (Zubair, J, & C, 2016; Raj & Ray, 2018). The 5 categories of the MITBIH data are as follows: normal (N), supraventricular ectopic (S), ventricular ectopic (V), fusion (F), and unknown (Q). The dataset is separated into training and testing groups with a ratio of 80-20. The training group includes 87,554 signals, divided into imbalanced 5 classes.

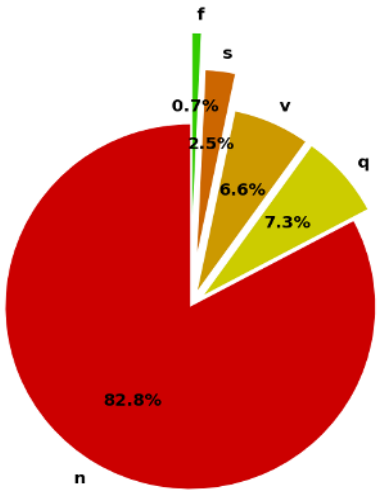

Figure 2.2: Percentages of ECG Categories

The original classes N, Q, V, S, and V compounds the training group with the percentages 82.8%, 7.3%, 6.6%, 2.5%, and 0.7% respectively. Figure 2.2 shows the percentages of ECG classes within the training set, whereas **Table 2.1** shows the number of samples for each category. To evaluate the performance of the proposed model with taking into consideration balanced classes, we opt to generate synthetic data using bootstrapping procedure which uses random sampling with replacement. Consequently, each class of the augmented training data has 72,471 signals.

Table 2.1: The Distribution of ECG Classes Within the Training Set

Class	Code	Diagnosis	No. of samples
0	N	Normal	72,471
4	Q	Unknown	6,431
2	V	ventricular ectopic	5,788
1	S	supraventricular ectopic	2,223
3	F	Fusion	641

2.2. Artificial Intelligence

AI which is commonly used to refer to artificial intelligence which is considered as a manner of enabling computers to be intelligent. It is a computer science branch in which we can script code to program the machine to behave intelligently.
As stated by John McCarthy, who is the father of AI, "The science and engineering of making intelligent machines, especially intelligent computer programs".

2.2.1. Motivation & Intuition behind AI

The motivation and intuition behind AI originate from the brain of humans which is the super powerful gift delivered to humanity. The top intelligent species on earth are humans. And Neocortex is the reason behind this powerful brain. Neocortex gives us the power to act, function, think, recall, and so on. Commonly there is a myth saying that whales are superior to humans in being smart, which is not correct because whales have in their neocortex only 4-5 layers; which do not have a chance to be compared to human beings that have more significant neocortex with six layers.
Our brain is wired such that no images are stored by the Neocortical Memory, instead of that, patterns (whether spatial or temporal patterns) get stored there. The brain is able to predict what it encounters and what it should hear or see, when these memories or patterns are recalled by the brain. A surprise sense is triggered when things get predicted incorrectly, however, usual feeling happens to the human when predicting correctly.
In childhood, parents are used to teaching their kids the English alphabet whose letters compose words when getting combined; in turn words compose sentences when getting combined; then, sentences compose paragraphs. Kids start to learn from the surroundings as they grow older, like when they are playing. For example, when a kid falls, the brain would recall avoiding the same mistakes. Then kids start to learn more developed things like outcomes, for example, 45 is the dot product of 9 and 5, and C letter comes after B and before D.
Similarly, humans wanted the machine to be enabled to learn depending on the data fed into it. Then, the machine is expected to have enough intelligence to make proper decisions taking into account what it has previously learned.
According to Wikipedia, “AI, which is sometimes called as machine intelligence, is the ability of machines to demonstrate intelligence, in comparison to the nature intelligence showed by humans and animals. AI researches are referred to as the studies of “intelligent agents” within computer science: devices that perceive their surroundings and make decisions that maximize their probabilities of accomplishing goals successfully.”

2.2.2. Why do we need Intelligent Machines?

Automation of tasks that people do redundantly is the main reason behind the necessity of AI. For example, in terms of sophisticated calculations, computers are faster and more accurate than humans in performing math operations on several-digits numbers. They can give results in a matter of milliseconds while humans cannot.

Analogously, if a computer is trained on some records or data, based on these records, it can simply be able to predict future outcomes, since it can learn and understand some structures and patterns from the past records or data.

For instance, let us assume that doctors need to see on a brain MRI image, which is degraded to the extent of lacking sharpness and clarity. In such scenarios, doctors can face difficulty in judging on such images, and here where the AI role comes. The resolution of MRI images can be enhanced by benefiting from a machine learning algorithm applied previously on high-resolution images. As a result, bad MRI images can be projected into a super-resolution space.

2.3. Machine Learning

Basically, ML is a manner to perform AI. Like AI, ML is a computer science branch through which you study or devise the design of algorithms that are able to learn.

There are a variety of ML algorithms as such as:

- Naive Bayes,
- Decision trees,
- Support vector machine
- Random forest
- K-means clustering,
- K-nearest neighbor,
- Hidden Markov model,
- Gaussian mixture model.

In ML, we can use one of the algorithms aforementioned, which enables machines to understand and learn automatically without getting programmed repeatedly.

The algorithms receive data which have diverse features or attributes, giving us a decision boundary depending on the data provided. The algorithm trained itself, when it has interpreted and learned the data, at this point, we can test our algorithm by exposing inputs as test data points and expect some results to be given by it.

For instance, let us assume we aim to predict a house price, by providing a dataset including 1000 houses with the same attributes or features such as the cost and the number of rooms. Now, we intend to use a ML algorithm like decision trees and feed this dataset of two features into it. In this case, the input of the algorithm will be the room number, whereas the predicted output is the price of the house. This algorithm will attempt to learn and recognize a relationship that links the cost of the house to the number of rooms.

At the testing phase, we just need to provide the number of rooms to the algorithm as an input, and it is expected to predict the house price correctly.

As shown in Figure 2.3 , AI includes several subcategories. and ML is considered one of them and includes a lot of learning algorithms.

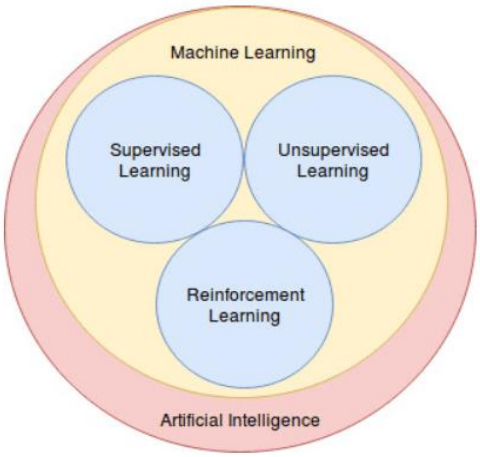

Figure 2.3: The Circle of AI

- **Supervised Learning:** In this kind of ML, the whole learning technique is controlled. These algorithms mainly aim to predict outcomes by providing training dataset and training classes or labels, also known as classification of data points. Since we inform the algorithm at the training time what labels it should predict, we call it supervised learning. The following example helps us understand it:

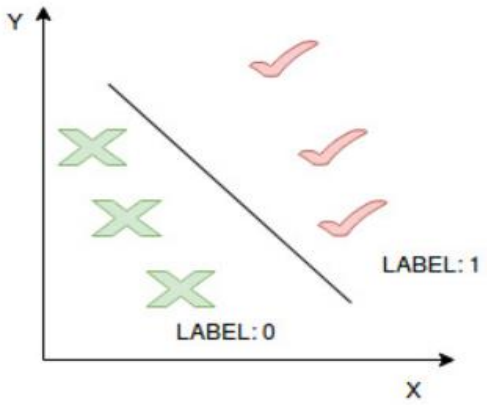

Figure 2.4: Supervised Learning

Let us assume we have 10k images, 5k dog and 5k cat images, each dog and cat image have a label one and zero respectively. The aim of supervised learning is to detect patterns within the data with having the labels as constraints. This technique aims to draw a boundary line that splits the images data in half, such that when we pass a testing dog image with no labels to the algorithm, it gives us 1 as a label, telling that the image is correctly classified/predicted as a dog (Figure 2.4).

- **Unsupervised Learning:** In contrast to supervised learning, training labels are not provided here for the training samples. The algorithms are developed to find proper patterns and structure within the data. When the algorithm apparently identifies the consistent patterns, it starts to cluster the analogous data points together, and the different ones are gathered in other clusters. We usually use these algorithms in analysis and visualization purposes, like converting data with high dimensions into lower ones (Figure 2.5).

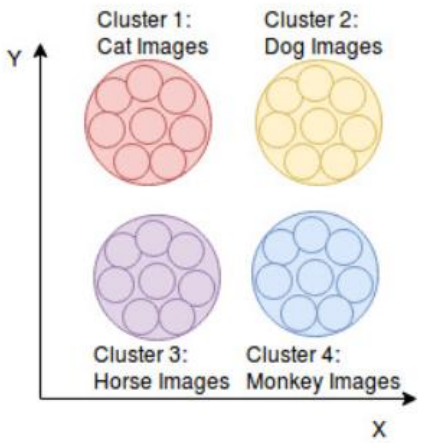

Figure 2.5: Unsupervised Learning

- **Reinforcement Learning**: It is one of the ML types that uses an agent (e.g. a robot) that is able to quantify results and take actions, thus, being able to learn how to act in an environment. If the robot performs a true response, it receives a point as a reward, which increases the confidence of the robot to perform similar actions. It utilizes Markov Decision Process (MDP).

  For instance, as shown in Figure 2.6 , the robot performs an action in the maze (environment), having the eyes as an interpreter, it receives a reward once a correct move is taken, and it takes the further action based on its current status.

Examples of cool applications regarding this technique are playing games (such as Mario, Chess), control system of traffic light, robotics, etc.

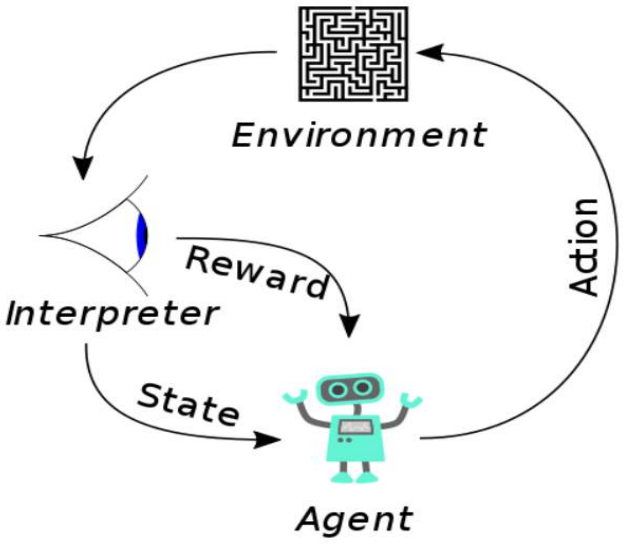

Figure 2.6: Reinforcement Learning

2.3.1. Related works to Machine Learning

Although ML methods showed decent ECG classification results, they still require pre-processing of input, such as noise removal or feature extraction. Besides, their learning curves might suffer from overfitting and exhibit less accuracy as expose to unseen data. The following paragraph explains 3 examples of the recent ML methods.

In (Zhou & Li, 2016), they separated the MITBIH dataset into training and testing datasets each one has 22 recordings. They proposed a method wavelet packet entropy (WPE) to decompose the ECG signals, afterwards, they calculated entropy from the decomposed coefficients as representative features, and finally used Rain Forest (RF) as a classifier to classify the five original heartbeats. They concluded that the ability of WPE +RR is more powerful than that of ICA+RR, DWT+RR or WPD +RR for feature extraction. They achieved accuracy of 94.6% using RF which is higher than the state-of-the-art classifiers such as KNN, DT, PNN, and SVM which achieve accuracy of 90.94%0, 85.50%, 92.50%, and 89.03% respectively.

In (Sahoo, Sabut, Kanungo, & Behera, 2017), they proposed an enhanced algorithm to detect QRS complex and extract features using multiresolution wavelet transform, and finally classify four types of heartbeats (normal, LBBBs, RBBBs, and Paced beat) using MLP-BP and SVM classifiers. They enhanced the signals using discrete wavelet decomposition up to 10 levels. They removed the noise by discarding D1, D2 & A10, and detected QRS complex by selecting 15% of (D3+D4+D5). The algorithm achieved accuracy of 96.67% and 98.39% in NN and SVM classifiers respectively.

In (Raj & Ray, 2018) presented the first study which used SR with an overcomplete GD in ECGs classification in the literature. They decomposed an ECG signal into elementary waves using the GD. They used discrete wavelet transform (db4) and R-peak detection using Pan Tompkins as preprocessing the signals. For each significant atom in the dictionary, they extracted four features such as, time delay, frequency, width parameters, and square of expansion coefficient.

Afterwards, these features are concatenated and further classified using ML techniques such as, SVM, k-NN, PNN, and RBFNN. Further, the learning parameters of the classifiers are optimized using ABC and PSO techniques. Their method is evaluated over two analysis schemes, category-based scheme which is utilized to recognize 16 classes of ECGs, and personalized scheme in which MITBIH database is projected into 5 classes. In the personalized scheme, only 44 out of 48 subjects are included in the experiment; both the training and testing datasets have 22 subject recordings. The top accuracy they achieved in the personalized scheme is done using LSTSVM+PSO with accuracy of 89.93%.

2.4. Deep Learning

It is a subcategory of machine learning. Similar to machine learning, deep learning also has supervised, unsupervised, and reinforcement learning in it. As discussed earlier, the idea of AI was inspired by the human brain. So, let's try to connect the dots here, deep learning was inspired by artificial neural networks and artificial neural

networks commonly known as ANN were inspired by human biological neural networks. Deep learning is one of the ways of executing machine learning.

Like ML, DL is a subcategory of ML, and unsupervised, supervised, and reinforcement learnings are also included in DL. As aforementioned earlier, the human brain inspired the concept of AI. Artificial neural networks (ANNs) inspired DL, and ANNs are simulations to the biological human neural networks. DL is one of the approaches to execute ML.

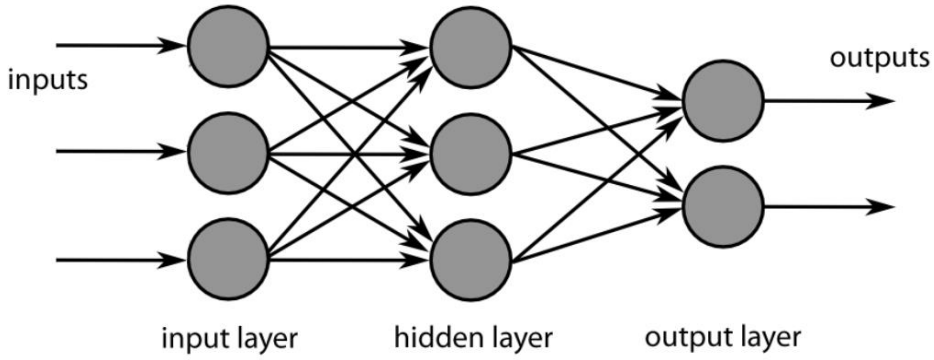

Figure 2.7: Shallow Neural Network

As you can see from the above figure, it is a shallow neural network which can be called Shallow Learning Network. A neural network will always have:

In Figure 2.7 , we will try to connect the dots in the network, this network is called Sallow Learning Network. NNs generally includes:

- **Input layer:** For example, it might be a time series data or image pixels.
- **Hidden layer**: It is common to look at it as weights that are initialized and eventually trained as training the NN.
- **Output layer:** This layer is responsible for providing the prediction of the input data fed into the network.

The NN is about an approximation function and seeks to learn the weights (parameters) in hidden layers. These weights when multiplied with the given input, they enable the network to predict an output close or similar to the target output.

The DL name comes from the idea of having many hidden layers stacked between the output and input layers.

2.4.1. DL Methods

Different methods are designed to implement DL. Each DL problem can be solved by a proposed method, based on several factors such as, the data type we have, which learning type is targeted, supervised or unsupervised, and the nature or domain of the problem itself.

Some of the deep learning methods are:

Here is a list of the DL methods:

- Convolutional Neural Network,
- Long short-term memory,
- Recurrent Neural Network,
- Generative Adversarial network,
- Denoising autoencoder,
- Sparse autoencoder,
- Auto Encoders,
- Variational autoencoder,
- Stacked autoencoder, etc.

Among the DL usages are:

- Feature extraction which is used to visualize data in lower-dimension spaces,
- Classification problems whether it is a binary or multi classes classification.
- Regression problems such as localization,
- Data preprocessing,
- Robotics,
- Object recognition, which is considered a regression and classification problem,
- Prediction of time series, such as weather prediction, stock prediction, etc.

2.4.2. Comparison between DL & ML

Functionality: ML usually receives input data, parses this data and aims to contextualize it (decisions) based on the learning gained during the training process. On the other hand, DL is a subset of ML that receives input data and makes intelligent and intuitive decisions via an ANN which has stacked layers.

Feature Extraction: ML is not well in extracting representative features from the input. To perform well, handcrafted features are utilized as an input. While DL is one of the best methods applied on raw data to extract meaningful features. It is able to extract hierarchical features, and not rely on handcrafted features such as gradients' histograms, local binary patterns. Layer-wise features learning can be achieved in DL, that means it learns features of low level and as moving through the layers, it increasingly learns abstract level features out of the data as shown in Figure 2.8

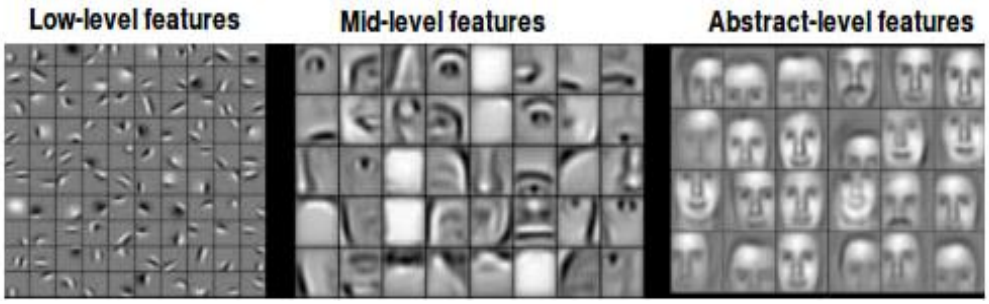

Figure 2.8: Layer-Wise Features Learned by a DL Model

Shape, position, textures, pixel values, orientation and color are examples of the features here. The level of the accuracy in features identification and extraction plays an essential role in the performance of traditional ML algorithms. When traditional ML feature extraction methods are applied, they do not cause problems significantly, since any tiny variation within the data causes changes in the extracted features by conventional methods of feature extraction such as histogram of orientated gradients, local binary pattern, etc. As for the DL networks, they learn the included features at various levels using stacked layers, and eventually merge them to compose larger pictures to create abstract representation.

Dependency of data: DL is Data Greedy, the more data we provide, the better it is expected to perform. Commonly, when we utilize more data, the number of layers (the network depth) increases causing more computation. Whereas ML algorithms can work well when we have small datasets.

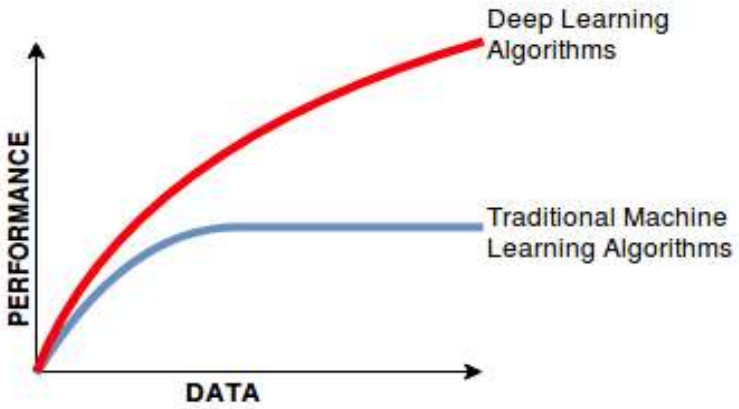

Figure 2.9: Performance (Y Axis) vs Data (X Axis) In DL and ML Algorithms

As shown above in Figure 2.9, we can tell that in traditional ML algorithms, even if we increase the data, the performance saturates and reaches a plateau after a while. While for the DL algorithms, the performance increases as the data increases.

Computation Power: As illustrated above, traditional ML algorithms can be trained and tested on CPUs which fairly offer decent specifications. On the other hand, DL networks depend on data, they require more power than what a CPU offers. In DL, we need to utilize graphical processing units (GPUs) which provide a tremendous number of cores, up to thousands, compared with CPUs that normally provide a small number of cores. The power of computation does not rely on the data amount only, but also on the deepness of our network. As we increase the number of layers or the data amount, more computation power is required.

Training and Inference Time: DL networks often need a training time that may range between a few hours and months. When we have a tremendous number of data, training networks normally requires time. In addition, as the number of layers gets increased in our network, the number of weights which are known as parameters will also increase, leading to slow training.

In DL networks, not only training takes a lot of time, but the inference process also takes time, since the test inputs are passed through a combination of layers in our network. A considerable time is usually consumed to implement a plenty of multiplication to produce the desired output.

As for traditional ML algorithms, they often require very less time ranges between a few minutes up to hours, however, the testing time in some ML algorithms may require a bit of time.

Problem-solving technique: When we use ML to find a solution for a problem, we are required to have the problem divided into various parts. Let us assume we aim to implement object recognition, for this we first walk over the whole image and see if an object is existing at each location, and where exactly it is located. Afterwards, among all the listed objects, we apply a ML algorithm let us say SVM with a feature extractor such as LBP to recognize requested objects. In contrast, in DL, we provide objects' rounding box coordinates and corresponding labels, then the network, on its own, can learn how to localize and classify objects.

Industry Ready: It is normally simple to decode how ML algorithms worked. In ML, we can understand and interpret what parameters are chosen and why in these algorithms. As for DL algorithms, they are about a black box, and not easy to be decoded. Although DL algorithms are able to achieve higher performance than humans, we do not look at them as reliable models when they get deployed in industries. ML algorithms such as decision trees, linear regression, random forest, etc. are very common in industries.

Output: The traditional ML algorithms generally result in a numerical output such as a classification or a score. Whilst DL algorithms produce various outputs such as text, speech, an element, a score, etc.

2.4.3. Related works to Deep Learning

The DL algorithms showed impressive results in classification problems. One of the major strengths of DL methods is that they are able to automatically learn features from inputs. The following paragraph explains 3 of the recent DL methods which achieved high accuracies.

In (Pandey & Janghel, 2019), they used a CNN to classify ECG heartbeats into five classes. It is found that the performance of the CNN model is improved after generating synthetic data using SMOTE technique. In their model, they did not use preprocessing steps such as feature extraction or selection. The CNN structure is composed of 11 layers, 4 CLs with filter size 27, 14, 3, and 1 are followed with 4 MaxPooling layers which diminish the feature map' size. Then, there are 3 fully connected layers (FCLs) consisting of 148, 50, and 5 neurons respectively. The CLs and the first two FCLs are activated using rectified linear unit (ReLU) function, whereas the output layer has the SOFTMAX activation function. They used different training and testing ratios such

as: 50-50, 60-40, 70-30, 80-20, 90-10. The highest accuracy achieved is 98.3 generated for the 70-30 training-testing ratio.

In (Zubair, J, & C, 2016), the authors proposed a model composed of two parts, feature extraction and CNN for the classification. 44 out of 48 records are selected from the MITBIH to use for training and evaluation. Each ECG beat is represented by 128 samples such that this data is fed into the input layer of the CNN. They used a 1D CNN consisting of three CLs, each one is followed with a pooling layer which has a subsampling factor of 2, then one MLP layer followed with a SOFTMAX layer. The kernel size of the first and second CL is set to 5. The overall performance of the proposed model shows a test accuracy of 92.7%.

In (Kachuee, Fazeli, & Sarrafzadeh, 2018), The authors suggested a method to transfer the knowledge acquired in classifying the five heartbeats of the MITBIH and PTB datasets, to a the myocardial infarction (MI) classification task. The CNN proposed model includes 1-D CL which has 32 kernels of size 5. The size and the stride of the used MaxPooling is 5 and 2 respectively. Afterwards, they added a predictor network which is composed of five residual blocks followed by 2 FCLs, each of them has 32 neurons. Finally, they used the output layer which has the activation of SOFTMAX to predict the output class probabilities. Their proposed model is able to predict overall accuracies of 93.4% and 95.9% on arrhythmias classification and MI classification respectively.

In (Acharya, et al., 2017), the authors developed a 9-layer deep CNN to automatically classify 5 different categories of the ECG heartbeats existing in the MITHIH dataset. The CNN consists of 3 CLs, 3 MaxPooling layers, and 3 FCLs. The CLs 1,3, and 5 have the kernel size of 3,4, and 4 respectively. The MaxPooling layers are used to decrease the size of the features map. The parameters for the kernel size are obtained using brute force technique, whereas the stride size for the convolution and pooling layers is set to 1 and 2 respectively. The first, second, and output FCLs have 30, 20, 5 neurons, respectively. The activation function for the layers 1,3,5,7,8 is Leaky ReLU, while the output layer has the SOFTMAX.

They conducted experiments on original and noise attenuated datasets. They augmented the dataset to even out the 5 imbalanced classes of heartbeats such that the total number of input signals including five ECG classes increased to 452,960. Then they denoised the input signals to remove high-frequency noise. In terms of the augmented data, their proposed CNN showed an accuracy of 94.03% and 93.47% over the original and noise free data. Whereas as for the highly imbalanced data, the accuracy decreased to 89.07% and 89.3% over the original and noise-free data.

2.5. Convolutional Neural Networks

CNNs are DL neural networks designed to process structured data arrays. CNNs have been used widely in computer vision and have become the state-of-the-art for image classifications. Moreover, they have also been used in natural language processing to classify texts.

CNNs show high efficiency in detecting patterns within input images, such as lines, circles, gradients, or even faces and eyes. This property makes CNNs very powerful in computer vision. Unlike earlier CV algorithms, CNNs are able to operate on raw images directly, without the need to perform preprocessing.

CNN is a feed forward net, and often includes up to 20 or 30 layers. The power of CNNs come from a special type of layer called convolutional layer.

CNNs consist of many convolutional layers stacked one on top of the other; each is capable of detecting complex patterns. For example, by using 25 CLs, we are able to recognize human faces, whereas with three or four CLs, we can recognize hand-written numbers.

The usage of CNNs can be looked at as a simulation to the human visual cortex, such that a series of layers process an input image and identifies complex patterns increasingly.

2.5.1. The Design of CNNs

CNNs are feed-forward multi layers, composed of staking up a set of hidden layers one on top of the other. This ordered structure enables CNNs to learn hierarchical features of inputs. Hidden layers are often convolutional layers followed by activation layers, and some of which are followed by pooling layers.

LeNet-5 which is proposed by Yann LeCun 1998, is able to recognize handwritten letters and considered as a simple CNN which helps us understand the substantial structure of early CNNs(Figure 2.10)

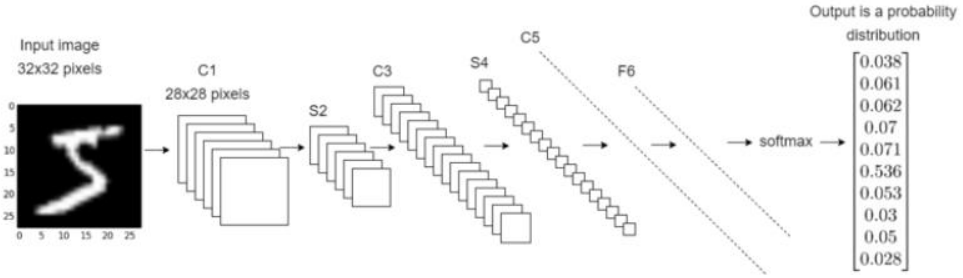

Figure 2.10: LeNet To Recognize Handwritten Letters

2.5.2. Explained Example about CNN Layers

LeNet receives a handwritten digit image with size of 32x32 as an input and feeds it through a series of subsequent layers. All layers, except the last one, are activated by tanh function:

- C1: The 1st CL. This is composed of 6 convolutional kernels (CKs), each one has the size of 5x5, which 'slide over' the input image. C1 produces 6 images, each has size of 28x28. The 1st layer of a CNN generally detects essential features such as corners and straight edges.

- S2: An average pooling layer, also known as a subsampling layer. In the C1 output, we average each 4 pixels to a single one. S2 shrinks the 6 images (28x28) by 2 and produces 6 output images, each as the size of 14x14.
- C3: The 2nd CL. This layer includes 16 CKs, each has the size of 5xc5, which slide over the 6 images each 14x14) again, generating 16 images, each of size 10x10.
- S4: The 2nd AvgPooling layer. S4 behaves the 16 images of size 10x10 to 16 images of size 5x5.
- C5: An FCL has 120 outputs. Each node of these output nodes connects to the entire 400 nodes (5x5x16) which belong to S4. At this level, the output is about 1D array with length of 120, and not an image anymore.
- F6: An FCL connecting the 120-outputs array to a following array with length of 10. At this point, each value of the array corresponds to one of the handwritten digits 0-9.
- Output Layer: This layer converts the F6 output to probabilities of 10 values which sum up to 1.

One of the simplest CNNs is LeNet-5 with 6 layers. This enables it to differentiate between handwritten digits 0-9, but not giving the power to recognize the alphabet of 26 letters or detect objects or faces. Recently, the most complex nets might have 30 layers or more and parameters count to millions, and also include branching, however, the simple building blocks of the CKs are still the same.

2.5.3. Convolutional Layer

The core building block within a CNN is the CL. We can look at the CL as a set of small square templates, named convolutional kernels (CKs), sliding over the image and finding patterns. When a part of the image matches the pattern of the kernel, the kernel provides a big positive number, whereas if no match, the kernel provides a smaller value or zero.

2.5.4. How to calculate a convolution on a matrix

Mathematically speaking, the kernel is nothing but a weighted matrix. For instance, in Figure 2.11, the kernel with size 3x3 can detect vertical lines.

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 2.11: Kernel With 3x3 Size

Let us imagine an input image 9x9 representing the summation sign. This includes two kinds of lines, vertical and horizontal, and cross lines. The image will be shown as the array in Figure 2.12.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.12: Input Image with Size 9x9

Imagine we need to try to detect the vertical line of the plus sign. To perform convolution, we slide the convolutional kernel over the image. In every position, we implement the dot product between every element of the convolutional kernel with the corresponding image element, afterwards, the results are summed up.

As the kernel has the width of 3, it can be placed only in 7 different positions horizontally in an image of the width 9. Therefore, the final result of the convolution applied on a CK 3x3 and an image 9x9 is a 7x7 new image (Figure 2.13)

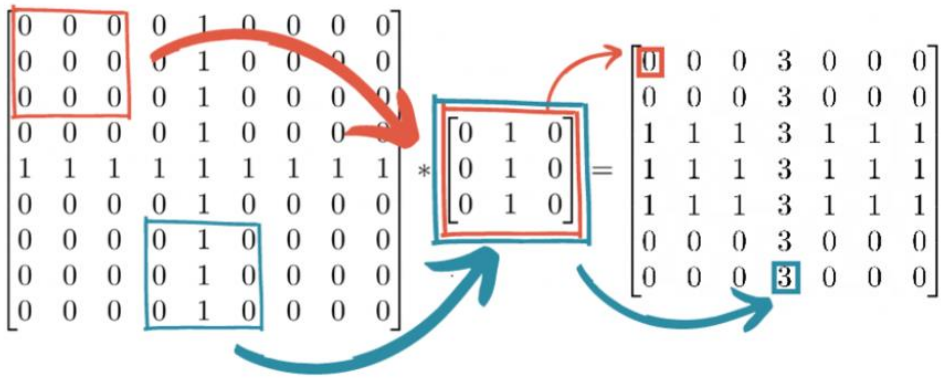

Figure 2.13: Convolution of CK With an Input Image

Hence within the example above, the kernel is placed at the upper left corner, and every element of the red box in the upper left location of the original image is multiplied with the corresponding element of the kernel. The result of that cell is zero in the upper left of the resulting matrix, since all the elemenst of the red box are zero.

Now let us look at the place of the blue box in the example above, it includes a segment of vertical line, thus, when the kernel is applied on this line, the result will be 3.

The CK is considered as a vertical line detector. As for the positions of the original image which includes a vertical line, it returns the value of 3, while it returns the value of 1 representing the horizontal line, and for the blank parts, it returns the value of 0.

Practically, the weights and biases are contained in the CK in an analogous way to the linear regression. Therefore, a given input pixel is multiplied with the weight and the bias then gets added.

2.5.5. Convolution example on an image

Let us assume the below 9x9 CK as in Figure 2.14 , which is a more complicated detector of vertical lines than the kernel utilized in the previous example:

$$\begin{bmatrix} 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.14: 9x9 CK

We also consider the below cat image with dimensions of 204x175 (Figure 2.15), which can be represented by a matrix with values ranging between 0 and 1, such that 0 indicates to black and 1 indicates to white.

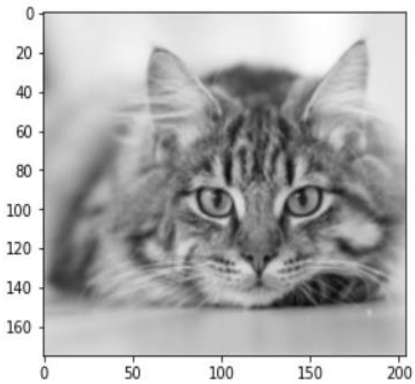

Figure 2.15: Cat Image of Size 204x175

By the implementation of the convolution, we notice that the filter has acted as a vertical line detector. The cat's vertical stripes are highlighted and detected in the output (Figure 2.16). The output image has the size of 196x167 which means that 8 pixels are ignored in both dimensions because of the 9x9 kernel.

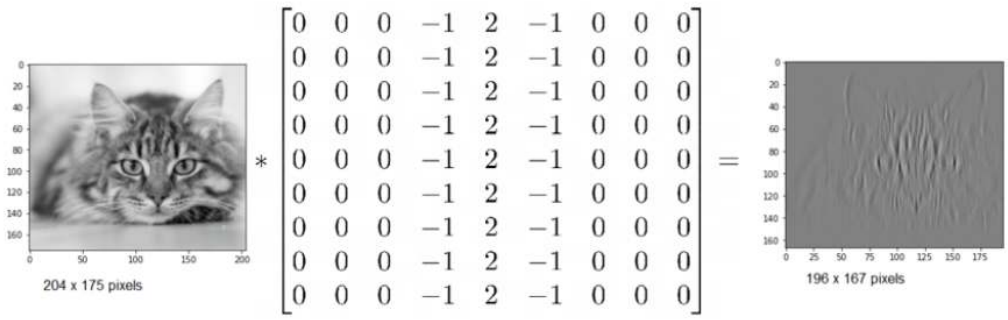

Figure 2.16: Convolution on A 2-D Image

Although applying the filter is simple, the most interesting powerful feature of the CK is the detection capability of curves, corners, vertical or horizontal lines, and other features. Recalling that the CL is composed of a series of CKs. Generally, the first layer of the CNN includes horizontal and vertical lines detectors, and diverse corner, diagonal and curve detectors. In fact, these feature detector kernels are learned by the NN during the training process, and not programmed and instructed by humans, and considered as the first phase of image recognition.

Following layers in the NN are able to depend on the features recognized by former layers and detect more sophisticated shapes.

2.5.6. Activation functions in CNNs

Post to having a given image passed through a CL, the output is passed through an AF. Among the common AFs is sigmoid function ((2.1)

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

And the rectified linear unit (ReLU) ((2.2) which is equivalent to taking the positive part of the input (Figure 2.17).

$$f(x) = \max(x, 0) \tag{2.2}$$

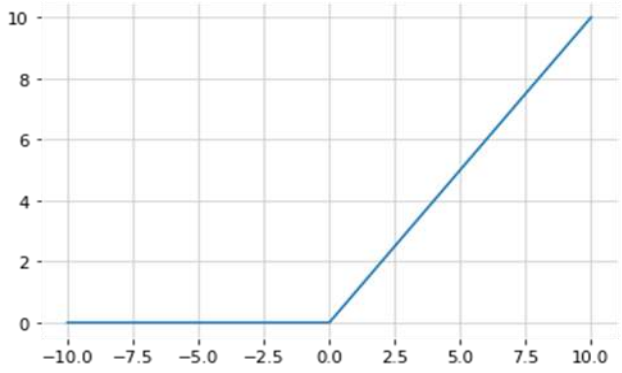

Figure 2.17: ReLU Activation Function

In fact, the AF plays a role in adding non-linearity to the CNN. All the layers of the NN could be expressed as a single matrix multiplication, if the AF was not applied. In the above cat example, when ReLU function is applied to the first layer, it results in strong contrast detecting the vertical lines, and eliminates the noise caused by other non-vertical features (Figure 2.18).

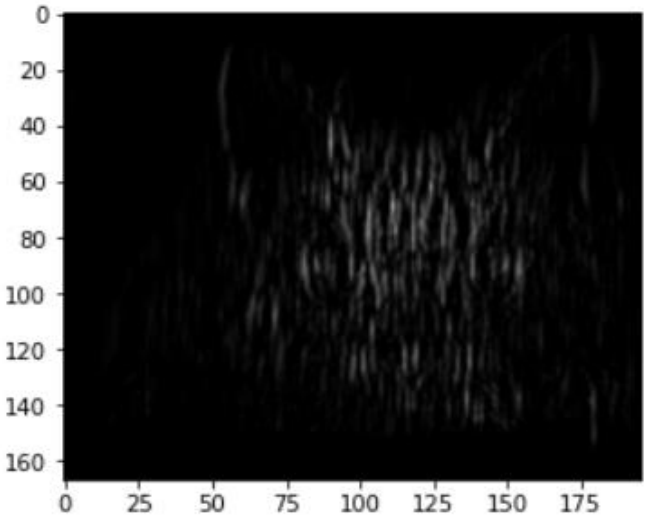

Figure 2.18: Effect of ReLU Function Through Convolution

2.5.7. Iterating Structure of a CNN

A typical CNN can be described as a series of CLs, followed by a AF, followed by a pooling layer (downscaling), iterated several times.

Through the iterated composition of these operations, simple features are detected by the 1st layer such as edges of a given image, and the 2nd layer can detect features of higher level. By the 10th layer, a CNN can detect fat sophisticated shapes such as noses. By the 20th layer, it can even be able to distinguish between human faces.

This ability is because of the iterative operations' layering, each of them is able to detect features of higher order than precedent layers.

2.5.8. CNNs vs Fully Connected Feedforward NNs

A CNN is a special type of a feedforward NN with less weights than a FC N.

In a FC feedforward NN, each node within the input is connected to each node in the 1st layer, and so on. There is no CK.

Therefore, in the above example of the 9x9 image as an input, the output of the 1st layer is a 7x7 image. If this were to be implemented using a FC feedforward NN, there would be:

$$\mathbf{9 \times 9 \times 7 \times 7 = 3969}\ \boldsymbol{connections}$$

However, in case we use a CL with a single 3x3 CK to implement this, there are:

$$3 \times 3 = 9\ connections$$

It is known that a CNN utilizes far less parameters than the alternative FC feedforward NN with the same dimensions of layers. This is due to the reusing of the network parameters as the CK slides over the image. This leads to giving the CNN the power of features detection within an image, regardless where the features are existing. "Translation invariance" terms are used to describe the CNNs' resilience.

Before inventing CNNs, "eigenfaces" was an early technique used in face recognition systems; it relied on comparing pixels of an input image directly. This technique was unsuccessful, due to not being translation invariant. For this reason, a FC network would not be suitable for image recognition because of the lack of translation invariance, in addition to, the need to train a network with a lot of weights.

Typically, the final layer of a CNN is a FL layer. For instance, the end layer of LeNet is able to translate an array with length 84 to another one of length 10, utilizing 840 connections.

2.5.9. Training a CNN in Image Classification

Fundamentally, the training process of a CNN is the same as the training process of another feedforward NN and utilizes the algorithm of backpropagation.

The network is built and initialized with randomized values for both biases and weights. Tuples of inputs and outputs which are images and classes are included within the training samples.

Let us consider whether we need to classify images as "do" or "cat" by training a CNN (Figure 2.19).

Every input image is fed into the whole network and the final layer which is the SoftMax layer provides a vector including probabilities' estimates. For instance, the output:

$$\begin{bmatrix} 0.71 \\ 0.29 \end{bmatrix}$$

can be described that we are 29% confident that the image is a dog and 71% confident that the image is a cat.

The cross-entropy loss function can be used, which refers to the network accuracy. The loss function is normally high when the network values are initialized randomly, and the goal of the training is to decrease the loss function as much as possible. As the loss function gets lower, the NN can show high accuracy as classifying the training set. The cross-entropy loss can be expressed using the formula (2.3):

$$L = -\sum_{c=1}^{M} y_o \log(p_o, c) \qquad 2.3)$$

The symbols of the cross-entropy loss are as follows:

- M – The classes count to be learnt by the classifier, in the cat-dog example, it is 2
- y_o,c – a binary value indicates whether the label c is correctly classified for the object o or not.

p_o, c – The predicted probability by the NN, and tells that the object o of the class c

By using the formula above, each image is passed through the network and the cross-entropy loss is calculated.

We aim to reduce the loss further through using calculus in calculating how the biases and weights of the NN need to be adjusted.

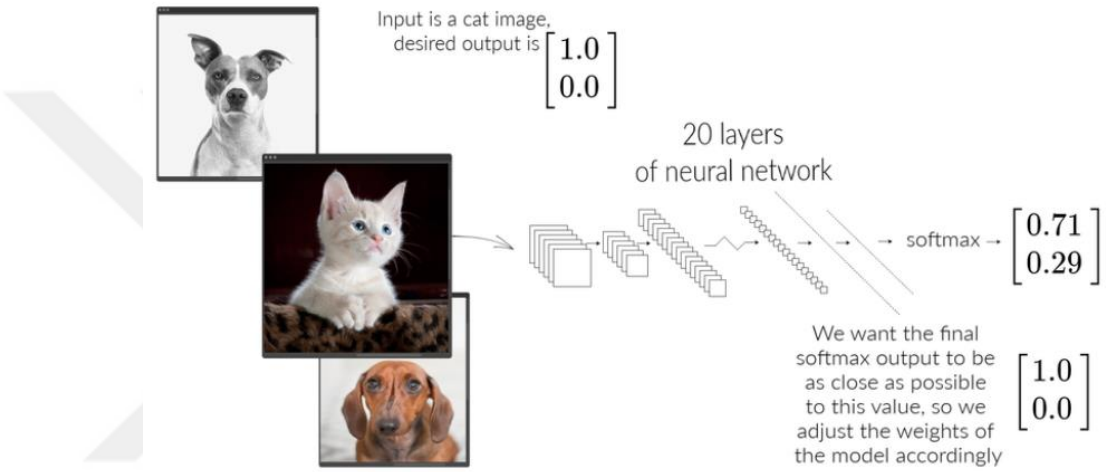

Figure 2.19: Cats and Dogs Classification Using CNN

Practically, we usually pass images through the network as batches, for example each batch can have 32, 64, 128 images. Afterwards, we change the network parameters after calculating the loss, and move to enter another batch for the same process.

We carry on this process until we make sure that the network shows the expected accuracy or there is no longer any improvement.

This process enables the 1st layer of a CNN to learn and detect vertical and horizontal edges without intervention by humans. As the training continues, the further layers learn how to detect beneficial features from the input images, based on the domain that the images are affiliated with. For instance, a CNN getting training on people images learns to collect relevant human features.

2.6. Sparse Representation

SR refers to a process in which we aim to generate a few non-zero elements (coefficients) representing a given input, depending on a predefined dictionary.

The SR is utilized to generate a hidden representation h(t) for a given signal x(t) This approach aims to generate appropriate sparse vectors which have many zero elements,

in an effort to having very-close signals to the original ones as being reconstructed by multiplying with a predefined dictionary A (FIGURE 2.20). The dictionary A is about a set of prototype signals (atoms).

The SR method is formally expressed with the objective function (2.4):

$$\min_{A} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} \left\| x^{(t)} - A\,\mathbf{h}^{(t)} \right\|_2^2 + \lambda \left\| \mathbf{h}^{(t)} \right\|_1 \qquad \textbf{2.4)}$$

Where $\boldsymbol{\lambda} \left\| \mathbf{h}^{(t)} \right\|_1$ is referred to as the sparsity penalty, where λ is sparsity coefficient.

and $\|A_i\|_2^2 \leq C \;\; determined\; constant;\; i = 1, \ldots, N$

We need to make a trade-off between the two terms within the equation, which are 1) the reconstruction error and 2) the sparse penalty; to have effective SR of signals. The first term represents the re construction error while the second one represents the sparsity penalty. Since SR problems are exponentially difficult (NP-hard), suboptimal solutions can be found using approximation algorithm s such as convex relaxation algorithms or greedy algorithms (Elad, n.d.).

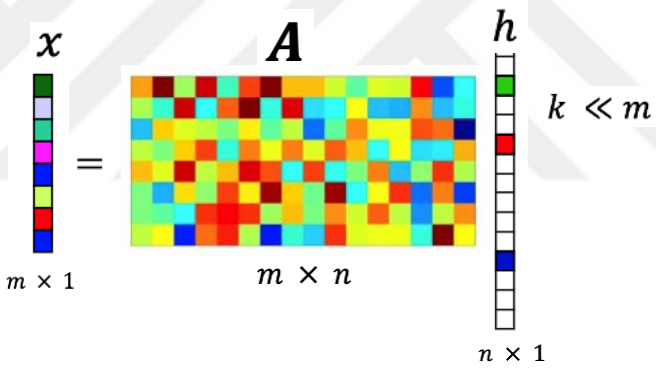

Figure 2.20: Sparse Representation

A simple representation of the sparse signal is shown in FIGURE 2.21, where the dots refer to the non-zero elements, while the majority of the sparse signal values are zeros.

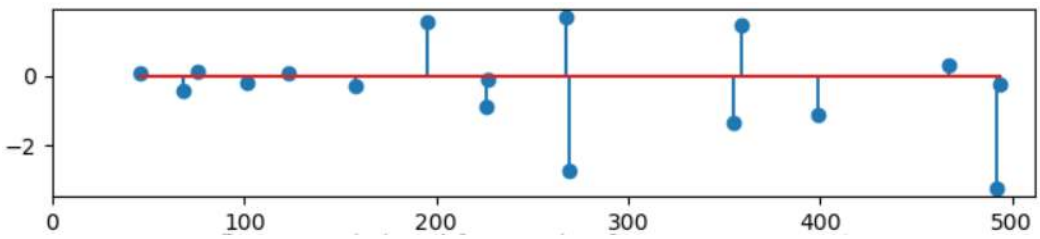

Figure 2.21: Sparse Signal

2.6.1. SR Algorithms

Since we knew that SR problems are NP-hard, therefore, we need to approximate in order to find proper solutions. We can use either relaxation methods such as Basis

Pursuit or greedy methods such as thresholding or OMP. The FIGURE 2.22 shows a set of greedy methods that span a wide range of accuracies and complexities.

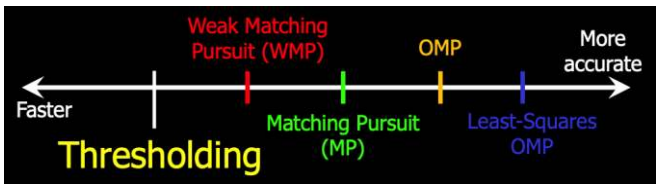

Figure 2.22: Greedy Algorithms

In the next paragraphs, we will demonstrate three of the SR algorithms briefly.

- **Basis Pursuit**

We take here the original problem of the SR which is too complicated and represented by the equation (2.5)

$$\min_{h} \|h\|_0 \quad s.t. \ \|Ah - x\|_2 \leq \varepsilon \qquad \textbf{2.5)}$$

We change here L_0 into L_1 and the problem become convex and manageable and expressed by the equation (2.6):

$$\min_{h} \|h\|_1 \quad s.t. \ \|Ah - x\|_2 \leq \varepsilon \qquad 2.6)$$

We do this process here because the authors in (Chen, Donoho, Johnstone, & Saunders, 1996) found that the equation often gives the solution for the original problem.

- **Matching Pursuit**

It is one of the greedy algorithms and proposed by (Mallat & Zhang, 1994). To find the support vector, which is the location of the non-zeros, one at a time, by seeping through all the atoms and seeing how much each of them bring the two sides of the equation (2.4) closer to each other. Then, the one with the closest effect is selected such that we stick with it and search again for the second one. This is generally the concept of this method.

- **Thresholding**

It is considered the dumbest algorithm that can be used in SR. We take the $\boldsymbol{x}$ then we take the inner product with all the atoms as in equation (**2.7**).

$$\hat{\mathbf{h}} = \boldsymbol{P_B}\{\boldsymbol{A^T x}\} \qquad 2.7)$$

Finally, we do thresholding, it means all the values which are too small are set to zero while the rest are left as they are.

2.7. Signal Filtering

In our work we used Discrete Wavelet Transform (DWT) which has been used in many engineering applications such as pattern recognition, computer vision, signal filtering and most commonly in image and signal compression.

We aim to eliminate the noise of high frequencies as preserving the wave form's character, involving the transitions of high frequencies at the peaks of the signal. Fourier based filters are considered not suitable to filter this kind of signal since it is non-stationary, and we need to keep the shape and locations of the peak.

There are 3 essential steps to implement signal filtering using wavelets, as the case with Fourier analysis:

- Using DWT to decompose the given signal.
- Using thresholding to filter the given signal within the wavelet space.
- Using inverse DWT (iDWT) to reconstruct the original signal utilizing the filtered signal.

In short, in DWT we basically decompose the signal into two parts: approximation and details. At a specific scale the details include the insignificant noise mostly and can be filtered and zeroed out utilizing thresholding without having an effect on the signal.
There are different wavelet families we can use such as Symlet, Daubechies, Harr, Coiflet, and Best Localized. In our work, we decomposed the ECG heartbeats at level 2 using the Daubeches2 wavelet. We set the threshold to a small value which is 0.01 to maintain the signals from getting distorted and keep high SNRs for all the training and testing signals.
We can use two kinds of thresholding in DWT, soft and hard thresholding. In the hard thresholding, we compare the details values at some scale with a fixed threshold, and the details are larger than the threshold, we keep the details value, otherwise we zero out the value. While in soft thresholding, we also compare the details value with the threshold, however here, if the details is larger than the threshold, we change details value to the amount expressed in the equation (2.8):

$$\boldsymbol{new\ D = \sin(D) * (|D| - Threshold)} \qquad \textbf{2.8)}$$

In our work, we tested our proposed model on the ECG dataset after having it filtered (denoised and reconstructed using DWT and iDWT respectively.) and sparsified.

FIGURE 2.23 shows an example of comparing the original with filtered ECG heartbeat.

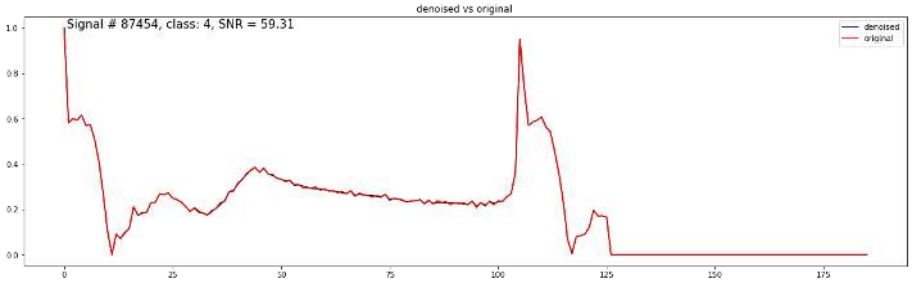

Figure 2.23: ECG Denoising Using DWT Daubechies 2

CHAPTER III: MATERIALS AND METHODS

A given ECG signal is normalized between 0 and 1 to handle the issue of amplitude scaling within MITBIH recordings. Afterwards, each heartbeat is resized to have a fixed length (187 samples) as in (Kachuee, Fazeli, & Sarrafzadeh, 2018). It is worth mentioning that the normalization is used to remove the effect of the amplitude scaling. In this study, we re-sample the signals to the sampling frequency of 125Hz as an input. To meet 1-D CNN requirements, we need to obtain a fixed beat length. In order to obtain the appropriate ECG segment length, we consider the standard normal heartbeat. A complete heartbeat in ECG starts from P wave and ends with T wave. Usually the P-R interval which is the time from the onset of the P wave to the start of the QRS complex ranges from 0.12 to 0.2s. The Q-T interval which is the time from the start of the QRS complex to the end of the T wave is up to about 0.42s, and the QRS duration is about 0.06-0.10s. According to the lowest possible heart rate of 40 BPM for normal people, the corresponding R-R interval is 1.5s (Cheng & Ye, 2020), (Luna., 2012). So, the total number of samples in 1.5 s is: 1.5*125=187 samples.

Using this length as segmenting makes sure that the QRS complexes are included within ECG beats. This would help identify normal people from arrhythmias disorders patients involved in PhysioNet db (ventricular ectopic, supraventricular ectopic, Fusion, or unknown). We are not covering all arrhythmias disorders in this study, and we might need to implement segmentation differently when the model is applied to other disorders. We believe that the normalization and segmentation are simple preprocessing and would not affect the in-question arrhythmias disorders. Other studies such as (Zhou & Li, 2016), (Raj & Ray, 2018) , and (Pandey & Janghel, 2019) opted to resize signals to 143, 256, and 360 samples respectively. Then, the data are divided into training and testing sets. Eventually, the sparsification and classification are handled. The steps of the proposed model are shown in Figure 3.1.

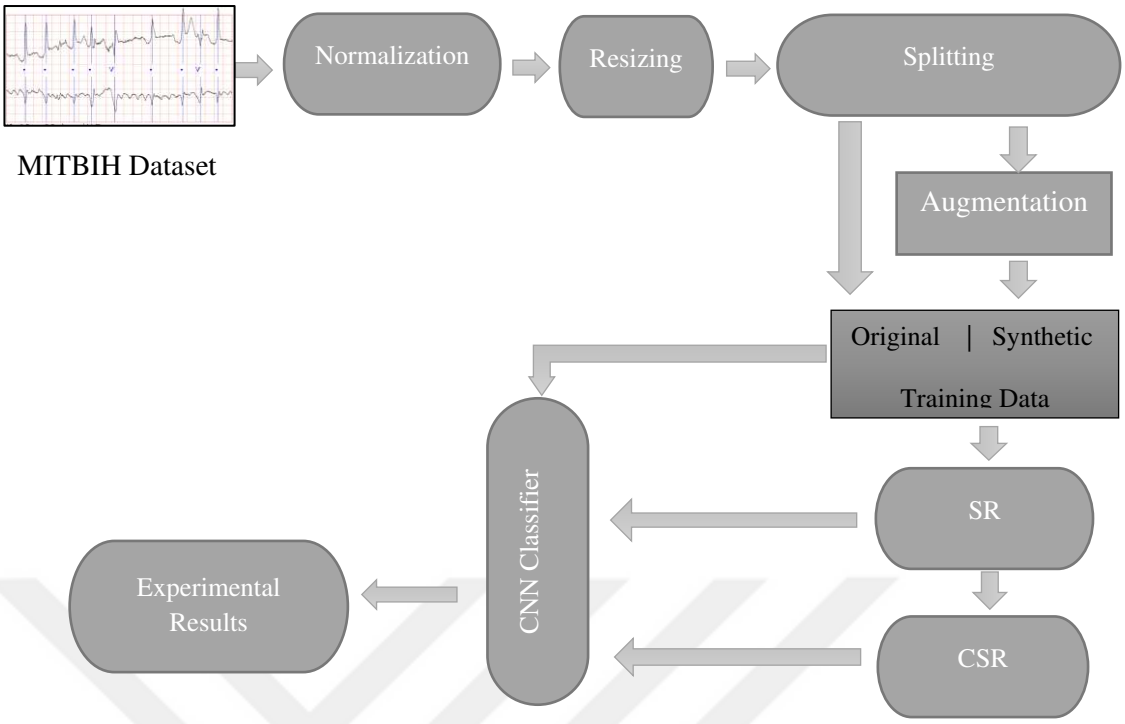

Figure 3.1: Block Diagram of The Proposed Model

3.1. Sparse Representation

SR aims to find sparse solutions for systems of linear equations. It is widely used in image processing, signal processing, ML, and medical imaging (Zhang, Xu, Yang, Li, & Zhang, 2015). In (Wright, Yang, Ganesh, Sastry, & Ma, 2009), SR was applied for face recognition. In (Yang, Wright, Huang, & Ma, 2010), an approach to single-image super-resolution was presented. It was based upon sparse signal representation. In terms of using SR with medical images, it can be used to implement efficient compression before real-time transmission of the volumes of data generated by ECG monitoring devices as in (Roy, et al., 2019). SR is also used in motor imagery EEG classification as reported in (She, Chen, Ma, Nguyen, & Zhang, 2018), where they proposed a novel approach called FDDL-ELM, which combines the discriminative power of extreme learning machine (ELM) with the reconstruction capability of SR. Other possible usage of SR is to extract significant features from ECG signals which can be further classified using classifiers as proposed in (Raj & Ray, 2018).

The SR is utilized to find a hidden representation for a given ECG signal, such that only very few elements are non-zeros whereas the majority of the elements are zeros. In this process, we need to respect that when we multiply the SR generated vector with a predefined dictionary, we will be able to have a signal close enough to the original input signal (Raj & Ray, 2018).

Therefore, we need to create a dictionary which can be adopted in the sparsification process. Moreover, we require to utilize a sufficient algorithm that can generate approximate proper SRs of input signals. The following two sections will demonstrate

how the required fixed dictionary is generated and the OMP algorithm which is utilized to generate SRs in our experiment.

3.1.1. Gabor Dictionary

The GD is an overcomplete dictionary whose width is commonly larger than its height; therefore, it is called an "overcomplete dictionary". And it is composed of a set of prototype signals called atoms. When the height of the dictionary is larger than its width, it is called under-complete dictionary which is not the case in our experiment.

The GD is inspired by (Wechsler, Liu, & H., 2002). It is a 2D frame which is modulated by an odd (sine function) and an even (cosine function) part. The equation (3.1) defines the Gabor kernels:

$$\boldsymbol{\psi_{\mu,v}(z)} = \frac{\left\|\boldsymbol{k_{\mu,v}}\right\|^2}{\boldsymbol{\sigma^2}} \boldsymbol{e^{-(\|k_{\mu,v}\|^2 \|z\|^2/2\sigma^2)}}\left[\boldsymbol{e^{ik_{\mu,v}Z}} - \boldsymbol{e^{-\sigma^2/2}}\right] \quad \textbf{3.1)}$$

Where $z = (x, y)$, v denotes the scale, while $\mu = 1$ and denotes the orientation of the Gabor frame.

σ is the STD of the Gaussian and equal to π , $\|.\|$ means the norm operator, and the wave vector is defined in the formula *(*3.2):

$$\boldsymbol{k_{\mu,v}} = \boldsymbol{k_v e^{i\phi_\mu}} \quad \textbf{(3.2)}$$

where $k_v = \frac{k_{max}}{f}$ and $\phi_\mu = \frac{\pi}{4}$

k_{max} is the maximum frequency, and f is the spacing factor between filters in the frequency domain. In this implementation, the GD is generated with single orientation and 8 scales. Figure 3.2 displays the generated predefined GD.

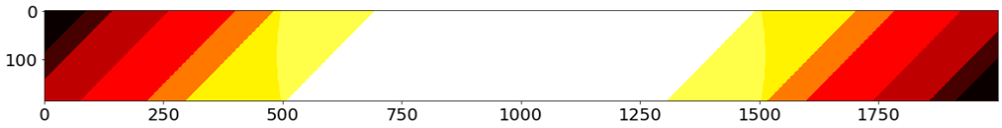

Figure 3.2: The Overcomplete Gabor Dictionary Used in SR

3.1.2. Orthogonal Matching Pursuit

In this work, the greedy algorithm OMP is implemented to produce appropriate sparse vectors through utilizing an over-complete GD.

The OMP is an enhanced version of the MP (Tawfic & Kayhan, 2017) (Tawfic & Kayhan, 2017) (Tawfic & Kayhan, 2015) which is a greedy iterative algorithm, such that it ensures the recent residual and the selected atoms are orthogonal at every iteration. Therefore, the OMP method is able to converge far faster than the MP one. The OMP is introduced and proposed by (Pati, Rezaiifar, & Krishnaprasad, 1993). This algorithm is shortly explained as follows.

During implementing the OMP, the number of the non-zero elements is set to a fixed value a priori, rather than generating a dynamic number of non-zero elements through comparing the residual with a predetermined threshold. Every input signal is represented by a sparse vector of the length 2000, such that only 40 elements out of it are non-zeros. The sparsity rate, in this approach, is utterly high and reaches up to 98%. **Table 3.1** shows the pseudocode algorithm of the OMP (Michael Elad).

Table 3.1: The Pseudocode algorithm of the used OMP

Our goal is Approximating the solution of

$$\min_x \|x\|_0 \; s.t. \; Ax = b$$

Initialization: $k = 0,\ x_0 = 0,\ r_0 = b - Ax_0 = b\,,\ Non - zeros = 40,\ and\ S_0 = \{\}$

For j=1 to Non-zeros:

1. Compute: $E(i) = \min_z \|z.a_i - r_{k-1}\|_2^2 \ for\ 1 \le i \le m$
2. Choose $i_0 \ s.t. \ \forall\, 1 \le i \le m,\ E(i_0) \le E(i)$
3. Update $S_k : S_k - S_{k-1} \cup \{i_0\}$
4. LS: $x_k = \min_x \|Ax - b\|_2^2 \ \ s.t. \sup\{x\} = S_k$
5. Update Residual: $r_k = b - Ax_k$

To ensure the implementation of SR is effective and provides appropriate alternative signals, we reconstruct the original signals though multiplying the sparsified signals with the fixed GD. The Peak signal-to-noise (PSNR) ratios are calculated for both training and testing signals. Figure 3.3 (a) shows that the 50% of the data have PSNRs range between 47 and 53 dB; while (b) shows the only 24 signals that are found with low PSNRs such that they range between 23 and <30 dB.

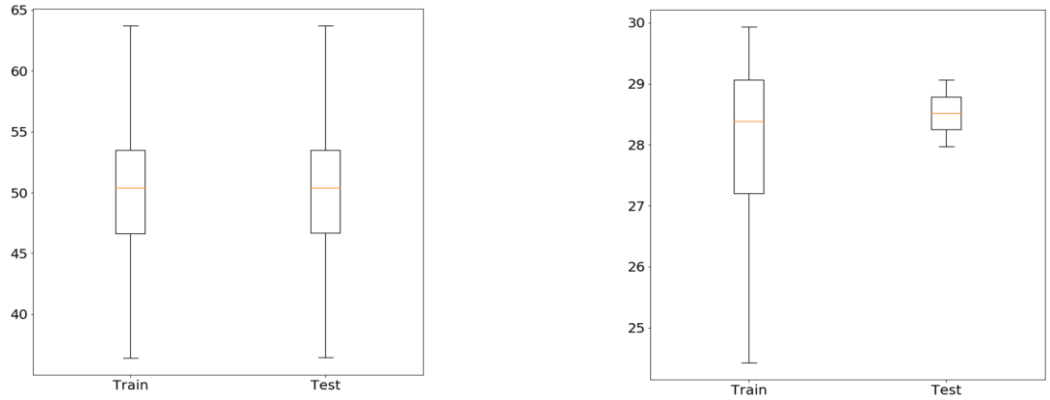

Figure 3.3: (a) PSNRs Of Train and Test Signals Sets; (b) PSNRs With <30 Values

An example of the bad signals with PSNRs less than 30 dB is shown in Figure 3.4. The following list shows the indices of the signals with less than 30 dB, [3423, 5995, 19244, 32584, 37795, 38276, 53166, 54637, 57382, 65793, 68443, 68558, 71557, 71854, 81538, 83332, 83573, 84120, 84197, 86031, 86102, 87532].

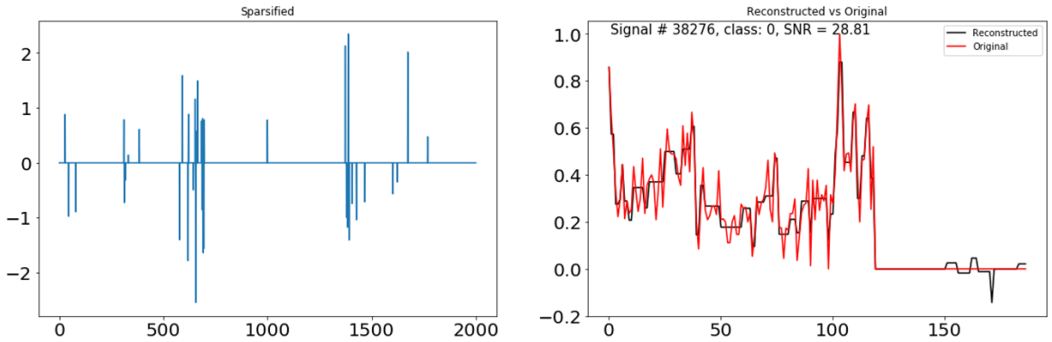

Figure 3.4: Sparsified Heartbeat with PSNR < 30

It is worth mentioning that the low-PSNRs sparsified signals could be optimized if the number of non-zero elements is increased more than 40. However, in terms of non-zeros in SR, we opted to select as lower non-zeros as possible, since as lower as better, with respect to achieving acceptable PSNRs.

In terms of denoised and sprasified input signals (DWT + SR), the Peak signal-to-noise (PSNR) ratios are also calculated for both training and testing signals. Figure 3.5 (a) shows that the 50% of the data have PSNRs range between 47 and 53 dB; while (b) shows the only 25 signals that are found with low PSNRs such that they range between 23 and <30 dB.

The following list shows the indices of the signals with less than 30 dB, [3423, 5995, 15266, 18070, 19244, 29844, 32382, 32584, 53166, 54637, 57382, 65793, 68443, 68558, 71557, 71854, 75393, 81538, 83573, 83599, 84120, 84197, 86031, 86102, 87532].

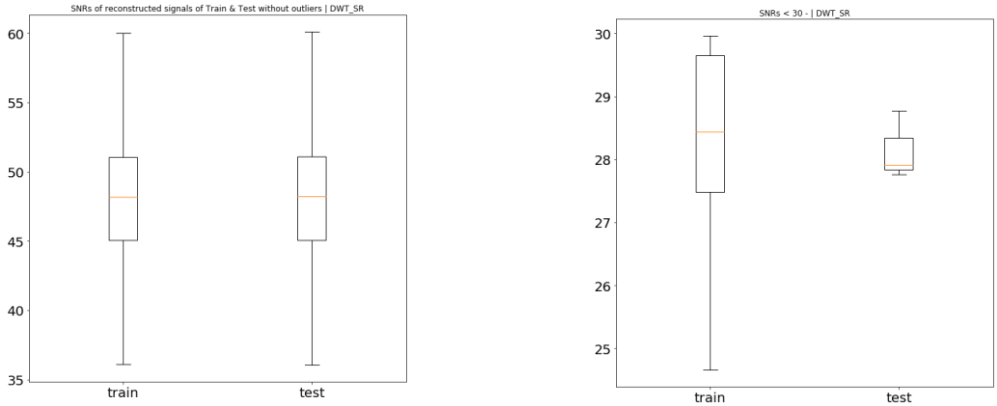

Figure 3.5:(a) PSNRs Of Train and Test Signals Sets; (b) PSNRs With <30 Values

Figure 3.6 shows examples of sparsified signals picked from the 5 ECG classes, along with their respective reconstructed and original signals.

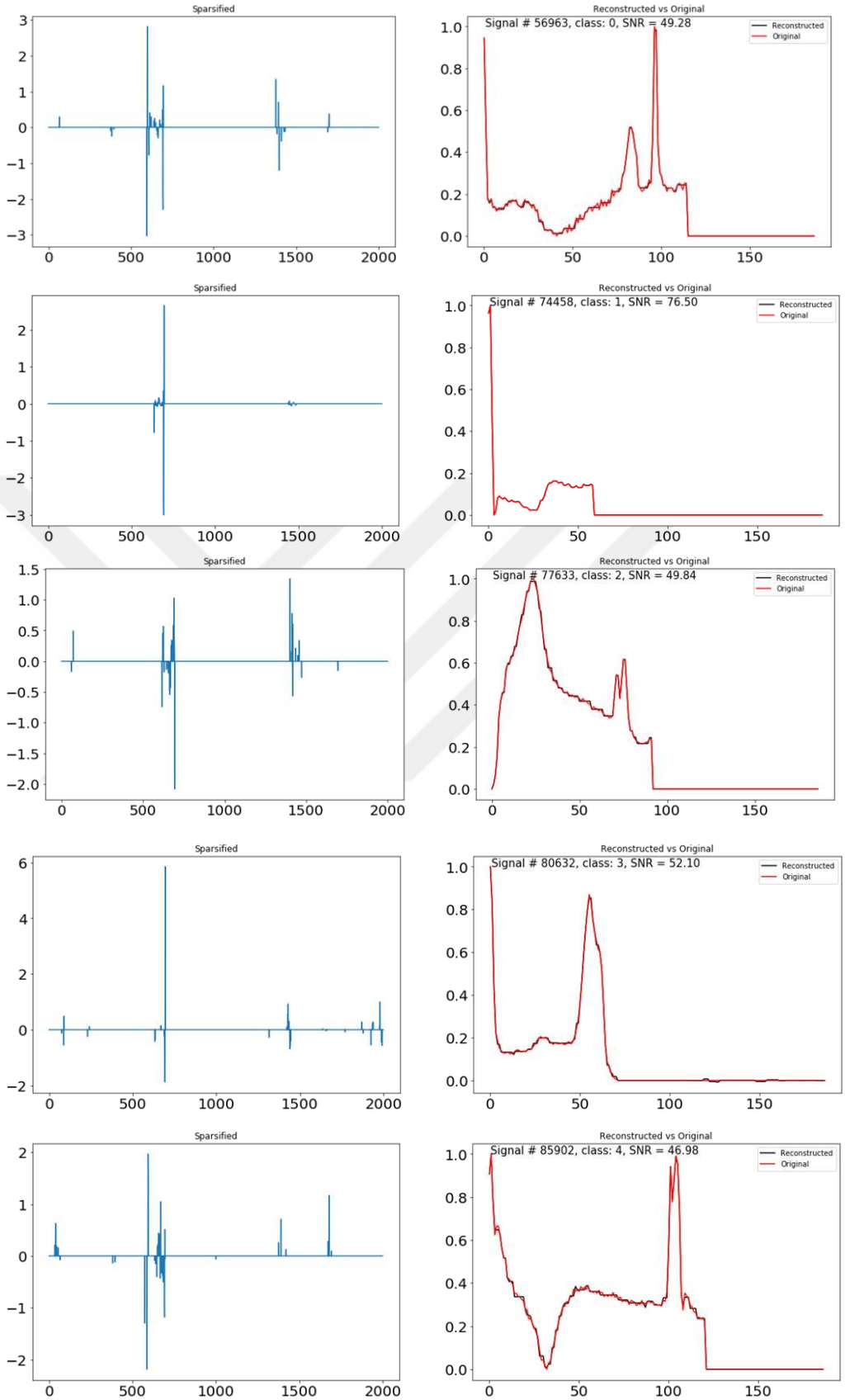

Figure 3.6: Sparsified Signals of 5 Classes with Their Original and Reconstructed Signals

Since we knew that the OMP is greedy algorithm, it is worth mentioning that generating the entire SR signals of both training and testing sets required roughly 6-7 hours on a local-CPU machine.

3.1.3. Compressed Sparse Representation (CSR)

Through the method of signal compression, we aim to reduce the amount of time required to train the model, while preserving the identification features of every input signal through utilizing its non-zeros and respective indices. Different approaches are tested in our experiment to implement CSR such as, using normalization of values and indices separately or together, concatenation or alternation of values and indices. For those approaches, the length of each signal of the CSR signals was 80. Eventually and based on experiment, we opted to go with implementing a dot product between the values (40) and values' indices (40) of the SR signals. Therefore, the length of an CSR signal has been halved to 40. The dot product helps present the weights of the values and its indices, through maintaining their magnitudes and signs, and keeping and the orders through multiplying correspondingly.

Figure 3.7 shows CSR signals for samples of the 5 ECG heartbeats.

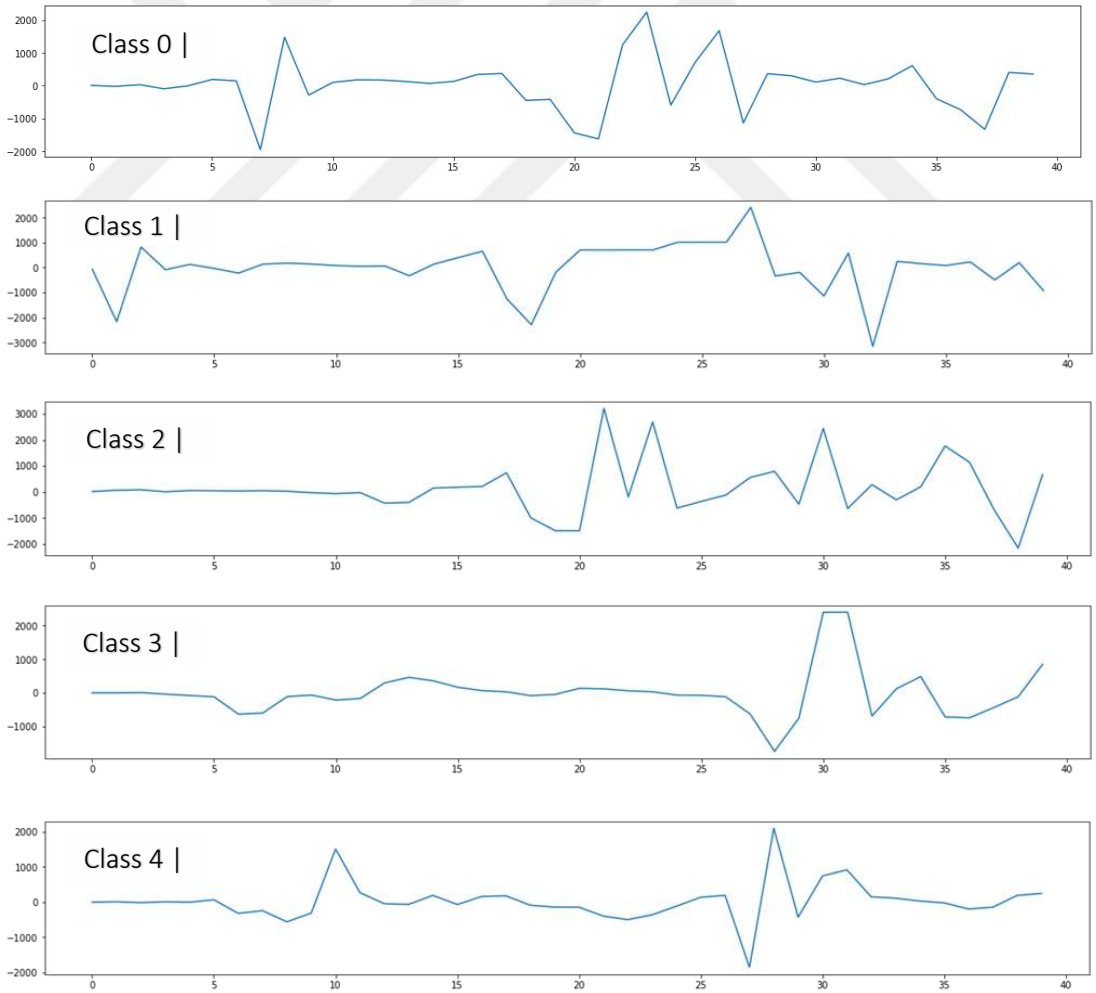

Figure 3.7: Examples of CSR Signals

3.2. Convolutional Neural Network

Our CNN classifier consists of five successive double 1-D CLs. The first layer includes 16 filters, while each one of the subsequent three layers consists of 32 filters with a kernel size of 7x1 and a stride 1. Each of these four layers is followed by 1-D AvgPooling which is responsible for shrinking the feature map and dropout (0.1) layers. The fifth CL consists of 256 filters with a kernel size of 7x1 and a stride 1; it is followed with a 1-D global AvgPooling layer and dropout (0.2).

The CLs are followed with 3 FCLs, each one of the first two layers consists of 64 neurons. The activation functions of all aforementioned layers are the ReLU to introduce non-linearity. The last FCL, which is the output layer, is composed of 5 neurons with SOFTMAX as activation function, which generates the probabilities of the five classes. AvgPooling is used instead of the MaxPooling, which is common in this problem, due to the nature of the sparse vectors whose non-zero elements can be positive and negative. Figure 3.1Figure 3.8 shows the proposed structure of the CNN.

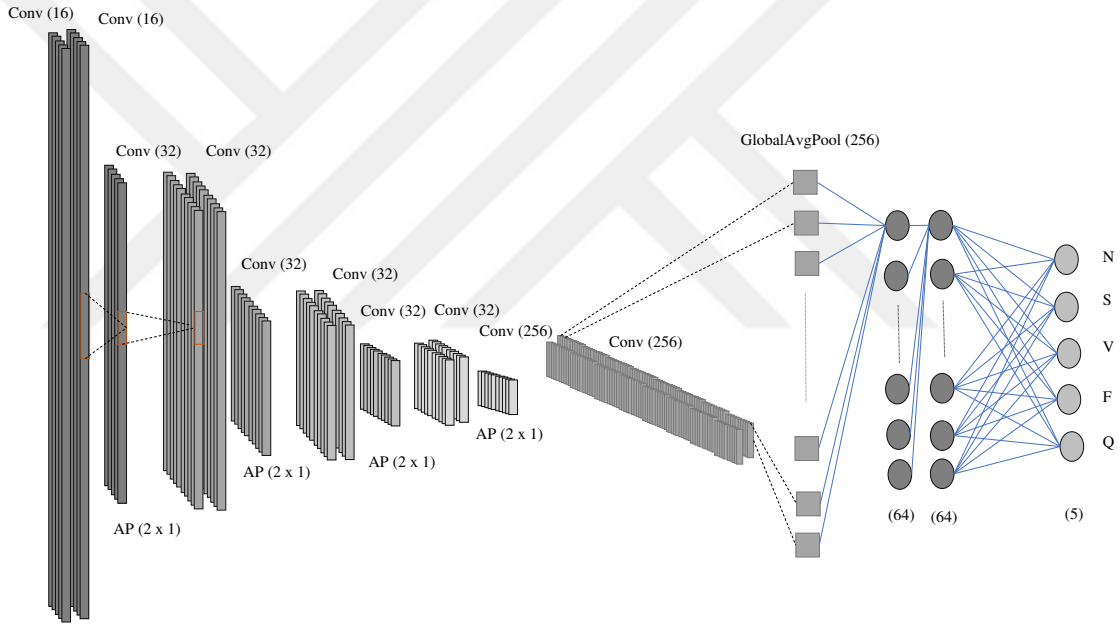

Figure 3.8: CNN with 1D Conv and 1D AvgPooling Layers Have Stride = 1 And Kernel of 7 and 2

The kernel size of convolutions is determined based on the nature of the SR signals. We calculated the average distance between all non-zero elements across all signals. We noticed that 50% of the distance values between consecutive non-zero elements ranges between nearly between 2 and 16, whereas the median of the distances is 5 as shown in Figure 3.9. However, after a plenty of tests and trials, we selected 7 to be the kernel size, which led to better actuaries.

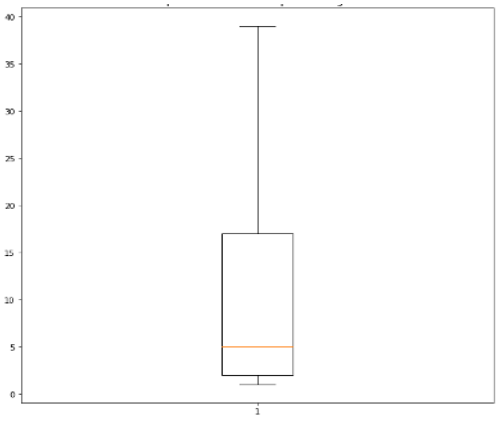

Figure 3.9: Distance Between Indices of Non-Zeros

CHAPTER IV: RESULTS AND DISCUSSIONS

4.1. Experimental Setup

4.1.1. Development Tools

Python is used as the main scripting tool, since it is very common and powerful in terms of developing and testing DL and ML algorithms. Local CPU is used to code and test all the snippets of the work parts such as coding and testing data preprocessing methods, OMP, DWT, and the generation of filtered and SR signals. Whereas a cloud GPU is used to train and test the CNN model. The set of packages and tools used in implementing this work are listed as follows:

- Python Packages
  - Python-keras 2.3.1
  - Tensorflow: 2.2.0-rc2
  - Sklearn: 0.22.2.post1
  - Scipy: 1.4.1
  - Numpy: 1.18.2
  - Matplotlib: 3.2.1
- IDEs:
  - Jupyter Notebook: for code organizing purposes
  - Google Colab Machine (GPU: Tesla P100-PCIE-16GB, Memory: 26G): for testing the CNN model.
  - Visual Studio Code: for debugging purposes

4.1.2. Evaluation Measures

The entire data points of the MITBIH database is used to evaluate the proposed model. The data is divided into two groups training set 80% and testing set 20%. The experiment considers the five existing ECG classes for classification. Two training experiments are conducted on balanced and imbalanced data. First, the model is trained using the original imbalanced-classes data. Second, the model is trained on synthetic data which are generated to compensate the imbalance in classes by up-sampling the data points of the minority classes (Acharya, et al., 2017). In all experiments presented in this section, three approaches of feeding the input to the model are applied. First the original signals are fed into the CNN without changing the shape of the data. Second, the signals are fed into the CNN after being sparsified. Third, the signals are fed into the CNN after being sparsified and compressed.

The classification report used to measure the performance of the model over the ECG classes is composed of four numerical metrics, which are sensitivity (recall), positive predictivity (precision), F1-score and accuracy. They are calculated by using true and false positives (TP and FP), true and false negatives (TN and FN).

To understand these terms simply, let us assume that we have two classes to classify: class 1 which refers to positive state and class 2 which refers to negative state. TABLE 4.1 shows the tabulation of the actual and predicted classes.

Table 4.1: Actual vs Predicted Classes

		Predicted class: Positive	Predicted class: Negative
Actual class	Positive	TP	FN
	Negative	FP	TN

TP indicates both actual and predicted classes are positive. FN indicates the actual class positive unlike the predicted which is negative. TN indicates both actual and predicted classes are negative. Finally, FP indicates actual class is negative whereas predicated class is positive.

The equations ((4.1, (4.2, 4.3, and 4.4) define the evaluation metrics:

$$P_P = \frac{TP}{TP + FP} \quad \textbf{(4.1)}$$

$$S_E = \frac{TP}{TP + FN} \quad \textbf{(4.2)}$$

$$F_S = 2 \times \frac{S_E \times P_P}{S_E + P_P} \quad \textbf{4.3)}$$

$$Acc = \frac{TP + TN}{TP + FP + FN + TN} \quad \textbf{4.4)}$$

We initialized the learning rate with the value (0.001). To optimize the training process, we used callbacks functionalities provided in keras, hence the epochs number did not exceed 50 for the experiments in this study. The first callback used is ModelCheckPoint to save the best model based on the validation accuracy metric. Second, ReduceOnPlateau callback is to reduce the learning rate by 10^{-1}, in case no better learning is achieved during 3 consecutive epochs. Third, EarlyStopping callback is to stop the training process in case the learning process is not increasing during 2 epochs following the last decrement of the learning rate.

The Confusion Matrix (CM) is used in the field of ML and particularly statistical classification problems, and also referred to as an error matrix. The CM is to show the predicted results crossed with the true class in classification problems. The predictions whether correct or incorrect are calculated and disaggregated by each class. The CM is able to tell us the error types made, such as how many classes misclassified.

The CM, as the classification report, uses the terms such as TP, FN, TN, and FP.

The CM helps calculate the evaluation metrics, such as accuracy, sensitivity, and precision which are described by the formulas in the experimental setup section.

The CM can be easily understood through plotting using heatmap technique in which data values are represented as colors.

4.2. Results

4.2.1. Non-sparse & CNN

Regarding the original non-sparsified datasets, and to be confident of the effectiveness of the model, we repeated testing the model over the training 5 times. Table 4.2 and Table 4.3 show the overall results and the results of each class individually for the augmented and imbalanced datasets respectively.

Table 4.2: Augmented Dataset - Results of Non-Sparsified Inputs Over Classes

		1	2	3	4	5
	Acc	98.79	98.73	98.63	98.75	98.54
Overall	SEN	94.51	94.54	93.84	93.40	94.07
	PPV	91.65	90.94	91.15	91.28	89.64
	F1	92.95	92.55	92.41	92.26	91.59
N	SEN	99.34	99.23	99.24	99.33	99.06
	PPV	99.43	99.44	99.44	99.39	99.37
	F1	99.38	99.34	99.34	99.36	99.21
S	SEN	87.95	88.31	88.49	86.87	86.33
	PPV	86.24	86.75	85.27	86.71	83.92
	F1	87.09	87.52	86.85	86.79	85.11
V	SEN	96.34	96.55	96.48	96.48	96.13
	PPV	97.89	96.68	97.08	96.81	96.80
	F1	97.11	96.61	96.78	96.64	96.47
F	SEN	89.51	89.51	85.80	85.19	89.51
	PPV	75.13	72.14	74.73	73.80	69.05
	F1	81.69	79.89	79.89	79.08	77.96
Q	SEN	99.44	99.13	99.19	99.13	99.31
	PPV	99.56	99.69	99.25	99.69	99.07
	F1	99.50	99.41	99.22	99.41	99.19

Table 4.3: Imbalanced Dataset - Results of Non-Sparsified Inputs Over Classes

		1	2	3	4	5
	Acc	98.88	98.87	98.90	98.77	98.80
Overall	SEN	92.71	90.96	91.08	91.69	91.19
	PPV	94.75	95.22	95.62	94.60	94.65
	F1	93.69	92.96	93.20	93.08	92.83
N	SEN	99.62	99.69	99.71	99.64	99.67
	PPV	99.29	99.18	99.22	99.15	99.18
	F1	99.46	99.43	99.46	99.40	99.42
S	SEN	84.35	83.09	82.73	81.65	81.83
	PPV	92.14	91.85	93.88	91.53	93.05
	F1	88.08	87.25	87.95	86.31	87.08
V	SEN	96.41	96.96	97.24	95.86	96.41
	PPV	96.61	97.57	96.70	97.20	96.81
	F1	96.51	97.26	96.97	96.52	96.61
F	SEN	83.95	75.93	76.54	82.10	79.01
	PPV	86.08	87.86	88.57	85.81	84.77
	F1	85.00	81.46	82.12	83.91	81.79
Q	SEN	99.19	99.13	99.19	99.19	99.01
	PPV	99.63	99.63	99.75	99.32	99.44
	F1	99.41	99.38	99.47	99.25	99.22

Figure 4.1 shows an example of the accuracy and loss curves for the original imbalanced non-SR training and testing sets. Where the best accuracy reached is 99.4541% and 98.8032% for training and testing respectively. Whereas the optimum loss achieved is 0.0153 and 0.0483 for the training and testing sets respectively. The learning process required 20 epochs; each required an average time of 22 seconds.

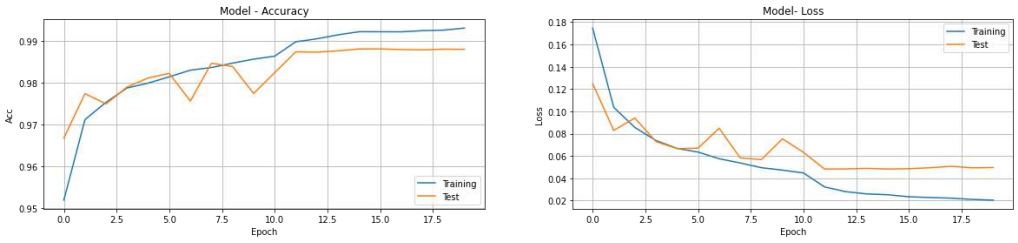

Figure 4.1: Accuracy and Loss of Non-SR Dataset

Figure 4.2 shows the CM calculated for the testing set after having the CNN trained. As shown, there is misclassification for the classes F and S compared with N, Q, and V classes. 16% of class S is classified as class N; and 11% and 10% of class F are classified as classes N and V respectively. The descending order of the classification sensitivities for the classes N, Q, V, S and F are 100%, 99%, 96%, 82% and 79% respectively.

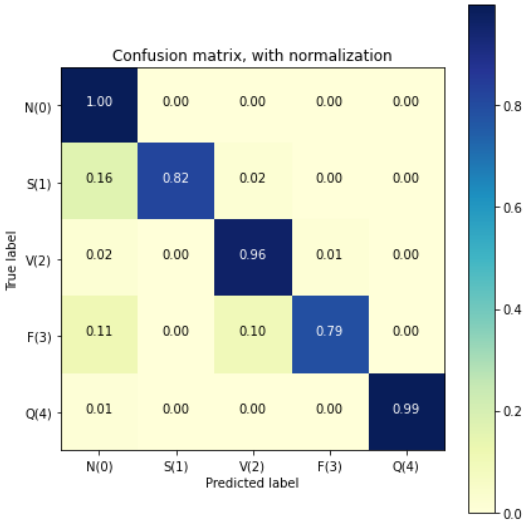

Figure 4.2: CM of Non-SR Dataset

4.2.2. SR & CNN

Regarding the sparsified datasets, and to be confident of the effectiveness of the model, we repeated testing the model over the training 5 times. Table 4.4 and Table 4.5 show the overall results and the results of each class individually for the augmented and imbalanced datasets respectively.

Table 4.4: Augmented Dataset - Results of Sparsified Inputs Over Classes

		1	2	3	4	5
	Acc	97.07	97.15	97.58	97.52	97.25
Overall	SEN	87.88	88.14	88.30	86.86	87.73
	PPV	86.97	86.54	88.67	88.83	85.63
	F1	87.41	87.32	88.47	87.79	86.60
N	SEN	98.47	98.24	98.66	98.91	98.37
	PPV	98.54	98.62	98.65	98.42	98.56
	F1	98.50	98.43	98.66	98.67	98.47
S	SEN	72.12	74.64	75.00	69.96	72.84
	PPV	73.71	73.58	78.68	79.07	73.91
	F1	72.91	74.11	76.80	74.24	73.37
V	SEN	92.68	92.89	93.30	92.54	92.33
	PPV	91.85	90.21	92.47	93.51	91.70
	F1	92.27	91.53	92.88	93.02	92.02
F	SEN	78.40	77.16	76.54	75.31	77.16
	PPV	72.99	73.10	76.07	74.85	66.49
	F1	75.60	75.08	76.31	75.08	71.43
Q	SEN	97.76	97.76	98.01	97.57	97.94
	PPV	97.76	97.21	97.46	98.30	97.52
	F1	97.76	97.48	97.73	97.93	97.73

Table 4.5: Imbalanced Dataset - Results of Sparsified Inputs Over Classes

		1	2	3	4	5
	Acc	97.51	97.31	97.27	97.45	97.67
Overall	SEN	83.52	81.94	80.89	82.98	76.59
	PPV	91.45	91.03	91.46	91.83	89.04
	F1	86.97	85.86	85.20	86.76	81.55
N	SEN	99.33	99.41	99.27	99.39	99.07
	PPV	98.03	97.87	97.74	97.97	96.97
	F1	98.67	98.63	98.50	98.67	98.01
S	SEN	63.67	62.59	58.27	61.33	54.32
	PPV	88.28	88.10	90.76	88.57	90.15
	F1	73.98	73.19	70.97	72.48	67.79
V	SEN	91.85	89.64	91.09	90.81	86.05
	PPV	93.60	94.13	90.22	93.53	88.37
	F1	92.72	91.83	90.65	92.15	87.19
F	SEN	66.05	61.11	60.49	66.05	51.23
	PPV	78.68	76.74	79.67	80.45	72.17
	F1	71.81	68.04	68.77	72.54	59.93
Q	SEN	96.70	96.95	95.32	97.32	92.27
	PPV	98.66	98.29	98.90	98.61	97.56
	F1	97.67	97.61	97.08	97.96	94.84

Figure 4.3 shows an example of the accuracy and loss curves for the original imbalanced SR training and testing sets. Where the best accuracy reached is 98.4900% and 97.4461% for training and testing respectively. Whereas the optimum loss achieved is 0.0540 and 0.0951 for the training and testing sets respectively. The learning process required 23 epochs; each required an average time of 38 seconds.

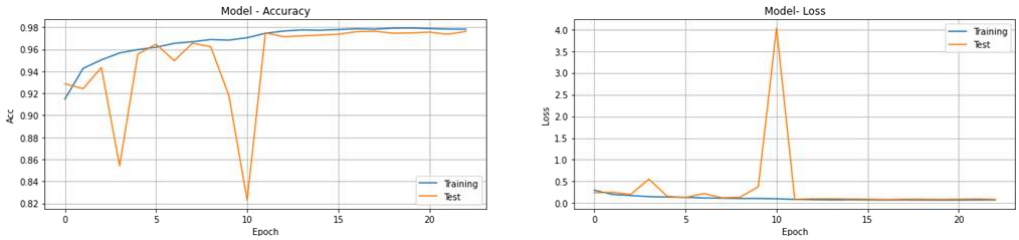

Figure 4.3: Accuracy and Loss of SR Dataset

Figure 4.4 shows the CM calculated for the testing set after having the CNN trained. As shown, there is misclassification for the classes F and S compared with N, Q, and V classes. 35% of class S is classified as class N; and 22% and 12% of class F are classified as classes N and V respectively. The descending order of the classification sensitivities for the classes N, Q, V, F and S are 99%, 97%, 91%, 66% and 61% respectively.

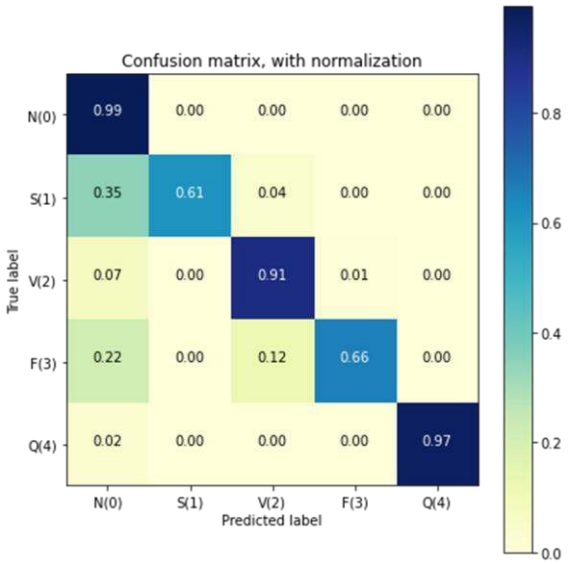

Figure 4.4: CM of SR Dataset

4.2.3. CSR & CNN

Regarding the compressed sparsified datasets, and to be confident of the effectiveness of the model, we repeated testing the model over the training 5 times. Table 4.6 and Table 4.7show the overall results and the results of each class individually for the augmented and imbalanced datasets respectively.

Table 4.6: Augmented Dataset - Results of Compressed Sparsified Inputs Over Classes

		1	2	3	4	5
	Acc	83.54	81.16	83.74	85.59	82.88
Overall	SEN	72.37	74.62	74.40	74.63	75.19
	PPV	53.63	51.75	53.71	55.56	53.57
	F1	58.67	58.06	59.21	62.14	59.89
N	SEN	84.68	80.93	84.69	86.36	82.96
	PPV	96.56	97.26	96.60	97.19	97.31
	F1	90.23	88.35	90.25	91.45	89.56
S	SEN	57.01	60.61	58.63	57.19	59.35
	PPV	19.92	17.96	20.07	28.34	19.87
	F1	29.53	27.71	29.91	37.90	29.77
V	SEN	78.25	83.84	76.11	83.43	83.84
	PPV	51.22	47.98	58.46	51.21	48.64
	F1	61.91	61.04	66.13	63.46	61.56
F	SEN	54.32	56.79	61.73	54.32	58.02
	PPV	17.50	22.44	17.18	26.43	24.61
	F1	26.47	32.17	26.88	35.56	34.56
Q	SEN	87.56	90.92	90.86	91.85	91.79
	PPV	82.97	73.10	76.21	74.63	77.44
	F1	85.20	81.04	82.89	82.35	84.01

Table 4.7: Balanced Dataset - Results of Compressed Sparsified Inputs Over Classes

		1	2	3	4	5
	Acc	93.84	94.05	94.01	93.99	93.94
Overall	SEN	66.97	67.52	66.59	67.74	67.26
	PPV	81.92	82.87	84.62	82.24	83.13
	F1	72.67	73.14	72.85	73.24	73.14
N	SEN	98.34	98.22	98.44	98.24	98.33
	PPV	94.98	95.24	95.03	95.18	95.07
	F1	96.63	96.71	96.71	96.69	96.68
S	SEN	41.73	42.45	39.93	42.45	42.45
	PPV	76.82	74.92	80.43	75.40	78.15
	F1	54.08	54.19	53.37	54.32	55.01
V	SEN	70.37	73.55	71.82	72.58	70.65
	PPV	82.58	82.75	83.67	83.02	82.17
	F1	75.99	77.88	77.29	77.45	75.97
F	SEN	36.42	33.95	33.95	36.42	35.80
	PPV	60.82	67.07	70.51	63.44	65.91
	F1	45.56	45.08	45.83	46.27	46.40
Q	SEN	88.00	89.43	88.81	88.99	89.05
	PPV	94.40	94.36	93.46	94.14	94.33
	F1	91.08	91.83	91.07	91.50	91.62

Figure 4.5 shows an example of the accuracy and loss curves for the original imbalanced CSR training and testing sets. Where the best accuracy reached is 94.9927 % and 93.2749 % for training and testing respectively. Whereas the optimum loss achieved is 0.1685 and 0.2291 for the training and testing sets respectively. The learning process required 23 epochs; each required an average time of 40 seconds.

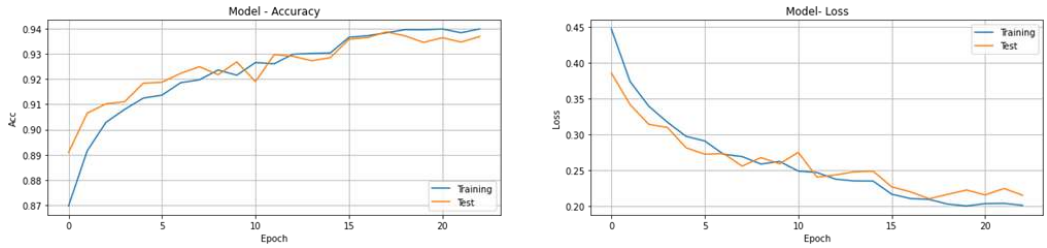

Figure 4.5: Accuracy and Loss of CSR Dataset

Figure 4.6 shows the CM calculated for the testing set after having the CNN trained. As shown, there is misclassification for the classes F and S compared with N, Q, and V classes. 55% of class S is classified as class N; and 59% of class F are classified as class N. The descending order of the classification sensitivities for the classes N, Q, V, S and F are 98%, 88%, 71%, 41% and 32% respectively.

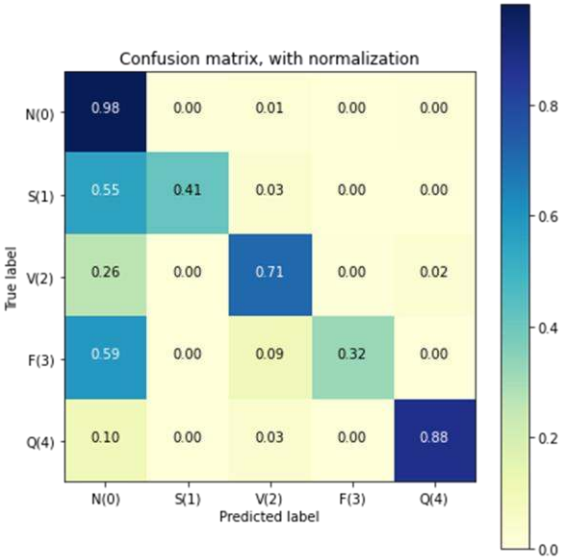

Figure 4.6: CM of CSR Dataset

4.2.4. Overall CNN Results

In terms of the imbalanced data, we reached average test accuracies of 98.84 %, 97.44%, and 93.96% over non-sparsified, sparsified, and compressed sparsified data respectively. While for the balanced augmented data, we achieved average test accuracies of 98.69%, 97.32%, and 83.38% over non-sparsified, sparsified, and compressed sparsified data respectively. The former results are illustrated in Table 4.8.

Table 4.8: Comparison of Imbalanced and Augmented Data Accuracies

Experiment	Imbalanced (%)	Augmented (%)
CNN	98.84 % ± 0.06 %	98.69 % ± 0.1 %
SR + CNN	97.44 % ± 0.16 %	97.32 % ± 0.23 %
CSR + CNN	93.96 % ± 0.08 %	83.38 % ± 1.6 %

In terms of the implementation speed, Figure 4.7 shows a comparison between the number of epochs and time in seconds required to train the network, for three kinds of the imbalanced inputs

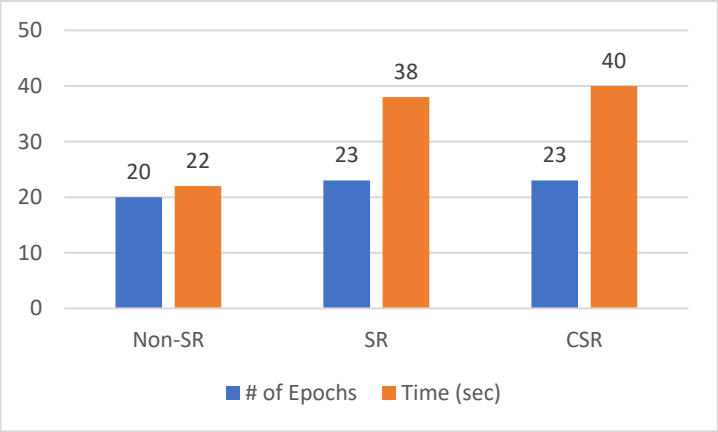

Figure 4.7: Implementation Speed Comparison

F1-score is a commonly better metric to look at rather than accuracy when there are imbalanced classes. Table 4.9 shows individually the results of the prediction, sensitivity, and F1-score for the ECGs classes over the imbalanced dataset. As illustrated, the class F and S show low performance in comparison with the other classes through all the 4 experiments.

Table 4.9: The Proposed Model's Results Over The 5 ECG Categories

Class	Metric	CNN	SR+CNN	DWT+SR+CNN	CSR+CNN
N	P_P	99.20	97.72	97.16	95.10
	S_E	99.67	99.29	99.08	98.31
	F_S	99.44	98.5	98.11	96.68
S	P_P	92.49	89.17	89.52	77.14
	S_E	82.73	60.04	54.21	41.8
	F_S	87.33	71.68	67.51	54.19
V	P_P	96.98	91.97	89.33	82.84
	S_E	96.57	89.89	86.55	71.8
	F_S	96.77	90.91	87.92	76.92
F	P_P	86.62	77.54	70.86	65.55
	S_E	79.51	60.99	51.60	35.31
	F_S	82.86	68.22	59.62	45.83
Q	P_P	99.55	98.41	96.87	94.14
	S_E	99.14	95.71	92.97	88.86
	F_S	99.35	97.03	94.88	91.42

We compared the study results with several recent ML and DL studies reported in the literature. Where there are a very few studies utilize SR over an overcomplete GD in heartbeat classifications domain. The first study reported using that is (Raj & Ray, 2018), they used three steps to extract ECG features, Daubechies wavelet 4 (db4), R peak detection, and SR using overcomplete GD. Afterwards, they fed the extracted features into different ML classifiers. The maximum accuracy they achieved is 89.93% using the classifier of Least Square Twin SVM. We achieved higher results than (Raj & Ray, 2018) in our model, the achieved accuracy is 97.44% when feeding the sparsified inputs to the CNN. Also, we used DWT to

denoise ECG signal, before feeding into CNN. However, the results of classification using the preprocessing is worse than our other methods; it decreased to 96.13% when the inputs were denoised, using db2, before being sparsified. That means our proposed model achieves higher performance without any preprocessing.

Our model also provides better results than the models proposed in (Zubair, J, & C, 2016; Kachuee, Fazeli, & Sarrafzadeh, 2018); they proposed CNN and residual CNN respectively, and achieved 92.7%, 93.4% respectively. The study (Acharya, et al., 2017) proposed a model which uses synthetic data of minority classes and CNN as a classifier, and they achieved 93.47%. Our model with the non-sparsified input achieves higher accuracy than models in (Sahoo, Sabut, Kanungo, & Behera, 2017; Pandey & Janghel, 2019), which showed high performance using ML and DL models. In (Sahoo, Sabut, Kanungo, & Behera, 2017), they achieved accuracy 98.39% by denoising inputs and using an SVM classifier. Whereas in (Pandey & Janghel, 2019), they achieved 98.3% with a deep CNN model, but with splitting the data to 70% for training and 30% for testing.

Table 4.10 shows our results compared with the recent studies of the heartbeat classification.

Table 4.10: Comparison of Heartbeat Classification Results (In %)

Work	Approach	A_{cc}	P_P	S_E	F_S
This study	- CNN	98.84	94.97	91.53	93.15
	- SR+CNN	97.44	90.96	81.18	85.27
	- DWT + SR + CNN	96.13	88.75	76.88	81.61
	- CSR + CNN	93.96	82.96	67.22	73.01
(Raj & Ray, 2018)	DWT + SR + LSTSVM + PSO	89.93	49.29	72.35	58.63
(Zubair, J, & C, 2016)	Deep CNN	92.7	-	-	-
(Kachuee, Fazeli, & Sarrafzadeh, 2018)	Deep Residual CNN	93.4	-	-	-
(Acharya, et al., 2017)	Augmentation + CNN	93.47	93.62	93.47	93.54
(Zhou & Li, 2016)	WPE + RR + RF	94.6	-	-	-
(Pandey & Janghel, 2019)	Deep CNN + SMOTE	98.3	95.25	84.77	84.77
(Sahoo, Sabut, Kanungo, & Behera, 2017)	DWT + SVM	98.39	-	96.86	-

4.2.5. ELM Results

We also had the chance to test the SR inputs against the method of Extreme Learning Machine (ELM) which is a feedforward NN with a single hidden layer or multi-hidden layers used in various domains such as regression, sparse approximation, clustering, classification, feature learning and compression. ELM aims to beat the slow gradient using learning algorithms. The smallest least square method of general linear systems and the Moore-Penrose inverse are utilized in ELM.

These hidden nodes are basically initialized randomly and never updated or may be copied from their ancestors with no changes. In most scenarios, we have the output weights of those hidden nodes learned in one step, without the need to implement iterations as in CNNs. The authors (Huang, Zhu, & Siew, 2004)of this algorithm gave it the name "Extreme Learning Machine".
According to the authors, ELM is able to show good performance in generalization and learns faster than backpropagation networks thousands of times. They claim also that it can beat SVM in both regression and classification applications.
FIGURE 4.8 shows the accuracy curves of the single hidden layer ELM for a number of neurons ranging between 1 and 100, over three different kinds of data, the non-sparsified, SR, and filtered & SR inputs.
And TABLE 4.11 shows the [1 to 100 nodes] performance results by showing accuracies, time, and the best number of nodes which produces higher accuracies.

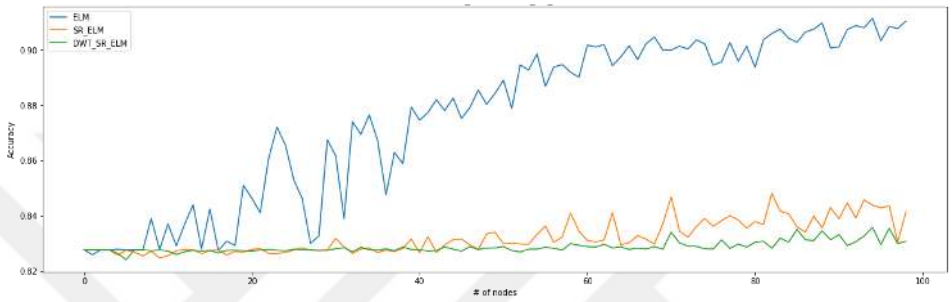

Figure 4.8: Single-Hidden Layer ELM Accuracy Curves for Nodes Between 1 And 100

Table 4.11: Single-Hidden Layer Elm Performance For Nodes Between 1 And 100

Experiment	Accuracy	Time (sec)	Best Nodes #
ELM	91.14	27	95
SR + ELM	84.80	72	83
DWT + SR + ELM	83.57	72	95

FIGURE 4.9 shows the accuracy curves of the single hidden layer ELM for a number of neurons ranging between 1 and 2000, over three different kinds of data, the non-sparsified, SR, and filtered & SR inputs.

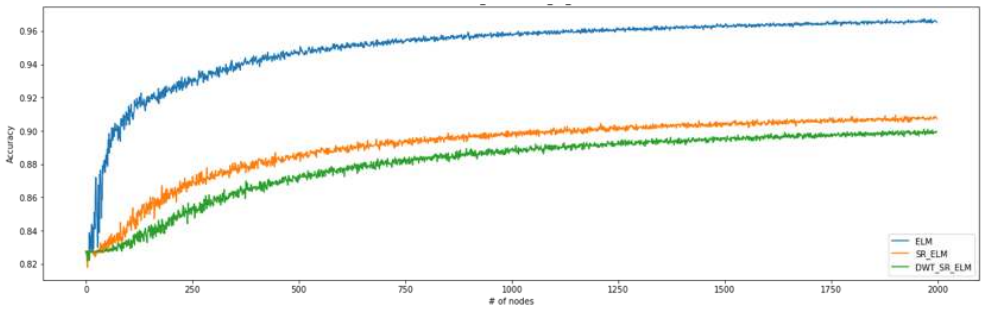

Figure 4.9: Single-Hidden Layer ELM Accuracy Curves for Nodes Between 1 And 2000

And TABLE 4.12 shows the [1 to 2000 nodes] performance results by showing accuracies, time, and the best number of nodes which produces higher accuracies.

Table 4.12: Single-Hidden Layer ELM Performance for Nodes Between 1 And 2000

Experiment	Accuracy	Time (h:m:s)	Best Nodes #
ELM	96.72	02:25:28	1970
SR + ELM	90.84	04:09:56	1902
DWT + SR + ELM	9.10	04:08:22	1969

CHAPTER V: CONCLUSIONS

5.1. Future Research Recommendations

We have concluded that ECG signals can be efficiently SR-ed using OGD as maintaining perfect PSNRs of retrieved signals. Then, SR-ed signals can be classified effectively using CNNs. Unlike (Acharya, et al., 2017), the augmentation of minor classes has shown a negative effect on accuracy generally.

The non-SR-ed data has produced superior accuracy over the SR-ed one, likewise, the SR-ed data has superseded the CSR-ed one. However, we believe CNN's structure and parameters can be better optimized to produce higher results using CSR-ed data.

Different approaches have been tested to implement CSR such as, normalization of values and indices together or apart, concatenation and alternation of values and indices. Further trails are recommended to achieve better CSR.

Although the proposed model has accomplished satisfactory results compared with the literature, the SR-ed and CSR-ed data have not superseded the non-SR-ed unlike what we expected. There are several settings to be further examined such as, kernel and stride sizes, dataset size and ECG classes' count. In fact, those options make the comparison among available ML/DL models extremely difficult.

REFERENCES

Acharya, U. U., Oh, S. L., Hagiwara, Y., Tan, J. H., Adam, M., Gertych, A., & Tan, a. R. (2017). A Deep Convolutional Neural Network Model to Classify Heartbeats. *Computers in Biology and Medicine, 89*, 389-396.

Chen, S., Donoho, D., Johnstone, I., & Saunders, M. (1996). Basis Pursuit.

Cheng, Y., & Ye, Y. (2020). Multi-label Arrhythmia Classification from Fixed-length Compressed ECG Segments in Real-time Wearable ECG Monitoring.

Elad, M. (n.d.). *Sparse Modeling in Image Processing and Deep Learning (Keynote Talk)*. (IEEE SigPort) Retrieved June 15, 2020, from http://sigport.org/2259

Huang, G.-B., Zhu, Q.-Y., & Siew, C. (2004). Extreme learning machine: A new learning scheme of feedforward neural networks. *IEEE International Conference on Neural Networks - Conference Proceedings*, 985 - 990.

Kachuee, M., Fazeli, S., & Sarrafzadeh, M. (2018). Ecg heartbeat classification: A deep transferable representation. *2018 IEEE International Conference on Healthcare Informatics (ICHI)*.

Luna., A. B. (2012). Characteristics of the Normal Electrocardiogram: Normal ECG Waves and Intervals. In Wiley-Blackwell, *Clinical Electrocardiography*.

Mallat, S., & Zhang, Z. (1994). Matching Pursuit with Time-Frequency Dictionaries. *Signal Processing, IEEE Transactions on*, 3397 - 3415.

Mark, G. B., & G., R. (May-June 2001). The impact of the MIT-BIH Arrhythmia Database,. *IEEE Engineering in Medicine and Biology Magazine, 20*(3), 45-50.

Michael Elad. (n.d.). *Sparse & Redundant Representations and Their Applications in Signal and Image Processing*. Retrieved June 22, 2020, from https://studio.edx.org/asset-v1:IsraelX+236862.2x+3T2018+type@asset+block@Section_0_Slides.pdf

Pandey, S. K., & Janghel, R. R. (2019). Automatic detection of arrhythmia from imbalanced ECG database using CNN model with SMOTE. *Australasian Physical & Engineering Sciences in Medicine, 42*(4), 1129-1139.

Pati, Y. C., Rezaiifar, R., & Krishnaprasad, P. S. (1993). orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA,*, 40-44.

PhysioNet. (2001). *Physionet MIT-BIH arrhythmia database*.

Raj, S., & Ray, K. C. (2018). Sparse representation of ECG signals for automated recognition of cardiac arrhythmias. *Expert Systems with Applications, 105*(0957-4174), 49-64.

Roy, R., Roy, A., Mukherjee, A., Ghosh, A., Bhattacharyya, S., & Naskar, M. (2019). Sparse Encoding Algorithm for Real-Time ECG Compression. 31-38.

Sahoo, S., Sabut, S., Kanungo, B., & Behera, S. (2017). Multiresolution wavelet transform based feature extraction and ECG classification to detect cardiac abnormalities. *Measurement*, 55-66.

She, Q., Chen, K., Ma, Y., Nguyen, T., & Zhang, Y. (2018). Sparse Representation-Based Extreme Learning Machine for Motor Imagery EEG Classification. *Computational Intelligence and Neuroscience, 2018*.

Tawfic, I. S., & Kayhan, S. K. (2015). Strong recovery conditions for least support orthogonal matching pursuit in noisy case. *Electronics Letters, 51(17)*, 1368-1370.

Tawfic, I. S., & Kayhan, S. K. (2017). An improved stopping condition guarantee recovery of sparse signal via Subspace Pursuit method. *ISA transactions*, 149-160.

Tawfic, I. S., & Kayhan, S. K. (2017). Improving recovery of ECG signal with deterministic guarantees using split signal for multiple supports of matching pursuit (SS-MSMP) algorithm. *Computer Methods and Programs in Biomedicine*, 39-50.

Wechsler, Liu, C., & H. (2002, April). Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition. *in IEEE Transactions on Image Processing, 11*(4), 467-476. Retrieved June 15, 2020

Wright, J., Yang, A. Y., Ganesh, A., Sastry, S. S., & Ma, Y. (2009). Robust Face Recognition via Sparse Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 31*(2), 210-227.

Yang, J., Wright, J., Huang, T. S., & Ma, Y. (2010). Image Super-Resolution Via Sparse Representation. *IEEE Transactions on Image Processing, 19*(11), 2861-2873. doi:doi: 10.1109/TIP.2010.2050625.

Zhang, Z., Xu, Y., Yang, J., Li, X., & Zhang, D. (2015). A Survey of Sparse Representation: Algorithms and Applications. *IEEE Access, 3*, 490-530. doi:10.1109/ACCESS.2015.2430359

Zhou, M., & Li, T. (2016). Ecg classification using wavelet packet entropy and random forests. *Entropy, 18*(8), 285.

Zubair, M., J, K., & C, Y. (2016). An automated ECG beat classification system using convolutional neural networks. *IT Convergence and Security (ICITCS), 2016 6th International Conference on. IEEE.*