**Automated Detection of Cardiac Arrhythmia using Recurrent Neural Network**

**1. INTRODUCTION:**

Cardiac arrhythmia is a condition where irregular heart rhythms occur. According to World Health Organization (WHO), about 17 million people in the world die every year due to cardiovascular diseases. This is about 31% of the total deaths globally. According to the statistics of American Heart Association (AHA), one out of every three deaths in US is related to cardiovascular diseases. The deaths due to cardiovascular diseases are more than due to all types of cancer and chronic lower respiratory diseases combined. A 2014 study indicates that approximately 2 to 3% of the people in North American and European countries are affected by atrial fibrillation. A heart rate which is high (above 100 beats per minute in adults) is called tachycardia and a heart rate that is slow (below 60 beats per minute) is called bradycardia. If the beat is too early, then it is called premature contraction. Irregular beat is called fibrillation or flutter. Other than the criteria of heart rate, there are a number of other classifications for cardiac arrhythmia depending upon different types of criteria. Another type of classification is in terms of the site of origin of the irregular heart rate. Atrial arrhythmias originate in the atrioventricular (AV) node. The AV node is positioned between the atria (each of the two upper cavities of the heart from which blood is passed to the ventricles is referred to as atria) and the ventricles. Atrial fibrillation (AF), atrial flutter, atrial tachycardia, premature atrial contractions and sinus bradycardia are some examples of atrial arrhythmias. Atrial fibrillation and atrial flutter are examples of arrhythmia which may lead to serious consequences. In AF, the atrium is contracted in a very fast and irregular manner with the heart's electrical signals originating from a different part of the atria or in the adjacent pulmonary veins instead of sino-atrial (SA) node. The walls of the atria fibrillate (quiver very fast) instead of beating in a normal way, making atria unable to pump blood properly into the ventricles. Stroke and heart failure are two complications to which atrial fibrillation can lead to. Conditions like high blood pressure, overactive thyroid gland, coronary and rheumatic heart diseases can lead to AF. Atrial flutter has similar symptoms and complications as AF. But in atrial flutter, the advancement of electrical signals of the heart through the atria happens in a fast and regular manner instead of the irregular manner in which it happens in AF. Ventricular arrhythmias are premature rhythms occurring in an ectopic ventricular focus. Ventricular fibrillation, ventricular tachycardia, premature ventricular contractions are some examples of ventricular arrhythmias. Some arrhythmias are symptomless and not at all life threatening. But some symptomless arrhythmias can even lead to serious complications like blood clotting, stroke, heart failure and sudden cardiac death. Arrhythmias occur when the electrical signals to the heart that co-ordinate heart beat are not working properly. The first step in the diagnosis of this abnormality is the analysis of electrocardiogram (ECG) and the confirmation that the ECG is not indicative of cardiac arrhythmia. ECG is a bio signal representing the activity of the autonomous nervous system (ANS) controlling heart rhythm. Thus, the electrical activity of the heart is recorded in ECG. It is a non-invasive and efficient tool to study cardiac rhythms and diagnose arrhythmias. The ECG signal is generated as a result of the following processes. The heartbeat is originated as an electric pulse from the SA node situated in the right atrium of the heart. After contracting both atria, this electric pulse, then activates atrioventricular (AV) node that connects electrically the atria and the ventricles. This is followed by the activation of both ventricles. The complete heart activity is represented in the ECG waveform. Abnormalities in the morphology of ECG waveforms are indicators of cardiac arrhythmias. ECG waveform is analysed to ascertain the risk associated with any type of arrhythmia.Extensive research has been done in the area of arrhythmia detection. The below are works in a serious type of arrhythmia called as myocardial infarction (MI) commonly known as heart attack. Data from a single lead ECG wasused for MI detection achieving an accuracy of 94.74%. Multiscale eigenspace analysis was carried out on 12 lead ECG data to achieve the same objective with an accuracy of 96%. Analysis of 12 nonlinear parameters extracted from 12 lead ECG data using discrete wavelet transform (DWT) were used to detect MI to achieve an accuracy of 98.8%. Deep learning techniques are now being increasingly employed in this area. The automated detection of normal and MI was conducted with CNN with an accuracy of 95.22%. An accuracy of 84.54% was achieved in the detection of inferior MI in ECG using CNN. Four types of arrhythmia were classified with an accuracy of 99.38% with MIT BIH data set along with another dataset as input. Classification of MIT Arrhythmia database of ECG into normal and abnormal was conducted using artificial neural network (ANN) achieving an accuracy of 96.77%. There are many works of classifying specific types of cardiac arrhythmia with ECG as normal input data. Often these specific cardiac arrhythmia cases addressed in most of the previous research work will be serious arrhythmia types like myocardial infarction. In short, researches were conducted into classifying normal ECG and many types of arrhythmia affected ECG. Cardiac arrhythmia, though identified by the irregularity in cardiac rhythm, is due to the anomalies happening in the heart. These anomalies cause anatomical differences in the structure of atria and ventricles, thus producing changes in its activation, depolarization and repolarisation. These changes are reflected as deviation of ECG waveform from its normal shape and size. Different types of cardiac arrhythmia are caused by unique factors, thus causing unique changes in the morphology of the ECG wave. The objective of this work is to develop an automated method for the diagnosis of cardiac arrhythmia. We perform a two class classification of the given ECG signal, whether cardiac arrhythmia is present or not. We use ECG recordings from the publically available MIT-BIH arrhythmia database in Physionet. The MIT-BIH arrhythmia database is the first generally available dataset which is widely used for ascertaining the efficiency of cardiac arrhythmia detection algorithms. We employ deep learning based analysis methods using CNN, CNN-RNN, CNN-LSTM, CNN-GRU. Our work can be an assisting automated tool to cardiologists for the initial screening of people having cardiac arrhythmia. The organisation of the paper is as follows: Section 2 presents brief descriptions of the deep learning techniques of CNN, CNN-RNN, CNN-LSTM and CNN-GRU used in this work. Section 3 tells about experiments conducted and results obtained. Section 4 includes discussions, conclusion and future work.

**1.1 Objective of the project:**

Cardiac arrhythmia is a condition where heart beat is irregular. The goal of this paper is to apply deep learning techniques in the diagnosis of cardiac arrhythmia using ECG signals with minimal possible data pre-processing. We employ convolutional neural network (CNN), recurrent structures such as recurrent neural network (RNN), long short-term memory (LSTM) and gated recurrent unit (GRU) and hybrid of CNN and recurrent structures to automatically detect the abnormality. Unlike the conventional analysis methods, deep learning algorithms don't have feature extraction based analysis methods. The optimal parameters for deep learning techniques are chosen by conducting various trails of experiments. All trails of experiments are run for 1000 epochs with learning rate in the range . We obtain five-fold cross validation accuracy of 0.834 in distinguishing normal and abnormal (cardiac arrhythmia) ECG with CNN-LSTM. Moreover, the accuracy obtained by other hybrid architectures of deep learning algorithms is comparable to the CNN-LSTM.

**2. LITERATURE SURVEY:**

**“A new pattern recognition method for detection and localization of myocardial infarction using T-wave integral and total integral as extracted features from one cycle of ECG signal.”**

In this paper we used two new features i.e. T-wave integral and total integral as extracted feature from one cycle of normal and patient ECG signals to detection and localization of myocardial infarction (MI) in left ventricle of heart. In our previous work we used some features of body surface potential map data for this aim. But we know the standard ECG is more popular, so we focused our detection and localization of MI on standard ECG. We use the T-wave integral because this feature is important impression of T-wave in MI. The second feature in this research is total integral of one ECG cycle, because we believe that the MI affects the morphology of the ECG signal which leads to total integral changes. We used some pattern recognition method such as Artificial Neural Network (ANN) to detect and localize the MI, because this method has very good accuracy for classification of normal signal and abnormal signal. We used one type of Radial Basis Function (RBF) that called Probabilistic Neural Network (PNN) because of its nonlinearity property, and used other classifier such as k-Nearest Neighbors (KNN), Multilayer Perceptron (MLP) and Naive Bayes Classification. We used PhysioNet database as our training and test data. We reached over 76% for accuracy in test data for localization and over 94% for detection of MI. Main advantages of our method are simplicity and its good accuracy. Also we can improve the accuracy of classification by adding more features in this method. A simple method based on using only two features which were extracted from standard ECG is presented and has good accuracy in MI localization.

**Multiscale energy and eigenspace approach to detection and localization of myocardial infarction:**

In this paper, a novel technique on a multiscale energy and eigenspace (MEES) approach is proposed for the detection and localization of myocardial infarction (MI) from multilead electrocardiogram (ECG). Wavelet decomposition of multilead ECG signals grossly segments the clinical components at different subbands. In MI, pathological characteristics such as hypercute T-wave, inversion of T-wave, changes in ST elevation, or pathological Q-wave are seen in ECG signals. This pathological information alters the covariance structures of multiscale multivariate matrices at different scales and the corresponding eigenvalues. The clinically relevant components can be captured by eigenvalues. In this study, multiscale wavelet energies and eigenvalues of multiscale covariance matrices are used as diagnostic features. Support vector machines (SVMs) with both linear and radial basis function (RBF) kernel and K-nearest neighbor are used as classifiers. Datasets, which include healthy control, and various types of MI, such as anterior, anteriolateral, anterioseptal, inferior, inferiolateral, and inferioposterio-lateral, from the PTB diagnostic ECG database are used for evaluation. The results show that the proposed technique can successfully detect the MI pathologies. The MEES approach also helps localize different types of MIs. For MI detection, the accuracy, the sensitivity, and the specificity values are 96%, 93%, and 99% respectively. The localization accuracy is 99.58%, using a multiclass SVM classifier with RBF kernel.

**Automated detection and localization of myocardial infarction using electrocardiogram: A comparative study of different leads:**

Identification and timely interpretation of changes occurring in the 12 electrocardiogram (ECG) leads is crucial to identify the types of myocardial infarction (MI). However, manual annotation of this complex nonlinear ECG signal is not only cumbersome and time consuming but also inaccurate. Hence, there is a need of computer aided techniques to be applied for the ECG signal analysis process. Going further, there is a need for incorporating this computerized software into the ECG equipment, so as to enable automated detection of MIs in clinics. Therefore, this paper proposes a novel method of automated detection and [localization](https://www.sciencedirect.com/topics/engineering/localisation) of MI by using ECG signal analysis. In our study, a total of 200 twelve lead ECG subjects (52 normal and 148 with MI) involving 611,405 beats (125,652 normal beats and 485,753 beats of MI ECG) are segmented from the 12 lead ECG signals. Firstly, ECG signal obtained from 12 ECG leads are subjected to [discrete wavelet transform](https://www.sciencedirect.com/topics/mathematics/discrete-wavelet-transform) (DWT) up to four levels of decomposition. Then, 12 [nonlinear features](https://www.sciencedirect.com/topics/engineering/nonlinear-feature) namely, approximate entropy (Eax), signal energy (Ωx), fuzzy entropy (Efx), Kolmogorov–Sinai entropy (Eksx), [permutation](https://www.sciencedirect.com/topics/mathematics/permutation) entropy (Epx), Renyi entropy (Erx), [Shannon entropy](https://www.sciencedirect.com/topics/engineering/shannon-entropy) (Eshx), Tsallis entropy (Etsx), wavelet entropy (Ewx), [fractal dimension](https://www.sciencedirect.com/topics/engineering/fractal-dimension) (FDx), [Kolmogorov complexity](https://www.sciencedirect.com/topics/computer-science/kolmogorov-complexity" \o "Learn more about Kolmogorov complexity from ScienceDirect's AI-generated Topic Pages) (Ckx), and [largest Lyapunov exponent](https://www.sciencedirect.com/topics/engineering/largest-lyapunov-exponent) (ELLEx) are extracted from these DWT coefficients. The extracted features are then ranked based on the t value. Then these features are fed into the k-nearest neighbor (KNN) classifier one by one to get the highest classification performance by using minimum number of features. Our proposed method has achieved the highest average accuracy of 98.80%, sensitivity of 99.45% and specificity of 96.27% in classifying normal and MI ECG (two classes), by using 47 features obtained from lead 11 (V5). We have also obtained the highest average accuracy of 98.74%, sensitivity of 99.55% and specificity of 99.16% in differentiating the 10 types of MI and normal ECG beats (11 class), by using 25 features obtained from lead 9 (V3). In addition, our study results achieved an accuracy of 99.97% in locating inferior posterior infarction by using only lead 9 (V3) ECG signal. Our proposed method can be used as an automated diagnostic tool for (i) the detection of different (10 types of) MI by using 12 lead ECG signal, and also (ii) to locate the MI by analyzing only one lead without the need to analyze other leads. Thus, our proposed algorithm and [computerized system](https://www.sciencedirect.com/topics/computer-science/computerized-system) software (incorporated into the ECG equipment) can aid the physicians and clinicians in accurate and faster location of MIs, and thereby providing adequate time available for the requisite treatment decision.

**Application of deep convolutional neural network for automated detection of myocardial infarction using ECG signals.:**

The electrocardiogram (ECG) is a useful diagnostic tool to diagnose various cardiovascular diseases (CVDs) such as myocardial infarction (MI). The ECG records the heart’s electrical activity and these signals are able to reflect the abnormal activity of the heart. However, it is challenging to visually interpret the ECG signals due to its small amplitude and duration. Therefore, we propose a novel approach to automatically detect the MI using ECG signals. In this study, we implemented a convolutional neural network (CNN) algorithm for the automated detection of a normal and MI ECG beats (with noise and without noise). We achieved an average accuracy of 93.53% and 95.22% using ECG beats with noise and without noise removal respectively. Further, no feature extraction or selection is performed in this work. Hence, our proposed algorithm can accurately detect the unknown ECG signals even with noise. So, this system can be introduced in clinical settings to aid the clinicians in the diagnosis of MI.

**Detection of inferior myocardial infarction using shallow convolutional neural networks:**

—Myocardial Infarction is one of the leading causes of death worldwide. This paper presents a Convolutional Neural Network (CNN) architecture which takes raw Electrocardiography (ECG) signal from lead II, III and AVF and differentiates between inferior myocardial infarction (IMI) and healthy signals. The performance of the model is evaluated on IMI and healthy signals obtained from Physikalisch-TechnischeBundesanstalt (PTB) database. A subject-oriented approach is taken to comprehend the generalization capability of the model and compared with the current state of the art. In a subject-oriented approach, the network is tested on one patient and trained on rest of the patients. Our model achieved a superior metrics scores (accuracy= 84.54%, sensitivity= 85.33% and specificity= 84.09%) when compared to the benchmark. We also analyzed the discriminating strength of the features extracted by the convolutional layers by means of geometric separability index and euclidean distance and compared it with the benchmark model.

**Neural network based arrhythmia classification using Heart Rate Variability signa:**

Heart Rate Variability (HRV) analysis is a non-invasive tool for assessing the autonomic nervous system and specifically it is a measurement of the interaction between sympathetic and parasympathetic activity in autonomic functioning. In recent years, HRV signal is mostly noted for automated arrhythmia detection and classification. In this paper, we have used a neural network classifier to automatic classification of cardiac arrhythmias into five classes. HRV signal is used as the basic signal and linear and nonlinear parameters extracted from it are used to train a neural network classifier. The proposed approach is tested using the MIT-BIH arrhythmia database and satisfactory results were obtained with an accuracy level of 99.38%.

**Clasification of arrhythmic ECG data using machine learning techniques.:**

In this paper we proposed a automated Artificial Neural Network (ANN) based classification system for cardiac arrhythmia using multi-channel ECG recordings. In this study, we are mainly interested in producing high confident arrhythmia classification results to be applicable in diagnostic decision support systems. Neural network model with back propagation algorithm is used to classify arrhythmia cases into normal and abnormal classes. Networks models are trained and tested for MIT-BIH arrhythmia. The different structures of ANN have been trained by mixture of arrhythmic and non arrhythmic data patient. The classification performance is evaluated using measures; sensitivity, specificity, classification accuracy, mean squared error (MSE), receiver operating characteristics (ROC) and area under curve (AUC). Our experimental results gives 96.77% accuracy on MIT-BIH database and 96.21% on database prepared by including NSR database also.

ECG signal generation and heart rate variability signal extraction: Signal Processing, Feature detection, and their correlation with cardiac diseases:

The sum total of millions of cardiac cell depolarization potentials can be represented by an electrocardiogram (ECG). Inspection of the P–QRS–T wave allows for the identification of the cardiac bioelectrical health and disorders of a subject. In order to extract the important features of the ECG signal, the detection of the P wave, QRS complex, and ST segment is essential. Therefore, abnormalities of these ECG parameters are associated with cardiac disorders. In this work, an introduction to the genesis of the ECG is given, followed by a depiction of some abnormal ECG patterns and rhythms (associated with P–QRS–T wave parameters), which have come to be empirically correlated with cardiac disorders (such as sinus bradycardia, premature ventricular contraction, bundle-branch block, atrial flutter, and atrial fibrillation). We employed algorithms for ECG pattern analysis, for the accurate detection of the P wave, QRS complex, and ST segment of the ECG signal. We then catagorited and tabulated these cardiac disorders in terms of heart rate, PR interval, QRS width, and P wave amplitude. Finally, we discussed the characteristics and different methods (and their measures) of analyting the heart rate variability (HRV) signal, derived from the ECG waveform. The HRV signals are characterised in terms of these measures, then fed into classifiers for grouping into categories (for normal subjects and for disorders such as cardiac disorders and diabetes) for carrying out diagnosis.

**“Real-time detection of atrial fibrillation from short time single lead ECG traces using recurrent neural networks**

Atrial fibrillation (AF) is the predominant type of cardiac arrhythmia affecting more than 45 Million individuals globally. It is one of the leading contributors of strokes and hence detecting them in real-time is of paramount importance for early intervention. Traditional methods require long ECG traces and tedious preprocessing for accurate diagnosis. In this paper, we explore and employ deep learning methods such as RNN, LSTM and GRU to detect the Atrial Fibrillation (AF) faster in the given electrocardiogram traces. For this study, we used one of the well-known publicly available MIT-BIH Physionetdataset. To the best of our knowledge this is the first time Deep learning has been employed to detect the Atrial Fibrillation in real-time. Based on our experiments RNN, LSTM and GRU offer the accuracy of 0.950, 1.000 and 1.000 respectively. Our methodology does not require any de-noising, other filtering and preprocessing methods. Results are encouraging enough to begin clinical trials for the real-time detection of AF that will be highly beneficial in the scenarios of ambulatory, intensive care units and for real-time detection of AF for life saving implantable defibrillators.

**Instantaneous heart rate as a robust feature for sleep apnea severity detection using deep learning:**

Automated sleep apnea detection and severity identification has largely focused on multivariate sensor data in the past two decades. Clinically too, sleep apnea is identified using a combination of markers including blood oxygen saturation, respiration rate etc. More recently, scientists have begun to investigate the use of instantaneous heart rates for detection and severity measurement of sleep apnea. However, the best-known techniques that use heart rate and its derivatives have been able to achieve less than 85% accuracy in classifying minute-to-minute apnea data. In our research reported in this paper, we apply a deep learning technique called LSTM-RNN (long short-term memory recurrent neural network) for identification of sleep apnea and its severity based only on instantaneous heart rates. We have tested this model on multiple sleep apnea datasets and obtained perfect accuracy. Furthermore, we have also tested its robustness on an arrhythmia dataset (that is highly probable in mimicking sleep apnea heart rate variability) and found that the model is highly accurate in distinguishing between the two.

**3. SYSTEM ANALYSIS**

**3.1 Existing System**

This database has a major issue when applied to deep learning networks in the original format. It is due to the fact that these 48 sequences largely consist of two types of data (either related to normal or abnormal heartbeats). Due to this, beat-to-beat dependence is very much possible in the data sequences. Another issue is the difference in baseline voltages of different sequences. In order to tackle both these issues, we extracted individual heartbeats from continuous sequences of database. These extracted heartbeats are used to train our deep learning networks. LSTM introduced memory blocks instead of conventional simple RNN units to handle the problem of vanishing and exploding gradient. LSTMs can handle long term dependencies much better than the traditional RNNs whereby LSTMs can remember and connect previous information that really lags back so much in time compared to the present.

**Disadvantages of Existing System:**

* Less Accuracy

**3.2 Proposed System**

The proposed deep learning architecture for the classification of ECG recordings into either normal or arrhythmia is presented in Deep learning algorithms don't need explicit feature extraction and analysis like traditional machine learning based classifiers. It just passes raw input data to more than one hidden layer to obtain the optimal feature representation by itself. The newly formed feature representations are further passed as input to the fully connected layer (dense layer) which uses sigmoid activation function to produce output binary values 0 or 1 indicative of arrhythmia or normal ECG. It is an extension of feed forward network having feedback loops.This results in a cyclic graph. These loops are the short-term memory used to store and retrieve past information over time scales. Temporal tasks can be executed very effectively with this improvement. Unlike multilayer perceptrons (MLPs), RNNs can handle temporal sequences of arbitrary length. RNN can also share its parameters across time-steps to avoid not being generalized on dealing with the unseen sequence of arbitrary length. In short, RNNs are models that can effectively learn dynamic temporal behaviours for input-output sequences of any arbitrary length. RNN is used extensively, especially for long standing AI tasks in the field of machine translation, language modelling and speech recognition. The objective of this work is to develop an automated method for the diagnosis of cardiac arrhythmia. We perform a two class classification of the given ECG signal, whether cardiac arrhythmia is present or not. We use ECG recordings from the publically available MIT-BIH arrhythmia database in Physionet. The MIT-BIH arrhythmia database is the first generally available dataset which is widely used for ascertaining the efficiency of cardiac arrhythmia detection algorithms.

**Advantages of Proposed System:**

* High Accuracy

**Modules Information:**

1. Upload Arrhythmia Dataset: using this module we will upload dataset to application
2. Preprocess Dataset: using this module we will read all dataset values and then replace missing values with MEAN and then normalize training values and then selected important features from dataset by applying PCA algorithm. Dataset contains more than 270 columns and all this columns are not required so by using PCA we selected relevant features from dataset. After features selection we have splitted dataset into train and test where application using 80% dataset for training and 20% for testing
3. Run LSTM Algorithm: we will input 80% training data to LSTM to trained a model and then model will be applied on 20% test data to perform prediction and then calculate accuracy
4. Run CNN Algorithm: we will input 80% training data to CNN to trained a model and then model will be applied on 20% test data to perform prediction and then calculate accuracy
5. LSTM & CNN Training Graph: using this module we will plot CNN and LSTM training graph
6. Performance Table: using this module we will display both algorithms performance in tabular format.

**FUNCTIONAL REQUIREMENTS:**

**SOFTWARE REQIREMENTS:**

**System Attributes:**

1. X\_train, X\_test, y\_train, y\_test, pca
2. model, dataset
3. filename
4. X, Y

**Data base Requirements:**

No need

**USECASE:**

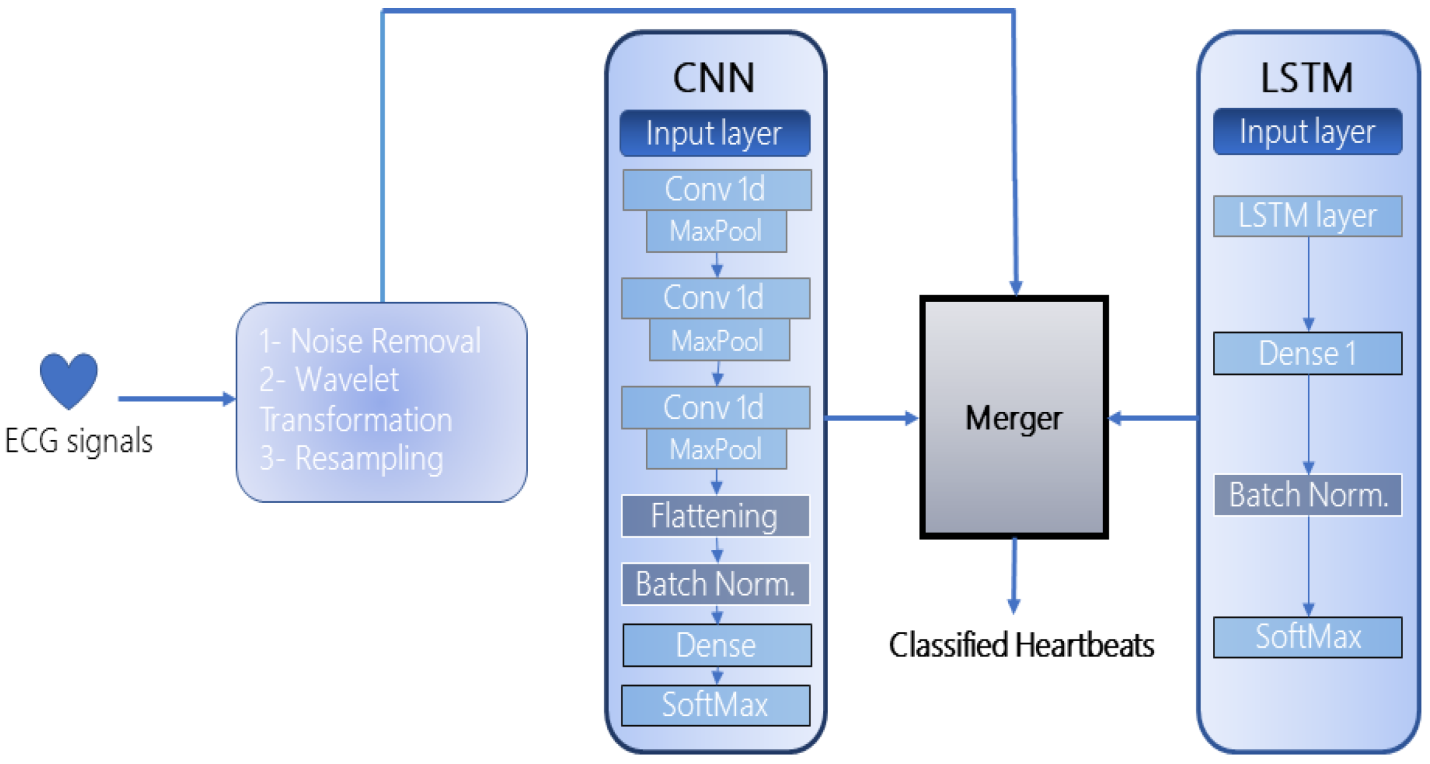
Use cases - Use cases describe the interaction between the system and external users that leads to achieving particular goals.

To implement this project we have designed following modules

* Upload Arrhythmia Dataset
* Preprocess Dataset
* Run LSTM Algorithm
* Run CNN Algorithm
* LSTM & CNN Training Graph
* Performance Table

**User Stories:** In this project you asked to design CNN and LSTM algorithm to predict Arrhythmia diseases with 7 different stages. To train both algorithm we have used MIT-BH dataset with 7 different disease stages.

**Work down Structure:**



**Prototype:**

python 3.7.0 or 3.7.4

opencv-python==4.5.1.48

keras==2.3.1

tensorflow==1.14.0

protobuf==3.16.0

h5py==2.10.0

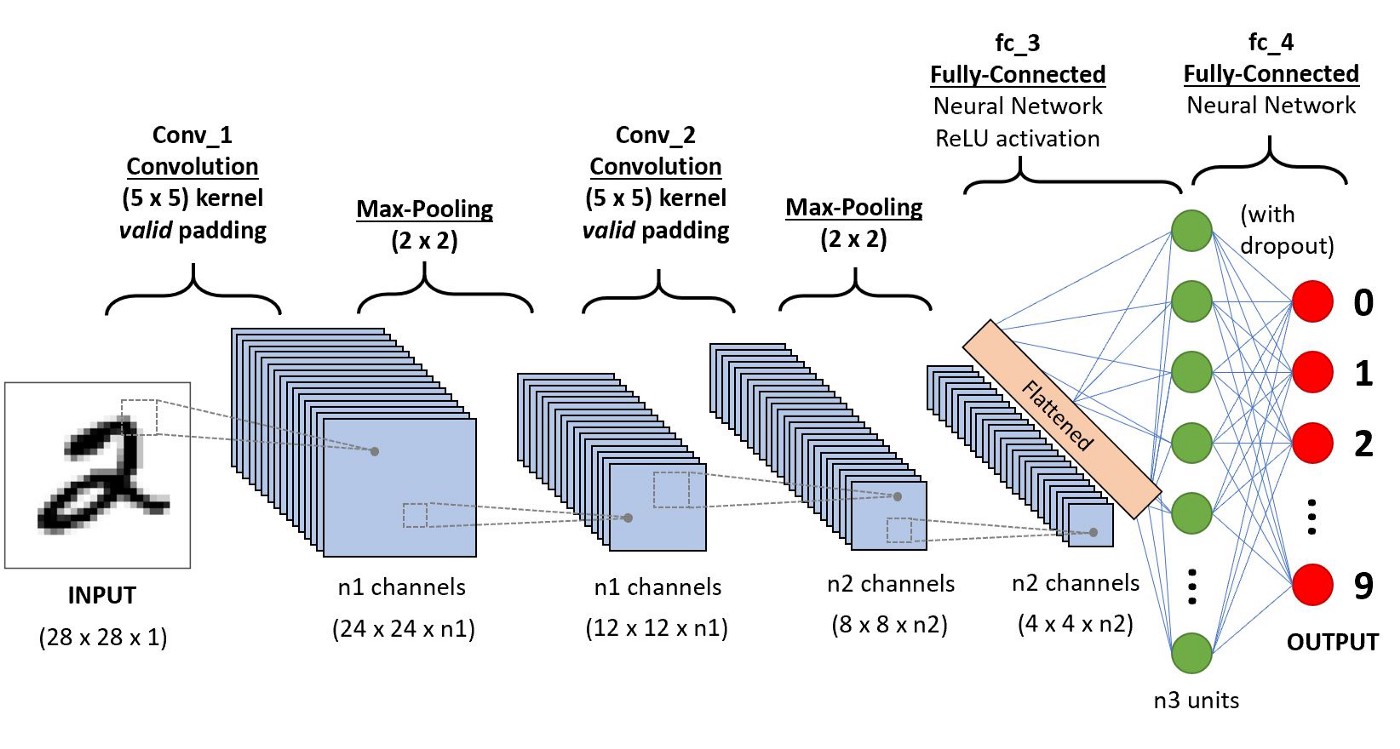
sklearn-extensions==0.0.2

scikit-learn==0.22.2.post1

Numpy

Pandas

**Models and Diagrams:**



**NON-FUNCTIONAL REQUIREMENT:**

**Usability:**  Usability is a quality attribute that assesses how easy user interfaces are to use. The word "usability" also refers to methods for improving ease-of-use during the design process.(how it was handle entire project easy)

**Security:** the quality or state of being secure: such as. a : freedom from danger : safety. b : freedom from fear or anxiety. c : freedom from the prospect of being laid off job security.

**Readability:** Readability is the ease with which a reader can understand a written text.

**Performance**: the execution of an action. : something accomplished : deed, feat. : the fulfillment of a claim, promise, or request : implementation. 3. : the action of representing a character in a play.

**Availability**: the quality or state of being available trying to improve the availability of affordable housing. 2 : an available person or thing.

**Scalability**: Scalability is the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands.

**3.3. PROCESS MODEL USED WITH JUSTIFICATION**

**SDLC (Umbrella Model):**

**Umbrella Activity**

**Umbrella Activity**

**Umbrella Activity**

1. Feasibility Study
2. TEAM FORMATION
3. Project Specification PREPARATION

Business Requirement Documentation

ANALYSIS & DESIGN

CODE

UNIT TEST

DOCUMENT CONTROL

ASSESSMENT

TRAINING

INTEGRATION & SYSTEM TESTING

DELIVERY/INSTALLATION

ACCEPTANCE TEST

Requirements Gathering

SDLC is nothing but Software Development Life Cycle. It is a standard which is used by software industry to develop good software.

**Stages in SDLC:**

* Requirement Gathering
* Analysis
* Designing
* Coding
* Testing
* Maintenance

**Requirements Gathering stage:**

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan. Each goal will be refined into a set of one or more requirements. These requirements define the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical processes to be managed, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.



These requirements are fully described in the primary deliverables for this stage: the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are *not* included in the requirements document.

The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages.

In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term requirements traceability.

The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.

* Feasibility study is all about identification of problems in a project.
* No. of staff required to handle a project is represented as Team Formation, in this case only modules are individual tasks will be assigned to employees who are working for that project.
* Project Specifications are all about representing of various possible inputs submitting to the server and corresponding outputs along with reports maintained by administrator.

**Analysis Stage:**

The planning stage establishes a bird's eye view of the intended software product, and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches.



The most critical section of the project plan is a listing of high-level product requirements, also referred to as goals. All of the software product requirements to be developed during the requirements definition stage flow from one or more of these goals. The minimum information for each goal consists of a title and textual description, although additional information and references to external documents may be included. The outputs of the project planning stage are the configuration management plan, the quality assurance plan, and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming Requirements stage, and high level estimates of effort for the out stages.

**Designing Stage:**

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.

  
When the design document is finalized and accepted, the RTM is updated to show that each design element is formally associated with a specific requirement. The outputs of the design stage are the design document, an updated RTM, and an updated project plan.

**Development (Coding) Stage:**

The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, and data management forms, data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts, and an online help system will be developed to guide users in their interactions with the software.



The RTM will be updated to show that each developed artifact is linked to a specific design element, and that each developed artifact has one or more corresponding test case items. At this point, the RTM is in its final configuration. The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an online help system that describes the operation of the software, an implementation map that identifies the primary code entry points for all major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

**Integration & Test Stage:**

During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability. During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data (or links to reference data source files) and production user list are compiled into the Production Initiation Plan.



The outputs of the integration and test stage include an integrated set of software, an online help system, an implementation map, a production initiation plan that describes reference data and production users, an acceptance plan which contains the final suite of test cases, and an updated project plan.

* **Installation & Acceptance Test:**

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loa ded onto the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the customer.

After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.



The primary outputs of the installation and acceptance stage include a production application, a completed acceptance test suite, and a memorandum of customer acceptance of the software. Finally, the PDR enters the last of the actual labor data into the project schedule and locks the project as a permanent project record. At this point the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.

**Maintenance:**

Outer rectangle represents maintenance of a project, Maintenance team will start with requirement study, understanding of documentation later employees will be assigned work and they will undergo training on that particular assigned category. For this life cycle there is no end, it will be continued so on like an umbrella (no ending point to umbrella sticks).

**3.4. Software Requirement Specification**

**3.4.1. Overall Description**

A Software Requirements Specification (SRS) – a [requirements specification](http://en.wikipedia.org/wiki/Requirements_specification) for a [software system](http://en.wikipedia.org/wiki/Software_system) is a complete description of the behavior of a system to be developed. It includes a set of [use cases](http://en.wikipedia.org/wiki/Use_case) that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. [Nonfunctional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements) are requirements which impose constraints on the design or implementation (such as [performance engineering](http://en.wikipedia.org/wiki/Performance_engineering) requirements, [quality](http://en.wikipedia.org/wiki/Quality_%28business%29) standards, or design constraints).

System requirements specification: A structured collection of information that embodies the requirements of a system. A [business analyst](http://en.wikipedia.org/wiki/Business_analyst), sometimes titled [system analyst](http://en.wikipedia.org/wiki/System_analyst), is responsible for analyzing the business needs of their clients and stakeholders to help identify business problems and propose solutions. Within the [systems development lifecycle](http://en.wikipedia.org/wiki/Systems_development_life_cycle) domain, the BA typically performs a liaison function between the business side of an enterprise and the information technology department or external service providers. Projects are subject to three sorts of requirements:

* [Business requirements](http://en.wikipedia.org/wiki/Business_requirements) describe in business terms what must be delivered or accomplished to provide value.
* Product requirements describe properties of a system or product (which could be one of several ways to accomplish a set of business requirements.)
* Process requirements describe activities performed by the developing organization. For instance, process requirements could specify .Preliminary investigation examine project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:
* **ECONOMIC FEASIBILITY**

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economical feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs. The system is economically feasible. It does not require any addition hardware or software. Since the interface for this system is developed using the existing resources and technologies available at NIC, There is nominal expenditure and economical feasibility for certain.

* **Operational Feasibility**

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization’s operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits. The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

* **TECHNICAL FEASIBILITY**

Earlier no system existed to cater to the needs of ‘Secure Infrastructure Implementation System’. The current system developed is technically feasible. It is a web based user interface for audit workflow at NIC-CSD. Thus it provides an easy access to .the users. The database’s purpose is to create, establish and maintain a workflow among various entities in order to facilitate all concerned users in their various capacities or roles. Permission to the users would be granted based on the roles specified. Therefore, it provides the technical guarantee of accuracy, reliability and security.

**3.4.2. External Interface Requirements**

**User Interface**

The user interface of this system is a user friendly python Graphical User Interface.

**Hardware Interfaces**

The interaction between the user and the console is achieved through python capabilities.

**Software Interfaces**

The required software is python.

**SYSTEM REQUIREMENT:**

**HARDWARE REQUIREMENTS:**

# Processor - Intel i3(min)

* Speed - 1.1 GHz
* RAM - 4GB(min)
* Hard Disk - 500 GB
* Key Board - Standard Windows Keyboard
* Mouse - Two or Three Button Mouse
* Monitor - SVGA

**SOFTWARE REQUIREMENTS:**

* Operating System - Windows10(min)
* Programming Language - Python

**4. SYSTEM DESIGN**

**CLASS DIAGRAM:**

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are represented with boxes which contain three parts:

* The upper part holds the name of the class
* The middle part contains the attributes of the class
* The bottom part gives the methods or operations the class can take or undertake

****

**USECASE DIAGRAM:**

A **use case diagram** at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as we



**SEQUENCE DIAGRAM**

A **sequence diagram** is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called **event diagrams**, **event scenarios**, and timing diagrams.



**COLLABORATION DIAGRAM:**

A collaboration diagram describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing both the static structure and dynamic behaviour of a system.



**COMPONENT DIAGRAM:**

In the Unified Modelling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems.

Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component. This illustrates the service consumer - service provider relationship between the two components.

**DEPLOYMENT DIAGRAM:**

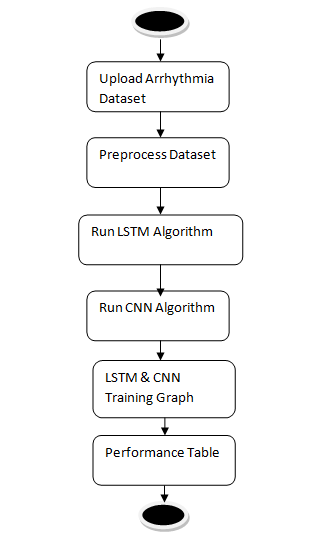
A **deployment diagram** in the Unified Modeling Language models the *physical* deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).

The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

****

**ACTIVITY DIAGRAM:**

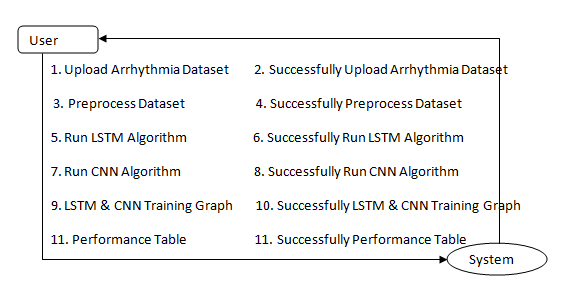
Activity diagram is another important diagram in UML to describe dynamic aspects of the system. It is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent



**Data flow :**

Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs. Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest. This analysis can be carried out in precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way.

As the name suggests, Data Flow Diagram (DFD) is an illustration that explicates the passage of information in a process. A DFD can be easily drawn using simple symbols. Additionally, complicated processes can be easily automated by creating DFDs using easy-to-use, free downloadable diagramming tools. A DFD is a model for constructing and analyzing information processes. DFD illustrates the flow of information in a process depending upon the inputs and outputs. A DFD can also be referred to as a Process Model. A DFD demonstrates business or technical process with the support of the outside data saved, plus the data flowing from the process to another and the end results.

****

**5. IMPLEMETATION**

**5.1 Python**

Python is a general-purpose language. It has wide range of applications from Web development (like: Django and Bottle), scientific and mathematical computing (Orange, SymPy, NumPy) to desktop graphical user Interfaces (Pygame, Panda3D). The syntax of the language is clean and length of the code is relatively short. It's fun to work in Python because it allows you to think about the problem rather than focusing on the syntax.

**History of Python:**

Python is a fairly old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991.

**Why Python was created?**

In late 1980s, Guido Van Rossum was working on the Amoeba distributed operating system group. He wanted to use an interpreted language like ABC (ABC has simple easy-to-understand syntax) that could access the Amoeba system calls. So, he decided to create a language that was extensible. This led to design of a new language which was later named Python.

**Why the name Python?**

No. It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

**Features of Python:**

**A simple language which is easier to learn**

Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#. Python makes programming fun and allows you to focus on the solution rather than syntax.

If you are a newbie, it's a great choice to start your journey with Python.

**Free and open-source**

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software’s written in it, you can even make changes to the Python's source code.

Python has a large community constantly improving it in each iteration.

**Portability**

You can move Python programs from one platform to another, and run it without any changes.

It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

**Extensible and Embeddable**

Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code.

This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

**A high-level, interpreted language**

Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and so on.

Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.

**Large standard libraries to solve common tasks**

Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself. For example: Need to connect MySQL database on a Web server? You can use MySQLdb library using import MySQLdb .

Standard libraries in Python are well tested and used by hundreds of people. So you can be sure that it won't break your application.

**Object-oriented**

Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem intuitively.

With OOP, you are able to divide these complex problems into smaller sets by creating objects.

**Applications of Python:**

**1. Simple Elegant Syntax**

Programming in Python is fun. It's easier to understand and write Python code. Why? The syntax feels natural. Take this source code for an example:

a = 2

b = 3

sum = a + b

print(sum)

**2. Not overly strict**

You don't need to define the type of a variable in Python. Also, it's not necessary to add semicolon at the end of the statement.

Python enforces you to follow good practices (like proper indentation). These small things can make learning much easier for beginners.

**3. Expressiveness of the language**

Python allows you to write programs having greater functionality with fewer lines of code. Here's a link to the source code of Tic-tac-toe game with a graphical interface and a smart computer opponent in less than 500 lines of code. This is just an example. You will be amazed how much you can do with Python once you learn the basics.

**4. Great Community and Support**

Python has a large supporting community. There are numerous active forums online which can be handy if you are stuck.

**5.2 Sample Code:**

from tkinter import \*

import tkinter

from tkinter import filedialog

import matplotlib.pyplot as plt

from tkinter.filedialog import askopenfilename

import numpy as np

import pandas as pd

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

import seaborn as sns

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

import os

import webbrowser

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import normalize

from sklearn.decomposition import PCA

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras.layers import LSTM

from keras.utils.np\_utils import to\_categorical

from keras.models import model\_from\_json

import pickle

from keras.layers import MaxPooling2D

from keras.layers import Activation, Flatten

from keras.layers import Convolution2D

main = tkinter.Tk()

main.title("Automated Detection of Cardiac Arrhythmia using Recurrent Neural Network")

main.geometry("1200x1200")

global X\_train, X\_test, y\_train, y\_test, pca

global model, dataset

global filename

global X, Y

accuracy = []

precision = []

recall = []

fscore = []

sensitivity = []

specificity = []

labels = ['Normal heart', 'Ischemic changes (Coronary Artery Disease)', 'Old Anterior Myocardial Infarction',

'Old Inferior Myocardial Infarction', 'Sinus tachycardy', 'Sinus bradycardy', 'Right bundle branch block']

def uploadDataset():

global filename, dataset

text.delete('1.0', END)

filename = filedialog.askopenfilename(initialdir="Dataset")

text.insert(END,str(filename)+" Dataset Loaded\n\n")

pathlabel.config(text=str(filename)+" Dataset Loaded")

dataset = pd.read\_csv(filename)

text.insert(END,str(dataset.head()))

label = dataset.groupby('279').size()

label.plot(kind="bar")

plt.show()

def preprocessDataset():

global X, Y, dataset, pca

global X\_train, X\_test, y\_train, y\_test

text.delete('1.0', END)

le = LabelEncoder()

dataset.fillna(0, inplace = True)

dataset["279"] = pd.Series(le.fit\_transform(dataset["279"].astype(str)))

temp = dataset.values

X = temp[:,0:temp.shape[1]-1]

Y = temp[:,temp.shape[1]-1]

indices = np.arange(X.shape[0])

np.random.shuffle(indices)

X = X[indices]

Y = Y[indices]

print(Y)

X = normalize(X)

pca = PCA(n\_components = 40)

X = pca.fit\_transform(X)

Y = to\_categorical(Y)

XX = np.reshape(X, (X.shape[0], X.shape[1], 1))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(XX, Y, test\_size = 0.2)

text.insert(END,"Total records found in dataset : "+str(X.shape[0])+"\n\n")

text.insert(END,"Different diseases found in dataset\n\n")

text.insert(END,str(labels)+"\n\n")

text.insert(END,"Dataset Train & Test Split Details\n\n")

text.insert(END,"Total records used to train LSTM & CNN : "+str(X\_train.shape[0])+"\n")

text.insert(END,"Total records used to test LSTM & CNN : "+str(X\_test.shape[0])+"\n")

def calculateMetrics(algorithm, predict, y\_test):

a = accuracy\_score(y\_test,predict)\*100

p = precision\_score(y\_test, predict,average='macro') \* 100

r = recall\_score(y\_test, predict,average='macro') \* 100

f = f1\_score(y\_test, predict,average='macro') \* 100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

text.insert(END,algorithm+" Accuracy : "+str(a)+"\n")

text.insert(END,algorithm+" Precision : "+str(p)+"\n")

text.insert(END,algorithm+" Recall : "+str(r)+"\n")

text.insert(END,algorithm+" FScore : "+str(f)+"\n")

conf\_matrix = confusion\_matrix(y\_test, predict)

total = sum(sum(conf\_matrix))

se = conf\_matrix[0,0]/(conf\_matrix[0,0]+conf\_matrix[0,1])

text.insert(END,algorithm+' Sensitivity : '+str(se)+"\n")

sp = conf\_matrix[1,1]/(conf\_matrix[1,0]+conf\_matrix[1,1])

text.insert(END,algorithm+' Specificity : '+str(sp)+"\n\n")

sensitivity.append(se)

specificity.append(sp)

text.update\_idletasks()

plt.figure(figsize =(6, 6))

ax = sns.heatmap(conf\_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="viridis" ,fmt ="g");

ax.set\_ylim([0,len(labels)])

plt.title(algorithm+" Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()

def runLSTM():

global X\_train, X\_test, y\_train, y\_test

global accuracy, precision, recall, fscore, sensitivity, specificity

text.delete('1.0', END)

accuracy.clear()

precision.clear()

recall.clear()

fscore.clear()

sensitivity.clear()

specificity.clear()

if os.path.exists('model/lstm\_model.json'):

with open('model/lstm\_model.json', "r") as json\_file:

loaded\_model\_json = json\_file.read()

lstm = model\_from\_json(loaded\_model\_json)

json\_file.close()

lstm.load\_weights("model/lstm\_model\_weights.h5")

lstm.\_make\_predict\_function()

else:

lstm\_model = Sequential()#defining deep learning sequential object

#adding LSTM layer with 100 filters to filter given input X train data to select relevant features

lstm\_model.add(LSTM(100,input\_shape=(X\_train.shape[1], X\_train.shape[2])))

#adding dropout layer to remove irrelevant features

lstm\_model.add(Dropout(0.2))

#adding another layer

lstm\_model.add(Dense(100, activation='relu'))

#defining output layer for prediction

lstm\_model.add(Dense(y\_train.shape[1], activation='softmax'))

#compile LSTM model

lstm\_model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

#start training model on train data and perform validation on test data

hist = lstm\_model.fit(X\_train, y\_train, epochs=100, batch\_size=16, validation\_data=(X\_test, y\_test))

#save model weight for future used

lstm\_model.save\_weights('model/lstm\_model\_weights.h5')

model\_json = lstm\_model.to\_json()

with open("model/lstm\_model.json", "w") as json\_file:

json\_file.write(model\_json)

json\_file.close()

f = open('model/lstm\_history.pckl', 'wb')

pickle.dump(hist.history, f)

f.close()

print(lstm.summary())

predict = lstm.predict(X\_test)

predict = np.argmax(predict, axis=1)

testY = np.argmax(y\_test, axis=1)

calculateMetrics("LSTM", predict, testY)

def runCNN():

global X, Y

XX = np.reshape(X, (X.shape[0], X.shape[1], 1, 1))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(XX, Y, test\_size = 0.2)

if os.path.exists('model/cnn\_model.json'):

with open('model/cnn\_model.json', "r") as json\_file:

loaded\_model\_json = json\_file.read()

cnn = model\_from\_json(loaded\_model\_json)

json\_file.close()

cnn.load\_weights("model/cnn\_model\_weights.h5")

cnn.\_make\_predict\_function()

else:

cnn = Sequential()

cnn.add(Convolution2D(32, 1, 1, input\_shape = (X\_train.shape[1], X\_train.shape[2], X\_train.shape[3]), activation = 'relu'))

cnn.add(MaxPooling2D(pool\_size = (1, 1)))

cnn.add(Convolution2D(32, 1, 1, activation = 'relu'))

cnn.add(MaxPooling2D(pool\_size = (1, 1)))

cnn.add(Flatten())

cnn.add(Dense(output\_dim = 256, activation = 'relu'))

cnn.add(Dense(output\_dim = y\_train.shape[1], activation = 'softmax'))

cnn.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

hist = cnn.fit(X\_train, y\_train, batch\_size=16, epochs=100, shuffle=True, verbose=2, validation\_data=(X\_test, y\_test))

cnn.save\_weights('model/cnn\_model\_weights.h5')

model\_json = cnn.to\_json()

with open("model/cnn\_model.json", "w") as json\_file:

json\_file.write(model\_json)

json\_file.close()

f = open('model/cnn\_history.pckl', 'wb')

pickle.dump(hist.history, f)

f.close()

print(cnn.summary())

predict = cnn.predict(X\_test)

predict = np.argmax(predict, axis=1)

testY = np.argmax(y\_test, axis=1)

calculateMetrics("CNN", predict, testY)

def graph():

f = open('model/lstm\_history.pckl', 'rb')

data = pickle.load(f)

f.close()

lstm\_accuracy = data['accuracy']

lstm\_loss = data['loss']

f = open('model/cnn\_history.pckl', 'rb')

data = pickle.load(f)

f.close()

cnn\_accuracy = data['accuracy']

cnn\_loss = data['loss']

plt.figure(figsize=(10,6))

plt.grid(True)

plt.xlabel('EPOCH')

plt.ylabel('Accuracy/Error Rate')

plt.plot(lstm\_accuracy, 'ro-', color = 'green')

plt.plot(lstm\_loss, 'ro-', color = 'blue')

plt.plot(cnn\_accuracy, 'ro-', color = 'orange')

plt.plot(cnn\_loss, 'ro-', color = 'red')

plt.legend(['LSTM Accuracy', 'LSTM Loss','CNN Accuracy','CNN Loss'], loc='upper left')

plt.title('LSTM Vs CNN Training Accuracy & Loss Graph')

plt.show()

def performanceTable():

output = '<table border=1 align=center>'

output+= '<tr><th>Dataset Name</th><th>Algorithm Name</th><th>Accuracy</th><th>Precision</th><th>Recall</th><th>FSCORE</th><th>Sensitivity</th><th>Specificity</th></tr>'

output+='<tr><td>MIT-BH Dataset</td><td>LSTM</td><td>'+str(accuracy[0])+'</td><td>'+str(precision[0])+'</td><td>'+str(recall[0])+'</td><td>'+str(fscore[0])+'</td><td>'+str(sensitivity[0])+'</td><td>'+str(specificity[0])+'</td></tr>'

output+='<tr><td>MIT-BH Dataset</td><td>CNN</td><td>'+str(accuracy[1])+'</td><td>'+str(precision[1])+'</td><td>'+str(recall[1])+'</td><td>'+str(fscore[1])+'</td><td>'+str(sensitivity[1])+'</td><td>'+str(specificity[1])+'</td></tr>'

output+='</table></body></html>'

f = open("output.html", "w")

f.write(output)

f.close()

webbrowser.open("output.html",new=1)

def close():

main.destroy()

font = ('times', 14, 'bold')

title = Label(main, text='Automated Detection of Cardiac Arrhythmia using Recurrent Neural Network')

title.config(bg='DarkGoldenrod1', fg='black')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=5,y=5)

font1 = ('times', 13, 'bold')

uploadButton = Button(main, text="Upload Arrhythmia Dataset", command=uploadDataset)

uploadButton.place(x=50,y=100)

uploadButton.config(font=font1)

pathlabel = Label(main)

pathlabel.config(bg='brown', fg='white')

pathlabel.config(font=font1)

pathlabel.place(x=560,y=100)

preprocessButton = Button(main, text="Preprocess Dataset", command=preprocessDataset)

preprocessButton.place(x=50,y=150)

preprocessButton.config(font=font1)

lstmButton = Button(main, text="Run LSTM Algorithm", command=runLSTM)

lstmButton.place(x=50,y=200)

lstmButton.config(font=font1)

cnnButton = Button(main, text="Run CNN Algorithm", command=runCNN)

cnnButton.place(x=50,y=250)

cnnButton.config(font=font1)

graphButton = Button(main, text="LSTM & CNN Training Graph", command=graph)

graphButton.place(x=50,y=300)

graphButton.config(font=font1)

ptButton = Button(main, text="Performance Table", command=performanceTable)

ptButton.place(x=50,y=350)

ptButton.config(font=font1)

exitButton = Button(main, text="Exit", command=close)

exitButton.place(x=50,y=400)

exitButton.config(font=font1)

font1 = ('times', 12, 'bold')

text=Text(main,height=25,width=100)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=400,y=150)

text.config(font=font1)

main.config(bg='LightSteelBlue1')

main.mainloop()

**6. TESTING:**

**Implementation and Testing:**

Implementation is one of the most important tasks in project is the phase in which one has to be cautions because all the efforts undertaken during the project will be very interactive. Implementation is the most crucial stage in achieving successful system and giving the users confidence that the new system is workable and effective. Each program is tested individually at the time of development using the sample data and has verified that these programs link together in the way specified in the program specification. The computer system and its environment are tested to the satisfaction of the user.

**Implementation**

The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion. The system may be requiring extensive user training. The initial parameters of the system should be modifies as a result of a programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained either on the inkjet or dot matrix printer, which is available at the disposal of the user. The proposed system is very easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

## Testing

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property functions as a unit. The test data should be chosen such that it passed through all possible condition. Actually testing is the state of implementation which aimed at ensuring that the system works accurately and efficiently before the actual operation commence. The following is the description of the testing strategies, which were carried out during the testing period.

### System Testing

Testing has become an integral part of any system or project especially in the field of information technology. The importance of testing is a method of justifying, if one is ready to move further, be it to be check if one is capable to with stand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical. When the software is developed before it is given to user to use the software must be tested whether it is solving the purpose for which it is developed. This testing involves various types through which one can ensure the software is reliable. The program was tested logically and pattern of execution of the program for a set of data are repeated. Thus the code was exhaustively checked for all possible correct data and the outcomes were also checked.

**Module Testing**

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example the job classification module is tested separately. This module is tested with different job and its approximate execution time and the result of the test is compared with the results that are prepared manually. The comparison shows that the results proposed system works efficiently than the existing system. Each module in the system is tested separately. In this system the resource classification and job scheduling modules are tested separately and their corresponding results are obtained which reduces the process waiting time.

**Integration Testing**

After the module testing, the integration testing is applied. When linking the modules there may be chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct. Thus the mapping of jobs with resources is done correctly by the system.

**Acceptance Testing**

When that user fined no major problems with its accuracy, the system passers through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation.

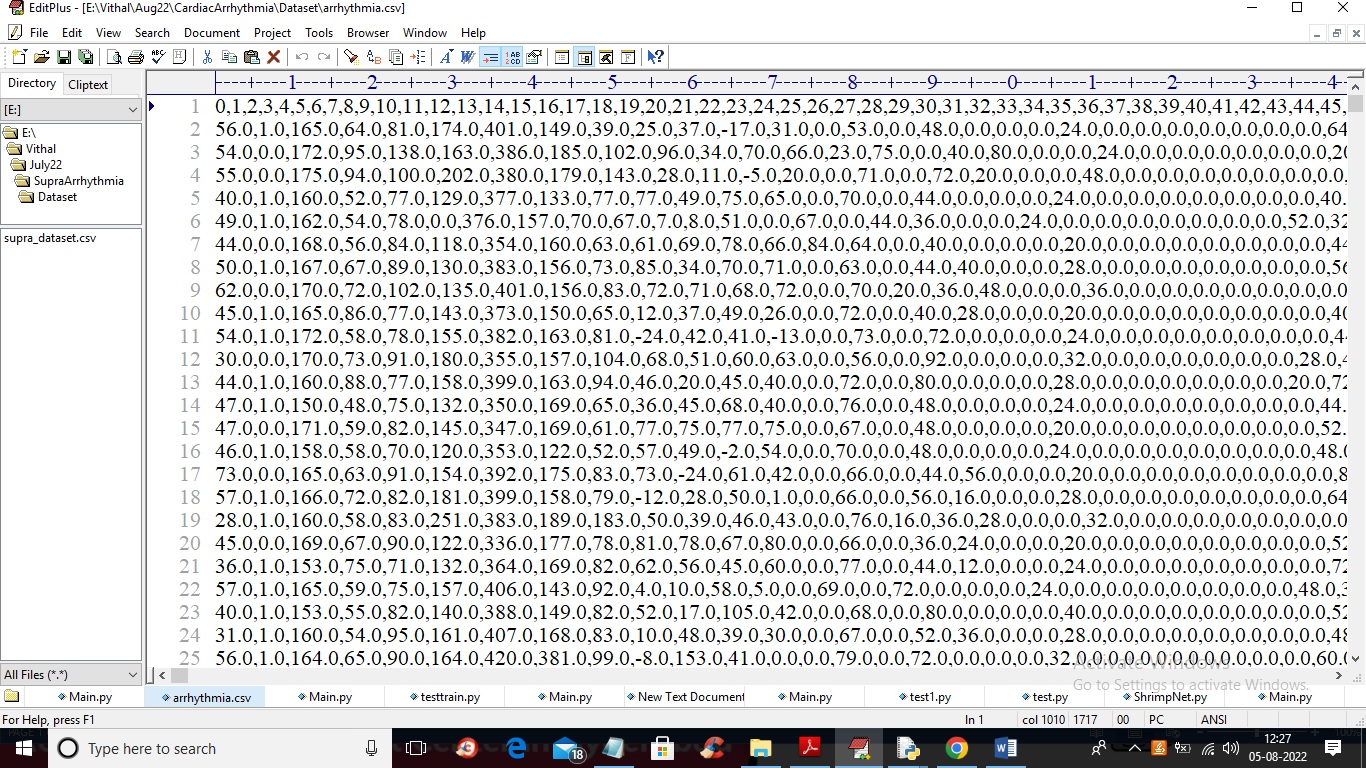
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Id** | **Test Case Name** | **Test Case Desc.** | **Test Steps** | | | | **Test Case Status** | **Test Priority** |
| **Step** | **Expected** | | **Actual** |
| 01 | Upload Arrhythmia Dataset | Verify Upload Arrhythmia Dataset or not | If Arrhythmia Dataset may not upload | we cannot do any further operations | we can do further operations | | High | High |
| 02 | Preprocess Dataset | Verify Preprocess Dataset or not | If Dataset may not Preprocessed | we cannot do any further operations | we can do further operations | | High | High |
| 03 | Run LSTM Algorithm | Verify Run LSTM Algorithm or not | If LSTM Algorithm May not be Run | we cannot do any further operations | we can do further operations | | High | High |
| 04 | Run CNN Algorithm | Verify Run CNN Algorithm or not | If CNN Algorithm not Run | We cannot run  operation | We can Run the Operation | | High | High |
| 05 | LSTM & CNN Training Graph | Verify LSTM & CNN Training Graph or not | If LSTM & CNN Training Graph not be plot | we cannot do any further operations | we can do further operations | | High | High |
| 06 | Performance Table | Verify Performance Table or not | If Performance Table may not be done | we cannot do any further operations | we can do further operations | | High | High |

**7. SCREENSHOTS:**

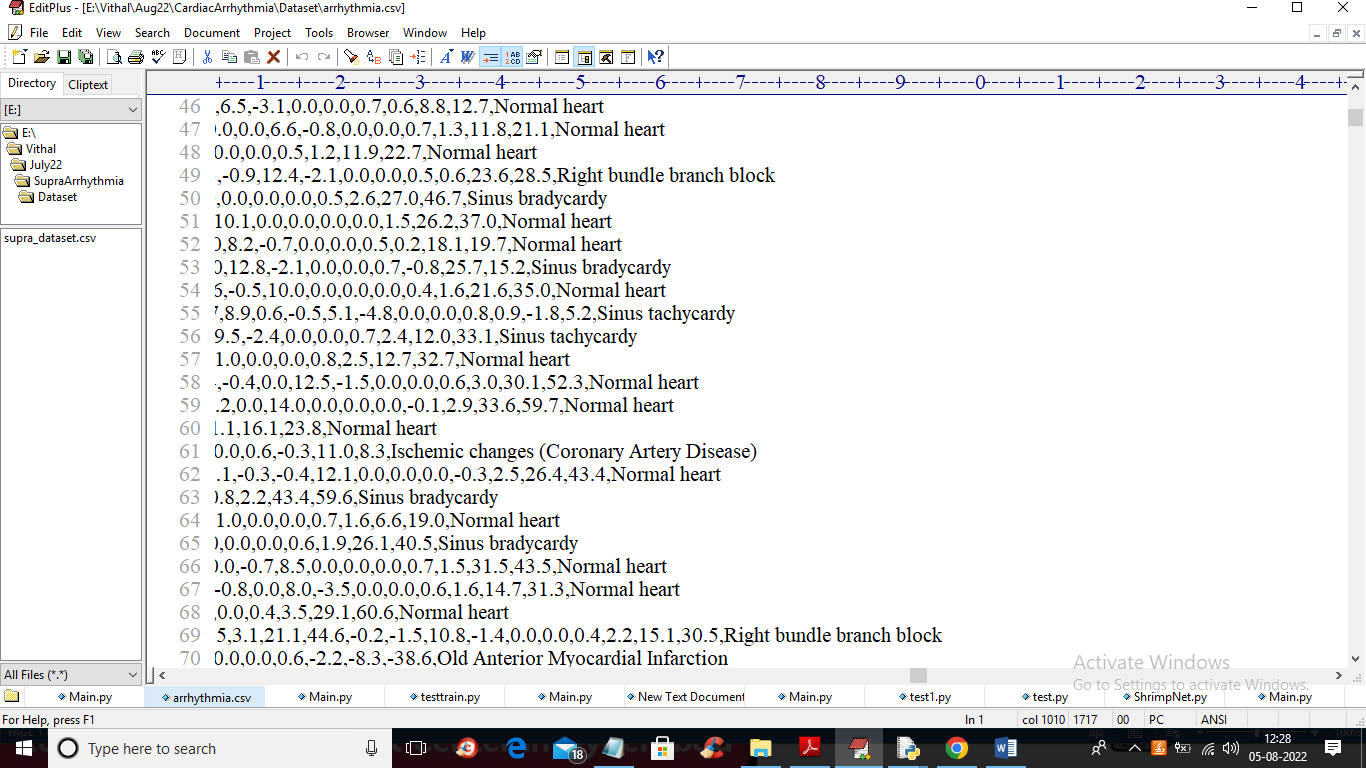
In this project you asked to design CNN and LSTM algorithm to predict Arrhythmia diseases with 7 different stages. To train both algorithm we have used MIT-BH dataset with 7 different disease stages.

You told to get 95% accuracy which is difficult to get with LSTM but with CNN we got 97% accuracy.

To train both algorithm we have used below dataset

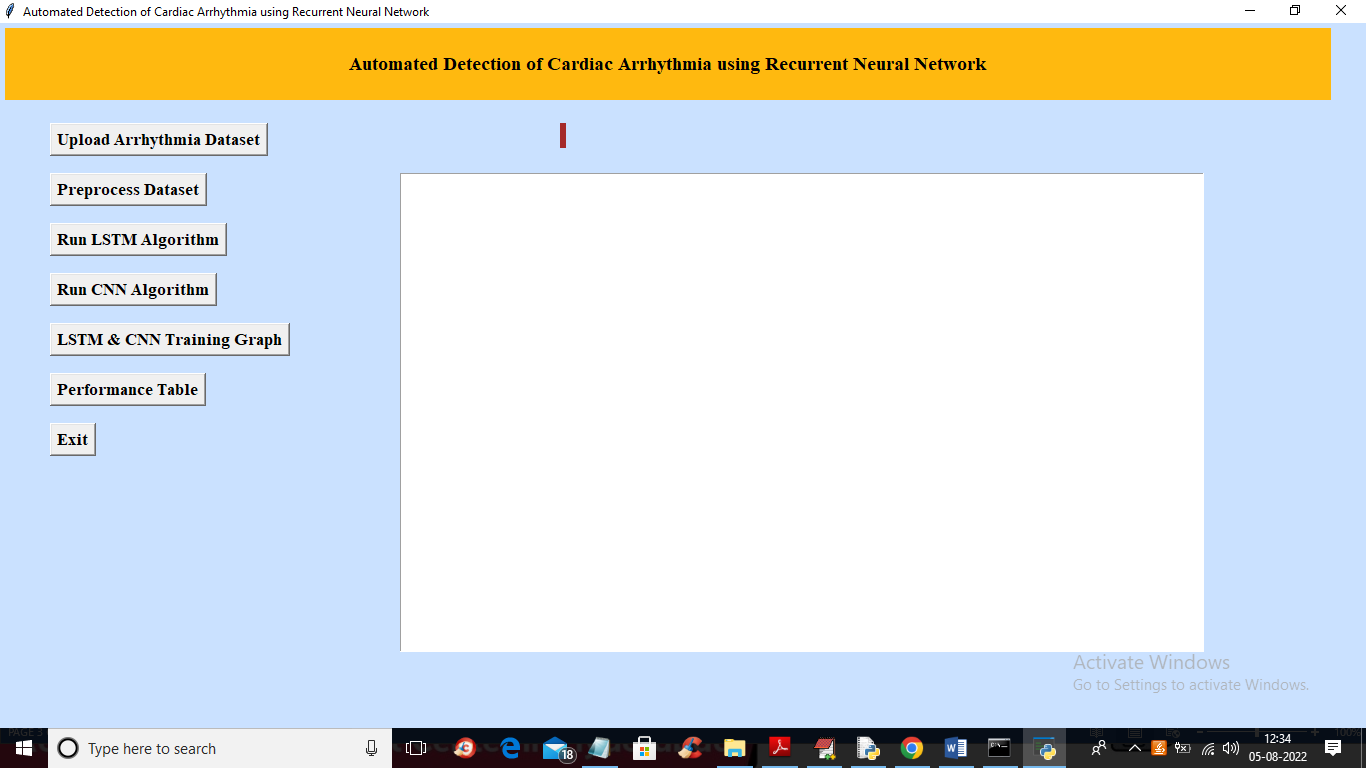


In above dataset screen first row contains column names and remaining rows contains dataset values and in last column we can see class label as disease name

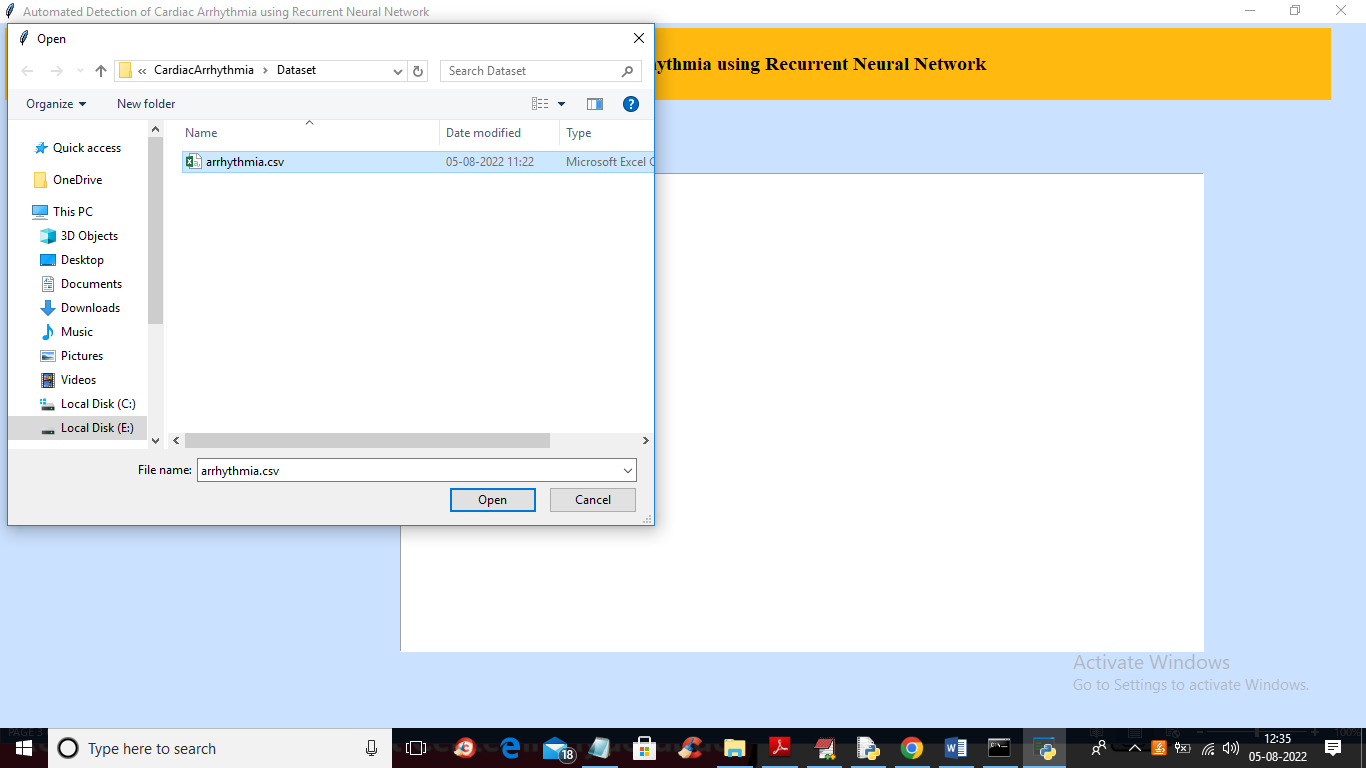


In above screen in last column we can see disease name and by using above dataset we are training both algorithms

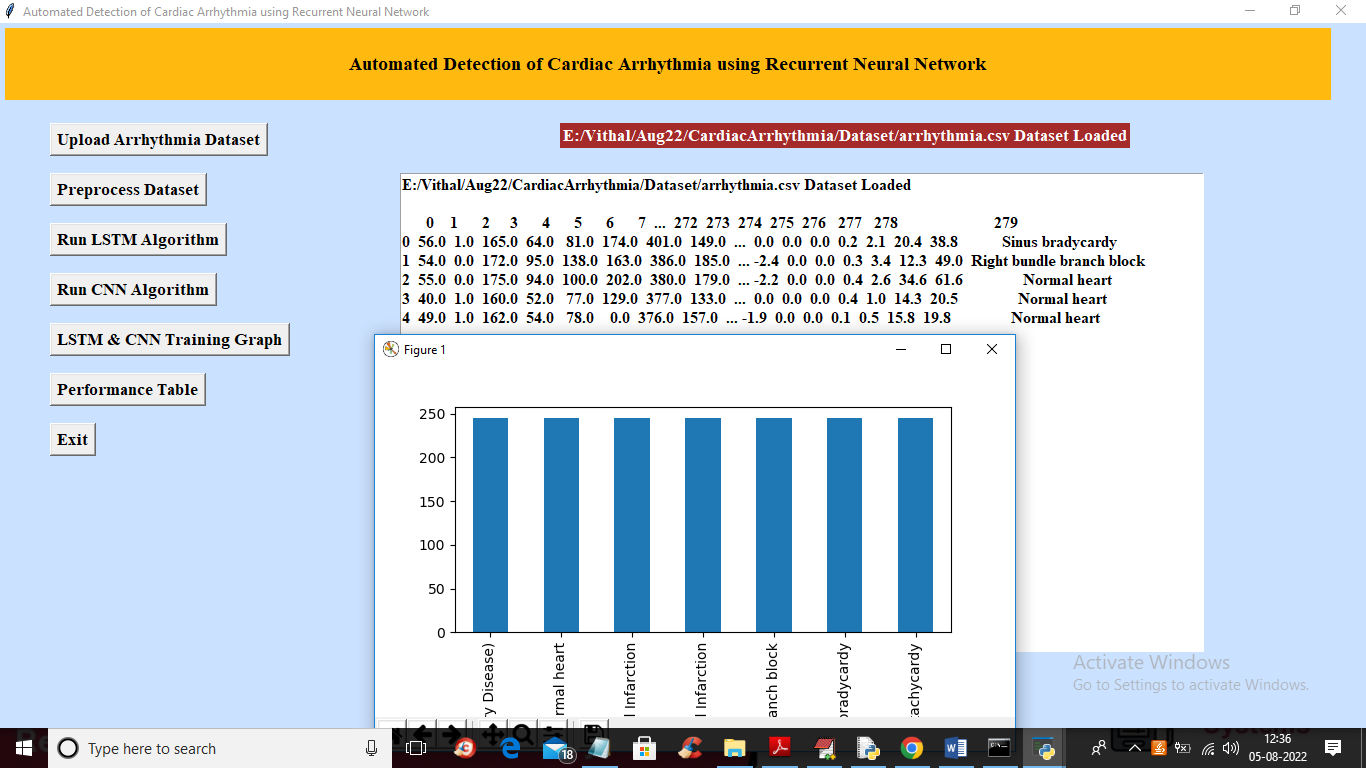
To run project double click on ‘run.bat’ file to get below screen



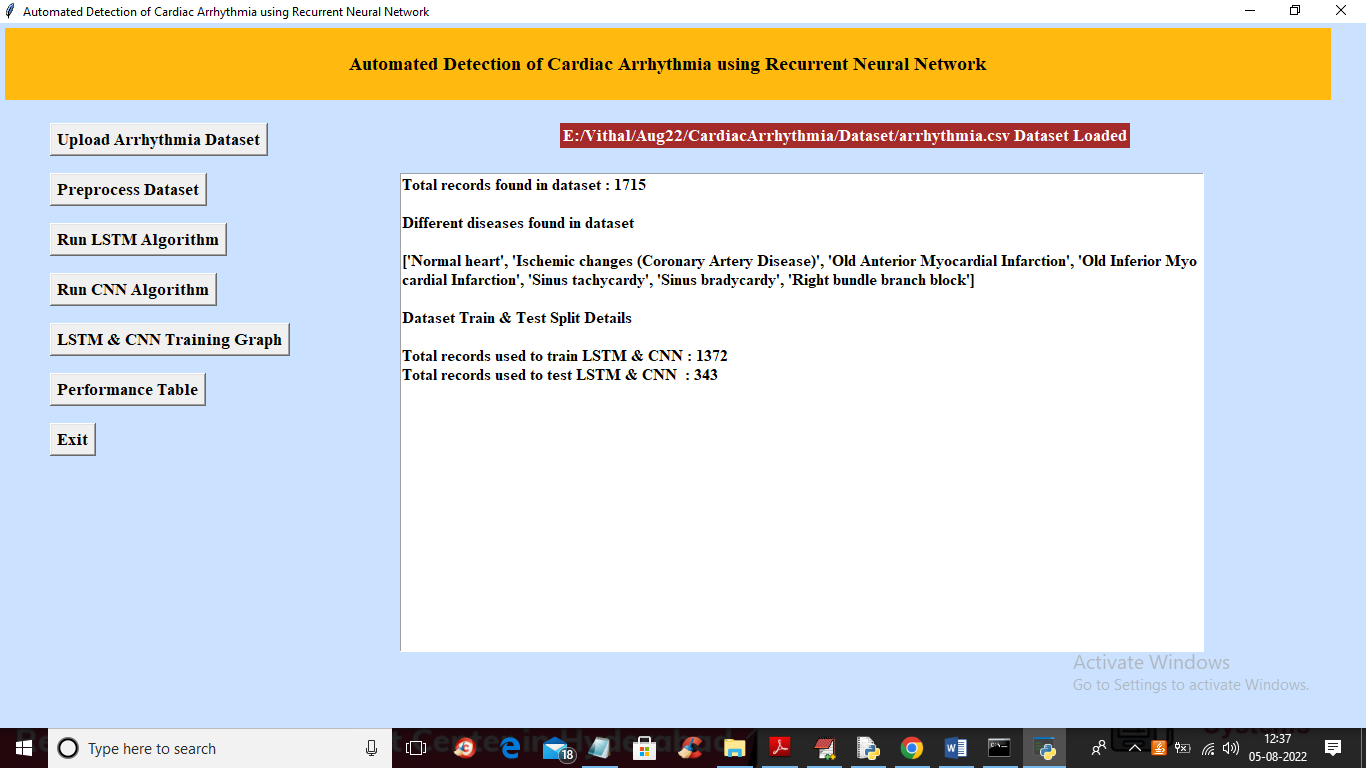
In above screen click on ‘Upload Arrhythmia Dataset’ button to upload dataset and get below output



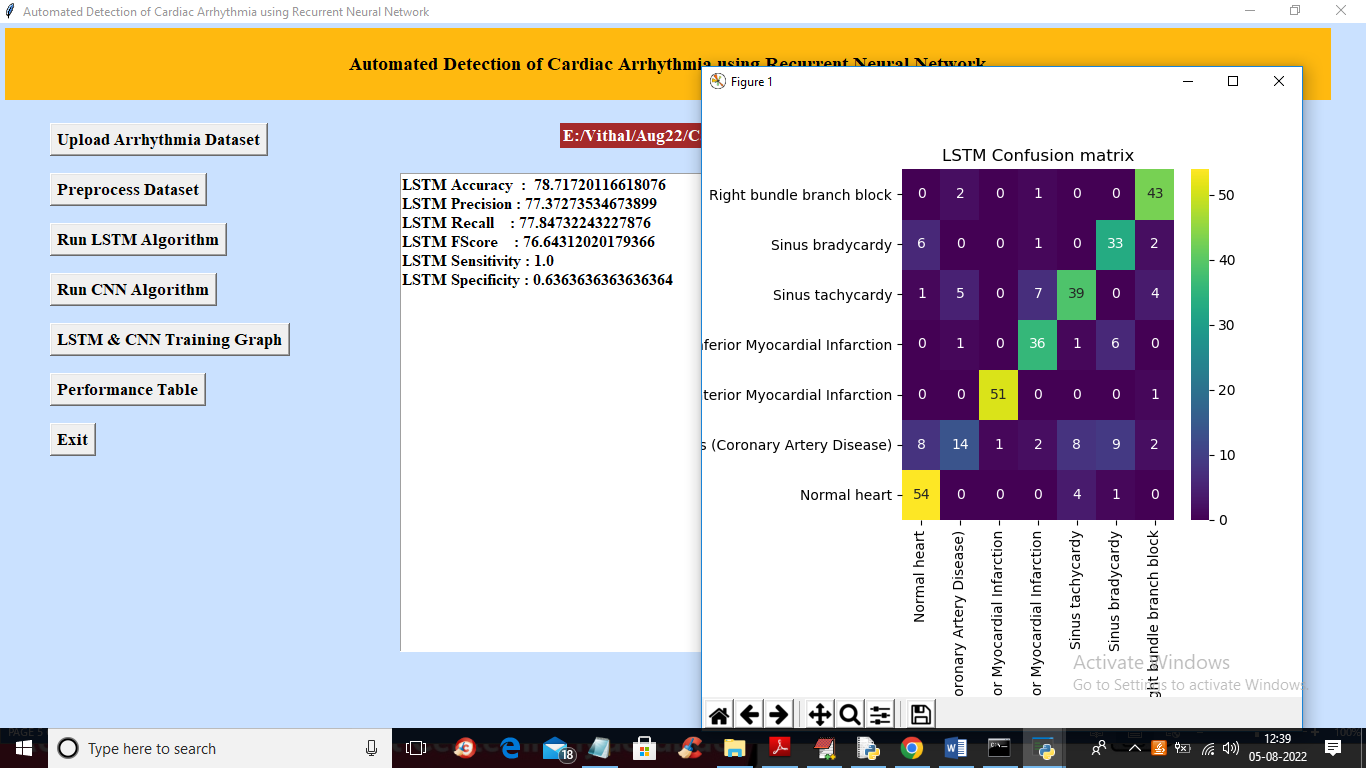
In above screen selecting and uploading ‘Arrhythmia’ dataset and then click on ‘Open’ button to load dataset and get below output



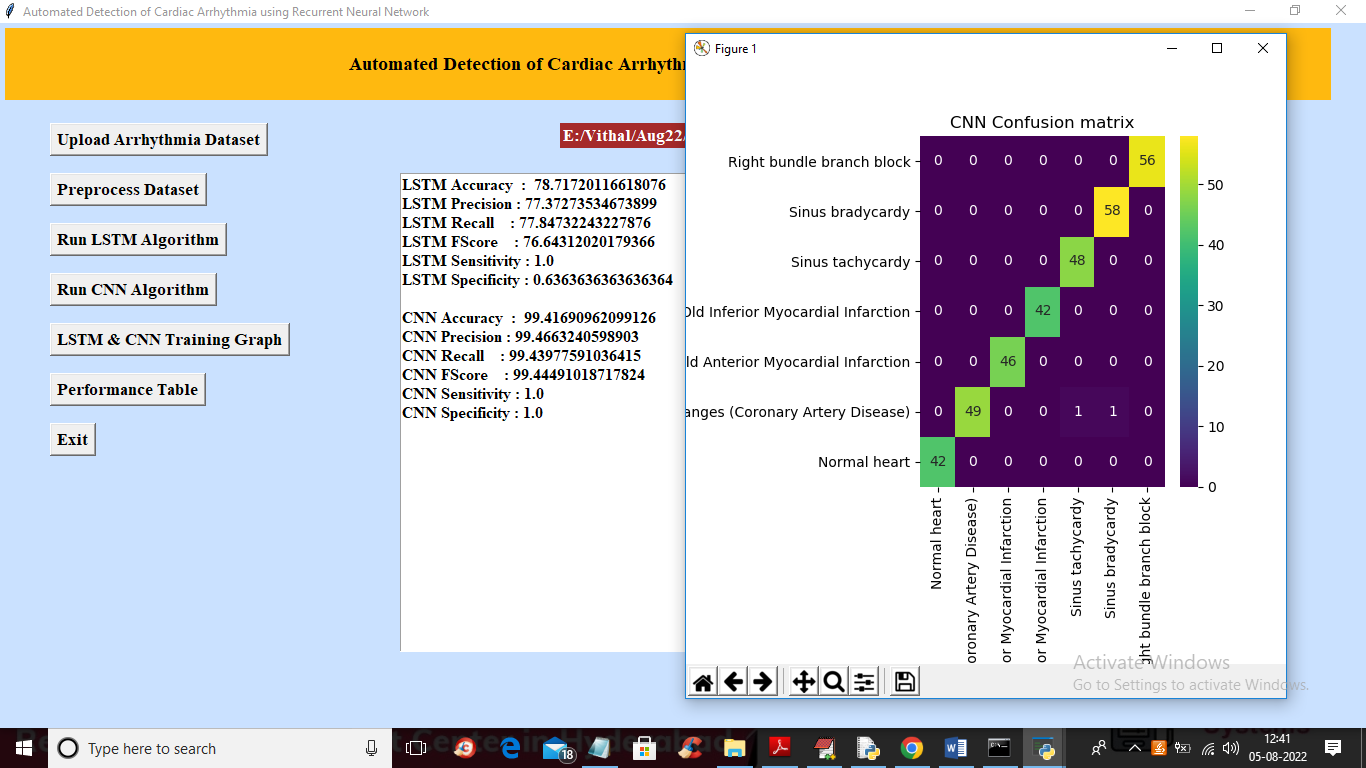
In above screen we can see dataset loaded and in graph x-axis represents 7 different disease stages and y-axis represents number of records found for that disease in dataset and in above screen we can see dataset contains some non-numeric values but algorithm accept only numeric values so close above graph and then click on ‘Preprocess Dataset’ button to process dataset and then split into train and test



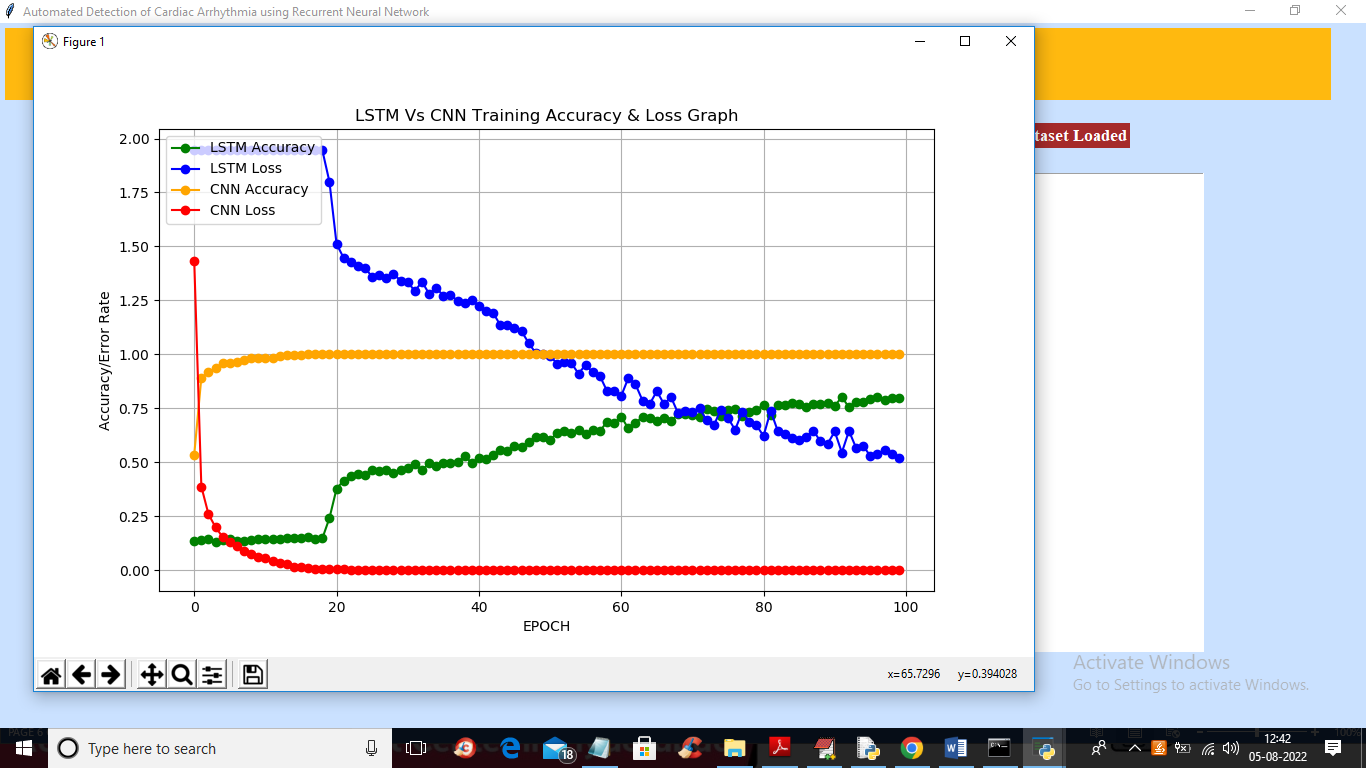
In above screen all dataset converted to numeric format and we can see total dataset size with train and test split details and displaying names of disease and now click on ‘Run LSTM Algorithm’ button to train LSTM with above process dataset and get below output



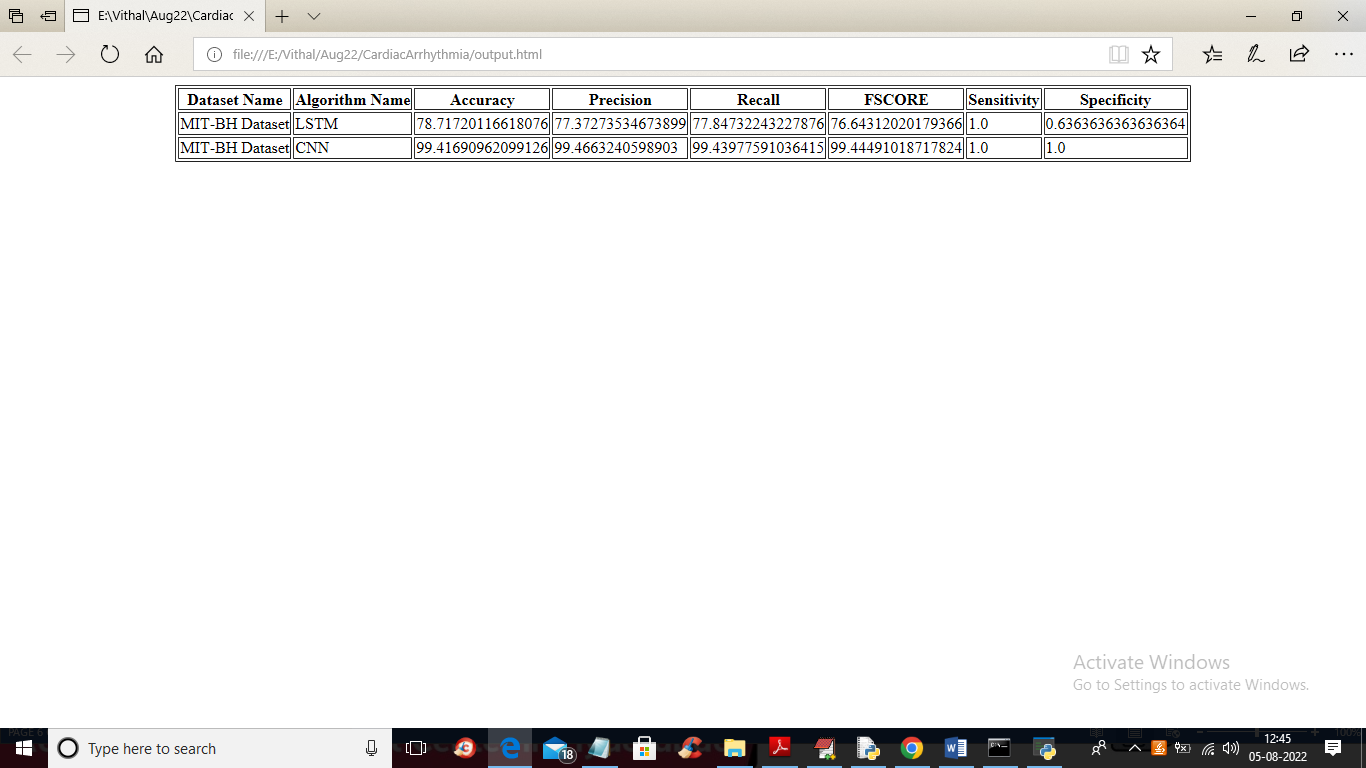
In above screen with LSTM we got 78% accuracy and in confusion matrix graph x-axis represents Predicted classes and y-axis represents TRUE classes and all blue colour boxes count are wrong prediction and different colour boxes count are correct prediction and we can see LSTM predicted so many wrong classes and now close above graph and then click on ‘Run CNN Algorithm’ button to train CNN and get below output



In above screen with CNN we got 99% accuracy and in confusion matrix graph only 2 counts in blue colour boxes are wrong prediction and rest are correct prediction. Now click on ‘LSTM & CNN Training Graph’ button to get below graph.



In above graph x-axis represents training epoch and y-axis represents training accuracy and loss values and green colour line represents LSTM accuracy and orange colour line represents CNN accuracy and red colour line represents CNN loss and blue line represents LSTM loss and in above graph we can see both algorithms accuracy got increase in every epoch and loss get decrease and now close above graph and then click on “Performance Table’ button to get below output



In above screen we can see output metrics of both algorithms in tabular format

**8. CONCLUSION:**

Cardiac arrhythmia is basically an irregularity in heart rhythm. Some types of cardiac arrhythmia can lead to complications like stroke, heart attack and may even lead to sudden cardiac death. So, timely detection and diagnosis of arrhythmia is very important. Once arrhythmia is detected, next stage of identification of category of arrhythmia can be done. We developed an automated non-invasive system based on deep learning networks to per form the basic classification of a given ECG data as belonging to normal ECG or abnormal (having arrhythmia) ECG using the most popular publically available MIT-BIH arrhythmia database. We compared the performance using a variety of deep learning architectures of CNN, CNN-RNN, CNN-LSTM and CNN-GRU and obtained an accuracy of 0.834. With concern on computational cost, we are not able to train more complex architecture. The reported results can be further improved by using more complex deep learning architecture. The complex network architectures can be trained by using advanced hardware and following distributed approach in training that we are incompetent to try. We have discussed the role of deep learning techniques such as CNN and recurrent structures in the task of arrhythmia classification. The highlight of the proposed method is that it doesn't need any noise filtering and feature engineering mechanisms. The results obtained prove that the performance of our method is better than other published results in effectively classifying ECG as belonging to normal or arrhythmia class. Though deep learning networks produces excellent results, the disadvantage lies in the insufficient understanding of the complex inner mechanisms of the deep learning networks. This could be overcome by re modeling the nonlinear deep networks to a linear form by computing eigenvalues and eigenvectors in different time steps. The future work can be the collection of real world datasets from hospitals having cardiac care units and the application of the same methodologies to the real datasets.

**9. REFERENCES:**

[1] Naser Safdarian, Nader Jafarnia, and GholamrezaAttarodi. (2014) “A new pattern recognition method for detection and localization of myocardial infarction using T-wave integral and total integral as extracted features from one cycle of ECG signal.” Journal of Biomedical Science and Engineering 7(10):818-824.

[2] Sharma L.N., Tripathy R.K., SamarendraDandapat. (2015) “Multiscale energy and eigenspace approach to detection and localization of myocardial infarction.” IEEE Transactions on biomedical engineering62(7):1827-37.

[3] Rajendra Acharya U., Hamildo Fujita, Vidya K Sudarshan, Shu Lih Oh, Muhammad Adam, et al. (2016) “ Automated detection and localization of myocardial infarction using electrocardiogram: A comparative study of different leads.” Knowledge-Based Systems99: 146-156.

[4] Rajendra Acharya U., Hamildo Fujita, Shu Lih Oh, Yuki Hagiwara, Jen Hong Tan, and Muhammad Adam. (2017) “Application of deep convolutional neural network for automated detection of myocardial infarction using ECG signals.” Information Sciences415: 190-198.

[5] Tahsin Reasat and Celia Shahnaz. (2017) “Detection of inferior myocardial infarction using shallow convolutional neural networks.”arXiv preprint arXiv:1710.01115.

[6] Babak Mohammadzadeh-Asl and SeyedKamaledinSetarehdan. (2006) “Neural network based arrhythmia classification using Heart Rate Variability signal.” In 14th EuropeanSignal Processing Conferencepages 1-4.

[7]Abhinav-Vishwa, Lal M.K., Dixit S, and Vardwaj P. (2011) “Clasification of arrhythmic ECG data using machine learning techniques.” International Journal of Interactive Multimedia and Artificial Intelligence1(4):67-70.

[8]Goutham Swapna, Dhanjoo N Ghista, Roshan Joy Martis, Alvin PC Ang, and SubbhuraamVinithaSree. (2012) “ECG signal generation and heart rate variability signal extraction: Signal Processing, Feature detection, and their correlation with cardiac diseases.” Journal of Mechanics in Medicine and Biology12(4):1240012

[9] Sujadevi V.G., Soman K.P., and Vinayakumar R. (2017) “Real-time detection of atrial fibrillation from short time single lead ECG traces using recurrent neural networks.” In The International Symposium on Intelligent Systems Technologies and Applications pages 212-221 Springer.

[10] Rahul K. Pathinarupothi ,Vinayakumar R., EkanathRangan, Gopalakrishnan E., and Soman. K.P.(2017) “ Instantaneous heart rate as a robust feature for sleep apnea severity detection using deep learning.” In IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)pages 293-296.

[11] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. (2000) “PhysioBank, PhysioToolkit, and PhysioNet.” Circulation101 (23):215-220.

[12] Jeffrey L. Elman. (1990) “Finding structure in time.”Cognitive Science 14(2): 179-211.

[13] Sepp Hochreiter, and Jürgen Schmidhuber. (1997) “ Long short-term memory.” Neural Computation9(8): 1735-1780.

[14] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. (1999) “Learning to Forget: Continual prediction with LSTM. “Neural Computation12 (10): 2451 – 2471.

[15] Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. (2002) “Learning precise timing with LSTM recurrent networks.” The Journal of Machine Learning Research 3(Aug):115-143.

[16]Kyunghyun Cho, Bart van Merrienboer, CaglarGulcehre, DzmitryBahdanau, FethiBougares, Holger Schwenk, and YoshuaBengio. (2014) “Learning phrase representations using RNN encoder-decoder for statistical machine translation.” arXiv preprint arXiv:1406.1078.

[17] François Chollet et al (2015) Keras

[18]Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. (2016) “TensorFlow: A system for large-scale machine learning.” In OSDI, volume 16, pages265-283.

[19] Paul J. Werbos. (1990) “Backpropagation through time: what it does and how to do it.” Proceedings of the IEEE78 (10): 1550-1560. [20] Reza Moazzezi. (2011) “Change-based population coding.” PhD thesis, UCL (University College London )