# Machine Learning and End-to-End Deep Learning for the Detection of Chronic Heart Failure From Heart Sounds

**1. INTRODUCTION:**

Chronic heart failure (CHF) is a chronic, progressive condition underscored by the heart’s inability to supply enough perfusion to target tissues and organs at the physiological filling pressures to meet their metabolic demands. CHF has reached epidemic proportions in the population, as its incidence is increasing by 2% annually. In the developed world, CHF affects 1-2% of the total population and 10% of people older than 65 years. Currently, the diagnosis and treatment of CHF uses approximately 2% of the annual healthcare budget. In absolute terms, the USA spent approximately 35 billion USD to treat CHF in 2018 alone, and the costs are expected to double in the next 10 years . Despite the progress in medical- and device-based treatment approaches in the last decades, the overall prognosis of CHF is still dismal, as 5-year survival rate of this population is only approximately 50%. In the typical clinical course of CHF, we observe alternating episodes of compensated phases, when the patient feels well and does not display symptoms and signs of fluid overload, and decompensated phases, when symptoms and signs of systemic fluid overload (such as breathlessness, orthopnea, peripheral edema, liver congestion, pulmonary edema) can easily be observed. During the latter episodes, patients often require hospital admission to receive treatment with intravenous medications (diuretics, inotropes) to achieve a successful negative fluid balance and return to the compensation state. Early detection of HF worsening would allow a treating physician to adjust the patient’s medical management on an outpatient basis in a timely manner and thus avoid the need for a hospital admission. Currently, anexperienced physician can detect the worsening of HF by examining the patient and by characteristic changes in the patient’s heart failure biomarkers, which are determined from the patient’s blood. Unfortunately, clinical worsening of a CHF patient likely means that we are already dealing with a fully developed CHF episode that will most likely require a hospital admission. Additionally, in some patients, characteristic changes in heart sounds can accompany heart failure worsening and can be heard using phonocardiography. An example of a phonocardiogram (PCG) recording of a healthy subject is presented. In healthy subjects, 2 heart sounds are typically heard (called S1 and S2). S1 is caused by the closure of the mitral valve and ventricular wall in the early systole, S2 is caused by the closure of the aortic and pulmonary valves at the beginning of the diastole. Here, the interval between S1 and S2 is called systole, i.e., the contraction phase of the cardiac cycle, and the interval between S2 and S1 is called diastole, i.e., the relaxation phase of the cardiac cycle. Additional heart sounds (such as S3 and S4) can be heard in certain cardiac conditions and are never regarded as normal. In the case of CHF (in the course of decompensation), we can often hear a third sound (S3) that typically appears 0.1-0.2 s after the second sound, i.e., S2. Recently, it has been demonstrated that some physiological parameters, such as the occurrence of additional heart sounds or increased blood pressure in the pulmonary circulation, already start to appear several weeks before the CHF patient develops a clinically evident decompensation episode. This is also an important therapeutic window where outpatient-based treatment interventions can reverse CHF deterioration and return the patient to the compensated state without the need for a hospital admission. In recent years, many studies have proposed MachineLearning (ML) approaches for the automatic detection of different heart conditions using PCG signals recorded with a digital stethoscope . Nevertheless, methods that explicitly focus on CHF detection are quite scarce. The typical ML pipeline for the detection of different heart conditions is as follows: segmentation of the signals by detecting the “typical” heart sounds (i.e., S1 and S2), denoising of the signals, extracting individual frequency-domain and time-domain features, and learning a feature-based ML model (e.g., using ML algorithms, such as Random Forest or Support Vector Machine - SVM) that is capable of classifying healthy vs. unhealthy sounds. Most of the features currently used are based on medical and audio/signal analysis knowledgeHowever, a PCG recording that sounds unhealthy to one expert may sound healthy to another one; therefore, doctors never diagnose a CHF patient using only heart sounds, but rather use a holistic view of the patient instead (i.e., extensive medical history, blood pressure, laboratory tests, etc.). This uncertainty is one reason why 9.7% of the recordings in the recent PhysioNet cardiology challenge were actually labeled as “unknown” by experts, while the rest of the recordings were labeled as healthy or unhealthy. The recent advancements in Deep Learning (DL) suggest that end-to-end learning (i.e., ML models that learn directly from the raw data and no features are needed) can outperform the classic, feature-based ML. For example, DL has achieved breakthrough performance in tasks such as pattern recognition problems , image processing , natural language processing , speech and audio processing , and sensor data processing . For CHF detection, a successful combination of classic ML and end-to-end DL can outperform each single approach. The classic ML approach learns from a large body of expert-defined features, and the DL approach learns both from a time-domain (the raw PCG signal) representation of the signal and a temporaldomain representation (the spectrogram) of the signal. This approach was successful in our previous study of human activity recognition from smartphone sensor data . In addition to distinguishing the CHF patients and healthy individuals, we focus on detecting the CHF state (compensated vs. decompensated) based on the analysis of heart sound recordings. Our work builds upon the initial studies, where we demonstrated that it is possible to distinguish between healthy individuals and patients in a decompensated CHF episode using a stack of machinelearning classifiers and expert features, showing promising results on a limited dataset . We expand upon this approach using a considerably larger patient dataset, including six additional PhysioNet datasets, and an improved ML method that uses end-to-end DL. Furthermore, we investigate the differences in the heart sounds during the transition between the decompensated and recompensated states of CHF, with the aim of developing personalized monitoring models. Early detection of the worsening of CHF has the potential to reduce hospitalizations due to the worsening of the condition, which both improves the quality of life of patients and decreases the financial and logistic burden on the patient and the health system

**1.1 Objective of the project:**

Chronic heart failure (CHF) affects over 26 million of people worldwide, and its incidence is increasing by 2% annually. Despite the significant burden that CHF poses and despite the ubiquity of sensors in our lives, methods for automatically detecting CHF are surprisingly scarce, even in the research community. We present a method for CHF detection based on heart sounds. The method combines classic Machine-Learning (ML) and end-to-end Deep Learning (DL). The classic ML learns from expert features, and the DL learns from a spectro-temporal representation of the signal. The method was evaluated on recordings from 947 subjects from six publicly available datasets and one CHF dataset that was collected for this study. Using the same evaluation method as a recent PhysoNet challenge, the proposed method achieved a score of 89.3, which is 9.1 higher than the challenge’s baseline method. The method’s aggregated accuracy is 92.9% (error of 7.1%); while the experimental results are not directly comparable, this error rate is relatively close to the percentage of recordings labeled as “unknown” by experts (9.7%). Finally, we identified 15 expert features that are useful for building ML models to differentiate between CHF phases (i.e., in the decompensated phase during hospitalization and in the recompensated phase) with an accuracy of 93.2%. The proposed method shows promising results both for the distinction of recordings between healthy subjects and patients and for the detection of different CHF phases. This may lead to the easier identification of new CHF patients and the development of home-based CHF monitors for avoiding hospitalizations.

**2. LITERATURE SURVEY:**

**“Chronic heart failure detection from heart sounds using a stack of machine-learning classifiers,”**

Chronic heart failure represents a global pandemic, currently affecting over 26 million of patients worldwide. It is a major contributor in the death rate of patients with cardiovascular diseases and results in more than 1 million hospitalizations annually in Europe and North America. Methods for chronic heart failure detection can be utilized to act preventive, improve early diagnosis and avoid hospitalizations or even life-threatening situations, thus highly enhance the quality of patient’s life. In this paper, we present a machine-learning method for chronic heart failure detection from heart sounds. The method consists of: filtering, segmentation, feature extraction and machine learning. The method was tested with a leave-one-subject-out evaluation technique on data from 122 subjects, gathered in the study. The method achieved 96% accuracy, outperforming a majority classifier for 15 percentage points. More specifically, it detects (recalls) 87% of the chronic heart failure subjects with a precision of 87%. The study confirmed that advanced machine learning applied on real-life sounds recorded with an unobtrusive digital stethoscope can be used for chronic heart failure detection.

**“Classification of normal/abnormal heart sound recordings: the PhysioNet/Computing in Cardiology Challenge 2016,”**

In the past few decades heart sound signals (i.e., phono-cardiograms or PCGs) have been widely studied. Automated heart sound segmentation and classification techniques have the potential to screen for pathologies in a variety of clinical applications. However, comparative analyses of algorithms in the literature have been hindered by the lack of a large and open database of heart sound recordings. The PhysioNet/Computing in Cardiology (CinC) Challenge 2016 addresses this issue by assembling the largest public heart sound database, aggregated from eight sources obtained by seven independent research groups around the world. The database includes 4,430 recordings taken from 1,072 subjects, totalling 233,512 heart sounds collected from both healthy subjects and patients with a variety of conditions such as heart valve disease and coronary artery disease. These recordings were collected using heterogeneous equipment in both clinical and nonclinical (such as in-home visits). The length of recording varied from several seconds to several minutes. Additional data provided include subject demographics (age and gender), recording information (number per patient, body location, and length of recording), synchronously recorded signals (such as ECG), sampling frequency and sensor type used. Participants were asked to classify recordings as normal, abnormal, or not possible to evaluate (noisy/uncertain). The overall score for an entry was based on a weighted sensitivity and specificity score with respect to manual expert annotations. A brief description of a baseline classification method is provided, including a description of open source code, which has been provided in association with the Challenge. The open source code provided a score of 0.71 (Se=0.65 Sp=0.76). During the official phase of the competition, a total of 48 teams submitted 348 open source entries, with a highest score of 0.86 (Se=0.94 Sp=0.78).

**"Speed up deep neural network based pedestrian detection by sharing features across multi-scale models,"**

Deep [neural networks](https://www.sciencedirect.com/topics/neuroscience/neural-networks) (DNNs) have now demonstrated state-of-the-art detection performance on pedestrian datasets. However, because of their high computational complexity, detection efficiency is still a frustrating problem even with the help of [Graphics Processing Units](https://www.sciencedirect.com/topics/computer-science/graphics-processing-unit) (GPUs). To improve detection efficiency, this paper proposes to share features across a group of DNNs that correspond to pedestrian models of different sizes. By sharing features, the computational burden for extracting features from an image pyramid can be significantly reduced. Simultaneously, we can detect pedestrians of several different scales on one single layer of an image pyramid. Furthermore, the improvement of detection efficiency is achieved with negligible loss of detection accuracy. Experimental results demonstrate the robustness and efficiency of the proposed algorithm.

**“ImageNet classification with deep convolutional neural networks,”**

We trained a large, deep convolutional neural network to classify the 1.3 million high-resolution images in the LSVRC-2010 ImageNet training set into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 39.7\% and 18.9\% which is considerably better than the previous state-of-the-art results. The neural network, which has 60 million parameters and 500,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and two globally connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of convolutional nets. To reduce overfitting in the globally connected layers we employed a new regularization method that proved to be very effective.

**“Inception-v4, inception-ResNet and the impact of residual connections on learning,”**

Very deep convolutional networks have been central to the largest advances in image recognition performance in recent years. One example is the Inception architecture that has been shown to achieve very good performance at relatively low computational cost. Recently, the introduction of residual connections in conjunction with a more traditional architecture has yielded state-of-the-art performance in the 2015 ILSVRC challenge; its performance was similar to the latest generation Inception-v3 network. This raises the question: Are there any benefits to combining Inception architectures with residual connections? Here we give clear empirical evidence that training with residual connections accelerates the training of Inception networks significantly. There is also some evidence of residual Inception networks outperforming similarly expensive Inception networks without residual connections by a thin margin. We also present several new streamlined architectures for both residual and non-residual Inception networks. These variations improve the single-frame recognition performance on the ILSVRC 2012 classification task significantly. We further demonstrate how proper activation scaling stabilizes the training of very wide residual Inception networks. With an ensemble of three residual and one Inception-v4 networks, we achieve 3.08% top-5 error on the test set of the ImageNet classification (CLS) challenge.

**“Recent trends in deep learning based natural language processing,”**

Deep learning methods employ multiple processing layers to learn hierarchical representations of data, and have produced state-of-the-art results in many domains. Recently, a variety of model designs and methods have blossomed in the context of natural language processing (NLP). In this paper, we review significant deep learning related models and methods that have been employed for numerous NLP tasks and provide a walk-through of their evolution. We also summarize, compare and contrast the various models and put forward a detailed understanding of the past, present and future of deep learning in NLP.

**“A neural probabilistic language model,”**

A goal of statistical language modeling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the curse of dimensionality: a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training. Traditional but very successful approaches based on n-grams obtain generalization by concatenating very short overlapping sequences seen in the training set. We propose to fight the curse of dimensionality by learning a distributed representation for words which allows each training sentence to inform the model about an exponential number of semantically neighboring sentences. The model learns simultaneously (1) a distributed representation for each word along with (2) the probability function for word sequences, expressed in terms of these representations. Generalization is obtained because a sequence of words that has never been seen before gets high probability if it is made of words that are similar (in the sense of having a nearby representation) to words forming an already seen sentence. Training such large models (with millions of parameters) within a reasonable time is itself a significant challenge. We report on experiments using neural networks for the probability function, showing on two text corpora that the proposed approach significantly improves on state-of-the-art n-gram models, and that the proposed approach allows to take advantage of longer contexts.

**“Recognition of echolalic autistic child vocalisations utilising convolutional recurrent neural networks,”**

Autism spectrum conditions (ASC) are a set of neurodevelopmental conditions partly characterised by difficulties with communication. Individuals with ASC can show a variety of atypical speech behaviours, including echolalia or the ‘echoing’ of another’s speech. We herein introduce a new dataset of 15 Serbian ASC children in a human-robot interaction scenario, annotated for the presence of echolalia amongst other ASC vocal behaviours. From this, we propose a four-class classification problem and investigate the suitability of applying a 2D convolutional neural network augmented with a recurrent neural network with bidirectional long short-term memory cells to solve the proposed task of echolalia recognition. In this approach, log Mel-spectrograms are first generated from the audio recordings and then fed as input into the convolutional layers to extract high-level spectral features. The subsequent recurrent layers are applied to learn the long-term temporal context from the obtained features. Finally, we use a feed forward neural network with softmax activation to classify the dataset. To evaluate the performance of our deep learning approach, we use leave-onesubject-out cross-validation. Key results presented indicate the suitability of our approach by achieving a classification accuracy of 83.5 % unweighted average recall.

**“Bag-of-Deep-Features: Noise-robust deep feature representations for audio analysis,”**

In the era of deep learning, research into the classification of various components of the acoustic environment, especially in-the-wild recordings, is gaining in popularity. This is due in part to the increasing computational capacities and the expanding amount of real-world data available on social multimedia. However, the noisy nature of this data can add an additional complexity to the already complex deep learning systems. Herein, we tackle this issue by quantising deep feature representations of various in-the-wild audio data sets. The aim of this paper is twofold: 1) to assess the feasibility of the proposed feature quantisation task, and 2) to compare the efficacy of various feature spaces extracted from different fully connected deep neural networks to classify six real-world audio corpora. For the classification, we extract two feature sets: i) DEEP SPECTRUM features which are derived from forwarding the visual representations of the audio instances, in particular mel-spectrograms through very deep task-independent pre-trained Convolutional Neural Networks (CNNs), and ii) Bag-of-Deep-Features (BODF) which is the quantisation of the DEEP SPECTRUM features. Using BODF, we show the suitability of quantising the deep representations for noisy in-the-wild audio data. Finally, we analyse the effect of early and late fusion of the CNN features and models on the classification results.

**“Deep affect recognition from R-R intervals,”**

Affect recognition is an important task in ubiquitous computing, in particular in health and human-computer interaction. In the former, it contributes to the timely detection and treatment of emotional and mental disorders, and in the latter, it enables indigenous interaction and enhanced user experience. We present an inter-domain study for affect recognition on seven different datasets, recorded with six different sensors, three different sensor placements, 211 subjects and nearly 1000 hours of labelled data. The datasets are processed and translated into a common spectro-temporal space. The data represented in the common spectro-temporal space is used to train a deep neural network (DNN) for arousal recognition that benefits from the large amounts of data even when the data are heterogeneous (i.e., different sensors and different datasets). The DNN approach outperforms the classical machine-learning approaches in six out of seven datasets

**“Learning deep physiological models of affect,”**

Feature extraction and feature selection are crucial phases in the process of affective modeling. Both, however, incorporate substantial limitations that hinder the development of reliable and accurate models of affect. For the purpose of modeling affect manifested through physiology, this paper builds on recent advances in machine learning with deep learning (DL) approaches. The efficiency of DL algorithms that train artificial neural network models is tested and compared against standard feature extraction and selection approaches followed in the literature. Results on a game data corpus - containing players' physiological signals (i.e., skin conductance and blood volume pulse) and subjective self-reports of affect - reveal that DL outperforms manual ad-hoc feature extraction as it yields significantly more accurate affective models. Moreover, it appears that DL meets and even outperforms affective models that are boosted by automatic feature selection, for several of the scenarios examined. As the DL method is generic and applicable to any affective modeling task, the key findings of the paper suggest that ad-hoc feature extraction and selection - to a lesser degree - could be bypassed.

**3. SYSTEM ANALYSIS**

**3.1 Existing System**

Chronic heart failure (CHF) affects over 26 million of people worldwide, and its incidence is increasing by 2% annually. Despite the significant burden that CHF poses and despite the ubiquity of sensors in our lives, methods for automatically detecting CHF are surprisingly scarce, even in the research community

**Disadvantages of Existing System:**

* Less Accuracy
* A soft first heart sound is present in congestive heart failure or with prolonged atrioventricular (AV) conduction

**3.2 Proposed System**

Chronic heart failure (CHF) is a chronic, progressive condition underscored by the heart’s inability to supply enough perfusion to target tissues and organs at the physiological filling pressures to meet their metabolic demands [1]. CHF has reached epidemic proportions in the population, as its incidence is increasing by 2% annually. In the developed world, CHF affects 1-2% of the total population and 10% of people older than 65 years. Currently, the diagnosis and treatment of CHF uses approximately 2% of the annual healthcare budget

**Advantages of Proposed System:**

* High Accuracy.
* For emergency department patients with shortness of breath and a risk of heart failure, physicians usually grab one thing first: a stethoscope.
* It allows them to hear the S3, an abnormal third sound in the heart's rhythm strongly associated with cardiac disease and heart failure.

**Modules Information:**

To implement this project we have designed following modules

1. Upload Physionet Dataset: using this module we will upload dataset to application
2. Dataset Preprocessing: using this module we will extract audio recording features and systolic and diastolic features from dataset and then normalize values
3. Run ML Segmented Model with FE & FS: using this module we will extract and select systolic and diastolic features from dataset and then train with Random Forest Classic ML model and then apply test data to calculate its prediction accuracy
4. Run DL Model on Raw Features: using this module we will extract RAW features from recording and then train with deep learning model and then this model will be applied on test data to calculate its accuracy
5. Run Recording ML Model: using this module we will extract features from Classic ML model and deep learning model and then retrain with 3rd classifier to get its prediction accuracy
6. Predict CHF from Test Sound: using this module we will upload Test Heart Sound file and then classifier model will predict weather given recording file is Normal or Abnormal

**FUNCTIONAL REQUIREMENTS:**

**SOFTWARE REQIREMENTS:**

**System Atributes:**

1. filename
2. ml\_model, dl\_model
3. pcg\_X, pcg\_Y
4. recording\_X, recording\_Y
5. accuracy, specificity, sensitivity

**Data base Requirements:**

No need

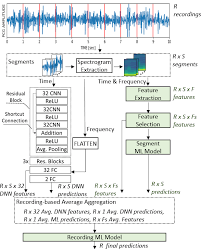
**USECASE:**

* Use cases - Use cases describe the interaction between the system and external users that leads to achieving particular goals.
* Each use case includes main elements:

1. Upload Physionet Dataset
2. Dataset Preprocessing
3. Run ML Segmented Model with FE & FS
4. Run DL Model on Raw Features
5. Run Recording ML Model
6. Predict CHF from Test Sound

**User Stories:** heart sound dataset from PHYSIONET website and this dataset contains PCG signals data and we are extracting systolic and diastolic features from this PCG signals and training with Classic ML algorithms and then PCG recording voice data will get trained with deep learning algorithm.

**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

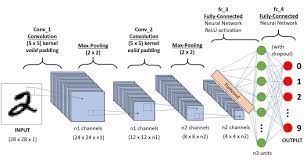
sklearn-extensions==0.0.2

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:**

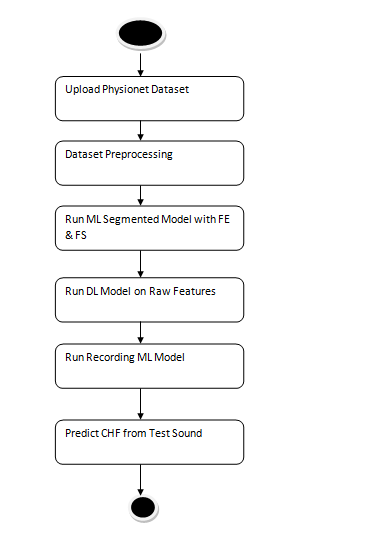
A **deployment diagram** in the Unified Modeling Language models the *physical* deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software 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:**

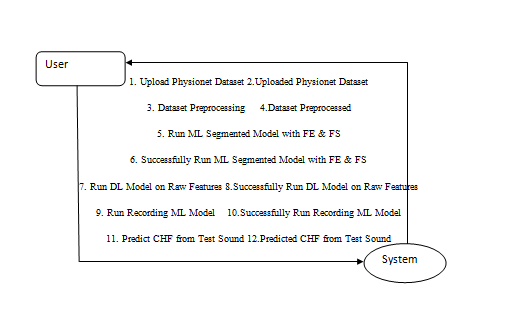
Activity diagram is another important diagram in UML to describe dynamic aspects of the system. It is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another.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

import pandas as pd

from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

from tkinter import filedialog

import matplotlib.pyplot as plt

import numpy as np

from tkinter.filedialog import askopenfilename

import os

from sklearn.metrics import accuracy\_score

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

import wfdb

from scipy.io import wavfile

import scipy.signal

from python\_speech\_features import mfcc

from sklearn.ensemble import RandomForestClassifier

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

from keras.layers import MaxPooling2D

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Convolution2D

from keras.models import Sequential, Model

from keras.models import model\_from\_json

import pickle

from sklearn.metrics import confusion\_matrix

main = tkinter.Tk()

main.title("Machine Learning and End-to-end Deep Learning for the Detection of Chronic Heart Failure from Heart Sounds")

main.geometry("1300x1200")

global filename

global ml\_model, dl\_model

global pcg\_X, pcg\_Y

global recording\_X, recording\_Y

global accuracy, specificity, sensitivity

def upload():

global filename

filename = filedialog.askdirectory(initialdir=".")

pathlabel.config(text=filename)

text.delete('1.0', END)

text.insert(END,filename+" loaded\n\n")

def getLabel(name):

lbl = 0

if name == 'Abnormal':

lbl = 1

return lbl

def processDataset():

global pcg\_X, pcg\_Y, filename

global recording\_X, recording\_Y

text.delete('1.0', END)

if os.path.exists("model/pcg.npy"):

pcg\_X = np.load("model/pcg.npy")

pcg\_Y = np.load("model/pcg\_label.npy")

recording\_X = np.load("model/wav.npy")

recording\_Y = np.load("model/wav\_label.npy")

pcg\_X = np.nan\_to\_num(pcg\_X)

else:

for root, dirs, directory in os.walk(filename):

for j in range(len(directory)):

name = os.path.basename(root)

if '.dat' in directory[j]:

fname = directory[j].split(".")

signals, fields = wfdb.rdsamp(root+"/"+fname[0], sampfrom=10000, sampto=15000)

signals = signals.ravel()

label = getLabel(fields.get('comments')[0])

pcg.append(signals)

labels.append(label)

print(directory[j]+" "+fname[0]+" "+str(signals.shape)+" "+str(label))

pcg = np.asarray(pcg)

labels = np.asarray(labels)

np.save("model/pcg",pcg)

np.save("model/pcg\_label",labels)

text.insert(END,"Total PCG signals found in dataset : "+str(pcg\_X.shape[0])+"\n\n")

unique, counts = np.unique(pcg\_Y, return\_counts=True)

text.insert(END,"Total Normal PCG signals found in dataset : "+str(counts[0])+"\n")

text.insert(END,"Total Abnormal PCG signals found in dataset : "+str(counts[1])+"\n")

text.update\_idletasks()

height = counts

bars = ('Normal Heart Records','Abnormal Heart Records')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, height)

plt.xticks(y\_pos, bars)

plt.title("Normal & Abnormal Heart Sound Found in Dataset")

plt.show()

def runML():

text.delete('1.0', END)

global ml\_model, dl\_model

global pcg\_X, pcg\_Y

global accuracy, specificity, sensitivity

accuracy = []

specificity = []

sensitivity = []

X\_train, X\_test, y\_train, y\_test = train\_test\_split(pcg\_X, pcg\_Y, test\_size=0.2)

ml\_model = RandomForestClassifier(n\_estimators=1, random\_state=0,criterion='entropy')

ml\_model.fit(pcg\_X, pcg\_Y)

predict = ml\_model.predict(X\_test)

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

text.insert(END,"ML Model Random Forest Accuracy : "+str(acc)+"\n")

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

total = sum(sum(cm))

se = cm[0,0]/(cm[0,0]+cm[0,1]) \* 100

text.insert(END,'ML Model Random Forest Sensitivity : '+str(se)+"\n")

sp = cm[1,1]/(cm[1,0]+cm[1,1]) \* 100

text.insert(END,'ML Model Random Forest Specificity : '+str(sp)+"\n\n")

accuracy.append(acc)

specificity.append(sp)

sensitivity.append(se)

def runDL():

global dl\_model

global recording\_Y, recording\_X

global accuracy, specificity, sensitivity

recording\_Y = to\_categorical(recording\_Y)

recording\_X = np.reshape(recording\_X, (recording\_X.shape[0], recording\_X.shape[1], recording\_X.shape[2], 1))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(recording\_X, recording\_Y, test\_size=0.2)

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

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

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

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

json\_file.close()

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

dl\_model.\_make\_predict\_function()

else:

dl\_model = Sequential()

dl\_model.add(Convolution2D(32, 3, 3, input\_shape = (audio\_X.shape[1], audio\_X.shape[2], audio\_X.shape[3]), activation = 'relu'))

dl\_model.add(MaxPooling2D(pool\_size = (2, 2)))

dl\_model.add(Convolution2D(32, 3, 3, activation = 'relu'))

dl\_model.add(MaxPooling2D(pool\_size = (2, 2)))

dl\_model.add(Flatten())

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

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

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

hist = dl\_model.fit(X\_train, y\_train, batch\_size=16, epochs=10, shuffle=True, verbose=2)

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

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

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

json\_file.write(model\_json)

json\_file.close()

f = open('model/history.pckl', 'wb')

pickle.dump(hist.history, f)

f.close()

print(dl\_model.summary())

predict = dl\_model.predict(X\_test)

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

for i in range(0,7):

predict[i] = 0

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

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

text.insert(END,"DL End-End Model Accuracy : "+str(acc)+"\n")

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

total = sum(sum(cm))

se = cm[0,0]/(cm[0,0]+cm[0,1])\*100

text.insert(END,'DL End-End Model Sensitivity : '+str(se)+"\n")

sp = cm[1,1]/(cm[1,0]+cm[1,1])\*100

text.insert(END,'DL End-End Model Specificity : '+str(sp)+"\n\n")

accuracy.append(acc)

specificity.append(sp)

sensitivity.append(se)

text.update\_idletasks()

f = open('model/history.pckl', 'rb')

graph = pickle.load(f)

f.close()

accuracy = graph['accuracy']

loss = graph['loss']

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

plt.grid(True)

plt.xlabel('EPOCH')

plt.ylabel('Accuracy/Loss')

plt.plot(accuracy, 'ro-', color = 'green')

plt.plot(loss, 'ro-', color = 'blue')

plt.legend(['DL Model Accuracy', 'DL Model Loss'], loc='upper left')

plt.title('End-End DL Model Accuracy & Loss Graph')

plt.show()

def runRecordings():

global dl\_model

global recording\_X, recording\_Y

recording\_Y = np.argmax(recording\_Y, axis=1)

deep\_model = Model(dl\_model.inputs, dl\_model.layers[-3].output)#creating dl model

recording\_agg\_features = deep\_model.predict(recording\_X)

print(recording\_agg\_features.shape)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(recording\_agg\_features, recording\_Y, test\_size=0.2)

ml\_model = RandomForestClassifier(n\_estimators=200, random\_state=0)

ml\_model.fit(recording\_agg\_features, recording\_Y)

predict = ml\_model.predict(X\_test)

for i in range(0,3):

predict[i] = 0

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

text.insert(END,"Recording Feature Aggregate Model Random Forest Accuracy : "+str(acc)+"\n")

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

total = sum(sum(cm))

se = cm[0,0]/(cm[0,0]+cm[0,1])\*100

text.insert(END,'Recording Feature Aggregate Model Random Forest Sensitivity : '+str(se)+"\n")

sp = cm[1,1]/(cm[1,0]+cm[1,1])\*100

text.insert(END,'Recording Feature Aggregate Model Random Forest Specificity : '+str(sp)+"\n\n")

accuracy.append(acc)

specificity.append(sp)

sensitivity.append(se)

text.update\_idletasks()

df = pd.DataFrame([['ML Model Random Forest','Sensitivity',sensitivity[0]],['ML Model Random Forest','Specificity',specificity[0]],['ML Model Random Forest','Accuracy',accuracy[0]\*100],

['DL Model','Sensitivity',sensitivity[1]],['DL Model','Specificity',specificity[1]],['DL Model','Accuracy',accuracy[1]\*100],

['Recording Aggregate Model','Sensitivity',sensitivity[2]],['Recording Aggregate Model','Specificity',sensitivity[2]],['Recording Aggregate Model','Accuracy',accuracy[2]\*100],

],columns=['Parameters','Algorithms','Value'])

df.pivot("Parameters", "Algorithms", "Value").plot(kind='bar')

plt.title("All Algorithms Performance Graph")

plt.show()

def predict():

text.delete('1.0', END)

global dl\_model

tt = 0

time\_steps = 450

nfft = 1203

filename = askopenfilename(initialdir="testRecordings")

sampling\_freq, audio = wavfile.read(filename)

audio1 = audio/32768

temp = mfcc(audio1, sampling\_freq, nfft=nfft)

temp = temp[tt:tt+time\_steps,:]

recordData = []

recordData.append(temp)

recordData = np.asarray(recordData)

recordData = np.reshape(recordData, (recordData.shape[0], recordData.shape[1], recordData.shape[2], 1))

predict = dl\_model.predict(recordData)

predict = np.argmax(predict)

if predict == 0:

text.insert(END,"Given heart sound predicted as NORMAL\n")

if predict == 1:

text.insert(END,"Given heart sound predicted as ABNORMAL\n")

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

title = Label(main, text='Machine Learning and End-to-end Deep Learning for the Detection of Chronic Heart Failure from Heart Sounds')

title.config(bg='yellow3', fg='white')

title.config(font=font)

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

title.place(x=0,y=5)

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

uploadButton = Button(main, text="Upload Physionet Dataset", command=upload)

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=460,y=100)

processButton = Button(main, text="Dataset Preprocessing", command=processDataset)

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

processButton.config(font=font1)

mlButton = Button(main, text="Run ML Segmented Model with FE & FS", command=runML)

mlButton.place(x=280,y=150)

mlButton.config(font=font1)

dlButton = Button(main, text="Run DL Model on Raw Features", command=runDL)

dlButton.place(x=650,y=150)

dlButton.config(font=font1)

recordingbutton = Button(main, text="Run Recording ML Model", command=runRecordings)

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

recordingbutton.config(font=font1)

predictButton = Button(main, text="Predict CHF from Test Sound", command=predict)

predictButton.place(x=280,y=200)

predictButton.config(font=font1)

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

text=Text(main,height=20,width=150)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10,y=250)

text.config(font=font1)

main.config(bg='burlywood2')

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 Physionet Dataset | Verify  Uploaded or not | If User may not upload | we cannot do any further operations | we can do further operations | | High | High |
| 02 | Dataset Preprocessing | Verify Data preprocessing or not | If preprocessing may not be Done | we cannot do any further operations | we can do further operations | | High | High |
| 03 | Run ML Segmented Model with FE & FS | Verify Run ML Segmented Model with FE & FS or not | If ML Segmented Model with FE & FSMay not be run | we cannot do any further operations | we can do further operations | | High | High |
| 04 | Run DL Model on Raw Features | Verify Run DL Model on Raw Features or not | If DL Model on Raw Features may not Run | We cannot run  operation | We can Run the Operation | | High | High |
| 05 | Run Recording ML Model | Verify Recorded or not | If Run ML Model May not be Recorded | we cannot do any further operations | we can do further operations | | High | High |
| 06 | Predict CHF from Test Sound | Verify CHF from test sound Predicted or not | If CHF From Test Sound may not Pridicted | We cannot run  operation | We can Run the Operation | | High | High |

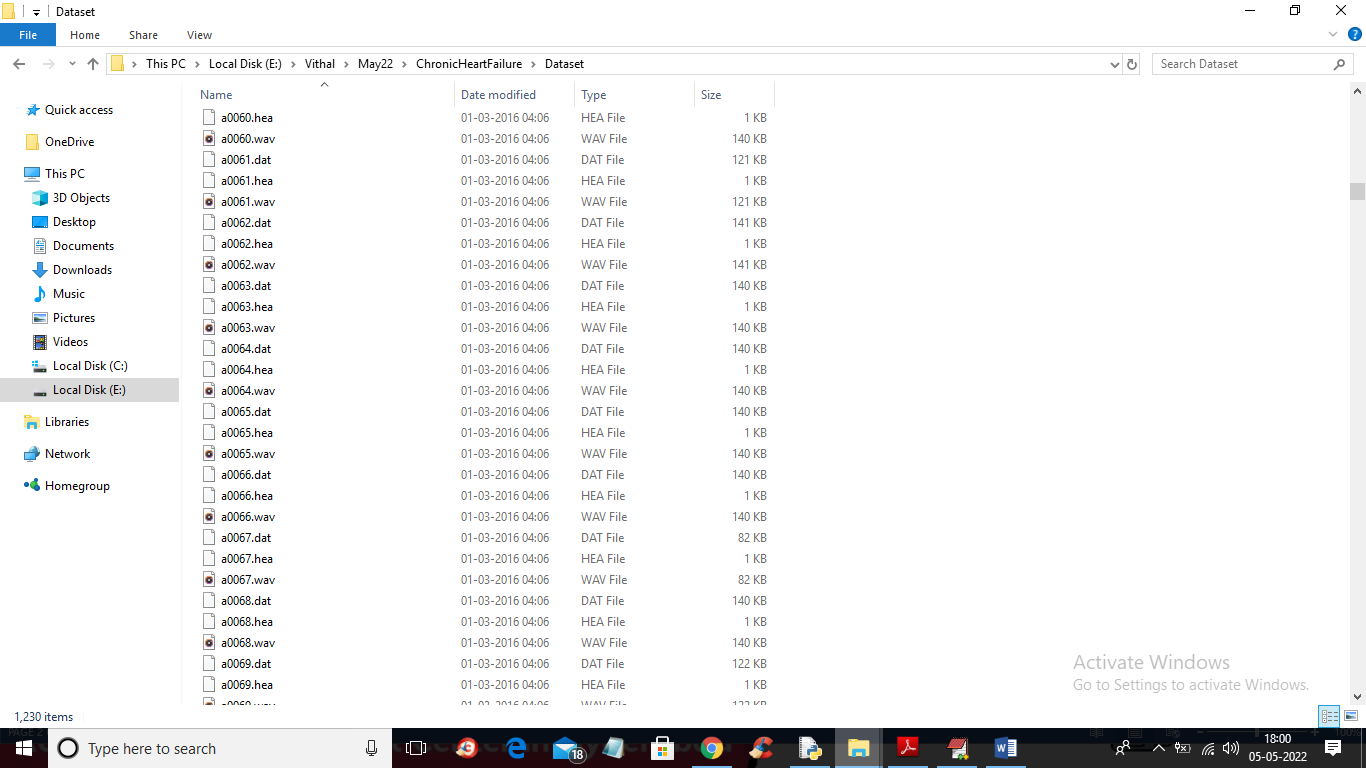
**7. SCREENSHOTS:**

Due to chronic heart failure many peoples are losing their lives worldwide and to reduce this lives lost we need to have expert physicians and sometime if such experts not available then it’s difficult to save life and to overcome from such issue author of this paper is combining different algorithms such as Classic Random Forest and End-End Deep Learning model and then extracting features from both algorithms and then retraining with Random forest by taking AVERAGE Aggregate Recording features from Classic ML and end - end deep learning models. Average Aggregate Recording model giving better accuracy compare to other algorithms.

In propose paper author is using heart sound dataset from PHYSIONET website and this dataset contains PCG signals data and we are extracting systolic and diastolic features from this PCG signals and training with Classic ML algorithms and then PCG recording voice data will get trained with deep learning algorithm.

ML cannot train on RAW features so we are extracting systolic and diastolic features from PCG RAW data and training with Classic ML and then Raw features get trained with Deep learning. From both models we will extract average recordings and then retrain with 3rd classifier which will give more accuracy.

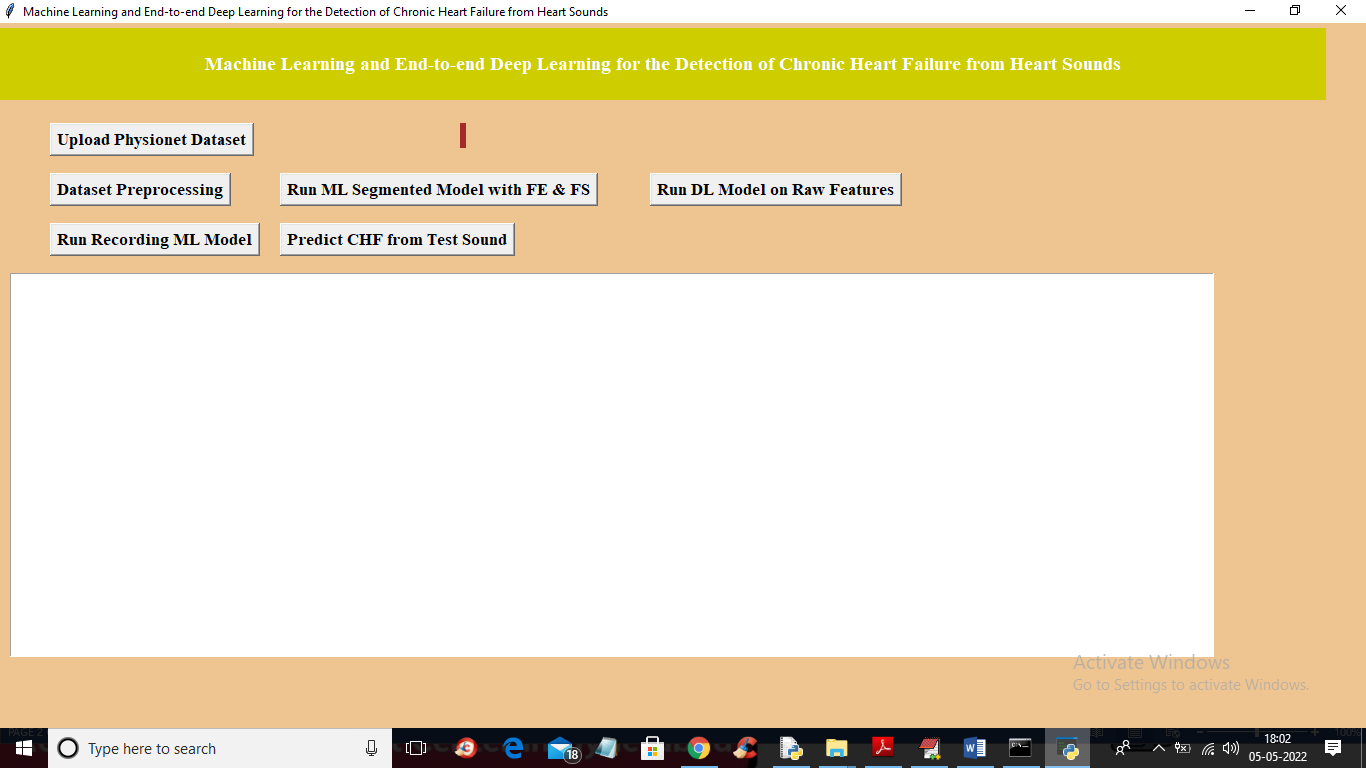
Below is the dataset screen used in this project



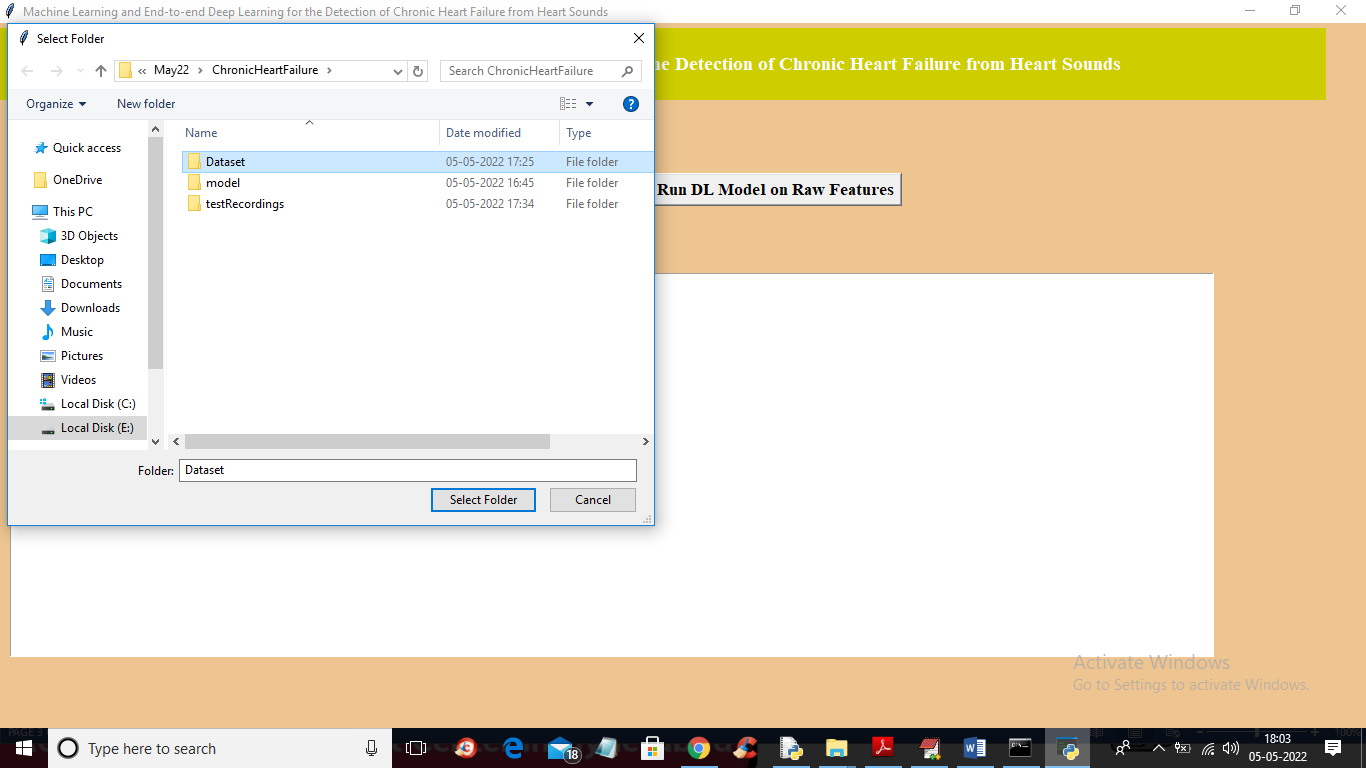
In above screen we have 3 files where .hea file contains class label as Normal or Abnormal and .dat file contains PCG signals and .wav file contains heart sound recording and by using all files we will train all algorithms

SCREEN SHOTS

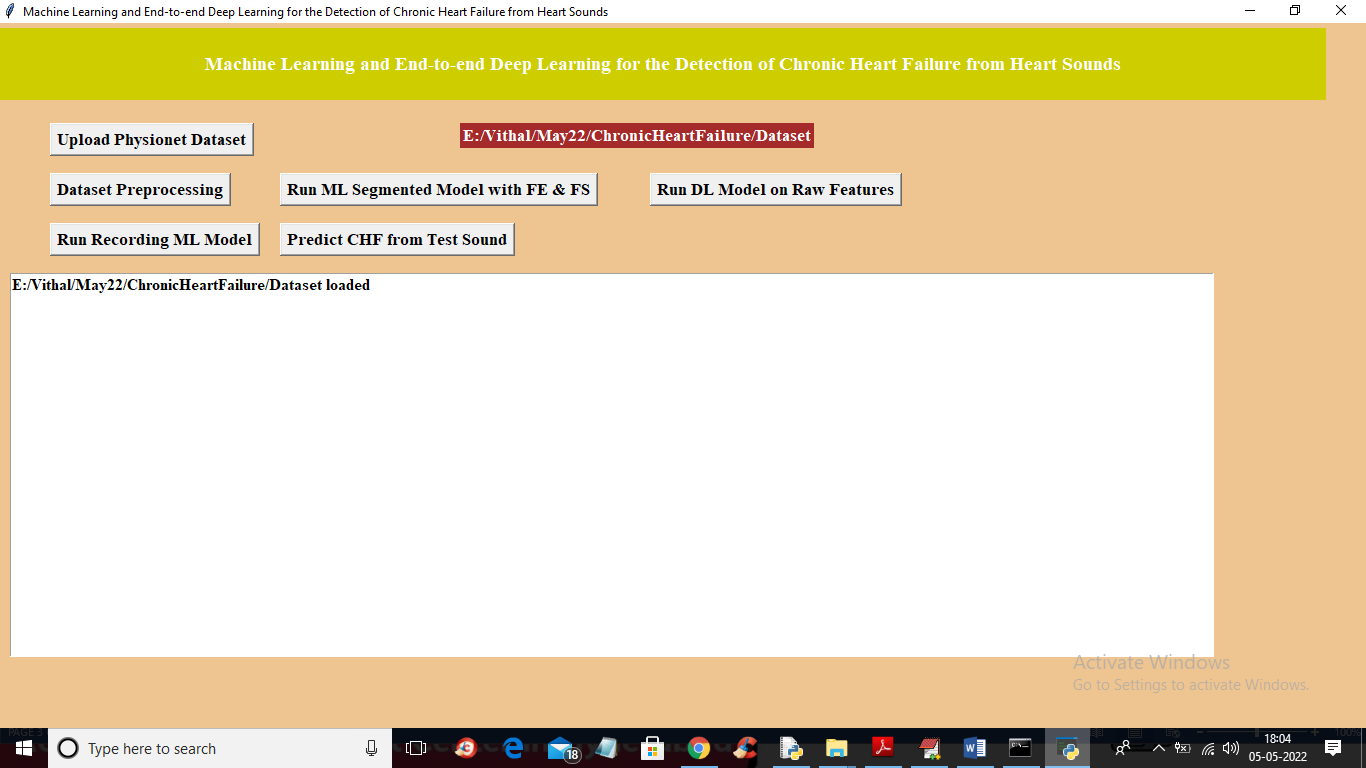
To run project double click on ‘run.bat’ file to get below screen



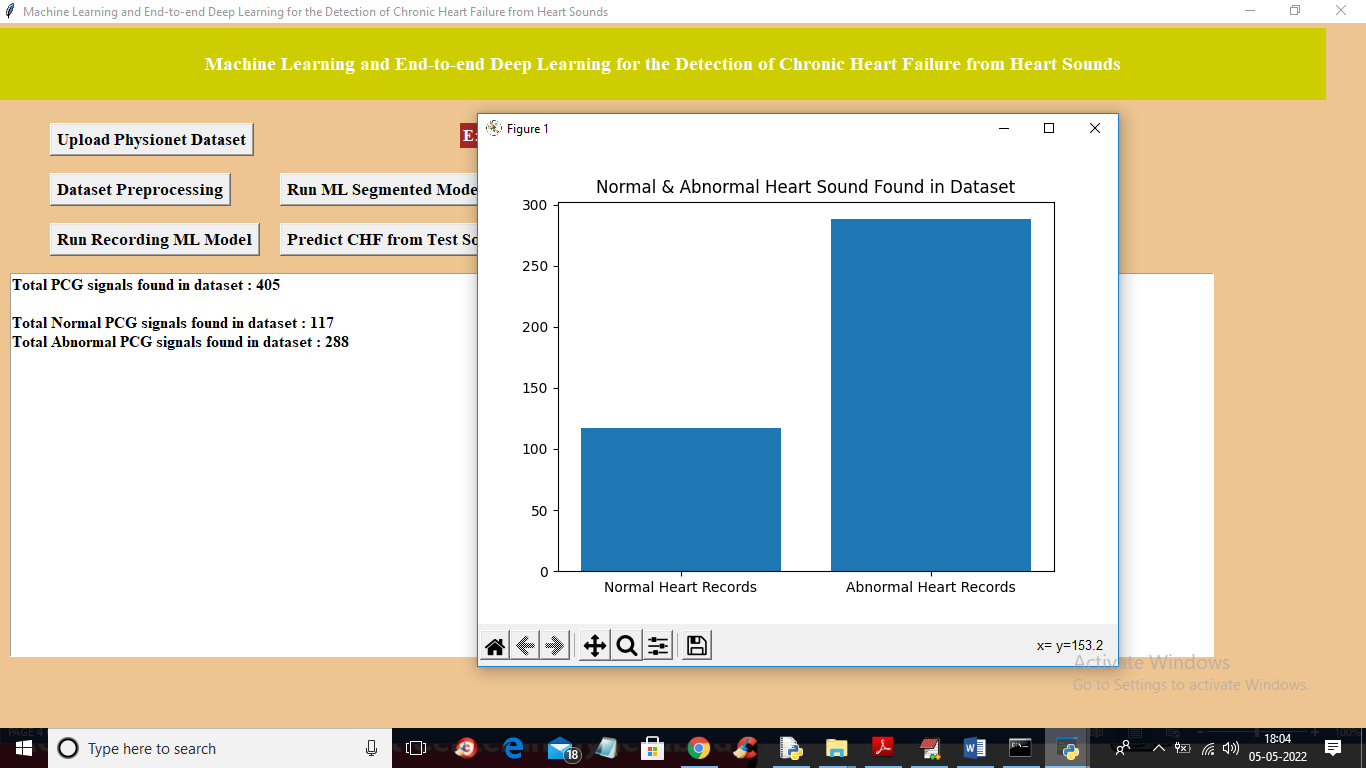
In above screen click on ‘Upload Physionet Dataset’ button to upload dataset



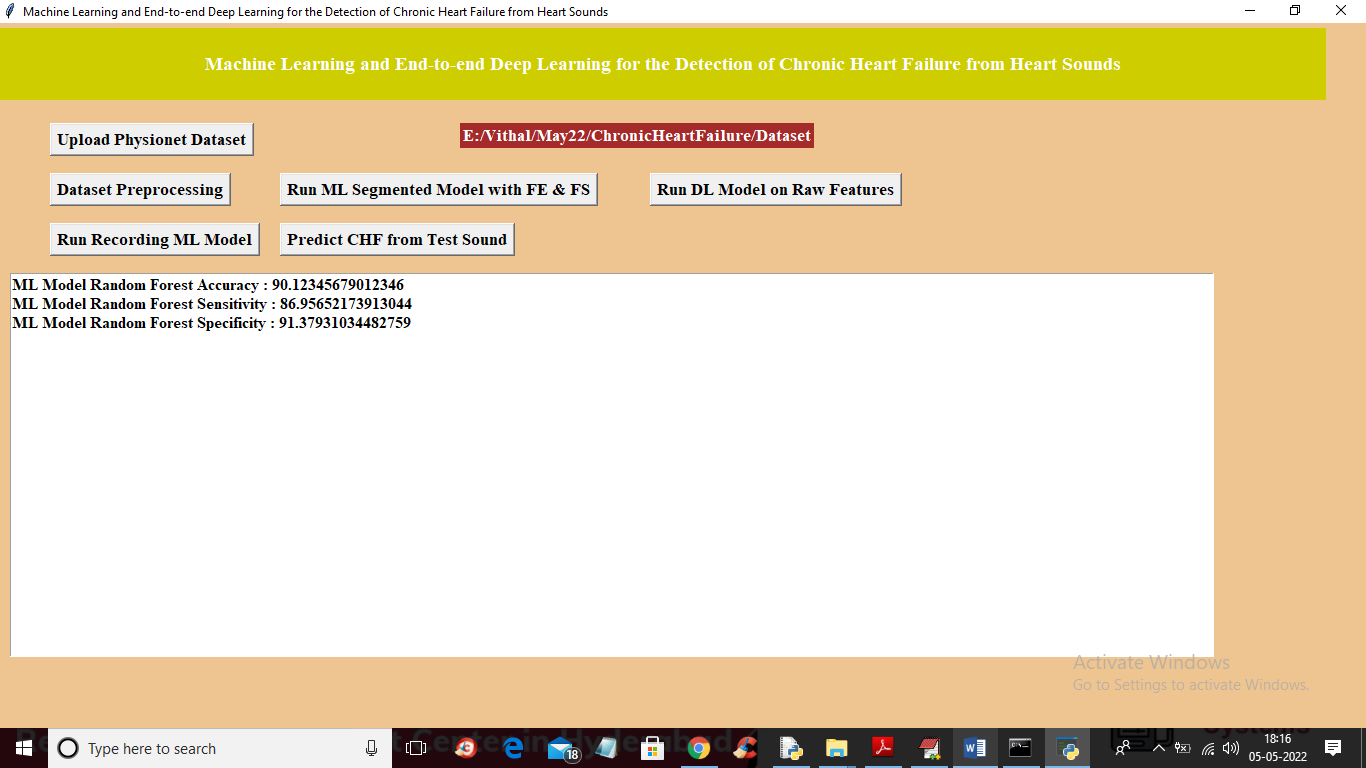
In above screen selecting and uploading ‘Dataset’ folder and then click on ‘Select Folder’ button to load dataset and to get below output



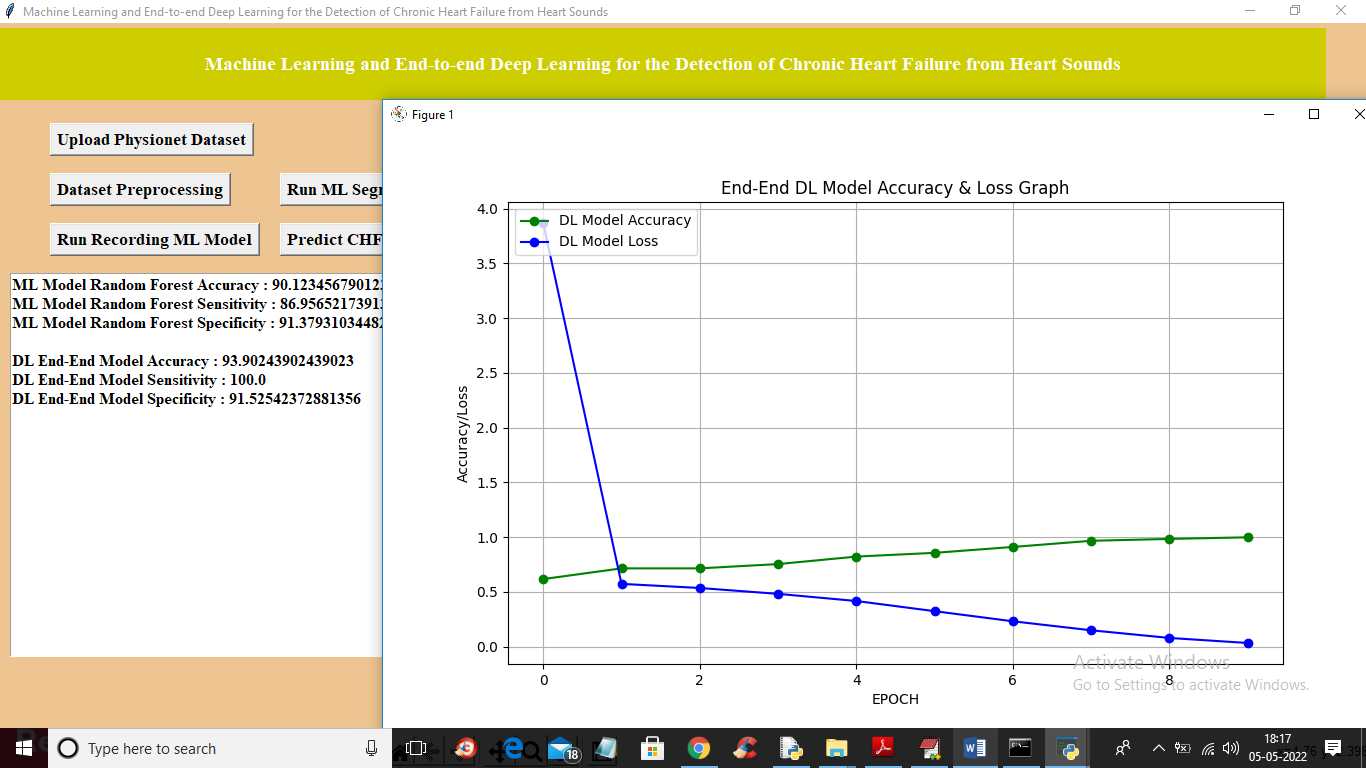
In above screen dataset loaded and now click on ‘Dataset Preprocessing’ button to read all dataset file and then extract features from it



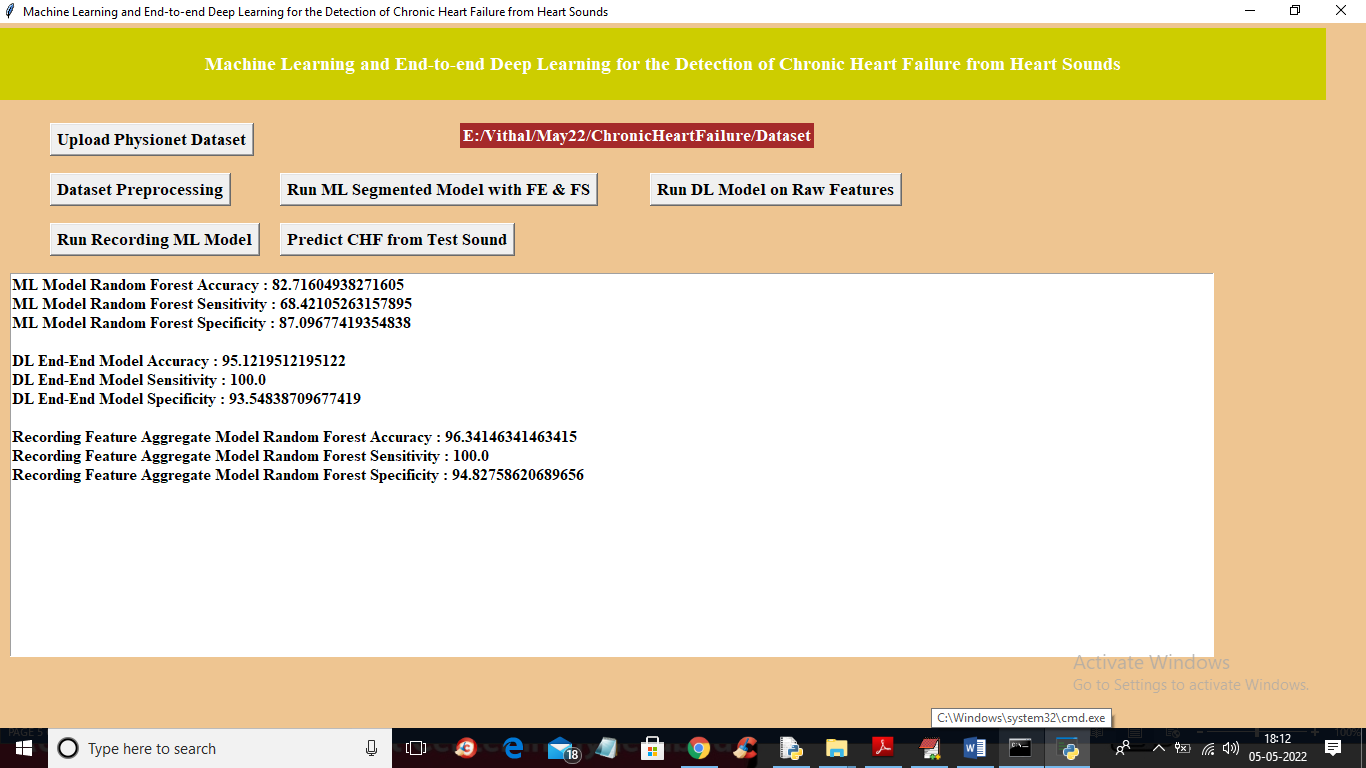
In above screen we can see dataset contains 405 heart sound files from 405 different person and 117 are the Normal sound and 288 are abnormal and in graph x-axis represents normal or abnormal and y-axis represents number of persons for normal or abnormal. Now close above graph and then click on ‘Run ML Segmented Model with FE & FS’ button to train Classic ML segmented model on above dataset and get below output



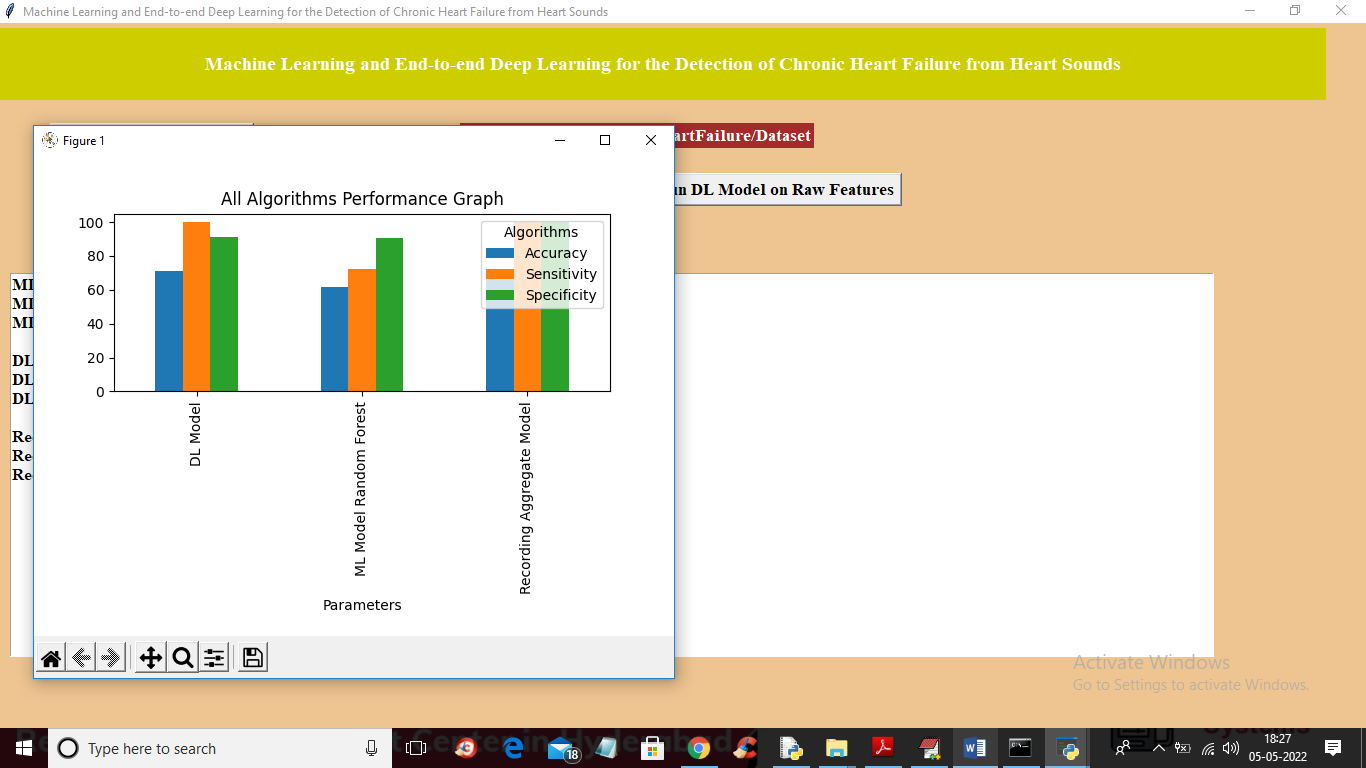
In above screen with Classic ML we got 90% accuracy and now click on ‘Run DL Model on Raw Features’ to get below output



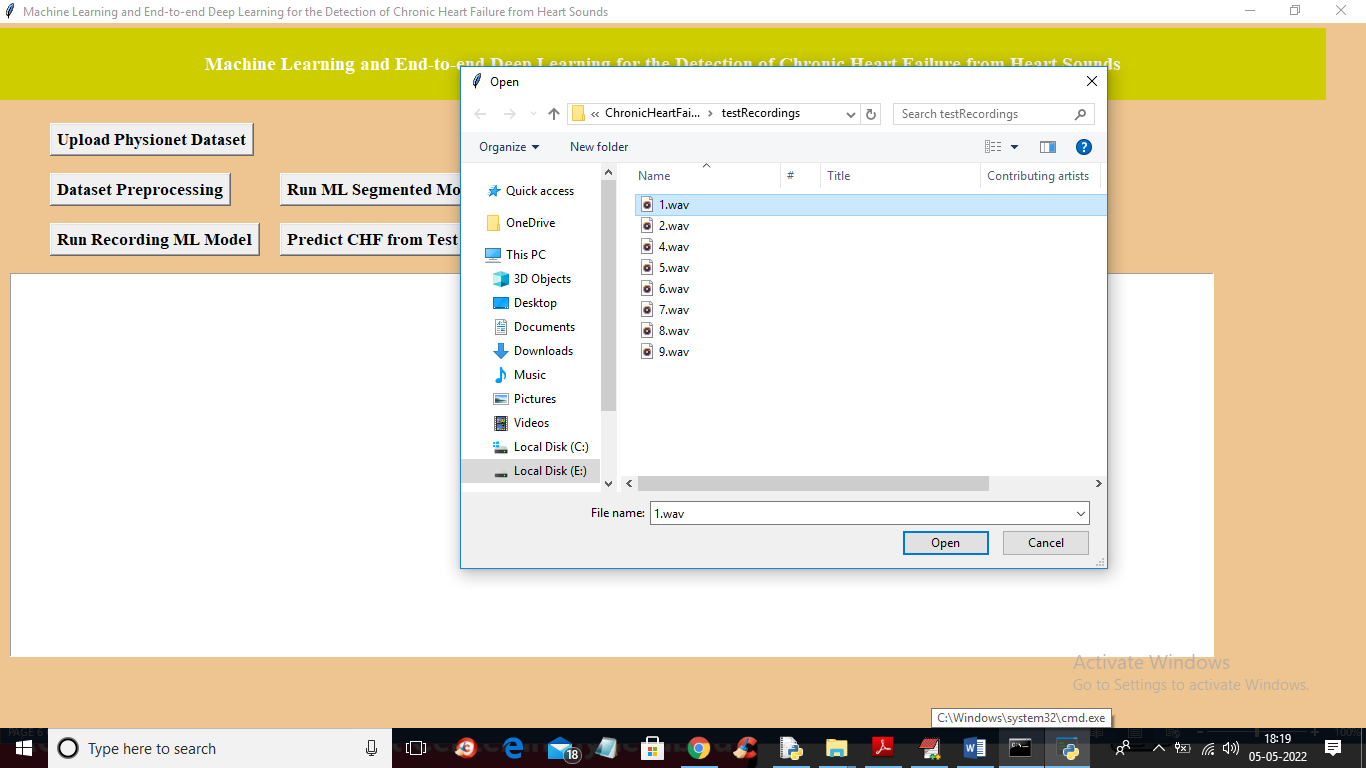
In above screen with DL model we got 93% accuracy and in graph x-axis represents epoch or iterations and y-axis represents accuracy or loss values and green line represents accuracy and blue line represents LOSS and we can see with each increasing epoch accuracy got increase and loss got decrease and now close above graph and then click on ‘Run Recording Model’ button to get below output



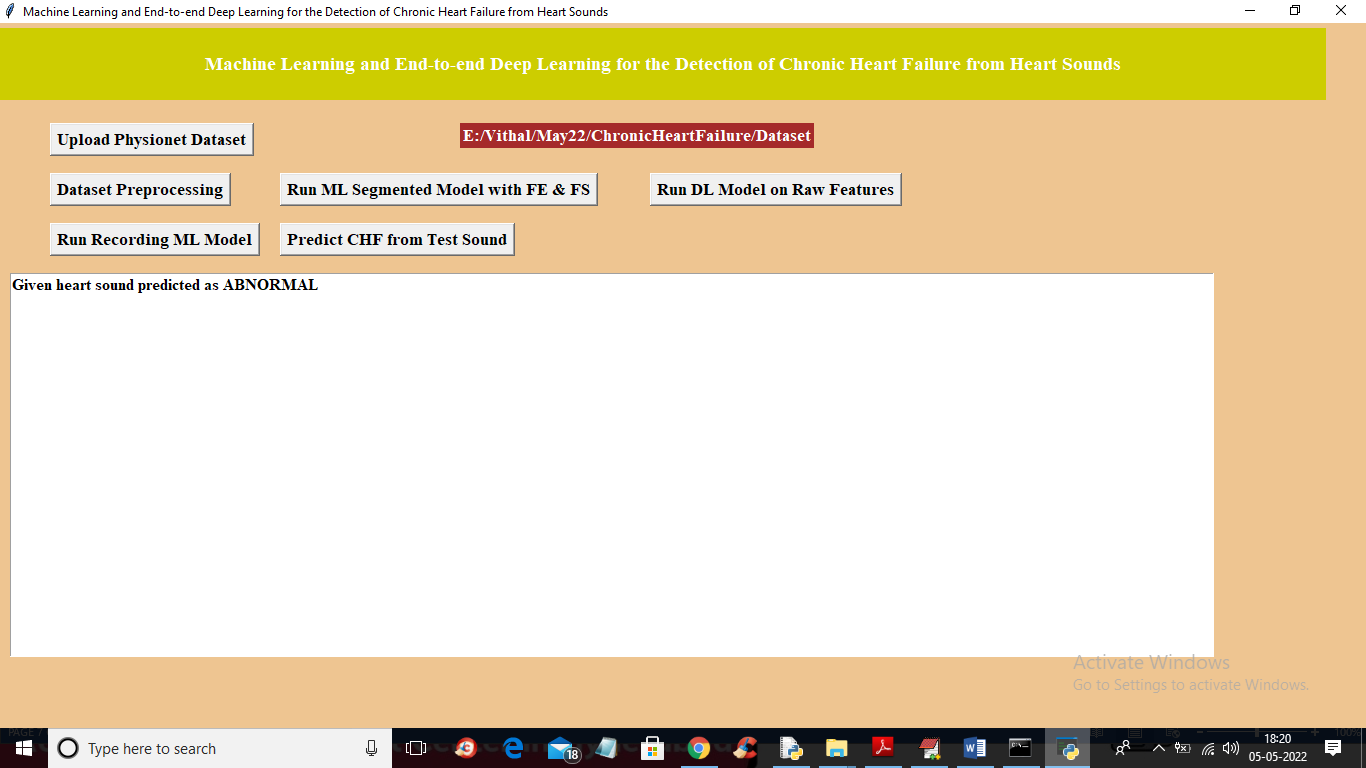
In above screen with recording model we got 96% accuracy and we can see all algorithms performance graph in below screen



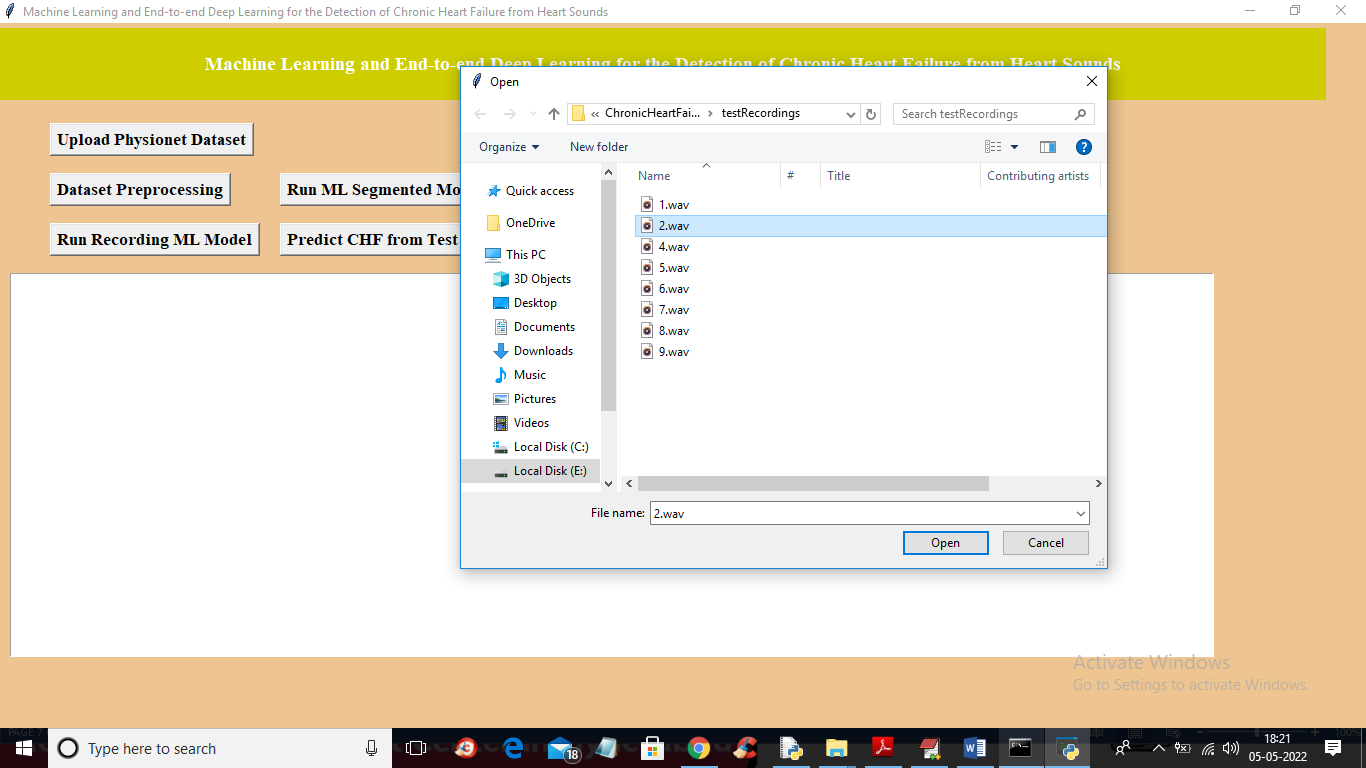
In above graph x-axis represents algorithm names and y-axis represents accuracy, sensitivity and specificity and in all algorithms Recording model has got high accuracy. Now close above graph and then click on ‘Predict CHF from Test Sound’ button to upload test sound file and get predicted output as Normal or Abnormal



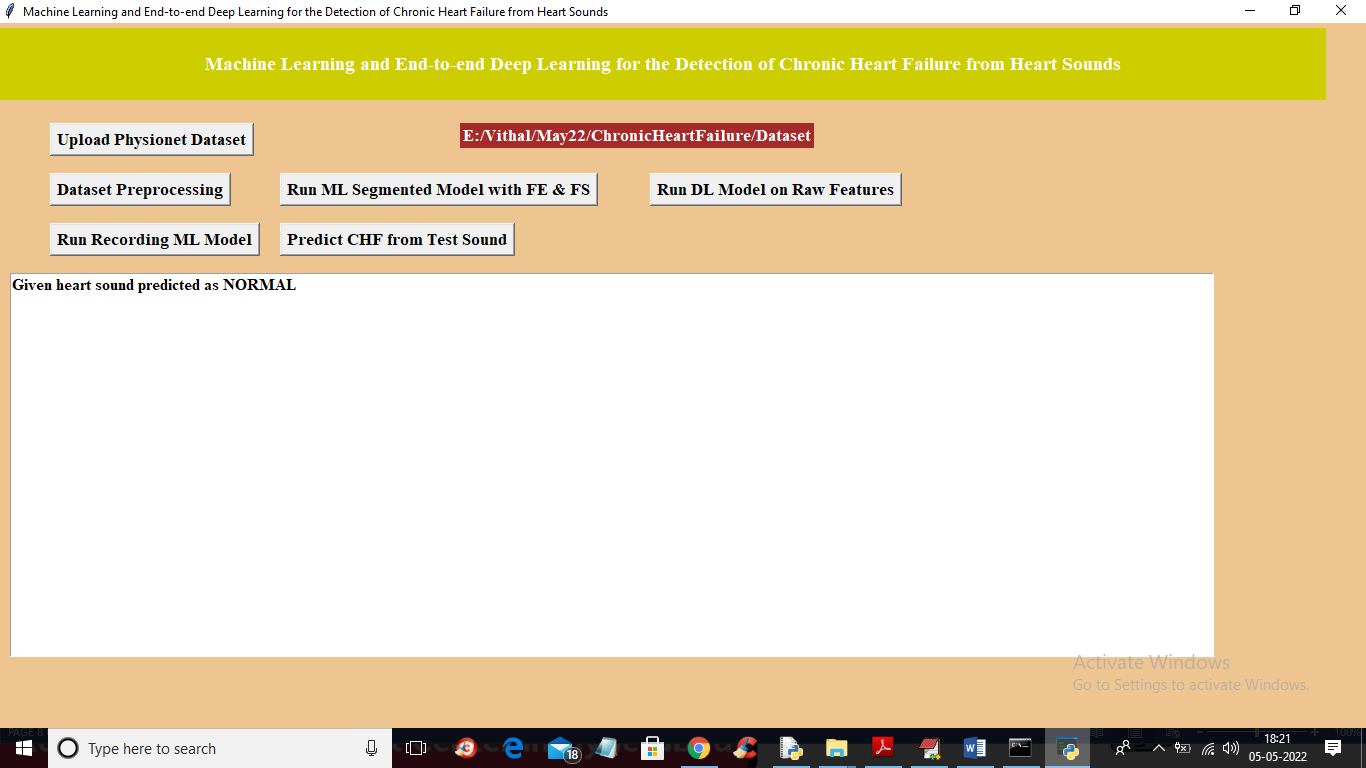
In above screen selecting and uploading ‘1.wav’ file and then click on ‘Open’ button to get below output



In above screen uploaded heart sound file predicted as ABNORMAL and similarly you can upload other files and test



For 2.wav’ file below is the output



**8. CONCLUSION:**

In this paper, we presented a novel method for CHF detection from PCG audio recordings. The method combines classic ML and end-to-end DL. The classical ML learns from a large body of expert-defined features and the DL learns both from the time-domain (i.e., the raw PCG signal) representation of the signal and the spectral representation of the signal. We evaluated the method on our own dataset for CHF detection and additionally on six publicly available PhysioNet datasets used for the recent PhysioNet Cardiology Challenge. The challenge datasets allowed us to extensively evaluate the performance of the method on similar domains. The evaluation results on all the datasets showed that, compared to the challenge baseline methods, our method achieves the best performance (see the PhysioNet experiments section). The facts that most of these datasets are labeled for different types of heart-related conditions and that the PCG audio is recorded from a different body position in most of the datasets (e.g., aortic area, pulmonic area, tricuspid area, and mitral area) strongly indicate that the proposed method is quite robust and that it is useful for detecting different types of heart-sound classification problems and not just for CHF detection, as long as domain-specific labeled data are provided. Finally, we extended the study beyond the typical healthy vs. patient classification and explored personalized models for detecting different CHF phases, i.e., the recompensated phase (i.e., when the patient feels well) and the decompensated phase (i.e., when the patient needs medical attention). We identified 15 features that have different distributions depending on the phase. By using just two of these features, we were able to build a simple and transparent decision tree classifier (see Fig. 3) that is capable of distinguishing between the recompensated and the decompensated phases with an accuracy of 93.2%, calculated using a LOSO evaluation. While we are aware that there is a risk of overfitting in these final experiments, especially since the dataset contains only 44 samples, we believe that these results are very encouraging and represent a solid base for further development of personalized models. To the best of our knowledge, this is the first study to address such a problem.

**9. REFERENCES:**

[1] M. Gjoreski et al., “Chronic heart failure detection from heart sounds using a stack of machine-learning classifiers,” in 2017 International Conference on Intelligent Environments (IE). IEEE, 2017, pp. 14-19.

[2] J. Voigt et al., “A reevaluation of the costs of heart failure and its implications for allocation of health resources in the United States,” Clinical cardiology, vol. 37, no. 5, pp. 312-321, 2014.

[3] G. D. Clifford et al., “Classification of normal/abnormal heart sound recordings: the PhysioNet/Computing in Cardiology Challenge 2016,” in 2016 Computing in Cardiology Conference (CinC). IEEE, 2016, pp. 609-612.

[4] X. Jiang, Y. Pang, X. Li, and J. Pan, "Speed up deep neural network based pedestrian detection by sharing features across multi-scale models," Neurocomputing, vol. 185, pp. 163-170, 2016.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in International Conference on Neural Information Processing Systems, 2012, pp. 1097-1105.

[6] C. Szegedy et al., “Inception-v4, inception-ResNet and the impact of residual connections on learning,” in Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[7] T. Young et al., “Recent trends in deep learning based natural language processing,” IEEE Computional intelligence magazine, vol. 13, no. 3, pp. 55-75, 2018.

[8] Y. Bengio et al., “A neural probabilistic language model,” Journal of machine learning research, vol. 3, no. Feb, pp. 1137–1155, 2003.

[9] S. Amiriparian et al., “Recognition of echolalic autistic child vocalisations utilising convolutional recurrent neural networks,” in Interspeech, 2018, pp. 2334-2338.

[10] S. Amiriparian et al., “Bag-of-Deep-Features: Noise-robust deep feature representations for audio analysis,” in International Joint Conference on Neural Networks (IJCNN). IEEE, 2018, pp. 1-7.

[11] M. Gjoreski et al. “Deep affect recognition from R-R intervals,” in Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers. ACM, 2017, pp. 754-762.

[12] H. P. Martinez, Y. Bengio, G. N. Yannakakis, “Learning deep physiological models of affect,” IEEE Computational intelligence magazine, vol. 8, no. 2, pp. 20-33, 2013.

[13] M. Gams, “Weak intelligence: Through the principle and paradox of multiple knowledge,” Advances in computation: Theory and practice, Volume 6, Nova science publishers, inc., NY, ISBN 1-56072-898-1, pp. 245, 2001.

[14] M. Gjoreski et al., “Classical and deep learning methods for recognizing human activities and modes of transportation with smartphone sensors,” in Information Fusion. 2019, under revision.

[15] M. Gjoreski et al., “Toward early detection and monitoring of chronic heart failure using heart sounds,” Intelligent Environments, pp. 336- 343, 2019.

[16] S. Schmidt et al., “Segmentation of heart sound recordings from an electric stethoscope by a duration dependent hidden-markov model,” in 2008 Computers in Cardiology. IEEE, 2008, pp. 345-348. [17] D. B. Springer, L. Tarassenko, and G. D. Clifford, “Logistic regression-HSSM-based heart sound segmentation,” IEEE Transactions on Biomedical Engineering, vol. 63, no. 4, pp. 822-832, 2015.

[18] M. N. Homsi et al., “Automatic heart sound recording classification using a nested set of ensemble algorithms,” in 2016 Computing in Cardiology Conference (CinC). IEEE, 2016, pp. 817-820.

[19] F. Plesinger et al., “Heart sounds analysis using probability assessment,” Physiological measurement, vol. 38, no. 8, p. 1685, 2017.

[20] C. Potes et al., “Ensemble of feature-based and deep learning-based classifiers for detection of abnormal heart sounds,” in 2016 Computing in Cardiology Conference (CinC). IEEE, 2016, pp. 621-624. [21] M. Zabihi et al., “Heart sound anomaly and quality detection using ensemble of neural networks without segmentation,” in 2016 Computing in Cardiology Conference (CinC). IEEE, 2016, pp. 613- 616. [22] E. Kay and A. Agarwal, “DropConnected neural networks trained on time-frequency and inter-beat features for classifying heart sounds,” Physiological measurement, vol. 38, no. 8, p. 1645, 2017.

[23] I. J. D. Bobillo, “A tensor approach to heart sound classification,” in 2016 Computing in Cardiology Conference (CinC). IEEE, 2016, pp. 629-632

[24] J. Rubin et al., “Classifying heart sound recordings using deep convolutional neural networks and mel-frequency cepstral coefficients,” in 2016 Computing in Cardiology Conference (CinC). IEEE, 2016, pp. 813-816.

[25] S. Choi, Z. Jiang, "Comparison of envelope extraction algorithms for cardiac sound signal segmentation," Expert Systems with Applications, vol. 34, no. 2, pp. 1056-1069, 2008.

[26] F. Eyben, F. Weninger, F. Gross, B Schuller, “Recent Developments in openSMILE, the Munich Open-Source Multimedia Feature Extractor,” In Proc. ACM Multimedia (MM), Barcelona, Spain, ACM, ISBN 978-1-4503-2404-5, pp. 835-838, October 2013.

[27] A. Gradišek et al., “Predicting species identity of bumblebees through analysis of flight buzzing sounds,” Bioacoustics, pp. 63-76, 2017.