A project report on

APPLICATION OF MACHINE LEARNING AND DEEP LEARNING FOR DETECTION OF NON-ALCOHOLIC FATTY LIVER DISEASE (NAFLD)

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

by

AVIREDDY NVSRK ROHAN(19BCE1180)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April, 2023

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I here by declare that the thesis entitled "APPLICATION OF MACHINE LEARNING AND DEEP LEARNING FOR DETECTION OF NON-ALCOHOLIC FATTY LIVER DISEASE (NAFLD) " submitted by me, for the award of the degree of Bachelor of Technology in Computer Science and Engineering, Vellore Institute of Technology, Chennai, is a record of bonafide work carried out by me under the supervision of Guide Name

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: Signature of the Candidate

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **"APPLICATION OF MACHINE LEARNING AND DEEP LEARNING FOR DETECTION OF NON-ALCOHOLIC FATTY LIVER DISEASE (NAFLD)"** is prepared and submitted by **Avireddy NVSRK Rohan(19BCE1180)** to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** programme is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:

Name: Dr./Prof. Geetha . S

Date:

Signature of the Examiner 1	Signature of the Examiner 2
Name:	Name:
Date:	Date:

Approved by the Head of Department
B. Tech. CSE

Name: Dr. Nithyanandam P

Date: 24 – 04 – 2023

(Seal of SCOPE)

ABSTRACT

Electronic Business offers innovative ways of doing business through Internet. Internet revolution has introduced number of advanced technologies to access organizations all over the world in an efficient but simple manner. Internet traffic is increasing exponentially as a result of wireless access, mobile computing and other innovative communication
technologies.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Geetha S, Professor, SCOPE, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Machine Learning and Image Processing.

It is with gratitude that I would like to extend thanks to our honorable Chancellor, Dr. G. Viswanathan, Vice Presidents, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan and Mr. G V Selvam, Assistant Vice-President, Ms. Kadhambari S. Viswanathan, Vice-Chancellor, Dr. Rambabu Kodali, Pro-Vice Chancellor, Dr. V. S. Kanchana Bhaaskaran and Additional Registrar, Dr. P.K.Manoharan for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dean, Dr. Ganesan R, Associate Dean Academics, Dr. Parvathi R and Associate Dean Research, Dr. Geetha S, SCOPE, Vellore Institute of Technology, Chennai, for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Nithyanandam P, Head of the Department, Project Coordinators, Dr. Abdul Quadir Md, Dr. Priyadarshini R and Dr. Padmavathy T V, B. Tech. Computer Science and Engineering, SCOPE, Vellore Institute of Technology, Chennai, for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staff at Vellore Institute of Technology, Chennai, who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: Name of the student

AVIREDDY NVSRK ROHAN

Chapter 1

1. INTRODUCTION

Non-alcoholic Fatty Liver Disease (NAFLD) has become one of the most prevalent causes of chronic liver diseases in the world.The global prevalence of NAFLD is 25.24 % with the highest prevalence in the Middle East and South America and the lowest in Africa[1].A recent systematic review estimated the global prevalence of NAFLD in diabetic patients to be around 58% . A patient is diagnosed with NAFLD when there is a greater percentage of fat in his liver (>5%). Excess alcohol consumption is also one of the factors that can cause fatty liver disease. However, if fatty liver is found in a patient who consumes almost little to no alcohol then the patient may be diagnosed with Non-alcoholic fatty liver disease. NAFLD can range from simple fatty liver (NAFL) and may progress into more serious stages such as NASH and cirrhosis. In some cases, it may also lead to HCC.NAFLD is an insidious disease where the patient may experience little to no symptoms, even in the advanced stages in some cases. Various methods of diagnosis have been developed to detect NAFLD. The surest way of diagnosing NAFLD is a biopsy. However, given the procedure's invasiveness and associated complications during and after the procedure such as infection, bleeding, etc, it is not advisable to use this method for a larger population. Hence, medical professionals actively worked to detect it through non-invasive tests such as b-mode ultrasound scanning, elastography, complete blood panel, and so on. However, factors such as requiring a trained medical professional to assess these tests and testing errors that can arise as a result of human involvement such as high inter-operator and intra-operator variability pose great difficulty to come up with an accurate diagnosis for this disease. The rise of AI in medicine has shown significant promise in the identification, diagnosis, and staging of patients with NAFLD[2]. Specifically, ML and DL algorithms are not only capable detect NAFLD but also assess its severity with laudable accuracy. Our study aims to find ways to efficiently apply ML and DL in two specific areas that we believe have great potential for further experimentation. We present our novel approaches using two types of datasets.One predominantly involving numerical indicators of various compounds that are obtained from blood and uring tests and the other involving b-mode ultrasound image data.. Further sections discuss previous work in this field and provide detailed explanations about how we engineered the data sets appropriately for our studies, the algorithms we tested, and their performances.

Chapter 2

2.RELATED WORK

The following subsections present the review of literature done as a part of this study.

2.1.LITERATURE ABOUT ML USING LABORATORY DATA

Previous works that employed machine learning have focused on developing automatic classifiers to classify the presence or absence of the disease[citing required]. Some studies proposed methods capable of assessing the severity of the illness [citing required]. The importance of including specific bio-markers such as FIB-4 and FLI was also emphasized in some studies. Studies using laboratory data (values of bio-markers) collected from blood tests experimented predominantly with models such as Logistic regression, SVM, etc [citing required]. Since predicted labels from the laboratory can depend on a large set of bio-markers, researchers often performed correlation analysis and shortened the dependent variable list.[citing required]. It is also important to note that symptoms of excess alcohol consumption may be confounded with those indicating NAFLD which some studies did not take into account while performing their analyses. Data from NHANES studies have often been taken into consideration. Since NHANES is a multi-year cross-sectional study, the data sets used by the researchers kept changing. [3] experimented with different versions of some machine learning models like Decision trees, SVM, KNN, etc. The authors also experimented with ensembling with trees and it is worth noting that they performed better than the other models mentioned in their work. [4] provided an extended study that compared traditional machine learning models, Gradient boosted trees, and deep learning models like CNN and the LSTM. However, it was interesting to note that the importance (weight) given to the features varied across models in some cases, in [3] and also in many other studies, ALT was considered to be an essential feature--but in case of [4]--- more weight was given to the BMI of a person for the XGBoost (extreme gradient boosting) model. Surprisingly, neural network models achieved lower AUROC, specificity, and accuracy than the XGBoost model. So it is interesting to note that tree-based models are capable of outperforming deep learning methods, however, the XGBoost model achieved lower sensitivity. [5] also used LR, SVM, RF, D.T but particularly to find out whether these models are capable of identifying NASH or advanced liver fibrosis using commonly available clinical and biochemical data. They also analyzed feature importance using the SHAP (SHapley

Additive ExPlanations) method[citing required]. Additionally, accepted liver function indexes[citing required] were considered as features for their work. The random forest model once again outperformed other models.

2.2.LITERATURE ABOUT ML ON ULTRASOUND DATA

The utility of a few traditional machine learning models spans beyond just laboratory indicators.

[6] implemented random forest and SVM models on 120 ultrasound images of the liver diagnosed with 3 levels of severity(mild, moderate, severe) whose region of interest was manually mapped [verify]. Features were extracted using multiple image processing techniques, which then were fed to the SVM and the Random Forest. Results indicated that the Random forest without performing feature selection outperformed the SVM after feature selection indicating better generalizability of the Random Forest compared to SVM even without complex preprocessing. [7] employed a genetic algorithm to automatically segment the ROI in ultrasound images and then trained a J48 decision tree and a voting-based classifier on the extracted features from those ROI's . J48 algorithm achieved 93.12% while the voting-based classifier achieved 95.71%. Another study conducted by [8] tested an ANN against redundancy-eliminated features extracted, such as Intensity histogram, GLRLM, and Invariant moments, and achieved more than 90\% accuracy. Some studies also proposed deep learning models that were slightly modified. [9] 's study proposed a 1-D CNN model to classify the radio-frequency ultrasound images assisted with MRI for the presence of NAFLD. The classifier achieved 96% accuracy for NAFLD diagnosis, Sensitivity of 97%; Specificity, 94%; Positive predictive value, 97%; Negative predictive value, of 94%. Many studies tried to circumvent the obstacle of a limited amount of ultrasound data by making use of transfer learning[citing required]. The study conducted by [10] demonstrated the use of a cascaded network to segment the liver-kidney area on US images. Transfer learning was employed for segmenting the liver and kidney (L-K) on parasagittal US images, The second network also segmented the images by checking the presence of a ring that is typically located around the kidney and cropping of the L-K area from the original US images. These cropped L-K areas are provided as input to the SteatosisNet, for grading the severity of the fatty liver disease. It achieved high sensitivity of 99.78% and , a specificity of 100%.

Chapter 3

3.Objectives

- Eliminating the need for manual extraction of ROI from ultrasound images by implementing an automated algorithm
- To compare the efficiency of traditional methods requiring ROI with more sophisticated deep learning
- To train deep learning models on sequential image data computation
- Compare the performance of the transfer learning approach and a cascaded sequential neural network on sequential image data.
- Improve the classification specificity, sensitivity and auroc by implementing a novel ensemble of MLP and RF model on NHANES 2017-18 data.

Chapter 4

4.DATASETS AND PREPROCESSING

This section contains a description of the data sets that have been chosen for the three approaches mentioned in the previous section and also involves the study of the performance of machine learning and deep learning algorithms. Fibroscan elastography and laboratory data were chosen for studying the application of machine learning and deep learning algorithms for the approach involving laboratory data along with liver stiffness and Controlled attenuation parameter that were collected during the NHANES 2017-18 study. B-mode ultrasound images from the Department of Internal Medicine, Hypertension and Vascular Diseases, Medical University of Warsaw, Poland for the approach involving the detection of NAFLD from ultrasound images. Information in detail about each dataset is presented in the following subsections.

4.1.NHANES 2017-18 DATA

The National Center for Health Statistics (NCHS), Division of Health and Nutrition Examination Surveys (NHANES), part of the Centers for Disease Control and Prevention (CDC), has conducted a series of health and nutrition surveys since the early 1960s.[]. The survey consists of a cross-sectional interview, examination, and laboratory data collected from a complex multistage, stratified, clustered probability sample representative of the civilian, non-institutionalizedand population with oversampling of non-Hispanic blacks, Hispanics, Asians, low-income persons, and older adults. The survey was approved by the Centers for Disease Control and Prevention ethics review board, and all participants provided written informed consent. To Understand the epidemiology of fatty liver disease in the general population and to develop interventions that will reduce the public health burden, liver fat and stiffness bio-markers were assessed for the first time in NHANES 2017-2018 sample by transient elastography.

4.1.1.CAP AND VCTE

Knowing that fat affects ultrasound propagation, a novel attenuation parameter has been developed to detect and quantify steatosis. This parameter is based on the ultrasonic properties of the radiofrequency back-propagated propagated signals acquired by the Fibroscan. It is called controlled attenuation parameter (CAP) because it was devised to specifically target the liver.It is measured in db/m \cite{}. Stiffness (E) is expressed in kilopascals (kPa) and represents the resistance of the material to deformation. While stiff materials, such as concrete, exhibit low strain even at high stress, soft materials such as biological soft tissues exhibit large strain even at low stress.[11]. Liver stiffness is an excellent bio-marker to identify liver fibrosis, which in turn-- is one of the symptoms of the progression of fatty liver disease.

4.1.2.EXCLUSION CRITERIA OF THE STUDY

The performances shed further define how we filtered out some data samples after acquiring the data from NHANES's website. The exclusion criteria devised here has been improvised from [12].

● Patients who did not go through the examination

● Patients who do not have >=10 reliable measures OR whose IQR/MEDIAN >30 (QUARTILE BASED CV) \cite{}.

- Consumed alcohol is defined as an average daily consumption of ≥20 g/day and ≥30 g/day for women and men respectively
- With serum hepatitis B surface antigen
- With hepatitis C
- If hepatitis B or C was reported
- With human immunodeficiency virus (HIV)
- Is pregnant at the time of examination.

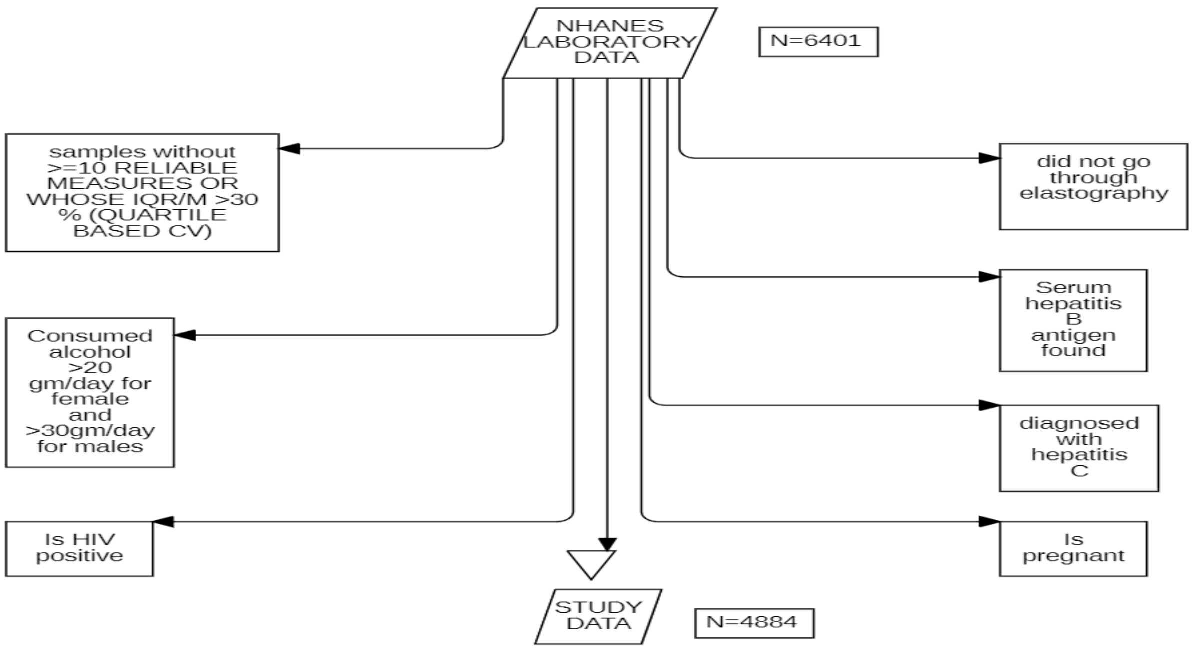

NHANES LABORATORY DATA
N=6401
samples without >=10 RELIABLE MEASURES OR WHOSE IQR/M >30 % (QUARTILE BASED CV)
did not go through elastography
Consumed alcohol >20 gm/day for female and >30gm/day for males
Serum hepatitis B antigen found
diagnosed with hepatitis C
Is HIV positive
Is pregnant
STUDY DATA
N=4884

Fig1. Caption 2b wr10

4.1.3.CUT-OFFS FOR ANNOTATING THE DATA

Condition	**Class label**	**Support**
CAP <302 and VCTE<8.2	Normal	3617
CAP >=302	NAFLD	1267
CAP<302 and VCTE>13.6	Cryptogenic cirrhosis	95
274<=CAP<302 and 8.2<=VCTE<=13.6	Borderline steatosis	57

CAP<274 and 8.2<=VCTE<=13.6	Control	49

Final dataset parameters	Frequency
Dataset size	4884
NAFLD/Normal	3617, 1267
Number of columns	265

4.2.B-MODE ULTRASOUND DATA

Our study included 55 severely obese patients admitted for bariatric surgery (mean age 40.19.1, mean BMI 45.95.6, 20\% males) (laparoscopic sleeve gastrectomy). The ultrasound data were collected at the Medical University of Warsaw's Department of Internal Medicine, Hypertension and Vascular Diseases during a preoperative cardiac echocardiographic evaluation 1-2 days before surgery.

The Medical University of Warsaw's Ethical Committee approved the study, and all patients provided informed consent for echocardiography and abdominal ultrasound examination. During the bariatric surgery, each patient underwent a wedge liver biopsy as part of the routine protocol implemented at the Department of General, Transplant, and Liver Surgery, Medical University of Warsaw, Poland [23]. Tissue was extracted from the left liver lobe's subcapsular region.

A single pathologist performed the histopathological evaluation by the Clinical Research Network's recommendations [24]. The percentage of hepatocytes with fatty infiltration was used to determine the degree of steatosis. A fatty liver was defined as having more than 5\% steatosis in hepatocytes. There were 38 patients diagnosed with fatty liver.

DATASET NAME	**SIZE OF THE DATA**	**TOTAL SAMPLES USED IN THE DATA**	**resolution**	**class distribution**	**Conditions for data coll.**	**Sampling Rate**
B-mode liver ultrasound images	52 MB	55x 10= 550	434×636	38 (NAFLD) 17(NORMAL)	The fatty liver was defined to have more than 5% hepatocytes with steatosis	2.5 Mhz

Chapter 5

5.DESIGN AND IMPLEMENTATION

This section elaborates on the algorithms that were chosen for each of the three approaches discussed earlier.

5.1.DETECTING NAFLD USING ENSEMBLE DEEP LEARNING METHOD

Previous attempts discussed strongly suggest that the Random Forest model performs significantly well over the traditional machine learning models\cite{} and in some cases better than deep learning models\cite{}. However, the advantages of incorporating deep learningplayed a vital role in ing a large feature set cannot be ignored. For instance, the recent work of Deore.et.al \cite{} developed an improved network intrusion detection model to detect malicious internet traffic and presented a novel machine learning method, an ensemble of Multi-layer perceptron networks with Random Forest. The work also compared its proposed method with existing methods and achieved improved Accuracy, precision, and recall. Fundamentally, we believe that Deore. et.al's approach is analogous to our problem i.e training algorithms to classify data samples where the number of features is more. So, keeping its potential in mind, we present our findings after experimenting with an MLP and Random Forest ensemble.

5.1.1.ARCHITECTURE OF ENSEMBLE OF RANDOM FOREST AND MLP

The model is an ensemble of three different machine learning models: Random Forest, a 3-layer Multi-Layer Perceptron (MLP), and a 5-layer MLP. Each model is trained on the NHANES laboratory data to predict the presence or absence of NAFLD in patients.

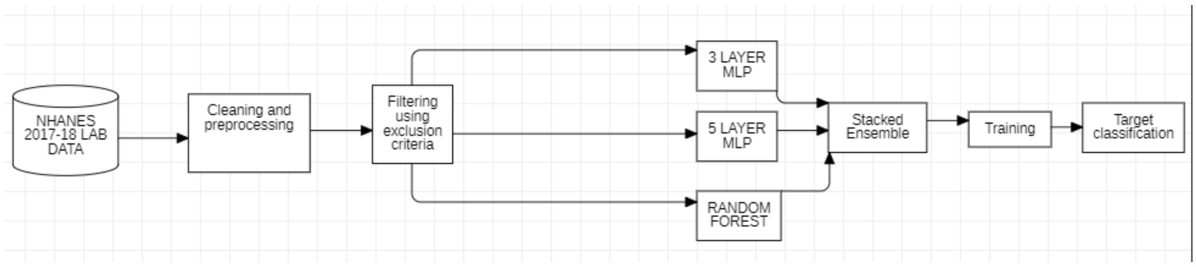
The stacked ensemble approach combines the strengths of each individual model to improve overall prediction performance. The predictions from each model are fed as input to a meta-learner, which learns how to optimally combine the predictions to make a final prediction.

Random forest is a powerful algorithm that can handle high-dimensional datasets with complex interactions between features, but it may not capture non-linear relationships between features as effectively as neural networks.

On the other hand, MLPs can capture complex non-linear relationships between features, but they may overfit or underfit the data if the architecture is not well-tuned. By ensembling these models with different strengths and weaknesses, we can improve the generalization performance of the model and reduce the risk of overfitting or underfitting.

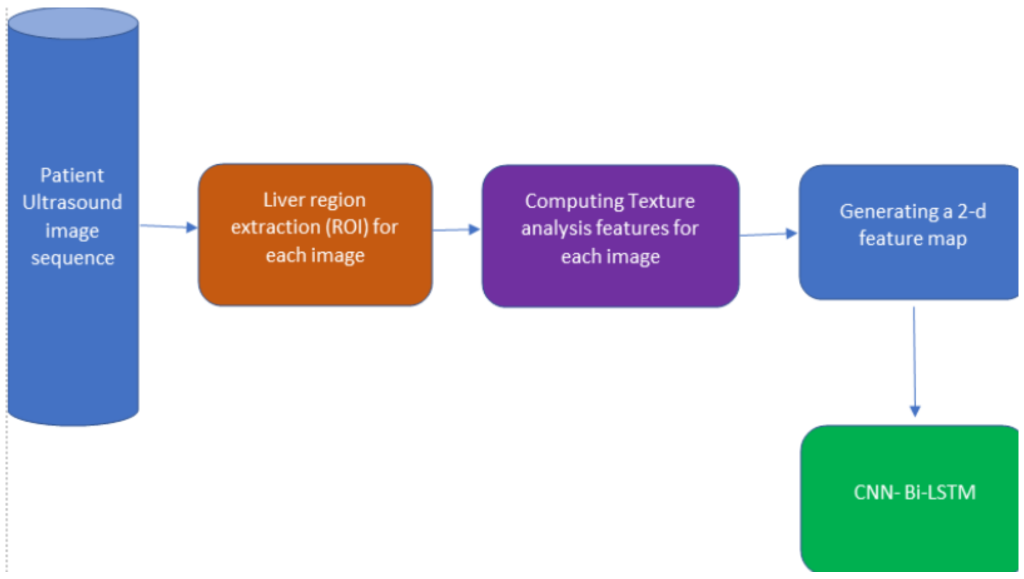
Furthermore, by varying the number of layers in the MLPs, we can capture different levels of abstraction in the data, which can improve the model's ability to capture complex relationships between features at different levels of granularity.

5.1.2.MODEL CONFIGURATION



5.2.DETECTING NAFLD USING DEEP LEARNING ON ULTRASOUND IMAGES

In recent years, the neural network model plays a vital role in the field of computer vision such as image classification, face recognition, handwritten digit recognition, etc,. Generally, Convolutional Neural Network (CNN) model is the predominant position of image classification. To achieve better performance, Long-Short Term Memory (LSTM) technique and Bidirectional Long-Short Term Memory (BiLSTM) method are applied in RNN model. The basic principles of the CNN model, LSTM and BiLSTM models are given in the subsequent section

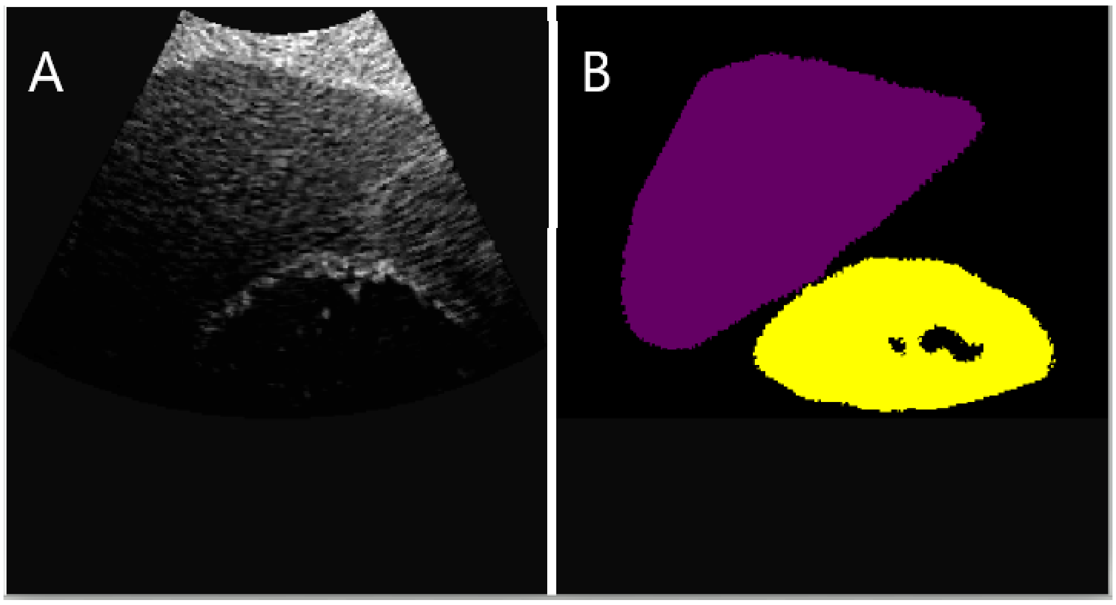

Patient Ultrasound image sequence
Liver region extraction (ROI) for each image
Computing Texture analysis features for each image
Generating a 2-d feature map
CNN- Bi-LSTM

Neural networks have played a vital role in computer vision i.e face recognition, image classification, etc. The Convolutional Neural network (CNN) model has been the go-to model for image classification. Previous works demonstrated the use of neural networks to diagnose NAFLD, however, a lot of factors affect the quality of the ultrasound image and may impact the information captured, hence it is imperative to make use of multiple ultrasound images that are captured from a single patient to effectively diagnose the disease. In this aspect, sequential neural networks like Long-short term memory are helpful. CNN can help in capturing useful information from stand-alone images.We propose a CNN+ Bi-LSTM architecture as a model which trains directly on the ultrasound image sequences without ROI segmentation. The 2D CNN can learn spatial features from the ultrasound images and the BI-LSTM can learn temporal dependencies between the frames in the sequence. For these methods extracting ROI is necessary. For that extraction, we implement U-net algorithm[13] to segment the liver region from the ultrasound images. We then extract texture features from the isolated liver region. These features are then fed as input to our CNN+Bi-LSTM model. Since each patient has 10 images, a unique feature map is constructed using these texture analysis features and then the feature map serves as the input for the CNN+Bi-LSTM model. By passing the feature map through a CNN, we are essentially allowing the network to learn temporal relationships between the extracted features across different images in the sequence. The output feature map generated by the CNN should therefore contain information not only about the individual images but also about the temporal relationships between them. By passing this output feature map through a Bi-LSTM layer, we further allow the network to learn long-term temporal dependencies in the sequence.

The concept of stacking features to form feature maps has been proven to be effective in multiple works\cite{}\cite{}\cite{}. However, the apparent assumption that we are basing this idea on is that by generating feature maps from multiple sequential images, the resulting feature maps may capture some temporal information. This assumption is based on the fact that the features extracted from different frames of the same sequence are correlated and can provide information about the temporal changes in the image features.

5.2.1.EXTRACTING LIVER REGION USING U-NET ALGORITHM

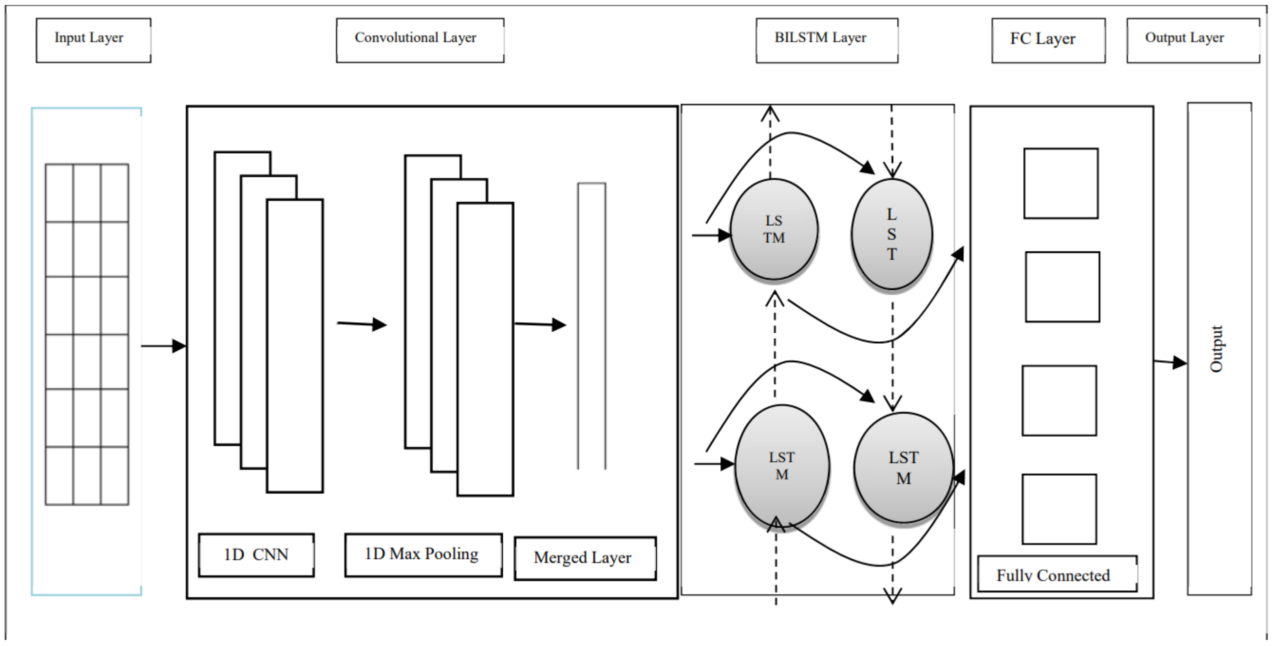
To extract the liver region from the ultrasound images, first we identified a pretrained algorithm called U-net. A dataset comprising of 300 ultrasound scans where the liver region has been segmented is used to fine-tune this model to detect and isolate the liver region from the liver-kidney parasaggital ultrasound image data. However, it is to be noted that the ultrasound image dataset that has been used to fine-tune the dataset does not contain strictly parasaggital images of liver-kidney. The fine-tuned model is then used to detect and isolate the liver region from our b-mode ultrasound image data. Here is a sample liver region color code in purple extracted from the ultrasound image.The kidney region is represented by yellow color. Liver regions of all 550 images from 55 patients have been isolated.



5.2.2.ARCHITECTURE OF CNN+BI-LSTM

A CNN+Bi-LSTM model is a deep learning architecture that combines a convolutional neural network (CNN) and a bidirectional long short-term memory (Bi-LSTM) network. This type of model is particularly useful for processing sequences of images or other types of data where temporal relationships between input data points are important.The CNN component of the model is typically used to extract features from the input data, in this case, a sequence of ultrasound images. The CNN consists of one or more layers of convolutional filters that learn to identify patterns in the input data. These patterns can represent features such as edges, corners, or other structures that are relevant to the problem being solved.The output of the CNN is a set of feature maps, which represent the presence of different patterns in the input data. These feature maps are then passed to the Bi-LSTM component of the model.

The Bi-LSTM network is a type of recurrent neural network (RNN) that is designed to capture temporal relationships in sequences of data. The Bi-LSTM network consists of two layers of LSTM cells, one processing the sequence in the forward direction and the other processing it in the reverse direction. The outputs of these two layers are then concatenated to produce the final output.The Bi-LSTM network is trained to learn temporal patterns in the input data that are relevant to the problem being solved. For example, in the case of liver ultrasound images, the Bi-LSTM network might learn to identify temporal changes in the texture or appearance of the liver over time, which could be indicative of liver disease.The final output of the model is typically a prediction or classification of some kind, such as the presence of liver disease in the patient. This prediction is based on the features extracted by the CNN and the temporal patterns learned by the Bi-LSTM network.Overall, a CNN+Bi-LSTM model is a powerful deep learning architecture that is well-suited to processing sequences of data where temporal relationships are important. This type of model has shown promising results in a variety of applications, including speech recognition, natural language processing, and medical imaging analysis.



5.2.3.MODEL CONFIGURATION

The input shape to the model is a 5-dimensional tensor of shape (batch_size, seq_len, channels, height, width), where batch_size is the number of sequences in the batch, seq_len is the length of each sequence (i.e., the number of images in each sequence), channels is the number of channels in each image (e.g., 3 for RGB), and height and width are the height and width of each image, respectively.

The first part of the model is a convolutional neural network (CNN) with two convolutional layers followed by ReLU activations and max pooling. The first convolutional layer has 32 filters with a kernel size of 3, while the second convolutional layer has 64 filters with a kernel size of 3. The max pooling operation reduces the spatial dimensions of the output by a factor of 2. The CNN processes each image in the sequence independently.The output of the CNN is then reshaped into a 3-dimensional tensor of shape (batch_size, seq_len, features), where features is the number of output channels from the second convolutional layer (which is 64 in this case).The reshaped output is then fed into a bidirectional LSTM (long short-term memory) layer with lstm_units hidden units.

The LSTM processes the sequence of features across time (i.e., across the seq_len dimension) and generates a sequence of hidden states with shape (batch_size, seq_len, lstm_units * 2), where the last dimension is multiplied by 2 because the LSTM is bidirectional.The output of the LSTM is passed through a dropout layer with a rate of dropout_rate, which randomly sets some of the values in the tensor to zero to prevent overfitting.

Finally, the output of the dropout layer is fed through a fully connected layer with 3 output units, which correspond to the 3 classes in the classification task.

5.2.4.FEATURE EXTRACTION USING TEXTURE ANALYSIS TECHNIQUES

Texture analysis methods such as GLCM, Haralick, and Fractal have been used in medical imaging to analyzing the texture patterns of liver ultrasound images, which can help in the detection and diagnosis of NAFLD.

5.2.4.1.GLCM(Gray-Level Cooccurence matrix)

GLCM is a statistical method used to analyze the spatial relationship between pixels in an image. It involves creating a matrix that counts the number of times pairs of pixels with specific values and spatial relationships occur in the image. From this matrix, several texture features can be calculated, including contrast, homogeneity, entropy, and correlation.

Probability Matrix(P):

where N(i, j) is the number of occurrences of pixel pairs (i, j) with a certain distance and direction.

Contrast:

$Contrast = ∑\_\{i = 1\}^\{G\}∑\_\{j = 1\}^\{G\} \backslash frac\{(i − j)^2 P(i, j)\}\{(\backslash sigma\_i)^2 + \backslash sigma\_j$

Energy:

$Energy = \sum i = 1G\sum j = 1GP2(i, j)$

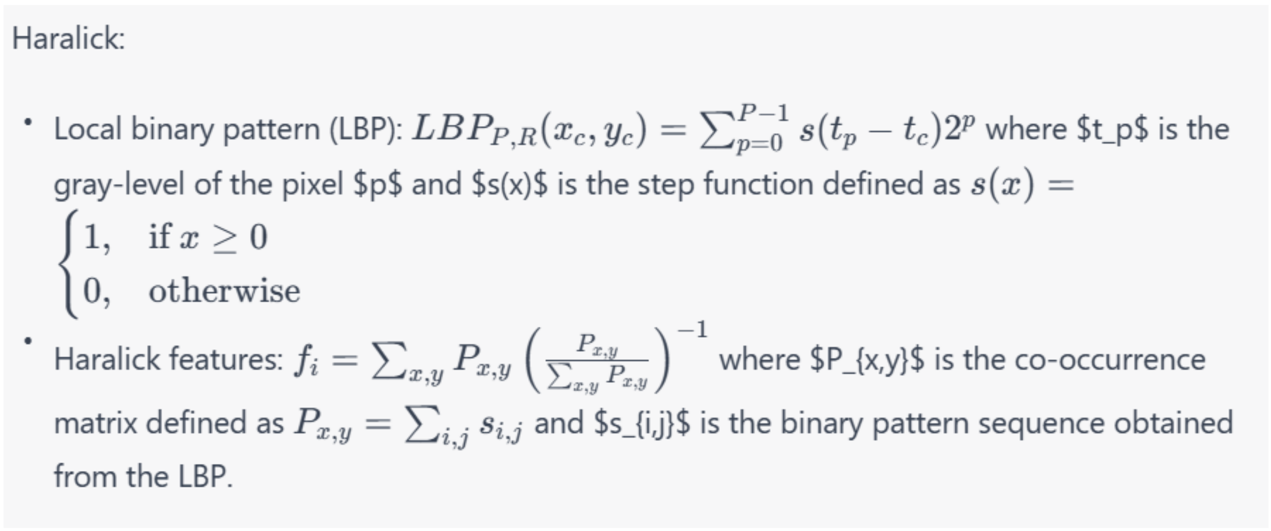
Homogeneity:

$Homogeneity = \sum i = 1G\sum j = 1G1 + |i - j|P(i, j)$

where G is the number of gray levels in the image.

5.2.4.2.Haralick Feature technique

Haralick features are a set of texture features that were originally developed for analyzing textures in satellite imagery. These features are based on the co-occurrence matrix and measure various aspects of texture such as contrast, uniformity, and complexity. The Haralick features can capture variations in texture patterns, which can be useful for detecting the presence of fatty liver disease.

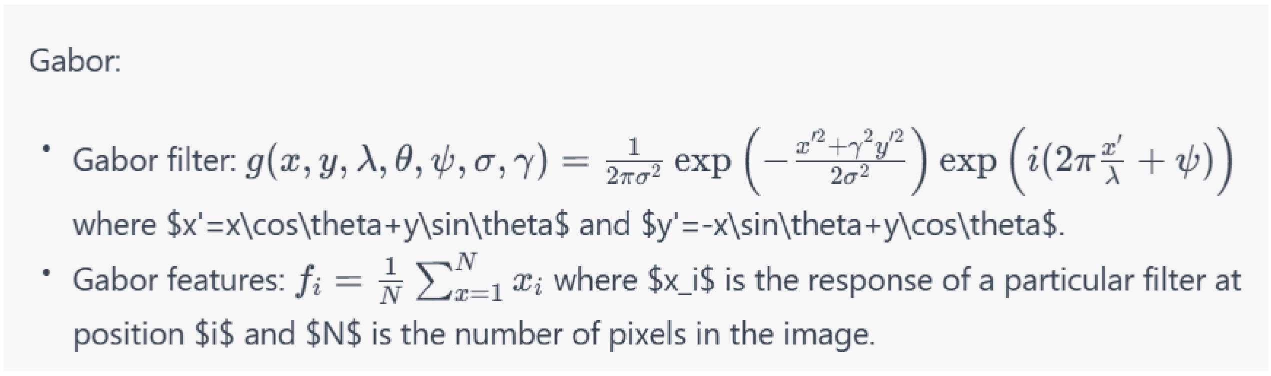


Haralick:

- Local binary pattern (LBP): $LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(t_p - t_c)2^p$ where \$t_p\$ is the gray-level of the pixel \$p\$ and \$s(x)\$ is the step function defined as $s(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$
- Haralick features: $f_i = \sum_{x,y} P_{x,y} \left(\frac{P_{x,y}}{\sum_{x,y} P_{x,y}} \right)^{-1}$ where \$P_{x,y}\$ is the co-occurrence matrix defined as $P_{x,y} = \sum_{i,j} s_{i,j}$ and \$s_{i,j}\$ is the binary pattern sequence obtained from the LBP.

5.2.4.3.Gabor Feature technique

Gabor features are computed by convolving an image with a bank of Gabor filters at different orientations and scales. The resulting filter response maps are then used to extract statistical measures such as mean, variance, energy, and entropy. These measures capture information about the spatial frequency and orientation content of the image,

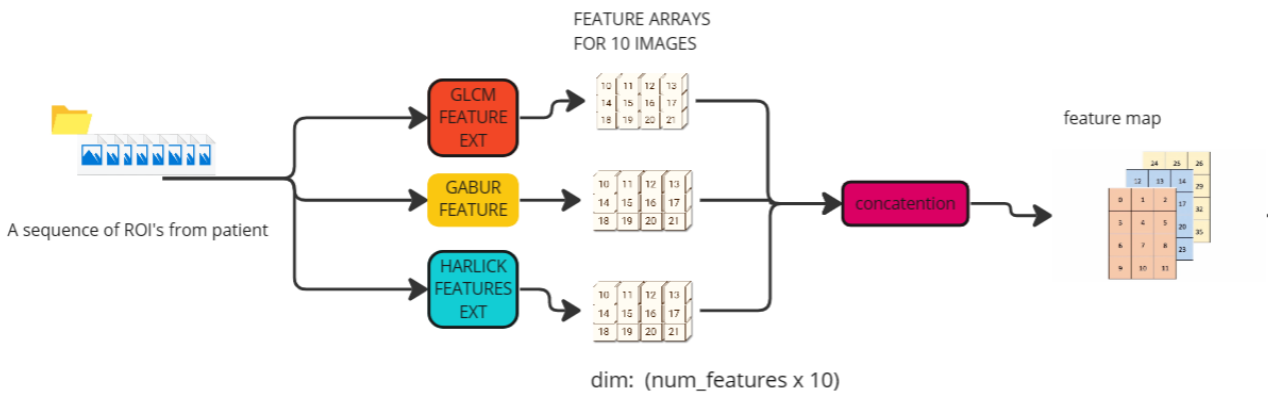


Gabor:

- Gabor filter: $g(x, y, \lambda, \theta, \psi, \sigma, \gamma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i(2\pi\frac{x'}{\lambda} + \psi)\right)$ where $x'=x\cos\theta+y\sin\theta$ and $y'=-x\sin\theta+y\cos\theta$.
- Gabor features: $f_i = \frac{1}{N}\sum_{x=1}^{N} x_i$ where x_i is the response of a particular filter at position i and N is the number of pixels in the image.

5.2.5. Generating feature maps from texture techniques

To generate a feature map from these three methods, we concatenated the feature vectors obtained from each method for each ultrasound image. This will result in a combined feature vector. This process is repeated for all images in a sequence. Once we obtained the combined feature vectors for all images in a sequence, they were stacked to create a 2D tensor. Each element in the tensor would represent the feature vector for a particular image in the sequence.



This 2D tensor is passed through CNN to generate a set of feature maps. The CNN would generate its own set of feature maps by learning and identifying relevant patterns in the input data. These feature maps are then be passed through a Bi-LSTM to capture the temporal dependencies between the feature maps. The output of the Bi-LSTM can be used as input for further analysis or classification tasks.

Feature extraction technique	Number of features extracted per image	Tensor dimensions
GLCM	5	
HARALICK	1	55,10,14
GABUR-WAVELET	8	

Chapter 6

6.RESULTS AND DISCUSSION

Results for both methods have been presented separately in this section.

6.1.NHANES 2017-18 DATA

Apart from the stacked ensemble of the 3-layer MLP, 5-layer MLP, and Random Forest, each model has also been individually tested. The highest macro recall (sensitivity) of 0.997 was achieved by the proposed stacked ensemble model while the lowest was 0.942 achieved by the 3-layer MLP model. Both the random forest model and the proposed method achieved similar specificity of 0.998 while the lowest was 0.966 obtained by 5-layer MLP. There proposed method achieved the highest accuracy of 99.8%. The Random Forest model's accuracy was close to but less than the proposed method. Overall the highest AUC score was obtained by the proposed method. Here we can observe a good improvement achieved in creating an ensemble involving deep learning models.

Model	Mac rec	Specificity	Acc (%)	AUC
3-Layer MLP	0.942	0.970	95.5	0.942
5-Layer MLP	0.944	0.966	95.4	0.944
RandomForest	0.989	0.998	99.3	0.989
Proposed stacked ensemble*	0.997	0.998	99.8	0.997

6.2.B-MODE ULTRASOUND IMAGE DATA

7.UNIQUE CONTRIBUTIONS

- Removing the barriers to physician involvement through automating the process of ROI extraction.
- Developed models that can train on sequential images and make use of temporal information.
- Generating and utilizing feature maps instead of training directly on the image, based on the correlation between features between sequences. Also making it easier for the models to train on sequential image data.
- Presenting a simplistic way of ensembling models to classify clinical data.

8.CONCLUSION AND FUTURE WORK

9.APPENDIX

10. APPENDIX

Code for NHANES 2017-18 laboratory data:

CODE FOR DATA INTEGRATION

```
# -*- coding: utf-8 -*-
"""NHANES_2017_2018.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1Eih4rWV4hxGfbZqS6IDu_ERKoSde
tS1R
```

```
"""

from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score

!pip install xport
!pip install csv

p='/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD/DATA/nhanes_
data_files/VID_E_2007_2008.XPT'.split("/")
p


import xport, csv
import os
import pandas as pd
def xpt_to_csv(filepath, save_dir):
   with open(filepath, 'rb') as f:
      df = xport.to_dataframe(f)
   #print (df.columns.values)
   temp = filepath.split('/')[9]
   print(temp)
   name=temp.split(".")[0]
   print(name)
   savepath = save_dir +'/' + name + '.csv'
   df.to_csv(savepath)

xpt_to_csv('/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD/DATA
/nhanes_data_files/ALB_CR_E_2007_2008.XPT',
'/content/sample_data/CSVfiles')
```

```
xpt_to_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD/DATA
/nhanes_data_files/2017_2018/UCPREG_J.XPT","/content/drive/MyDrive/CA
PSTONE_19BCE1180/NAFLD/DATA/nhanes_data_files/2017_2018_CSV")

import xport, csv
import os
import pandas as pd

data_dir                                                                    =
'/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD/DATA/nhanes_dat
a_files/2017_2018'

# Function to convert an .XPT file to .csv format
def xpt_to_csv(filepath, save_dir):
    with open(filepath, 'rb') as f:
        df = xport.to_dataframe(f)
    #print (df.columns.values)
    temp = filepath.split('/')[9]
    print(temp)
    name=temp.split(".")[0]
    print(name)
    savepath = save_dir +'/' + name + '.csv'
    df.to_csv(savepath)

    return True
for xpt_file in os.scandir(data_dir):
    #print (xpt_file)
    if xpt_file.is_file():
     xpt_loc = xpt_file.path
```

```
                                        sav_loc              =
'/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD/DATA/nhanes_dat
a_files/2017_2018_CSV'
    try:
      print (xpt_loc)
      print (sav_loc)
      flag = xpt_to_csv(xpt_loc, sav_loc)
    except:
      continue
    #'''
    if (flag == True):
      print ("Success", xpt_loc)
    else:
      print ("Not a success", xpt_loc)
    #'''


"""# Exclusion

#### DROPPING PATIENTS WHO DID NOT GO THROUGH
EXAMINATION (LUAXSTAT =3)
"""

elast=elast[(elast["LUAXSTAT"]==1) | (elast["LUAXSTAT"]==2)]

"""#### DROPPING PATIENTS WHO DO NOT HAVE >=10 RELIABLE
MEASURES OR WHOSE IQR/M >30 % (QUARTILE BASED CV)"""

elast=elast[(elast["LUARXNC"]!=2) | (elast["LUARXNC"]!=3)]
```

```
"""#### DROPPING PATIENTS WHO HAD OTHER ESTABLISHED CAUSES OF LIVER DISEASE

1. CONSUMED ALCOHOL defined as an average daily consumption of ≥20 g/day and ≥30 g/day for women and men (ALQ_J.CSV)

2. serum hepatitis B surface antigen (HEPABD_J.CSV)

3. hepatitis C  (HEPC_J.CSV)

4. if hepatitis B or C was reported

5. human immunodeficiency virus (HIV) (HIV_J.CSV)

6. Is pregnant

LBDHBG (SURFACE ANTIGEN FOR HEP B)
"""


hep_b=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD/DATA/nhanes_data_files/2017_2018_CSV/HEPBD_J.csv")
hep_b_pos=hep_b[hep_b[("LBDHBG")]==1]

elast.shape

hep_b_pos_seq=hep_b_pos['SEQN']
for i in hep_b_pos_seq:
   elast.drop(elast[elast['SEQN']== i].index,inplace=True)

elast.shape
```

```
"""Hepatitis C RNA (LBXHCR)"""

hep_c=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFL
D/DATA/nhanes_data_files/2017_2018_CSV/HEPC_J.csv")
hep_c_pos=hep_c[hep_c[("LBXHCR")]==1]

elast.shape

hep_c_pos_seq=hep_c_pos['SEQN']
for i in hep_c_pos_seq:
   elast.drop(elast[elast['SEQN']== i].index,inplace=True)

elast.shape

"""HIV (LBXHIV1)"""

hiv_1=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFL
D/DATA/nhanes_data_files/2017_2018_CSV/HIV_J.csv")
hiv_1_pos=hiv_1[hiv_1["LBXHIV1"]==1]

elast.shape

hiv_pos_seq=hiv_1_pos['SEQN']
for i in hiv_pos_seq:
   elast.drop(elast[elast['SEQN']== i].index,inplace=True)

elast.shape

"""URINE PREGNANCY TEST(URXPREG)"""
```

```
preg=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD
/DATA/nhanes_data_files/2017_2018_CSV/UCPREG_J.csv")
preg_pos=preg[preg["URXPREG"]==1]

elast.shape

preg_pos_seq=preg_pos['SEQN']
for i in preg_pos_seq:
   elast.drop(elast[elast['SEQN']== i].index,inplace=True)

elast.shape

elast.to_csv("filtered_file.csv")

elast=pd.read_csv("/content/filtered_file.csv")
elast.drop("Unnamed: 0.1",inplace=True,axis=1)
elast

"""EXCESS ALCOHOL CONSUMERS:
 ≥20 g/day and ≥30 g/day for women and men

dem=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD
/DATA/nhanes_data_files/2017_2018_CSV/DEMO_J_2017_2018.csv")
dem_men=dem[dem["RIAGENDR"]==1]
dem_fem=dem[dem["RIAGENDR"]==2]
dem_men_seq=dem_men["SEQN"]
dem_fem_seq=dem_fem["SEQN"]

alc[alc["SEQN"]==93706.0]["ALQ130"]==np.NaN

alc_not_nan["ALQ130"].unique()
```

```
alc_not_nan

alc=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD/
DATA/nhanes_data_files/2017_2018_CSV/ALQ_J.csv")
alc_not_nan=alc.dropna(subset=["ALQ130"])
'''
for i in dem_men_seq:
   if (alc_not_nan["SEQN"]==i) and (alc_not_nan["ALQ130"]>3):

alc_not_nan.drop(alc_not_nan[alc_not_nan["SEQN"]==i].index,inplace=True)
for j in dem_fem_seq:
   if (alc_not_nan[(alc_not_nan["SEQN"]==j)].ALQ130>2):

alc_not_nan.drop(alc_not_nan[alc_not_nan["SEQN"]==j].index,inplace=True)
alc.shape'''

df= pd.merge(dem, alc, on='SEQN', how='inner')

df.dropna(subset=["ALQ130"],inplace=True)
df_m=df[df["RIAGENDR"]==1]
df_f=df[df["RIAGENDR"]==2]
df_m=df_m[df_m["ALQ130"]>3]
df_f=df_f[df_f["ALQ130"]>2]
n_safe=list(df_m["SEQN"])+list(df_f["SEQN"])
len(n_safe)

for i in n_safe:
   elast.drop(elast[elast['SEQN']== i].index,inplace=True)

elast.shape
```

```
"""MERGING"""

elast=pd.read_csv("/content/LUX_imputed.csv")

elast.drop("Unnamed: 0",inplace=True,axis=1)
elast.drop("Unnamed: 0.1",inplace=True,axis=1)
"""## Merging datasets"""

import pandas as pd
import numpy as np

elast=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD/DATA/NHANES_CSV/LUX_J_2017_2018.csv")
elast
```

CODE FOR EXCLUSION CRITERIA:

```
"""# Exclusion

#### DROPPING PATIENTS WHO DID NOT GO THROUGH EXAMINATION (LUAXSTAT =3)
"""

elast=elast[(elast["LUAXSTAT"]==1) | (elast["LUAXSTAT"]==2)]

"""#### DROPPING PATIENTS WHO DO NOT HAVE >=10 RELIABLE MEASURES OR WHOSE IQR/M >30 % (QUARTILE BASED CV)"""

elast=elast[(elast["LUARXNC"]!=2) | (elast["LUARXNC"]!=3)]
```

```
"""#### DROPPING PATIENTS WHO HAD OTHER ESTABLISHED CAUSES OF LIVER DISEASE

1. CONSUMED ALCOHOL defined as an average daily consumption of ≥20 g/day and ≥30 g/day for women and men (ALQ_J.CSV)

2. serum hepatitis B surface antigen (HEPABD_J.CSV)

3. hepatitis C  (HEPC_J.CSV)

4. if hepatitis B or C was reported

5. human immunodeficiency virus (HIV) (HIV_J.CSV)

6. Is pregnant

LBDHBG (SURFACE ANTIGEN FOR HEP B)
"""

hep_b=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NAFLD/DATA/nhanes_data_files/2017_2018_CSV/HEPBD_J.csv")
hep_b_pos=hep_b[hep_b[("LBDHBG")]==1]

elast.shape

hep_b_pos_seq=hep_b_pos['SEQN']
for i in hep_b_pos_seq:
   elast.drop(elast[elast['SEQN']== i].index,inplace=True)

elast.shape
```

```
"""Hepatitis C RNA (LBXHCR)"""

hep_c=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/N
AFLD/DATA/nhanes_data_files/2017_2018_CSV/HEPC_J.csv")
hep_c_pos=hep_c[hep_c[("LBXHCR")]==1]

elast.shape

hep_c_pos_seq=hep_c_pos['SEQN']
for i in hep_c_pos_seq:
   elast.drop(elast[elast['SEQN']== i].index,inplace=True)

elast.shape

"""HIV (LBXHIV1)"""

hiv_1=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NA
FLD/DATA/nhanes_data_files/2017_2018_CSV/HIV_J.csv")
hiv_1_pos=hiv_1[hiv_1["LBXHIV1"]==1]

elast.shape

hiv_pos_seq=hiv_1_pos['SEQN']
for i in hiv_pos_seq:
   elast.drop(elast[elast['SEQN']== i].index,inplace=True)

elast.shape

"""URINE PREGNANCY TEST(URXPREG)"""
```

```
preg=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NA
FLD/DATA/nhanes_data_files/2017_2018_CSV/UCPREG_J.csv")
preg_pos=preg[preg["URXPREG"]==1]

elast.shape

preg_pos_seq=preg_pos['SEQN']
for i in preg_pos_seq:
   elast.drop(elast[elast['SEQN']== i].index,inplace=True)

elast.shape

elast.to_csv("filtered_file.csv")

elast=pd.read_csv("/content/filtered_file.csv")
elast.drop("Unnamed: 0.1",inplace=True,axis=1)
elast

"""EXCESS ALCOHOL CONSUMERS:
 ≥20 g/day and ≥30 g/day for women and men
 "

dem=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NA
FLD/DATA/nhanes_data_files/2017_2018_CSV/DEMO_J_2017_2018.csv"
)
dem_men=dem[dem["RIAGENDR"]==1]
dem_fem=dem[dem["RIAGENDR"]==2]
dem_men_seq=dem_men["SEQN"]
dem_fem_seq=dem_fem["SEQN"]

alc[alc["SEQN"]==93706.0]["ALQ130"]==np.NaN
```

```
alc_not_nan["ALQ130"].unique()

alc_not_nan

alc=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE1180/NAF
LD/DATA/nhanes_data_files/2017_2018_CSV/ALQ_J.csv")
alc_not_nan=alc.dropna(subset=["ALQ130"])
'''
for i in dem_men_seq:
   if (alc_not_nan["SEQN"]==i) and (alc_not_nan["ALQ130"]>3):

alc_not_nan.drop(alc_not_nan[alc_not_nan["SEQN"]==i].index,inplace=
True)
for j in dem_fem_seq:
   if (alc_not_nan[(alc_not_nan["SEQN"]==j)].ALQ130>2):

alc_not_nan.drop(alc_not_nan[alc_not_nan["SEQN"]==j].index,inplace=
True)
alc.shape'''

df= pd.merge(dem, alc, on='SEQN', how='inner')

df.dropna(subset=["ALQ130"],inplace=True)
df_m=df[df["RIAGENDR"]==1]
df_f=df[df["RIAGENDR"]==2]
df_m=df_m[df_m["ALQ130"]>3]
df_f=df_f[df_f["ALQ130"]>2]
n_safe=list(df_m["SEQN"])+list(df_f["SEQN"])
len(n_safe)
```

```
for i in n_safe:
   elast.drop(elast[elast['SEQN']== i].index,inplace=True)

elast.shape

"""MERGING"""

elast=pd.read_csv("/content/LUX_imputed.csv")

elast.drop("Unnamed: 0",inplace=True,axis=1)
elast.drop("Unnamed: 0.1",inplace=True,axis=1)
```

Annotating the data:

```
"""## Annotating merged data, i.e based on CAP score and Fibrosis score
(VCTE)"""

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

liver_data=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE118
0/NAFLD/DATA/nhanes_data_files/2017_2018_CSV/liver_nhanes_merged
.csv")

liver_data.drop("Unnamed: 0",inplace=True,axis=1)

liver_data
```

```
liver_data.info()

liver_data["LUXCAPM"].isnull().value_counts()

liver_data["LUXSMED"].isnull().value_counts()

for i in liver_data.columns:
  print(liver_data[i].isnull().value_counts("True"))

!pip install missingpy

import sys
import sklearn.neighbors._base
sys.modules['sklearn.neighbors.base'] = sklearn.neighbors._base

nafld.shape

a=list(liver_data.columns)

def intersection(lst1, lst2):
    lst3 = [value for value in lst1 if value in lst2]
    return lst3

b=[
"LBDSATLC",
"LBXSALSI","LBDSBUSI","LBDSCRSI","LBDSGBSI","LBDSGLSI",
"LBDSGTLC", "LBXSIR", "LBXSPHSI", "LBDSIRSI", "LBDSTBSI",
"LBDSTBLC","LBXSCASI",
"LBDSCHSI",    "LBDSTPSI",    "LBDSTRSI",    "LBDSUASI",
"URXCRS","URDUCRLC","URXUMA","URDUMALC",
"URXCRS",
"BMIHIP",
"BMIWAIST",
"BMIARMC",
"BMIARML",
"BMILEG",
"BMIHT",
"BMIHEAD",
"BMIRECUM",
"BMIWT",
"BMDSTATS",
"PEASCCT1",
"BPAARM",
"BPACSZ",
"BPAEN1",
"BPAEN2",
"BPAEN3",
"BPAEN4",
"LBXLYPCT",
"LBXMOPCT",
"LBXNEPCT",
"LBXEOPCT",
"LBXBAPCT",
"LBDCOTLC",
"LBDHCTLC",
"LBDBCRSI",
"LBDBCRLC",
"LBDBCOSI",
"LBDBCOLC",
"SDDSRVYR",
"RIDSTATR",
"RIDRETH1",
"RIDEXMON",
"DMQMILIZ",
"DMQADFC",
"DMDCITZN",
"DMDYRSUS",
"DMDEDUC3",
"DMDEDUC2",
"DMDMARTL",
"SIALANG",
"SIAPROXY",
"SIAINTRP",
"FIALANG",
"FIAPROXY",
"FIAINTRP",
"MIALANG",
"MIAPROXY",
"MIAINTRP",
"AIALANGA",
"DMDHHSIZ",
"DMDFMSIZ",
"DMDHHSZA",
"DMDHHSZB",
"DMDHHSZE",
"DMDHRGND",
"DMDHRAGZ",
"DMDHREDZ",
"DMDHRMAZ",
"DMDHSEDZ",
"WTINT2YR",
"WTMEC2YR",
"SDMVPSU",
"SDMVSTRA",
"INDHHIN2",
"INDFMIN2",
"INDFMPIR",
"DXARLBV",
"DXARATV",
"DXARLTV",
"DXATRBV",
"DXATRTV",
"LBDFERSI",
"URDUASLC",
"URDUIOLC",
"WTSA2YR",
"WTSA2YR",
"URDUCMLC",
"URDTIME3",
"URDFLOW3",
"URXVOL3",
"URDTIME2",
"URDFLOW2",
"URXVOL2",
"URDTIME1",
"URDFLOW1",
"URXVOL1",
"WTSA2YR",
"URDUA3LC",
"URDUA5LC",
"URDUABLC",
"URDUACLC",
"URDUDALC",
"URDUMMAL",
"WTSAF2YR",
"LBDTRSI",
"LBDLDLSI",
"LBDLDMSI",
"LBDLDNSI",
"LBDTFRSI",
"LBDTCSI",
"SMQ020",
"SMD030",
"SMQ040",
"SMQ050Q",
"SMQ050U",
"SMD057",
"SMQ078",
"SMD641",
"SMD650",
"SMD093",
"SMDUPCA",
"SMD100BR",
"SMD100FL",
"SMD100MN",
"SMD100LN",
"SMD100TR",
"SMD100NI",
"SMD100CO",
"SMQ621",
"SMD630",
"SMQ661",
"SMQ665A",
"SMQ665B",
"SMQ665C",
"SMQ665D",
"SMQ670",
"SMQ848",
"SMQ852Q",
"SMQ852U",
"SMQ890",
"SMQ895",
"SMQ900",
"SMQ905",
"SMQ910",
"SMQ915",
"SMAQUEX2",
"LBDBPBSI",
"LBDBPBLC",
"LBDBCDSI",
"LBDBCDLC",
"LBDTHGSI",
"LBDTHGLC",
"LBDBSESI",
"LBDBSELC",
"LBDBMNSI",
"LBDBMNLC",
"LUAXSTAT",
"LUARXNC",
"LUARXND",
"LUARXND",
"LUATECH",
"WTSAF2YR",
"LBDINSI",
"LBDINLC",
"LBDHRPLC",
"LBDHDDSI",
"WTSAF2YR",
"LBDGLUSI",
"URDBBALC",
"URDDPHLC",
"URDDUPLC",
"URDBDCLC",
"URDCEPLC",
"URDBCPLC",
"WTSB2YR",
"LBDSF6LC",
"LBDSF5LC",
"LBDSF4LC",
"LBDSF3LC",
"LBDSF2LC",
"LBDSF1LC",
"WTFOL2YR",
"LBDRFOSI",
"LBDTIBSI",
"LBDUIBLC",
"LBDIRNSI"]
a=list(liver_data.columns)
k=intersection(a,b)
k

liver_data.drop(k,inplace=True,axis=1)

liver_data.drop("LUAPNME",inplace=True,axis=1)

liver_data.drop("SEQN",inplace=True,axis=1)
```

```
col=liver_data.columns

"""CATEGORICAL AND NUMERICAL SEPARATION"""

liver_data

"""### Multiple imputer """

liver_data["LUXCAPM"].fillna(liver_data["LUXCAPM"].median(),inpla
ce=True)
liver_data["LUXSMED"].fillna(liver_data["LUXSMED"].median(),inpla
ce=True)
liver_data["LUXCAPM"]

liver_data.isnull().values.any()

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(liver_data),columns=col)
liver_data=X_scaled

liver_data

# Install the fancyimpute library
!pip install fancyimpute

# Load the required libraries
import pandas as pd
import numpy as np
from fancyimpute import IterativeImputer
```

```
# Load your data into Python
data = liver_data

# Convert the data to a numpy array
data = np.array(data)

# Specify the imputation method
imp = IterativeImputer(random_state=0, sample_posterior=True,
imputation_order='random', n_nearest_features=None,
initial_strategy='mean', max_iter=10, tol=0.001)

# Impute the missing values
imputed_data = imp.fit_transform(data)

# Convert the imputed data back to a pandas dataframe



# Use the imputed data for further analysis

imputed_data = pd.DataFrame(imputed_data, columns=col)

liver_data=imputed_data

liver_data.to_csv("liver_imputed.csv")

nafld_l=[]
counter=0
for i,j in
zip(list(liver_data["LUXCAPM"]),list(liver_data["LUXSMED"])):
  counter+=1
  if i<302 and j<8.2:
    nafld_l.append(0)  #non nafld
  elif i>=302:
    nafld_l.append(1)  #nafld
  elif i<302 and j>13.6:
    nafld_l.append(2)  #cryptogenic cirrhosis
  elif 274<=i<302 and 8.2<=j<=13.6:
    nafld_l.append(3)  #borderline steatosis
  elif i<274 and 8.2<=j<=13.6:
    nafld_l.append(4)  #control
  else:
    print(i,j,counter)

nafld=pd.Series(nafld_l)

nafld.value_counts()

liver_data=pd.read_csv("/content/drive/MyDrive/CAPSTONE_19BCE118
0/NAFLD/DATA/liver_data.csv",index_col=False)
liver_data.drop(["Unnamed: 0","Unnamed: 0.1"],inplace=True,axis=1)
liver_data

liver_data["nafld_label"]=nafld

liver_data["nafld_label"].value_counts()

liver_data=liver_data[(liver_data["nafld_label"]==0)                    |
(liver_data["nafld_label"]==1)]

liver_data.shape
```

Train-test split:

```
"""### train test split"""

liver_data.shape

liver_data.drop("Unnamed: 0",inplace=True,axis=1)
liver_data.drop("Unnamed: 0.1",inplace=True,axis=1)

from sklearn.model_selection import train_test_split
# Load the data
df = liver_data
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,random_state=2)

X_train

import torch
from torch.utils.data import TensorDataset, DataLoader
import pandas as pd
from sklearn.model_selection import train_test_split
# Load the data from the CSV file using pandas
data = liver_data

# Split the data into inputs and labels
inputs = data.drop('nafld_label', axis=1).values
labels = data['nafld_label'].values
```

```
# Split the data into training and testing datasets
train_inputs, test_inputs, train_labels, test_labels = train_test_split(inputs,
labels, test_size=0.2,random_state=3)

# Convert the training and testing inputs and labels to tensors
train_inputs_tensor = torch.tensor(train_inputs, dtype=torch.float32)
train_labels_tensor = torch.tensor(train_labels, dtype=torch.long)
test_inputs_tensor = torch.tensor(test_inputs, dtype=torch.float32)
test_labels_tensor = torch.tensor(test_labels, dtype=torch.long)

# Create a PyTorch dataset from the tensors
train_dataset = TensorDataset(train_inputs_tensor, train_labels_tensor)

# Create a PyTorch DataLoader from the dataset
batch_size = 32
trainloader_p = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)

train_inputs.shape
```

Models

```
"""## MODEL"""

import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dropout
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
```

```
import matplotlib.pyplot as plt
batch_size = 32
epochs = 100

"""### 3 LAYER MLP"""

import torch
import torch.nn as nn
import torch.nn.functional as F

# Define the MLP model class for binary classification
class MLP3(nn.Module):
    def __init__(self):
        super(MLP3, self).__init__()
        self.fc1 = nn.Linear(265, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x

# Initialize the MLP model and loss function
model = MLP3()
criterion = nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001,
momentum=0.9)

# Train the model on the dataset
```

```
for epoch in range(200):
    running_loss = 0.0
    for i, data in enumerate(trainloader_p, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels.unsqueeze(1).float())
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    if epoch % 100 == 0:
        print("Epoch %d, Loss: %.3f" % (epoch + 1, running_loss / 5000))

print("Finished Training")

import torch
import pandas as pd



# Pass the test inputs tensor to the model for prediction
model.eval()
with torch.no_grad():
    test_outputs = model(test_inputs_tensor)

# Convert the predicted outputs tensor to a numpy array
test_predictions_3 = test_outputs.numpy()

# Print the predictions

preds=[]
```

```
for i in test_predictions_3:
  if i[0]>0.5:
    preds.append(1)
  else:
    preds.append(0)

from sklearn.metrics import classification_report
print(classification_report(test_labels,preds))

import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score,
roc_auc_score, f1_score

def binary_classification_metrics(y_true, y_pred):
    # Calculate accuracy
    accuracy = accuracy_score(y_true, y_pred)

    # Calculate macro-averaged precision, recall, specificity and F1-score
    precision = precision_score(y_true, y_pred, average='macro')
    recall = recall_score(y_true, y_pred, average='macro')
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    specificity = tn / (tn + fp)
    f1 = f1_score(y_true, y_pred, average='macro')

    # Calculate AUC score
    auc = roc_auc_score(y_true, y_pred)

    # Calculate macro-averaged AUC score
    # First, convert y_true and y_prob to one-hot format
    #n_classes = len(np.unique(y_true))
    #y_true_one_hot = np.eye(n_classes)[y_true]
    #y_prob_one_hot = np.zeros((len(y_prob), n_classes))
    #y_prob_one_hot[:, 1] = y_prob
            #macro_auc  =  roc_auc_score(y_true_one_hot,  y_prob_one_hot,
average='macro')

                return  {"accuracy":accuracy,"precision":  precision,"recall":
recall,"specificity": specificity,"f1-score": f1,"auc": auc}

import numpy as np
from sklearn.metrics import confusion_matrix, roc_auc_score

def calculate_metrics(y_true, y_pred):
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    specificity = tn / (tn + fp)
    f1_score = 2 * precision * recall / (precision + recall)
    auc_score = roc_auc_score(y_true, y_pred)

    return {'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'specificity': specificity,
            'f1_score': f1_score,
            'auc_score': auc_score}

print(calculate_metrics(test_labels,preds))

print(binary_classification_metrics(test_labels,preds))
```

```
"""### RANDOM FOREST"""

from sklearn.ensemble import RandomForestClassifier
# Train the random forest classifier
clf = RandomForestClassifier(n_estimators=50)
clf.fit(train_inputs, train_labels)

# Evaluate the model on the test set
acc = clf.score(test_inputs, test_labels)
print("Accuracy: {:.2f}%".format(acc * 100))

preds_r=clf.predict(test_inputs)
print(classification_report(test_labels,preds_r))

print(calculate_metrics(test_labels,preds_r))

print(binary_classification_metrics(test_labels,preds_r))

# Confusion matrix
tn, fp, fn, tp = confusion_matrix(test_labels, preds_r).ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)

# ROC curve and AUC
fpr, tpr, thresholds = roc_curve(test_labels,
clf.predict_proba(test_inputs)[:,1])
auroc = roc_auc_score(test_labels, clf.predict_proba(test_inputs)[:,1])

import matplotlib.pyplot as plt

plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.2f})'.format(auroc))
plt.show()

"""### 5 LAYER MLP"""

import torch
import torch.nn as nn
import torch.nn.functional as F

class MLPBinary(nn.Module):
    def __init__(self):
        super(MLPBinary, self).__init__()
        self.fc1 = nn.Linear(265, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 32)
        self.fc5 = nn.Linear(32, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = torch.sigmoid(self.fc5(x))
        return x

# Initialize the MLP model and loss function
model = MLPBinary()
```

```
criterion = nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001,
momentum=0.9)

# Train the model on the dataset
for epoch in range(200):
    running_loss = 0.0
    for i, data in enumerate(trainloader_p, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels.unsqueeze(1).float())
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    if epoch % 100 == 0:
        print("Epoch %d, Loss: %.3f" % (epoch + 1, running_loss / 5000))

print("Finished Training")

mlp3_rf_preds=pd.DataFrame({"rforest":preds_r,"mlp3":preds})
mlp3_rf_preds

mlp3_rf_preds["test_labels"]=test_labels

test_input_df=pd.DataFrame(test_inputs)

mlp3_rf_

test_input_df["row_number"]=test_input_df.reset_index().index
```

```
mlp3_rf_preds["row_number"]=mlp3_rf_preds.reset_index().index
temp=pd.merge(test_input_df,mlp3_rf_preds,on="row_number")
temp
```

Proposed method and result comparison and analysis:

```
"""STACKED ENSEMBLE OF 3-LAYER, RF AND 5-LAYER MLP"""

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler


# Create the base classifiers
mlp1 = MLPClassifier(hidden_layer_sizes=(100, 50, 25), max_iter=1000, random_state=42)
rf = RandomForestClassifier(n_estimators=100, random_state=42)
mlp2 = MLPClassifier(hidden_layer_sizes=(200, 100, 50, 25, 10), max_iter=1000, random_state=42)

# Create the stacked ensemble classifier
estimators = [('3-layer MLP', mlp1), ('RANDOM FOREST', rf), ('5-layer MLP', mlp2)]
stacked_ensemble = StackingClassifier(estimators=estimators, final_estimator=MLPClassifier(hidden_layer_sizes=(50,), max_iter=1000, random_state=42))

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler


# Create the base classifiers
mlp1 = MLPClassifier(hidden_layer_sizes=(100, 50, 25), max_iter=1000,
random_state=42)
rf = RandomForestClassifier(n_estimators=100, random_state=42)
mlp2 = MLPClassifier(hidden_layer_sizes=(200, 100, 50, 25, 10),
max_iter=1000, random_state=42)

# Create the stacked ensemble classifier
estimators = [('3-layer MLP', mlp1), ('RANDOM FOREST', rf), ('5-layer
MLP', mlp2)]
stacked_ensemble = StackingClassifier(estimators=estimators,
final_estimator=MLPClassifier(hidden_layer_sizes=(50,), max_iter=1000,
random_state=42))

# Fit the stacked ensemble on the training data
stacked_ensemble.fit(train_inputs, train_labels)


# Evaluate the stacked ensemble on the testing data
stacked_predicts=stacked_ensemble.predict(test_inputs)
print(classification_report(test_labels,stacked_predicts,digits=0,zero_divisi
on=1))

print(calculate_metrics(test_labels,stacked_predicts))
print(binary_classification_metrics(test_labels,stacked_predicts))

# Confusion matrix
tn, fp, fn, tp = confusion_matrix(test_labels, stacked_predicts).ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)

# ROC curve and AUC
fpr, tpr, thresholds = roc_curve(test_labels, stacked_ensemble.predict_proba(test_inputs)[:,1])
auroc = roc_auc_score(test_labels, stacked_ensemble.predict_proba(test_inputs)[:,1])

confusion_matrix = metrics.confusion_matrix(test_labels, stacked_predicts)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])

cm_display.plot()
plt.show()

plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.2f})'.format(auroc))
plt.show()

!pip install --upgrade mlxtend
```

```
"""# CROSS VALIDATION

### Stacked Ensemble with 5-fold CV
"""

from mlxtend.classifier import StackingCVClassifier
mlp1 = MLPClassifier(hidden_layer_sizes=(100, 50, 25), max_iter=1000,
random_state=42)
rf = RandomForestClassifier(n_estimators=100, random_state=42)
mlp2 = MLPClassifier(hidden_layer_sizes=(200, 100, 50, 25, 10),
max_iter=1000, random_state=42)
estimators = [('3-layer MLP', mlp1), ('RANDOM FOREST', rf), ('5-layer
MLP', mlp2)]
#stacked_clf =StackingCVClassifier(classifiers=[mlp1,
rf],meta_classifier=mlp2,cv=5,use_probas=True,use_features_in_secondar
y=True,verbose=0)
stacked_clf=StackingClassifier(estimators=estimators,
final_estimator=MLPClassifier(hidden_layer_sizes=(50,), max_iter=1000,
random_state=42),cv=5)

stacked_clf.fit(train_inputs,train_labels)

!pip install --upgrade scikit-learn

from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores =
learning_curve(estimator=stacked_clf,
                                    X=train_inputs,
                                    y=train_labels,
                                    train_sizes=np.linspace(0.1, 1.0, 10),
                                              cv=5,
                                              n_jobs=-1)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5,
label='Training accuracy')
plt.fill_between(train_sizes, train_mean + train_std, train_mean -
train_std, alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s',
markersize=5, label='Validation accuracy')
plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std,
alpha=0.15, color='green')

plt.xlabel('Number of training samples')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.ylim([0.8, 1.0])
plt.show()

# Predict on test set
y_pred = stacked_clf.predict(test_inputs)

cv_preds=[]
for i in y_pred:
  if i>0.5:
    cv_preds.append(1)
  else:
    cv_preds.append(0)

print(calculate_metrics(test_labels,cv_preds))

# Confusion matrix
tn, fp, fn, tp = confusion_matrix(test_labels, y_pred).ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)

# ROC curve and AUC
fpr, tpr, thresholds = roc_curve(test_labels,
stacked_clf.predict_proba(test_inputs)[:,1])
auroc = roc_auc_score(test_labels,
stacked_clf.predict_proba(test_inputs)[:,1])

print(classification_report(test_labels,y_pred))

import matplotlib.pyplot as plt

plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.2f})'.format(auroc))
plt.show()

"""### delong test to compare auc scores"""

!pip install pycm
```

```
!pip install -U pycm

from sklearn.metrics import roc_auc_score
from scipy.stats import norm

def delong_test(y_true, y_pred1, y_pred2):
    # Calculate AUC for each model
    auc1 = roc_auc_score(y_true, y_pred1)
    auc2 = roc_auc_score(y_true, y_pred2)

    # Calculate difference in predicted probabilities
    y_diff = y_pred1 - y_pred2

    # Calculate variance of the difference
    var_diff = (2 * (2 * auc1 - 3 * auc1**2 + auc1**3) +
               2 * (2 * auc2 - 3 * auc2**2 + auc2**3))
    var_diff /= len(y_diff)

    # Calculate z-score of the difference
    z_score = y_diff / np.sqrt(var_diff)

    # Calculate p-value
    p_value = 2 * norm.cdf(-np.abs(z_score))

    return {"pvalue": p_value, "z_score": z_score, "var_diff": var_diff}

from pycm import *
import numpy as np
from scipy.stats import wilcoxon

def delong_test(y_true, y_pred_model1, y_pred_model2):
```

```
y_true = np.array(y_true)
y_pred_model1 = np.array(y_pred_model1)
y_pred_model2 = np.array(y_pred_model2)

if len(y_true.shape) == 1:
    y_true = y_true.reshape(-1, 1)

if len(y_pred_model1.shape) == 1:
    y_pred_model1 = y_pred_model1.reshape(-1, 1)

if len(y_pred_model2.shape) == 1:
    y_pred_model2 = y_pred_model2.reshape(-1, 1)

if y_true.shape[1] != 1:
    raise ValueError("y_true must be a column vector")

if y_pred_model1.shape[1] != 1:
    raise ValueError("y_pred_model1 must be a column vector")

if y_pred_model2.shape[1] != 1:
    raise ValueError("y_pred_model2 must be a column vector")

n = y_true.shape[0]

# Calculate AUC scores for each model
auc_model1 = roc_auc_score(y_true, y_pred_model1)
auc_model2 = roc_auc_score(y_true, y_pred_model2)

# Calculate covariance matrix
S = np.cov(y_pred_model1, y_pred_model2, rowvar=False)
```

```
    # Calculate DeLong's covariance
    a = np.sum((y_pred_model1 - y_pred_model2) ** 2) / (2 * n ** 2)
    b = 2 * S[0, 1] / n
     c = (auc_model1 * (1 - auc_model1) + auc_model2 * (1 - auc_model2) - 2
* S[0, 1]) / (2 * (n - 1))
    var_cov = (a - b + c) ** 2 / (a ** 2 + 4 * b ** 2 + c ** 2)

    # Calculate test statistic and p-value
    test_statistic = (auc_model1 - auc_model2) / np.sqrt(var_cov)
    p_value = 2 * (1 - norm.cdf(np.abs(test_statistic)))

    return {"test_statistic": test_statistic, "pvalue": p_value}

# Collect true labels and predicted probabilities for each model on a
common test dataset
true_labels = np.array(test_labels)
preds_model1 = np.array(preds)
preds_model2 = np.array(preds_r)
preds_model3 = np.array(mlp5_preds)
preds_model4 = np.array(stacked_predicts)

# Perform pairwise DeLong tests
p_value_1_2 = delong_test(true_labels, preds_model1,
preds_model2)["pvalue"]
p_value_1_3 = delong_test(true_labels, preds_model1,
preds_model3)["pvalue"]
p_value_1_4 = delong_test(true_labels, preds_model1,
preds_model4)["pvalue"]
p_value_2_3 = delong_test(true_labels, preds_model2,
preds_model3)["pvalue"]
```

```
p_value_2_4 = delong_test(true_labels, preds_model2,
preds_model4)["pvalue"]
p_value_3_4 = delong_test(true_labels, preds_model3,
preds_model4)["pvalue"]


# Print the p-values for each comparison
print("P-value for 3-layer MLP vs random forest:", p_value_1_2)
print("P-value for 3-layer MLP vs 5-layer MLP:", p_value_1_3)
print("P-value for 3-layer MLP vs stacked ensemble:", p_value_1_4)
print("P-value for random forest vs 5-layer MLP:", p_value_2_3)
print("P-value for random forest vs stacked ensemble:", p_value_2_4)
print("P-value for 5-layer MLP vs stacked ensemble:", p_value_3_4)

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, precision_recall_curve

# Collect true labels and predicted probabilities for each model on a
common test dataset
true_labels = np.array(test_labels)
preds_model1 = np.array(preds)
preds_model2 = np.array(preds_r)
preds_model3 = np.array(mlp5_preds)
preds_model4 = np.array(stacked_predicts)

# Calculate AUC scores for each model
auc_model1 = roc_auc_score(true_labels, preds_model1)
auc_model2 = roc_auc_score(true_labels, preds_model2)
auc_model3 = roc_auc_score(true_labels, preds_model3)
auc_model4 = roc_auc_score(true_labels, preds_model4)
```

```
# Calculate other performance metrics for each model
acc_model1 = accuracy_score(true_labels, np.round(preds_model1))
acc_model2 = accuracy_score(true_labels, np.round(preds_model2))
acc_model3 = accuracy_score(true_labels, np.round(preds_model3))
acc_model4 = accuracy_score(true_labels, np.round(preds_model4))

prec_model1, recall_model1, _ = precision_recall_curve(true_labels,
preds_model1)
prec_model2, recall_model2, _ = precision_recall_curve(true_labels,
preds_model2)
prec_model3, recall_model3, _ = precision_recall_curve(true_labels,
preds_model3)
prec_model4, recall_model4, _ = precision_recall_curve(true_labels,
preds_model4)

fpr_model1, tpr_model1, _ = roc_curve(true_labels, preds_model1)
fpr_model2, tpr_model2, _ = roc_curve(true_labels, preds_model2)
fpr_model3, tpr_model3, _ = roc_curve(true_labels, preds_model3)
fpr_model4, tpr_model4, _ = roc_curve(true_labels, preds_model4)

# Create subplots for each performance metric
fig, axs = plt.subplots(2, 3, figsize=(15,10))

# Plot accuracy
axs[0, 0].bar([1, 2, 3, 4], [acc_model1, acc_model2, acc_model3,
acc_model4])
axs[0, 0].set_title("Accuracy")
axs[0, 0].set_xticks([1, 2, 3, 4])
axs[0, 0].set_xticklabels(["3-MLP", "RF", "5-MLP", "Stacked
ensemble"])
```

```
# Plot precision-recall curves
axs[0, 1].plot(recall_model1, prec_model1, label="3-MLP")
axs[0, 1].plot(recall_model2, prec_model2, label="RF")
axs[0, 1].plot(recall_model3, prec_model3, label="5-MLP")
axs[0, 1].plot(recall_model4, prec_model4, label="Stacked ensemble")
axs[0, 1].set_title("Precision-Recall Curve")
axs[0, 1].set_xlabel("Recall")
axs[0, 1].set_ylabel("Precision")
axs[0, 1].legend(loc="lower left")

# Plot ROC curves
axs[0, 2].plot(fpr_model1, tpr_model1, label="3-MLP")
axs[0, 2].plot(fpr_model2, tpr_model2, label="RF")
axs[0, 2].plot(fpr_model3, tpr_model3, label="5-MLP")
axs[0, 2].plot(fpr_model4, tpr_model4, label="Stacked ensemble")
axs[0, 2].set_title("ROC Curve")
axs[0, 2].set_xlabel("False Positive Rate")
axs[0, 2].set_ylabel("True Positive Rate")
axs[0, 2].legend(loc="lower right")

# Plot sensitivity
sensitivity_model1 = tpr_model1[np.where(fpr_model1 <= 0.05)[-1][-1]]
sensitivity_model2 = tpr_model2[np.where(fpr_model2 <= 0.05)[-1][-1]]
sensitivity_model3 = tpr_model3[np.where(fpr_model3 <= 0.05)[-1][-1]]
sensitivity_model4 = tpr_model4[np.where(fpr_model4 <= 0.05)[-1][-1]]
axs[1, 0].bar([1, 2, 3, 4], [sensitivity_model1, sensitivity_model2,
sensitivity_model3, sensitivity_model4])
axs[1, 0].set_title("Sensitivity @ 5% FPR")
axs[1, 0].set_xticks([1, 2, 3, 4])
axs[1, 0].set_xticklabels(["3-MLP", "RF", "5-MLP", "Stacked
ensemble"])
```

```
# Plot specificity
specificity_model1 = 1 - fpr_model1[np.where(tpr_model1 >= 0.90)[-1][0]]
specificity_model2 = 1 - fpr_model2[np.where(tpr_model2 >= 0.95)[-1][0]]
specificity_model3 = 1 - fpr_model3[np.where(tpr_model3 >= 0.90)[-1][0]]
specificity_model4 = 1 - fpr_model4[np.where(tpr_model4 >= 0.95)[-1][0]]
axs[1, 1].bar([1, 2, 3, 4], [specificity_model1, specificity_model2,
specificity_model3, specificity_model4])
axs[1, 1].set_title("Specificity @ 95% TPR")
axs[1, 1].set_xticks([1, 2, 3, 4])
axs[1, 1].set_xticklabels(["3-MLP", "RF", "5-MLP", "Stacked
ensemble"])

# Plot AUC scores
axs[1, 2].bar([1, 2, 3, 4], [auc_model1, auc_model2, auc_model3,
auc_model4])
axs[1, 2].set_title("AUC Score")
axs[1, 2].set_xticks([1, 2, 3, 4])
axs[1, 2].set_xticklabels(["3-MLP", "RF", "5-MLP", "Stacked
ensemble"])

# Display the plot
plt.tight_layout()
plt.show()
```

OUTPUT SCREENSHOTS:

Code for Ultrasound data:

11. REFERENCES

[1] Younossi, Z. M., Koenig, A. B., Abdelatif, D., Fazel, Y., Henry, L., & Wymer, M. (2016). Global epidemiology of nonalcoholic fatty liver disease—meta‐analytic assessment of prevalence, incidence, and outcomes. Hepatology, 64(1), 73-84.

[2] Aggarwal, P., & Alkhouri, N. (2021). Artificial Intelligence in Nonalcoholic Fatty Liver Disease: A New Frontier in Diagnosis and Treatment. Clinical liver disease, 17(6), 392–397. https://doi.org/10.1002/cld.1071

[3] Atsawarungruangkit, A., Laoveeravat, P., & Promrat, K. (2021). Machine learning models for predicting non-alcoholic fatty liver disease in the general United States population: NHANES database. World Journal of Hepatology, 13(10), 1417.

[4] Liu, Y. X., Liu, X., Cen, C., Li, X., Liu, J. M., Ming, Z. Y., ... & Zheng, S. S. (2021). Comparison and development of advanced machine learning tools to predict nonalcoholic fatty liver disease: An extended study. Hepatobiliary & Pancreatic Diseases International, 20(5), 409-415.

[5] Wu, Y., Yang, X., Morris, H. L., Gurka, M. J., Shenkman, E. A., Cusi, K., ... & Donahoo, W. T. (2022). Noninvasive Diagnosis of Nonalcoholic Steatohepatitis and Advanced Liver Fibrosis Using Machine Learning Methods: Comparative Study With Existing Quantitative Risk Scores. JMIR Medical Informatics, 10(6), e36997.
[6] Mihailescu, D. M., Gui, V., Toma, C. I., Popescu, A., & Sporea, I. (2013). Computer aided diagnosis method for steatosis rating in ultrasound images using random forests. Medical Ultrasonography, 15(3), 184-190.
[7] Gaber, A., Youness, H. A., Hamdy, A., Abdelaal, H. M., & Hassan, A. M. (2022). Automatic classification of fatty liver disease based on supervised learning and genetic algorithm. Applied Sciences, 12(1), 521.
[8] Kalyan, K., Jakhia, B., Lele, R. D., Joshi, M., & Chowdhary, A. (2014). Artificial neural network application in the diagnosis of disease conditions with liver ultrasound images. Advances in bioinformatics, 2014.
[9] Han, A., Byra, M., Heba, E., Andre, M. P., Erdman Jr, J. W., Loomba, R., ... & O'Brien Jr, W. D. (2020). Noninvasive diagnosis of nonalcoholic fatty liver disease and quantification of liver fat with radiofrequency ultrasound data using one-dimensional convolutional neural networks. Radiology, 295(2), 342-350.

[10] Rhyou, S. Y., & Yoo, J. C. (2021). Cascaded deep learning neural network for automated liver steatosis diagnosis using ultrasound images. Sensors, 21(16), 5304.

[11] Cao, Y. T., Xiang, L. L., Qi, F., Zhang, Y. J., Chen, Y., & Zhou, X. Q. (2022). Accuracy of controlled attenuation parameter (CAP) and liver stiffness measurement (LSM) for assessing steatosis and fibrosis in non-alcoholic fatty liver disease: A systematic review and meta-analysis. EClinicalMedicine, 51, 101547.

[12] Yip, T. F., Ma, A. J., Wong, V. S., Tse, Y. K., Chan, H. Y., Yuen, P. C., & Wong, G. H. (2017). Laboratory parameter‐based machine learning model for excluding non‐alcoholic fatty liver disease (NAFLD) in the general population. Alimentary pharmacology & therapeutics, 46(4), 447-456.

[13] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18 (pp. 234-241). Springer International Publishing.

[14]