Parameter setting and reliability test of a sensor system for infant carrier car seat sensing in a car using a dashboard sensor

Masters of Engineering
Information Technology

Aneeta Antony(1431461)
aneeta.antony@stud.fra-uas.de

Nitu Shrestha (1428296)
nitu.shrestha@stud.fra-uas.de

Sonam Priya (1427129)
sonam.priya@stud.fra-uas.de

Abstract— **Machine learning and Artificial intelligence are the applications of machines to perform human intelligence tasks such as performing actions and making decisions autonomously to solve a problem. This project uses machine learning techniques such as CNN and MLP to detect the presence of an infant in a car seat. Using Red Pitaya sensor system mounted on car's dashboard, we have collected over 40,000 ultrasonic measurement scans, with and without the infant doll, in various environments and settings. The dataset is used to train and test the MLP and CNN models for classifications and FFT analysis is used for feature extraction. Models were optimized using various parameters so that the best results could be achieved. Results are evaluated by confusion matrix and F1 Score which in turn validated each approach using various parameter settings. Finally, a robust and reliable sensor system has been achieved which could detect an infant in a car with high accuracy.**

Keywords—ML, FFT, CNN, MLP, Confusion Matrix, Ultrasonic Sensors, Red Pitaya.

I. INTRODUCTION

In recent years, advancements in sensor technology have paved the way for innovative applications within the automotive industry. One such application is person detection in the interior of a car, which has gained significant attention due to its potential to enhance safety and security. By utilizing advanced sensors, car manufacturers can now implement systems that detect the presence of individuals inside a vehicle. These sensors are designed to accurately identify and track occupants, ensuring their well-being and enabling various safety features. In the European new car assessment programme (Euro NCAP) 2025 roadmap, the child presence detection is mandatory for safety requirements. The United States also mandates that automakers install rear occupant alert systems. Therefore, it becomes increasingly crucial to detect passengers inside the car [1]. Proper monitoring of passengers boarding a car can increase effectiveness, reliability, and safety with proper control of airbags during a crash. Achieving a reliable and low-cost system using machine learning techniques is the main goal [2].

In general, video cameras, passive infrared (PIR), and ultrasound (US) sensors can be used for occupancy detection [3]. In this project, our focus is on Ultrasonic sensors. The performance capabilities of sensors to identify passengers is crucial. For machine learning models to produce meaningful outputs that autonomous systems can employ to carry out challenging tasks, they need sensor input data.

Pre-processing raw data is essential to raising the precision and dependability of machine learning algorithms. In this context, ultrasonic signals from human and non-human targets are gathered and processed to create the systems' input data. Understanding the importance of pre-processing raw data can help us develop more sophisticated autonomous systems and improve the efficiency and dependability of machine learning algorithms [4].

These are investigated in order to determine a pre-processing method that is efficient for feature extraction. In general, the following parameters are useful in separating humans from nonhumans: maximum amplitude, reflected energy and variation in Fourier transform shape over time [4].

II. LITRATURE REVIEW

A. *Ultrasonic Sensor*

Ultrasonic Sensors are devices which use ultrasonic sound waves for various applications such as measurement of distance, detection of objects and obstacle avoidance. In general, they use sound waves which have frequencies higher that 20KHz.

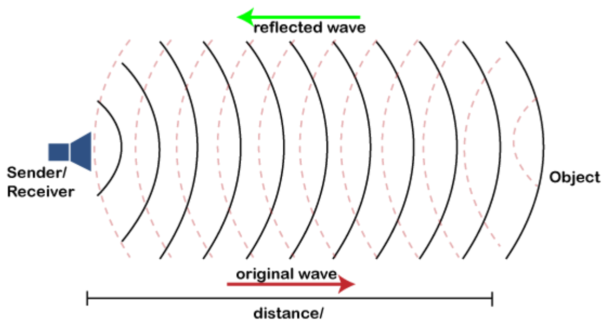


Fig. 1: Time of flight measurement of Ultrasonic sensor. Source adapted from [5].

Ultrasonic sensors operate by emitting high-frequency sound waves from a transmitter, which propagate through the air until they encounter an object in their path. Upon striking the object, the ultrasonic waves are reflected back to the sensor's receiver. The sensor then measures the time it takes for these waves to make the round trip. Using the speed of sound in the medium (usually air), the sensor calculates the distance to the object. This non-contact distance measurement capability makes ultrasonic sensors invaluable in various applications such as robotics, industrial automation, and parking assistance systems, where accurate proximity detection and obstacle avoidance are essential. Their reliability and simplicity contribute to their widespread use in diverse fields requiring precise distance measurements.

Ultrasonic sensors have the following characteristics:

- They are inexpensive.
- They are discrete and tiny.
- They are able to measure distances across smoke, dust, and humidity.
- They are low maintenance devices.

Because ultrasonic waves propagate slowly and quickly attenuate in air, they present two intrinsic challenges: (i) longer detection distance and (ii) shorter measuring time. An approach to these problems has been suggested: beam-forming with arrayed transmitters. In order to extend the detection distance, a number of research formed narrow beams for output amplification using square arrays. This directivity narrowing method works well for both raising output power and raising the precision of location estimation. To measure a large angular area with a narrow beam, many scans are needed. Without beam scanning, measurement can be done with a limited number of detections to produce a broad beam in relation to the detection plane [6].

B. *FFT (Fast Fourier Transform)*

An adept understanding of the Fast Fourier Transform is required for understanding the goals of the project. A more accurate depiction of infant presence and activities is possible through the use of Fourier Transform in sensor fusion, which enables a thorough analysis of the combined frequency components. It dissects a signal into its spectral components, yielding frequency data as an output. The utilization of FFTs extends to tasks such as analysing defects in machines and systems, ensuring quality control, and monitoring overall conditions.

The FFT (Fast Fourier Transform) is the best algorithm for implementing the DFT (Discrete Fourier Transformation). This method involves repeatedly sampling a signal in order to separate it into discrete frequency components, each of which is a single sinusoidal signal with variable amplitude, phase, and frequency [7].

C. *Red Pitaya - The sensor Equipment*

Electrical and electronic engineers use a wide variety of Tests and Measurement (T&M) instruments, which are crucial for projects and many types of engineering labs. Digital multimeters (DMM), oscilloscopes, signal generators, spectrum analyzers, frequency analyzers, and many more are examples of T&M equipment that are commonly used.

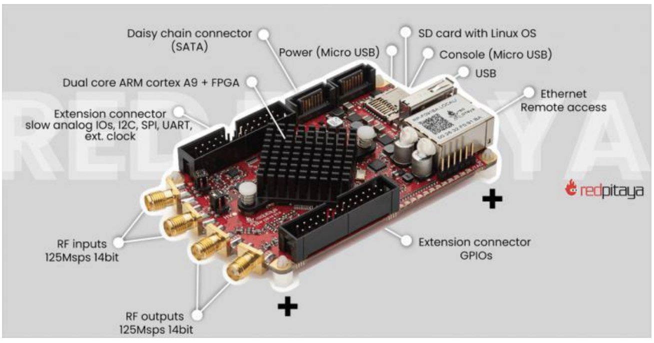
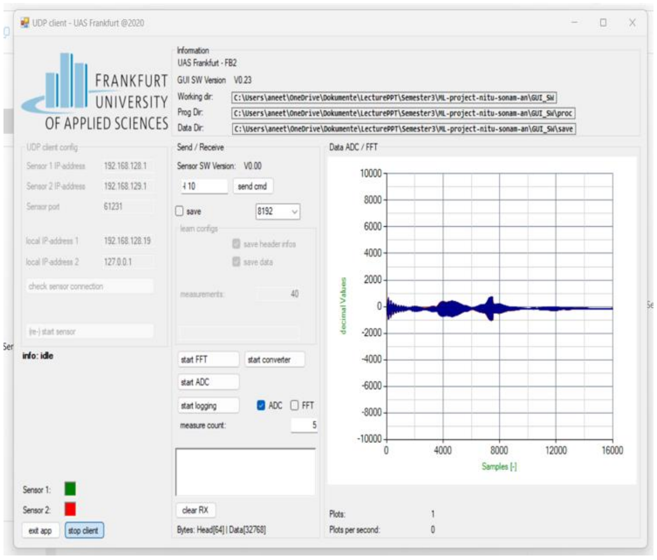

Daisy chain connector (SATA)
Power (Micro USB)
SD card with Linux OS
Console (Micro USB)
USB
Dual core ARM cortex A9 + FPGA
Ethernet Remote access
Extension connector slow analog IOs, I2C, SPI, UART, ext. clock
redpitaya
RF inputs 125Msps 14bit
Extension connector GPIOs
RF outputs 125Msps 14bit

Fig. 2: Red Pitaya Sensor. Source adapted from [8].

The Red Pitaya microchip STEM Lab board provides high bandwidth for the Analog-to-Digital Converter (ADC) and Digital-to-Analog Converter (DAC). It is a popular option for radio frequency applications due to its features, especially as a radio receiver and transmitter. The sensor equipment is the Red Pitaya sensor, available in the Lab for Autonomous System and Intelligent Sensors at Frankfurt University of Applied Sciences. To do the experiment and obtain the ADC data, the device was shipped with a preconfigured Linux system and the Lab's UDP_Client application.

D. *UDP_Client program*

To collect the data from Red Pitaya ultrasonic sensor system, we have used UDP_Client program developed by Frankfurt University of Applied Sciences. Fig 3 shows the UDP_Client program used in our program.


UDP client - UAS Frankfurt @2020

Information
UAS Frankfurt - FB2
GUI SW Version V0.23
Working dir: C:\Users\aneet\OneDrive\Dokumente\LecturePPT\Semester3\ML-project-nitu-sonam-an\GUI_SW
Prog Dir: C:\Users\aneet\OneDrive\Dokumente\LecturePPT\Semester3\ML-project-nitu-sonam-an\GUI_SW\proc
Data Dir: C:\Users\aneet\OneDrive\Dokumente\LecturePPT\Semester3\ML-project-nitu-sonam-an\GUI_SW\save

UDP client config
Sensor 1 IP-address 192.168.128.1
Sensor 2 IP-address 192.168.129.1
Sensor port 61231
local IP-address 1 192.168.128.19
local IP-address 2 127.0.0.1
info: idle
Sensor 1: 🟩
Sensor 2: 🟥

Send / Receive
Sensor SW Version: V0.00
-i 10
save 8192
learn configs
save header infos
save data
measurements: 40
ADC [✓] FFT [ ]
measure count: 5
Bytes: Head[64] | Data[32768]

Data ADC / FFT
Plots: 1
Plots per second: 0

Fig. 3: UDP Client program.

E. *Confusion Matrix*

The confusion matrix, also known as the error matrix, is depicted by a matrix describing the performance of a classification model on a set of test data Fig 4 [3].

Understanding Confusion Matrix:

- True Positives (TP): when the actual value is Positive and predicted is also Positive.
- True negatives (TN): when the actual value is Negative, and prediction is also Negative.

- False positives (FP): When the actual is negative, but prediction is Positive. Also known as the Type 1 error.
- False negatives (FN): When the actual is Positive, but the prediction is Negative. Also known as the Type 2 error.

Classification Measures:

In addition to the confusion matrix, there are several measures that help in better understanding and analyzing the performance of a classification model.

- Accuracy
  Accuracy represents the proportion of correct predictions out of the total predictions made.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- Precision
  Precision, also known as the positive predictive value, is the fraction of correctly predicted positive instances out of all predicted positive instances.

$$Precision = \frac{TP}{TP + FP}$$

- Sensitivity
  Sensitivity, recall, or the true positive rate (TPR) is the fraction of correctly identified positive instances out of all actual positive instances.

$$Sensitivity = \frac{TP}{TP + FN}$$

- Specificity
  Specificity indicates the proportion of correctly identified negative instances out of all actual negative instances. It's also known as the true negative rate.

$$Specificity = \frac{TN}{TN + FP}$$

- False Negative Rate
  The false negative rate is the proportion of actual positive instances that were incorrectly classified as negative.

$$False\ Negative\ Rate = \frac{FN}{FN + TP}$$

- F1 score
  The F1 Score is the harmonic mean of precision and sensitivity, providing a single metric to assess the model's accuracy.

$$F1\ Score = \frac{2\ X\ Precision\ X\ Sensitivity}{Precision + Sensitivity}$$

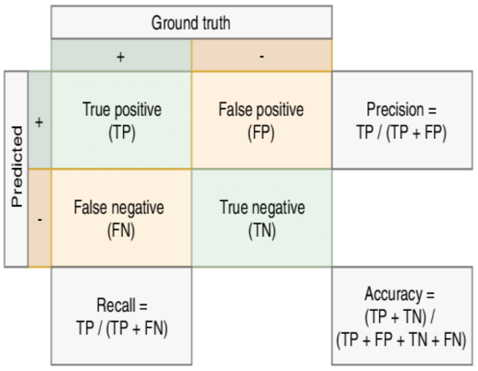


Fig. 4: Confusion Matrix.

F. CNN

In various domains such as surveillance, robotics, and autonomous vehicles, human detection plays a crucial role. With the advent of machine learning (ML) and deep learning (DL) algorithms, the performance of human detection systems has witnessed significant enhancements. Amongst the various architectures employed for human detection, convolutional neural networks (CNN) have proven to be highly successful, owing to their ability to learn and extract high-level features from images [9].

A Convolutional Neural Network (CNN) consists of input layers, multiple hidden layers, and a single output layer. Training begins with the transmission of data to the input layer. Subsequently, the data traverses through a series of hidden layers, each of which incorporates multiple filters and pooling layers. The insertion of these layers within the hidden layers enhances feature extraction, contributing to the network's overall effectiveness [10]

III. Environment Setup

The initial step is to setup the environment for building/running the project. To ensure accurate and reliable testing of the infant presence or absence inside a car seat using sensor, a carefully controlled environment is setup. This section explains the prerequisites for setting up or executing the project.

A. Experiment Setup

The Red Pitaya sensor is installed inside a car at the parking lot where the experiment is carried out. A baby doll is placed in the infant seat carrier and secured in the car seat using the seatbelt. The laptop is equipped with the UDP Client program, serving as the source for gathering data from the Red Pitaya sensor.

B. Installing Python and its packages

Initially, the system needs Python version 3.8 installed. Additionally, PyCharm, an Integrated Development Environment (IDE), was installed and utilized for development purposes. Once the previous steps have been completed successfully, the next step is to install the required packages. In this project, various packages were utilized, such as pandas, NumPy, matplotlib, and sklearn. These packages were installed in PyCharm as dependencies for the project. All dependencies were imported into the script as shown below:

import numpy as np

import os

import csv

import glob

import joblib

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

from sklearn.metrics import f1_score

from sklearn.metrics import confusion_matrix

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.model_selection import train_test_split

C. *Python Script*

Python scripts are typically developed using an Integrated Development Environment (IDE). For this project, the Python script was written using the PyCharm IDE. PyCharm offers a range of features like code analysis, a graphical debugger, an integrated unit tester, seamless integration with version control systems, as well as support for web development.

IV. Implementation

The Red Pitaya sensor is installed at the parking lot location where the experiment was carried out. The first thing we did was get acquainted with the measurement tools and procedure. For this, we measured and saved the dataset using the installed GUI_V0.23_2021-11-22, we measured and saved the dataset. With familiarization of the tools and procedures, we proceeded to take measurements. Data collection began with a dataset consisting of at least 200 events of the infant carrier without and with a baby doll, resulting in a total of at least 20,000 measurement scans. The required measurement scans are obtained with baby placed in the scanning bean of the sensor. The scanned data from sensor is then stored with ADC data and header details.

The experimental setup was properly noted, including all position conditions. This can be seen with the photos of the baby doll placed in baby seater as shown in Fig 5 and without baby doll in Fig 6. We varied systematically the lateral/vertical seat position and/or the angle of incidence of the sonic waves on to the baby doll but made sure to keep the incident beam hit the target properly to get accurate data collection.

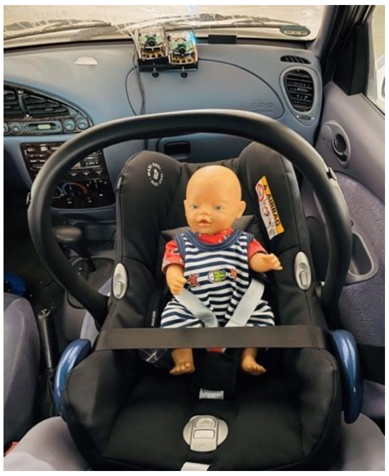


Fig. 5: Experiment with baby.

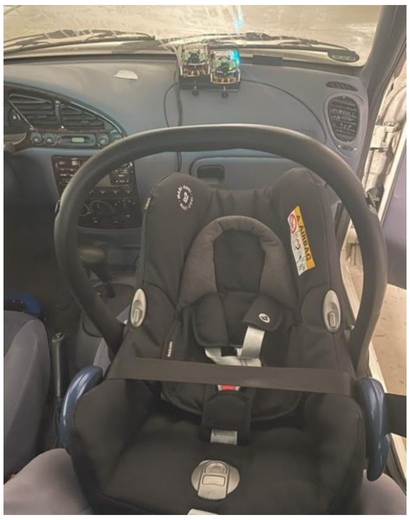
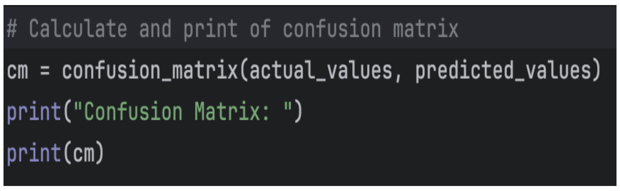


Fig. 6: Experiment without baby.

To accomplish the project's objective, we gathered readings and developed a code.

A. *Confusion Matrix Created*

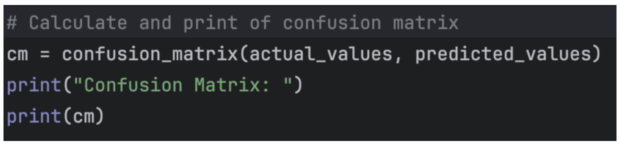
Firstly, we created a confusion matrix using the first set of 20,000 collected datasets. A confusion matrix is a table that is used to define the performance of a classification algorithm which visualizes and summarizes the performance of a classification algorithm. The table also compares the actual goal values to the predictions of the machine learning model. The code used to generate confusion matrix is show in figure 7.



```
# Calculate and print of confusion matrix
cm = confusion_matrix(actual_values, predicted_values)
print("Confusion Matrix: ")
print(cm)
```

Fig .7: Calculation and Print of Confusion matrix.

The code to display confusion matrix is:



```
# Calculate and print of confusion matrix
cm = confusion_matrix(actual_values, predicted_values)
print("Confusion Matrix: ")
print(cm)
```

Fig. 8: Display of confusion matrix.

The *Confusion Matrix* generated is shown in Fig 9. In the context of the confusion matrix utilized in this project, label '0' denotes no baby doll detection and '1' for baby doll detection in a seat.

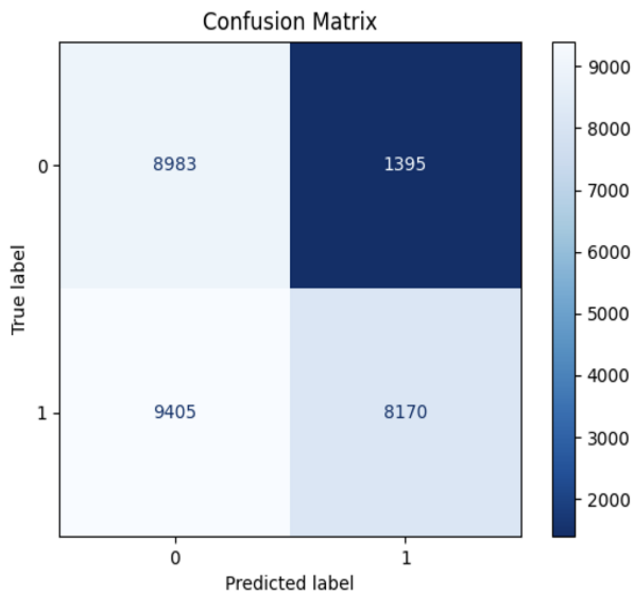
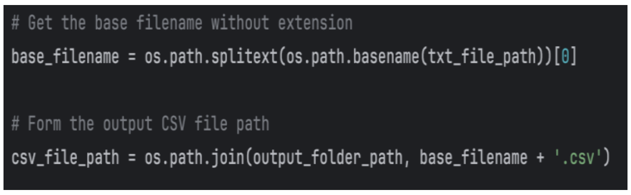


Fig 9: Confusion Matrix

B. *Conversion of ADC dataset to CSV file*

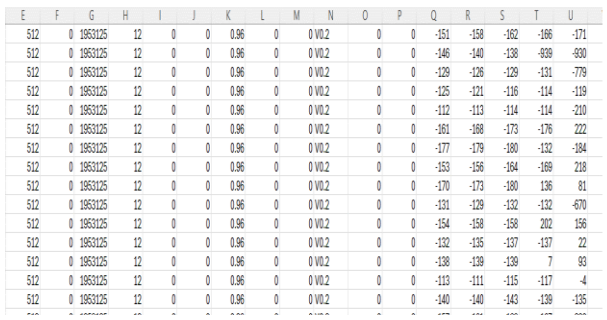
The ADC data is collected from the Red Pitaya sensor with and without baby doll is properly labeled and saved in separate text files. The text file is then converted as csv file and saved for the convenient use in our programming.



```
# Get the base filename without extension
base_filename = os.path.splitext(os.path.basename(txt_file_path))[0]

# Form the output CSV file path
csv_file_path = os.path.join(output_folder_path, base_filename + '.csv')
```

Fig. 10: Conversion of txt file to csv file.



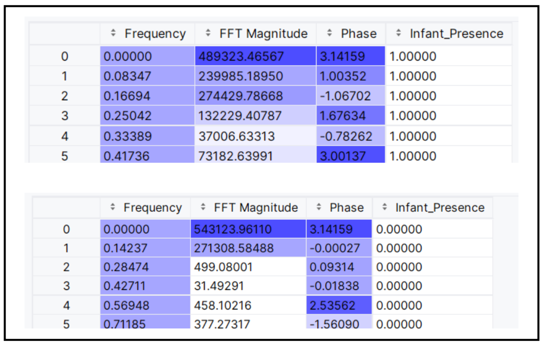
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-151	-158	-162	-166	-171
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-146	-140	-138	-939	-930
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-129	-126	-129	-131	-779
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-125	-121	-116	-114	-119
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-112	-113	-114	-114	-210
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-161	-168	-173	-176	222
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-177	-179	-180	-132	-184
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-153	-156	-164	-169	218
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-170	-173	-180	136	81
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-131	-129	-132	-132	-670
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-154	-158	-158	202	156
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-132	-135	-137	-137	22
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-138	-139	-139	7	93
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-113	-111	-115	-117	-4
512	0	1953125	12	0	0	0.96	0	0	V0.2	0	0	-140	-140	-143	-139	-135

Fig 11: ADC to csv file

C. *Calculation of FFT from the dataset*

After the conversion of ADC to csv, we calculate the FFT from the dataset. The data set is converted to FFT using NumPy inbuilt functions. The file is saved as NumPy data frame array for further use like training the MLP. During the conversion, we appropriately label the data as '1' for baby doll presence and '0' no baby doll respectively. The labeled NumPy array serves as valuable training data for machine learning models which can learn patterns from the frequency domain representation to classify instances based on the presence or absence of the baby doll.

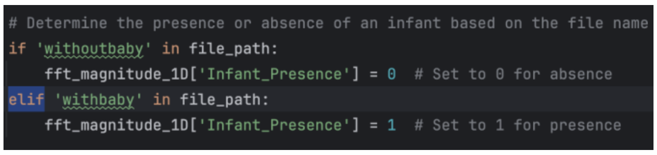
The code for FFT conversion is given in Fig 12 and labeling with/without baby in Fig 13.



	Frequency	FFT Magnitude	Phase	Infant_Presence
0	0.00000	489323.46567	3.14159	1.00000
1	0.08347	239985.18950	1.00352	1.00000
2	0.16694	274429.78668	-1.06702	1.00000
3	0.25042	132229.40787	1.67634	1.00000
4	0.33389	37006.63313	-0.78262	1.00000
5	0.41736	73182.63991	3.00137	1.00000

	Frequency	FFT Magnitude	Phase	Infant_Presence
0	0.00000	543123.96110	3.14159	0.00000
1	0.14237	271308.58488	-0.00027	0.00000
2	0.28474	499.08001	0.09314	0.00000
3	0.42711	31.49291	-0.01838	0.00000
4	0.56948	458.10216	2.53562	0.00000
5	0.71185	377.27317	-1.56090	0.00000

Fig. 12: FFT conversion Code.



```
# Determine the presence or absence of an infant based on the file name
if 'withoutbaby' in file_path:
    fft_magnitude_1D['Infant_Presence'] = 0  # Set to 0 for absence
elif 'withbaby' in file_path:
    fft_magnitude_1D['Infant_Presence'] = 1  # Set to 1 for presence
```

Fig. 13: labeling with/without baby.

D. *Random Forest Classification Modeling*

In this project, we used Random Forest (RF) classifier for modeling. Random Forest is a classifier comprising multiple decision trees trained on different subsets of the dataset. It then aggregates their predictions to enhance the overall predictive accuracy. Instead of relying on a single decision tree, Random Forest leverages the collective predictions of multiple trees, determining the final output based on the majority vote. The inclusion of a greater number of trees in the forest improves accuracy and helps mitigate overfitting is created [11].

The RF classifier was developed in Python utilizing the Scikit Learn machine learning framework. Scikit Learn seamlessly integrates with Python's NumPy and SciPy libraries, offering various classification algorithms, including Random Forest.

The dataset is divided into two types of data arrays: training and testing. Manually partitioning the dataset is unnecessary, as Scikit-learn's train-test split function automatically divides the dataset into random subsets. Additionally, a random state is specified for ensuring reproducibility of the operation.

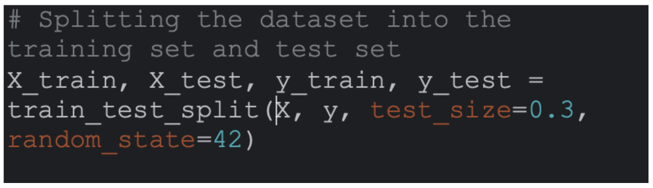
The random forest (RF) is a tree-structured ensemble learning method. The basic idea of building a random forest is [12]: First, randomly extract $\boldsymbol{K}$ new sample sets from the original data set with bootstrap resampling method, and use the new samples to construct $\boldsymbol{K}$ classification decision trees; then, suppose there are $\boldsymbol{m}$ feature, randomly select $\boldsymbol{m_{try}}$ features as candidate split attributes, and select one of $\boldsymbol{m_{try}}$ feature attributes for splitting according to the Gini index; the above process would be iterative executed, and finally, without cutting the decision tree, directly integrate the generated decision trees form a random forest to classify the new data, and the classification results are obtained by the majority voting strategy, the class with the most votes is the final classification result.

RF constructs different training sets to increase the difference between classification models, thereby improving the extrapolation prediction ability of combined classification models. Through k rounds of training, a classification model sequence $\boldsymbol{\{h_1(X), h_2(X), \ldots, h_K(X)\}}$ is obtained, and then they are used to form a multi-classification model system. The final classification result of the system adopts a simple majority voting method. Final classification decision is:

$$H(x) = \{arg_Y^{max}\} \sum_{\{i=1\}}^{K} I[\{\{h_i\}(x) = Y\}]$$

H(x) is a combined classification model, $\boldsymbol{h_t}$ is a classification model of a single decision tree, *Y* is an output class, and I(·) is an indicative function. The above equation illustrates the use of majority voting to determine the final classification [13].

The dataset is split into Training set and Test set. By default, Scikit-learn's train-test split function randomly divides the dataset into two subsets. This operation involves selecting a random state.



```
# Splitting the dataset into the
training set and test set
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3,
random_state=42)
```

Fig. 14: Splitting of dataset

- X, Y: The first option specifies the dataset to be used.
- Test size: This option determines the size of the testing dataset. The default condition is suitable for the size of the training group.
- Random state: It's a function that generates random numbers. To perform a random split, the default mode uses np. random.
- The accuracy and f1 score are calculated for this model.

The classification we used for this project is RF Classifier

Random Forest classifier is used for classification and prediction. The Random Forest classifier employs bagging techniques, utilizing decision tree classifiers as its base learners. It comprises multiple trees, each making individual predictions. The final classification decision is determined by the model based on the majority votes from these trees. Random Forest is employed to mitigate the overfitting issues commonly encountered with decision trees, leveraging the collective wisdom of multiple trees to enhance generalization performance of the model [14].

```
param_grid = {  'n_estimators': [50, 100, 150],
# Number of trees in the forest
   'max_depth': [None, 10, 20],  # Maximum depth
of the tree
   'min_samples_split': [2, 5, 10],  # Minimum
number of samples required to split an internal
node
 'min_samples_leaf': [1, 2, 4]  # Minimum number
of samples required to be at a leaf node
}
```

Fig. 15: FFT conversion Code.

For better performance of the RF classifier, we used Hyperparameter tuning. Hyperparameters are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning [15]. By tuning hyperparameters, we can enhance the performance of the model, such as improving accuracy, reducing overfitting, or increasing generalization to unseen data.

- n_estimators: This parameter specifies the number of trees in the forest. So, increasing the number of trees can improve model performance but also increases computational complexity.
- max_depth: This parameter controls the maximum depth of each tree in the forest.
- min_samples_split: It specifies the minimum number of samples required to split an internal node. Increasing this parameter can help prevent overfitting by ensuring that each leaf node contains a minimum number of samples.
- min_samples_leaf: It sets the minimum number of samples required to be at a leaf node. Increasing this parameter can also help prevent overfitting by controlling the size of the leaves in the trees.

E. *Multi-layer Perceptron (MLP) Classification Modeling*

MLP is a supervised learning algorithm that learns a function by training on a dataset. Multi-Layer Perceptron (MLP) is a type of artificial neural network including several layers of interconnected nodes, also called neurons used for various machine-learning applications, such as regression and classification [16]. The architecture of MLP consist of:

- Input layer: It consists of neurons that directly receive the features of the dataset. Each neuron in the input layer corresponds to a specific feature, and the total number of neurons in the input layer matches the total number of features in the dataset.

- Hidden layer: Between the input and output layers of an MLP classifier, there may be one or multiple hidden layers. The number of neurons in each hidden layer, a hyperparameter that can be adjusted, differs across the hidden layers. These hidden layers are crucial for capturing complex patterns within the data, allowing the model to recognize intricate relationships.

- Output layer: The output layer of the MLP classifier produces the final predictions or outputs by utilizing the information processed in the hidden layers.

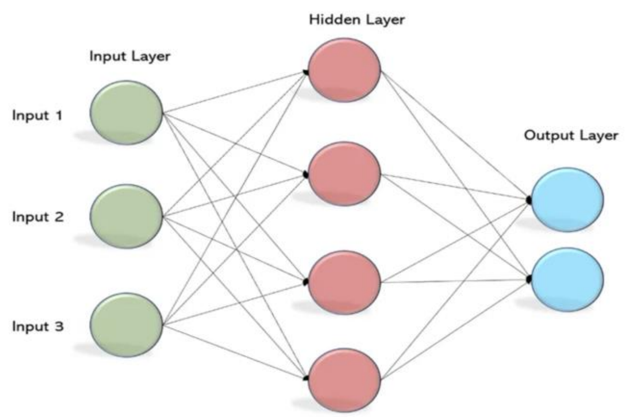


Fig. 16: MLP Classifier showing input layer, hidden layer, and output layer. Source adapted from [17].

In this Project, we have also implemented MLP classifier using Scikit-Learn. We loaded the necessary libraries using the code given below.

```
#Import MLP Classifier
from sklearn.neural_network import MLPClassifier
```

For Initializing the MLP classifier below code is used:

```
#Initializing the MLPClassifier

mlp_classifier =
MLPClassifier(hidden_layer_sizes=(7,),
activation='logistic', solver='adam', alpha=0.01,
                    batch_size='auto',
learning_rate='constant', learning_rate_init=0.01,
                    max_iter=100, shuffle=True,
random_state=42, tol=1e-3, verbose=False,
                    early_stopping=False,
validation_fraction=0.1)
```

F. *Convolutional Neural Network (CNN) Classification Modeling*

A Convolution Neural Network is a multi-layered artificial neural network capable of detecting complex features in data. In this project, a Convolutional Neural Network (CNN) classifier was employed using TensorFlow. The CNN model was constructed using the Sequential API from TensorFlow's Keras module, indicating a sequential layer-by-layer construction. The input layer was defined with a shape of (4,), specifying the input data shape. Subsequently, two dense (fully connected) layers were added, each comprising 64 and 32 neurons, respectively. Both layers utilized the Rectified Linear Unit (ReLU) activation function and L2 regularization with a regularization parameter of 0.001. To mitigate overfitting, a dropout layer with a dropout rate of 0.5 was incorporated. Finally, the output layer consisted of a single neuron with a sigmoid activation function, tailored for binary classification tasks.

The compiled model is trained on the training data (X_train and y_train) for 30 epochs, with each epoch comprising batches of size 128. This means that the model iterates through the entire training dataset 30 times, updating the model's weights after each batch of size 128.

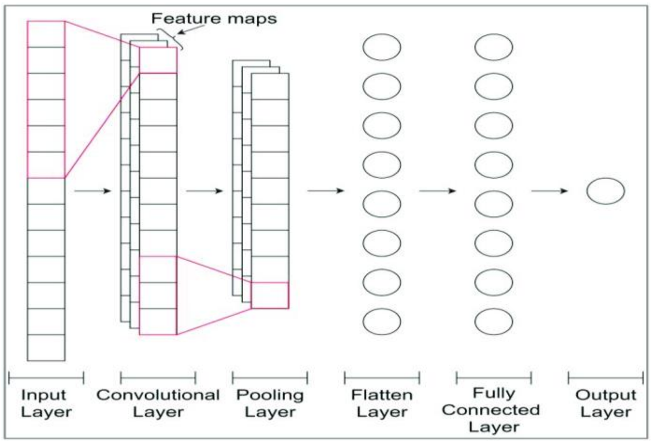


Fig. 17: Simple architecture of CNN network. Source adapted from [18].

The code to import TensorFlow is given below.

```
import tensorflow
from  tensorflow import keras
```

The code written is given below:

```
# Define a CNN classifier network
# CNN Model architecture
modelCNN.add(Conv1D(filters=64, kernel_size=3,
activation="relu", input_shape=(3, 1)))
modelCNN.add(MaxPooling1D(pool_size=1))

modelCNN.add(Flatten())
modelCNN.add(Dense(1052, activation='relu'))
modelCNN.add(Dense(1, activation='sigmoid'))  # Single
output neuron for binary classification

# Compile the model
modelCNN.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])

# Fit the model
modelCNN.fit(X_train_reshaped, y_train,  epochs=epochs,
batch_size=batch_size, verbose=verbose)

# Evaluate model
loss,accuracy =  modelCNN.evaluate(X_test, y_test,
```

batch_size=batch_size, verbose=verbose)
accuracy = accuracy 100.0 # Use accuracy metric, which is the second element in the evaluation result tuple*
print('Accuracy of Model: ', accuracy)

# Print model summary
modelCNN.summary()

V. Analysis and Results

This section provides a brief overview based on an analysis of the data set for various use cases such as detecting a baby doll or empty seat, or baby doll covered in a blanket. We experimented on the sensor position or the distance of the baby doll from the sensor in the second variation, which included the seating posture.

A. Task a, b

We familiarized ourselves with the equipment and measuring method. Then, after collection of data set with baby doll and without baby doll, we created confusion matrix. The confusion matrix for the initial dataset can be seen below.

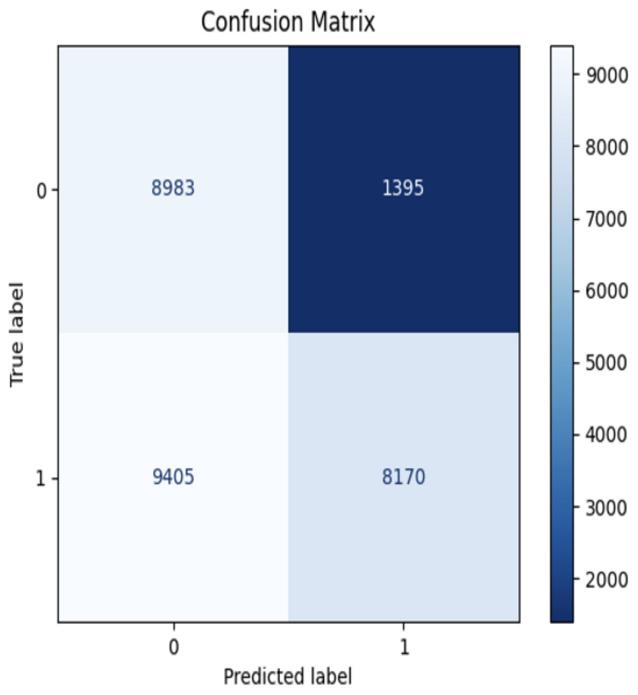


Fig. 19: Confusion matrix of initial dataset.

B. Task c

We calculated the FFT from the initial data set then created MLP program as well as RF Classifier to classify if the seat is empty or with baby doll. We used the calculated FFT data to train and test the MLP program and created the Confusion matrix. We also stored the pre-trained MLP data. The calculated confusion matrix can be seen below:

Below is the Random Forest Model characteristics:

Accuracy: 0.89
Precision: 0.93
Recall: 0.89
F1-score: 0.91
False Negative Rate: 0.11421319796954314
False Positive Rate: 0.10855949895615867

Below is the MLP Classifier Model characteristics:

Accuracy: 0.88
Precision: 0.90
Recall: 0.91
F1-score: 0.90
False Negative Rate: 0.09390862944162437
False Positive Rate: 0.16144745998608212

Below is the confusion matrix created using Random Forest and MLP classifier.

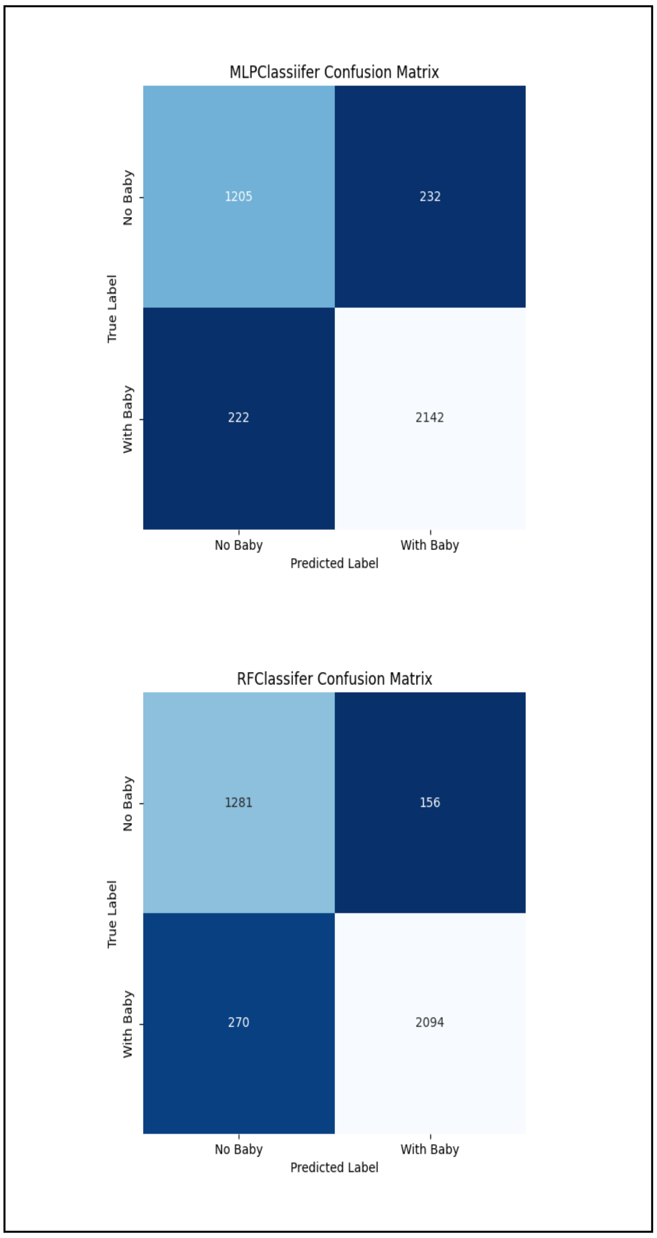


Fig. 20: The Created MLP models.

C. Task d

We collected a new set of data i.e. around 40,000 datasets and use the previously built MLP classifier and RF classifier to train the collected dataset. We calculated the confusion matrix from the trained dataset which is given in the figure Fig 20.

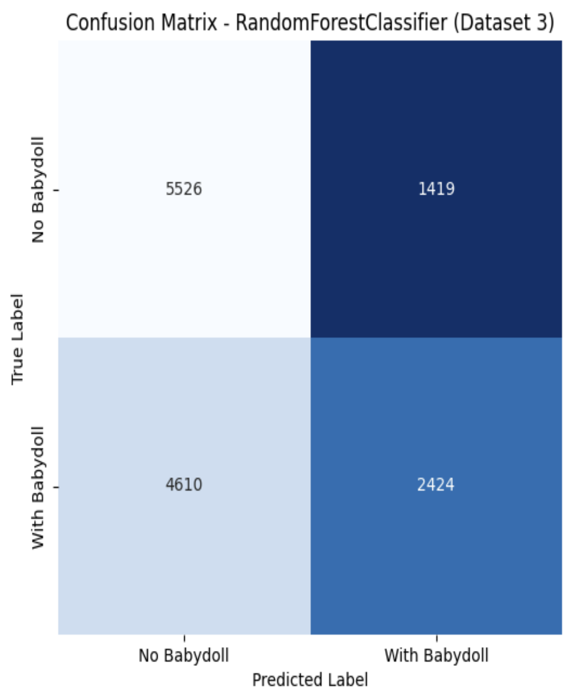


Fig. 21: Confusion Matrix using RF classifier.

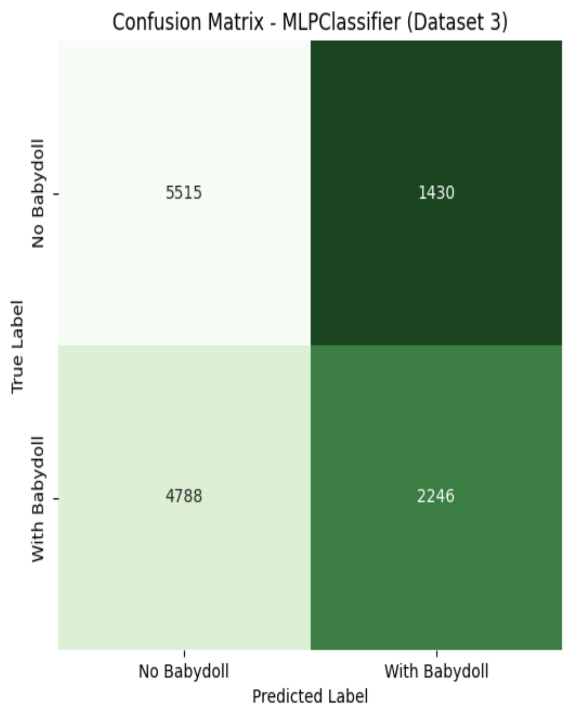


Fig. 22: Confusion Matrix using MLP classifier.

We checked the performance of the MLP classifier as well as RF classifier and got the graph which can be seen in figure. Receiver Operating Characteristic (ROC) Curve: ROC curve is a useful tool for visualizing and comparing the performance of classification.

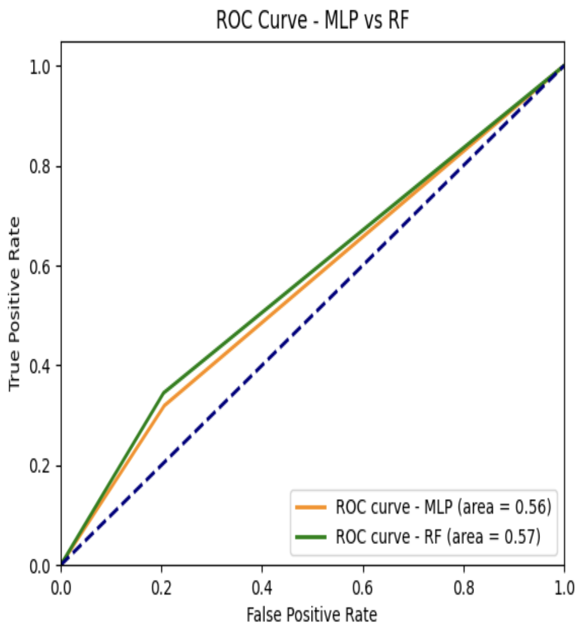


Fig. 23: ROC curve MLP vs RF classifier.

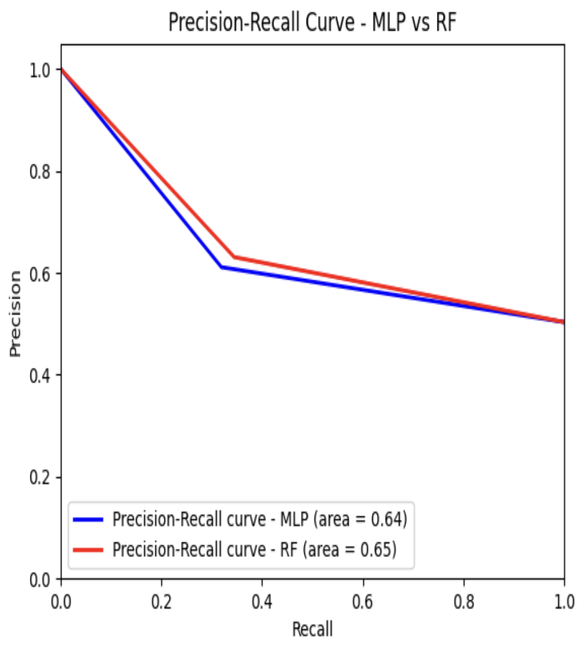


Fig. 24: Precision-Recall curve MLP vs RF classifier.

D. *Task e*

We created a CNN model for classification and trained it. We used the FFT calculated from the dataset 3 as input data and created the confusion matrix. We also stored the pre-trained CNN for further use. The confusion matrix can be seen below.

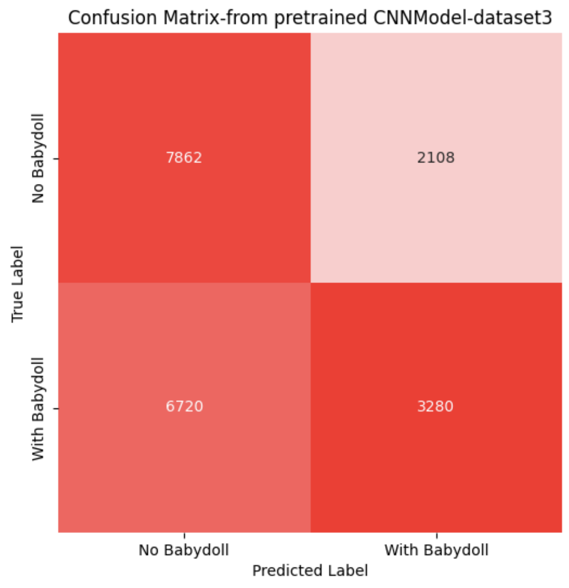


Fig. 25: Confusion Matrix from pre-trained dataset using MLP classifier.

The CNN Characteristics are:
Accuracy: 0.56
Precision: 0.61
Recall: 0.33
F1-score: 0.43
False Negative Rate: 0.672
False Positive Rate: 0.21143430290872617

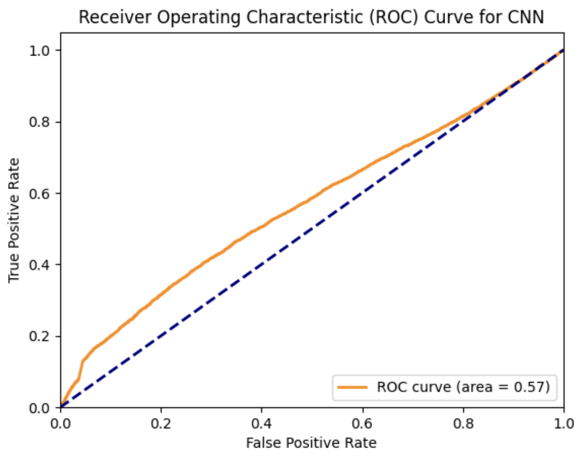


Fig. 26: ROC Curve generated using CNN.

E. Task f

Research and optimization on model

- **Optimization of MLP Classifier and RF Classifier:** In our project, we optimized MLP and RF classifiers by adjusting hyperparameters such as the number of hidden layers and learning rate for MLP, and the number of trees (n_estimators), maximum tree depth (max_depth), minimum samples required to split an internal node (min_samples_split), and minimum samples required at a leaf node (min_samples_leaf) for RF. These optimizations enhance the models' performance and generalization ability, allowing them to better handle complex classification tasks.

- **Optimization for CNN model:** The CNN model optimization involved tuning hyperparameters like the number of epochs, batch size, learning rate, and kernel size. We optimized the kernel size to improve training outcomes and adjusted the dropout regularization value to enhance performance. These refinements contributed to improved model accuracy and robustness.

After optimization, the MLP classifier and Random Forest models demonstrated improved performance. We generated ROC curves as well as Precision- Recall Curve for the MLP classifier and Random Forest (RF) classifier. The figures are given below.

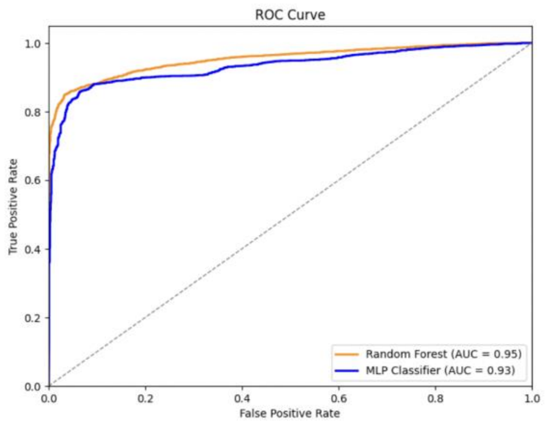


Fig. 27: ROC Curve.

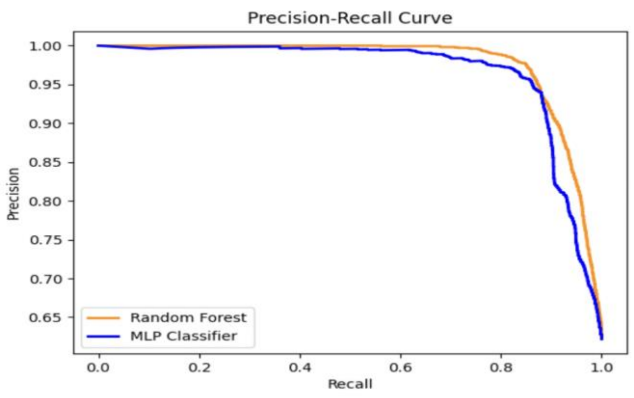


Fig. 28: Precision recall curve.

The table shows the performance metrices table of RF Classifier and MLP classifier.

	Metric	RF Classifier	MLP Classifier
1.	Accuracy	0 .887924	0.888187
2.	Precision	0.930667	0.935730
3.	Recall	0.885787	0.880711
4.	F1 Score	0.907672	0.907387

Table 1. Performance metrices of RF and MLP classifier.

How the classifiers (CNN, MLP) can be obfuscated:

- The CNN and MLP models were trained and tested across diverse datasets, revealing insights into their performance. Incorporating additional measurement labels improved accuracy but posed a risk of overfitting, obscuring the models' predictive capabilities. Emphasizing the importance of FFT magnitude, accurate and precise predictions were achieved.
- **Hyperparameter adjustments modifications**: They are vital, but certain parameters can adversely affect prediction capabilities if not optimized. Increasing layers with appropriate activation parameters enhances the model's capacity to deeply analyse each label and its associated values during training [19].
- **Obfuscated and Damaged data collected:** The presence of obfuscated or damaged data can significantly impair the classification ability and prediction precision of the model.
- **Model Complexity***:* The models we have used in our project, including CNN and MLP has very high probability of capturing very complex patterns. Because of this, there are high chances of overfitting, which eventually leads to uncertain results [20].
- **Noise and Variability***:* Dataset captured for our experiments are not completely noise free. Noise appears in the dataset because of multiple reasons. Because of this, model might not be able to identify the meaningful patterns, which could lead to uncertainty in the classification of result [21].
- **Data Imbalance**: Even distribution of the classes in the dataset is a key requirement for the success of our model. However, due to the imbalance in class distribution of our data, model
- **Boundary Cases:** Classification results are not very accurate when the predictions revolve around the edge cases scenarios. Minor changes near the edge case could produce false positive results, leading to underconfident classifications [22].

For Dataset 4: Baby covered in scarf.

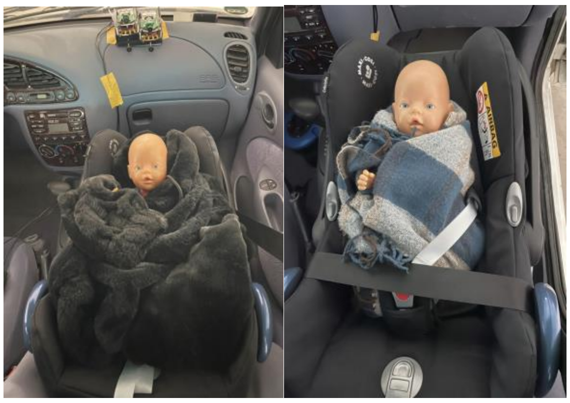


Fig. 29: Baby doll covered scarf.

Dataset 4 is applied to the MLP model to verify the label prediction precision and accuracy. Following table shows the results.

MLPClassifier Evaluation:	RandomForestClassifier Evaluation:
False Negative Rate: 0.5769166666666666 False Positive Rate: 0.20521564694082248 Cross-validation scores: [0.70650574 0.70650574 0.70650574 0.70650574 0.70650574] Mean CV score: 0.7065057403591404 Accuracy: 0.53 Precision: 0.83 Recall: 0.42 F1-score: 0.56	False Negative Rate: 0.5725833333333333 False Positive Rate: 0.2046138415245737 Cross-validation scores: [0.69634972 0.67692081 0.65911098 0.69561378 0.69443627] Mean CV score: 0.6844863114512806 Accuracy: 0.54 Precision: 0.83 Recall: 0.43 F1-score: 0.57

Table 2. MLP Classifier and RF classifier Evaluation for Dataset4.

The predicted Label Distribution over Baby with scarf and without baby like below:

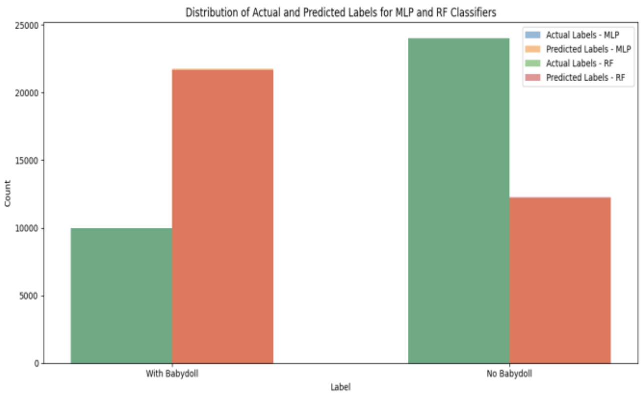


Fig. 30 Distribution of actual predicted labels for MLP and RF classifiers.

Dataset 5: Baby in carriage with sunshade pulled up.

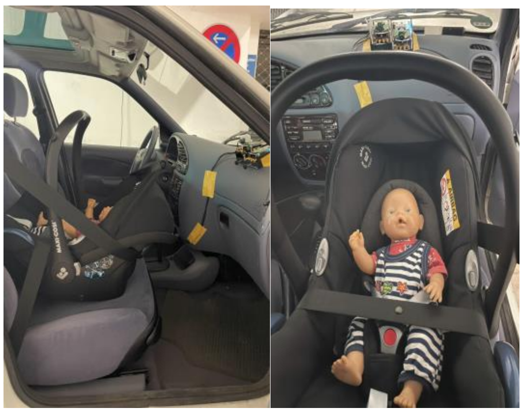


Fig. 31: Baby doll with sunshade pulled up.

Dataset 5 is applied to the MLP model and following results were obtained.

MLPClassifier Evaluation:	RandomForestClassifier Evaluation:
False Negative Rate: 0.197688054950577798 False Positive Rate: 0.20521564694082248 Cross-validation scores: [0.70949338 0.77110908 0.72341397 0.78429582 0.75850262] Mean CV score: 0.7493629757392901 Accuracy: 0.80 Precision: 0.82 Recall: 0.80 F1-score: 0.81	False Negative Rate: 0.17858937845535267 False Positive Rate: 0.2046138415245737 Cross-validation scores: [0.8767686 0.775445 0.72706527 0.78361105 0.87674047] Mean CV score: 0.8079260772030086 Accuracy: 0.81 Precision: 0.83 Recall: 0.82 F1-score: 0.82

Table 3. MLP Classifier and RF classifier Evaluation using Dataset 5.

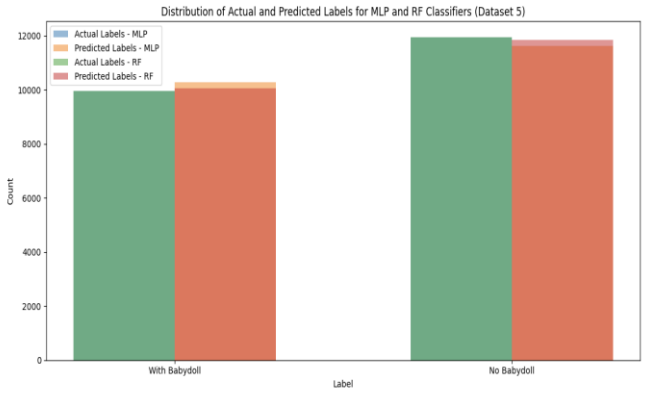


Fig. 32: Distribution of actual predicted labels for MLP and RF classifiers.

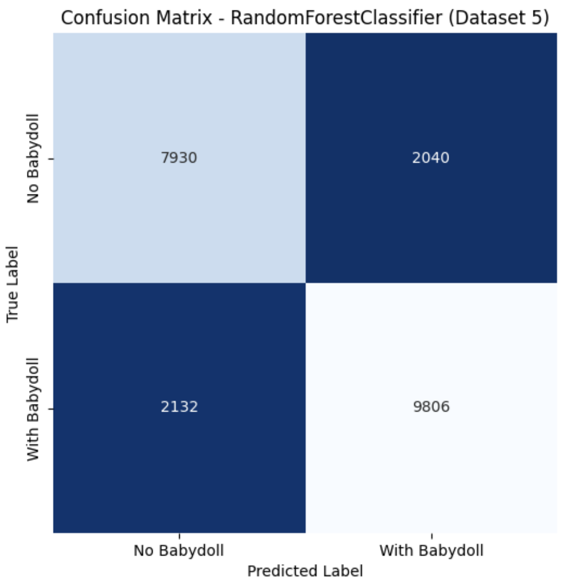


Fig. 33: Confusion matrix with dataset 5 for RF classifier.

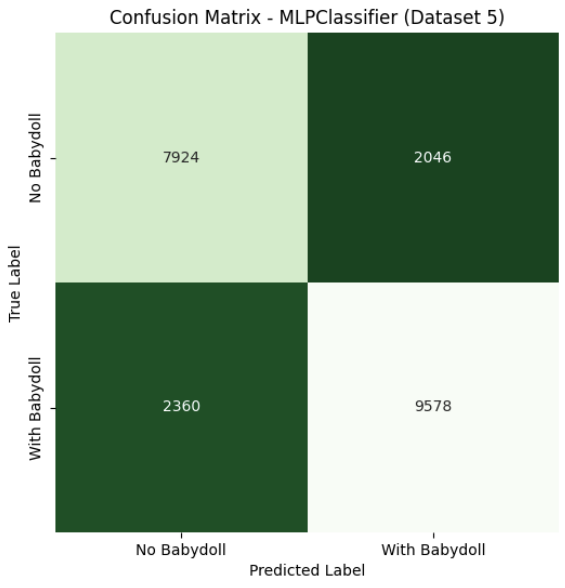


Fig. 34: Confusion matrix with dataset 5 for MLP classifier.

With Dataset 5, much better prediction, precision, and accuracy has been obtained.

Analysis on FFT of False predictions (False Positive Rate/ False Negative Rate).

The initial data set which was used to predict baby doll presence label, the MLP had a good FP Rate and FN Rate. However, the second dataset on testing on the pretrained Model showed fluctuations in the False negatives as well in case of False positives.

- We observed a variation in the sensor data we have obtained. The data after conversion to FFT were having different frequency range as compared to the experiment set up done initially, which gave us tampered dataset reading. So, to counter that, we have taken another set of readings to create a dataset, converted it to FFT with properly tuned sampling rate and window function and tested on the MLP and CNN. This has improved the FP and FN rate significantly.

- False predictions of the label were also observed. The reason behind this was the addition of unwanted experiment measurements including senor to baby carriage handle or sensor to baby head. This has led to overfitting of the model. The problem with the overfitting model of CNN and MLP is that it learns or memorize the training data instead of generalizing from it. This results in poor predictions on unknown dataset and scenarios. Overfitting models could indeed affect false predictions by increasing the likelihood of misclassification due to the model's overly specific fitting to the training set. [23]

VI. Conclusion

In our project, we have conducted experiments to verify and test parameters and also created a reliable, precise and accurate sensor system for infant carrier seat sensing in a car using a dashboard sensor (Red Pitaya).We have carried out 4 sets of data collection , in multiple scenarios including baby doll present, baby doll absent, baby doll covered in scarf and baby doll with the sunshade pulled up, with all having a minor movement of the baby. Starting with huge dataset collection of the raw ADC scan values to train and generate appropriate MLP Models, we modified various hyper parameter during each milestone to normalize and make a better and stable MLP for prediction of baby doll's presence in passenger seat. During the process, Confusion matrix, False negative and False positive rates were verified and taken into consideration. With this, we have reached a plateau of 65-70 % accuracy and precision of label prediction. The F1-Score comparison with the tasks we have performed over the 3 different models are like 0.46, 0.57, 0.82 and 0.90, thereby validating our improvements. To make a comparison with MLP, a CNN model has been created. From the research and analysis of the results obtained, we concluded that a proper label prediction within a range of 57-90 % has been achieved. Further improvements could be achieved with greater number of experiments with Bulk dataset. Moreover, including measurement parameters like distance sensor to baby head, the passenger seat carriage and the window could improve the F1 Score and the reliability.

Acknowledgment

We extend our sincere appreciation to Prof. Dr. Andreas Pech, our esteemed project advisor, whose valuable guidance has greatly contributed to the success of our project. The insights gained from the "Machine Learning" lectures proved invaluable in guiding us through various phases of the project. Collaboration among group members played a pivotal role in accomplishing this assignment. Additionally, we would like to acknowledge the abundance of online resources, reference materials, and books that aided us in achieving our goals. Special gratitude is owed to our mentor, Prof. Umansky, whose valuable support, and guidance were instrumental in overcoming challenges encountered throughout the project.

GitHub Link: https://github.com/antonyaneeta/ML-WiSe23-24.git

All Data sets available in OneDrive: DataSet1

Works Cited

[1] H. Song, Y. Yoo and H.-C. Shin, "In-Vehicle Passenger Detection Using FMCW Radar," 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9334014. [Accessed 2 march 2024].

[2] H. Abedi, C. Magnier and G. Shaker, "Passenger Monitoring Using AI-Powered Radar," 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9518503. [Accessed 2 March 2024].

[3] J. H. Jeppesen, R. H. Jacobsen, F. Inceoglu and T. S. Toftegaard, "A cloud detection algorithm for satellite imagery based on deep learning," 24 May 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425719301294?via%3Dihub. [Accessed 13 March 2024].

[4] S.-Y. Kwon and S. Lee, In-Vehicle Seat Occupancy Detection Using Ultra-Wideband Radar Sensors, Gdansk: 2022 23rd International Radar Symposium (IRS), 2022, pp. 275-278.

[5] JavaTpoint, "Sonar Definition," [Online]. Available: https://www.javatpoint.com/sonar. [Accessed March 2024].

[6] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," March 2021. [Online]. Available: https://www.researchgate.net/publication/350297716_Machine_Learning_Algorithms_Real-World_Applications_and_Research_Directions. [Accessed 13 March 2024].

[7] C. STOLL, "Introduction to Fast Fourier Transform (FFT) Analysis," Vibration Research Corp, 2024 . [Online]. Available: https://vibrationresearch.com/blog/fast-fourier-transform-fft-analysis/. [Accessed 13 March 2024].

[8] H. Interactive and Actuado, "STEMlab 125-14," RedPitaya, 2024. [Online]. Available: https://redpitaya.com/stemlab-125-14/?utm_source=google&utm_medium=cpc&utm_campaign=branded_products_s_worldwide&utm_content=a&utm_term=stemlab%20125%2014&utm_campaign=Branded+Products_(S)_Worldwide&utm_source=adwords&utm_medium=ppc&hsa_acc=1801228039&hsa. [Accessed 25 March 2024].

[9] A. Tsujii, T. Kasashima, H. Hatano and T. Yamazato, "Position Estimation of Slowly Moving Obstacles Using Ultrasonic Sensor Array," in *2022 IEEE International Ultrasonics Symposium (IUS)*, Venice, Italy, 2022, pp. 1-4.

[10] A. Upreti, "Convolutional Neural Network (CNN). A Comprehensive Overview," August 2022. [Online]. Available: https://www.researchgate.net/publication/362742399_Convolutional_Neural_Network_CNN_A_comprehensive_overview. [Accessed 15 March 2024].

[11] K. S. Gil, V. Anand, R. Gupta and P.-A. Hsiung, "Detection of Malware Using Machine Learning techniques on Random Forest, Decision Tree, KNeighbour, AdaBoost, SGD, ExtraTrees and GaussianNB Classifier," October 2023. [Online]. Available: https://www.researchgate.net/publication/376674723_Detection_of_Malware_Using_Machine_Learning_techniques_on_Random_Forest_Decision_Tree_KNeighbour_AdaBoost_SGD_ExtraTrees_and_GaussianNB_Classifier/references. [Accessed March 2024].

[12] K. Fang, J. Wu, J. Zhu and B. Xie, "A review of technologies on random forests," *Statistics and Information Forum*, vol. A26, pp. 32-38, 2011.

[13] D. Yuan, X. Y. J. Huang and J. Cui, "Improved random forest classification approach based on hybrid clustering selection," Shanghai, China, Chinese Automation Congress.

[14] A. Malik and R. Mangrulkar, "Random Forest Regression," 2022. [Online]. Available: https://www.sciencedirect.com/topics/engineering/random-forest#:~:text=Random%20Forest%20(RF)%20algorithm%20is,predicted%20by%20the%20individual%20trees.. [Accessed 25 March 2024].

[15] K. Nyuytiymbiy, "Parameters and Hyperparameters in Machine Learning and Deep Learning," 30 December 2020. [Online]. Available: https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac. [Accessed 24 March 2024].

[16] A. Nair, "Beginner's Guide To Scikit-Learn's MLP Classifier," 20 June 2019. [Online]. [Accessed 25 March 2024].

[17] R. Polanitzer, "A Multi-layer Perceptron Classifier in Python; Predict Digits from Gray-Scale Images of Hand-Drawn Digits from 0 Through 9," 29 December 2021. [Online]. Available: https://medium.com/@polanitzer/a-multi-layer-perceptron-

classifier-in-python-predict-digits-from-gray-scale-images-of-hand-drawn-44936176be33. [Accessed 25 March 2024].

[18] E. C. Nisa and Y. D. Kuan, "Comparative Assessment to Predict and Forecast Water-Cooled Chiller Power Consumption Using Machine Learning and Deep Learning Algorithms," January 2021. [Online]. Available: https://www.researchgate.net/publication/348502722_Comparative_Assessment_to_Predict_and_Forecast_Water-Cooled_Chiller_Power_Consumption_Using_Machine_Learning_and_Deep_Learning_Algorithms. [Accessed 26 March 2024].

[19] Panjeh, "scikit learn hyperparameter optimization for MLPClassifier," June 2023. [Online]. Available: https://panjeh.medium.com/scikit-learn-hyperparameter-optimization-for-mlpclassifier-4d670413042b. [Accessed 25 March 2024].

[20] M. A. Fatima Zahrae El-Hassani, N.-E. Joudar and K. Haddouch, "A New Optimization Model for MLP Hyperparameter Tuning: Modeling and Resolution by Real-Coded Genetic Algorithm," 4 March 2024. [Online]. Available: https://link.springer.com/article/10.1007/s11063-024-11578-0. [Accessed 28 March 2024].

[21] M. Kordos and A. Rusiecki, "Reducing noise impact on MLP training," 01 May 2015. [Online]. Available: https://link.springer.com/article/10.1007/s00500-015-1690-9. [Accessed 27 March 2024].

[22] "Classification and verification through the combination of the multi-layer perceptron and auto-association neural networks," August 2005. [Online]. Available: https://www.researchgate.net/publication/4202421_Classification_and_verification_through_the_combination_of_the_multi-layer_perceptron_and_auto-association_neural_networks. [Accessed 28 March 2024].

[23] H. Zou and T. Hastie, "Regularization and Variable Selection Via the Elastic Net".