# Applying deep learning/ machine learning algorithms to classify patients using EEG Signals

Aziz Zafar

June 10, 2023

## 1 Project Description

In this project, the task is to investigate and compare the performance of machine learning (ML), and deep learning (DL) algorithms as classification techniques for event-related potentials (ERPs) waveform. The focus is to analyze and evaluate the effectiveness of ML and DL algorithms in accurately classifying subjects based on their ERPs patterns. By Exploring and comparing the performance of these classification techniques, this thesis aims to contribute to the understanding and advancing ERPs-based classification methods in the field of neuroscience and cognitive research.

## 2 How to Install and Run the Project

The code and dataset used for this thesis have been included as part of the "attachment.zip" file, which is provided alongside this thesis. The "attachment.zip" file contains all the necessary files and resources to reproduce and verify the results presented in this thesis.

### 2.1 Raw

This file contains the raw EEG signal data for 91 patients, with 84 patients included in the study. Seven patients were excluded for reasons such as missing target stimuli annotation or subject group labels.

## 2.2 preprocessed_data

This file contains the clean and sorted epochs of EEG signals ready for feature extraction and classification.

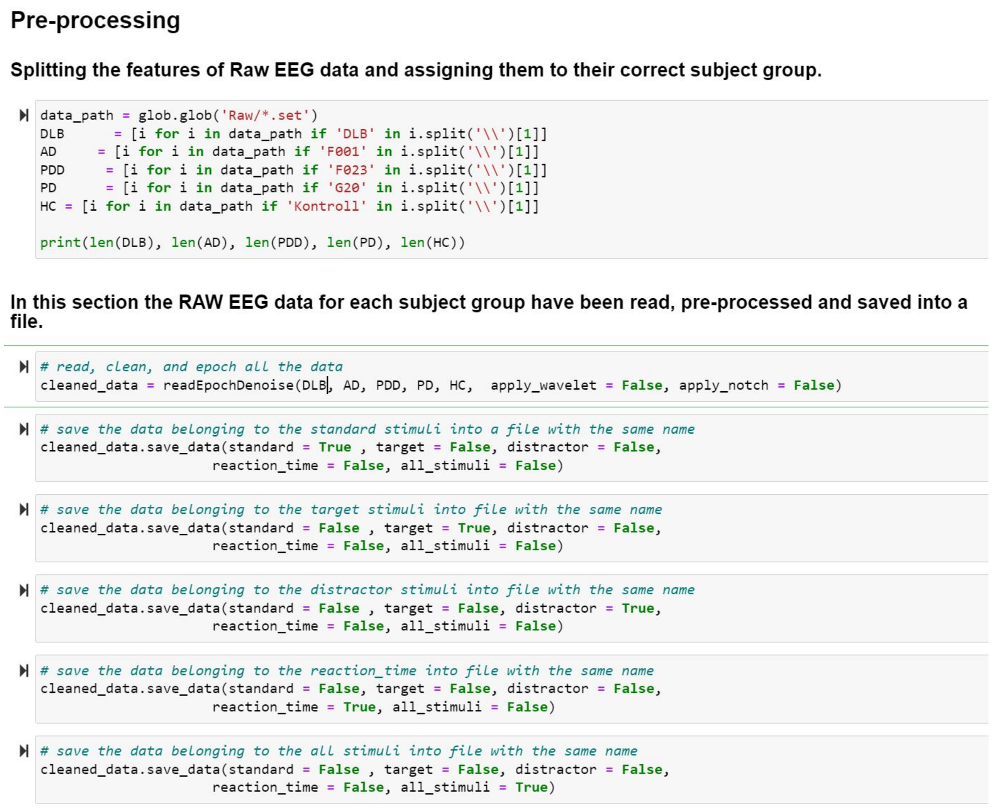
## 2.3 pasient_data

This Excel file contains information about the patients, and it's used to find the characteristics of the subject groups.

## 2.4 preprocessing.py

The "preprocessing.py" file includes a Python script that consists of two classes. The first class is responsible for reading, cleaning, and epoching the raw EEG data for each subject group individually. It contains methods or functions to perform data cleaning techniques, such as noise removal or artifact rejection, and epoching the data into segments of interest.

The second class utilizes the first class to read the data for each subject group, apply the necessary preprocessing steps using the methods from the first class, and save the preprocessed data into the "preprocessed_data" file as NumPy's ndarrays.


Pre-processing

Splitting the features of Raw EEG data and assigning them to their correct subject group.

```
data_path = glob.glob('Raw/*.set')
DLB      = [i for i in data_path if 'DLB' in i.split('\\')[1]]
AD     = [i for i in data_path if 'F001' in i.split('\\')[1]]
PDD     = [i for i in data_path if 'F023' in i.split('\\')[1]]
PD      = [i for i in data_path if 'G20' in i.split('\\')[1]]
HC = [i for i in data_path if 'Kontroll' in i.split('\\')[1]]

print(len(DLB), len(AD), len(PDD), len(PD), len(HC))
```

In this section the RAW EEG data for each subject group have been read, pre-processed and saved into a file.

```
# read, clean, and epoch all the data
cleaned_data = readEpochDenoise(DLB, AD, PDD, PD, HC,  apply_wavelet = False, apply_notch = False)
```

```
# save the data belonging to the standard stimuli into a file with the same name
cleaned_data.save_data(standard = True , target = False, distractor = False,
                       reaction_time = False, all_stimuli = False)
```

```
# save the data belonging to the target stimuli into file with the same name
cleaned_data.save_data(standard = False , target = True, distractor = False,
                       reaction_time = False, all_stimuli = False)
```

```
# save the data belonging to the distractor stimuli into file with the same name
cleaned_data.save_data(standard = False , target = False, distractor = True,
                       reaction_time = False, all_stimuli = False)
```

```
# save the data belonging to the reaction_time into file with the same name
cleaned_data.save_data(standard = False, target = False, distractor = False,
                       reaction_time = True, all_stimuli = False)
```

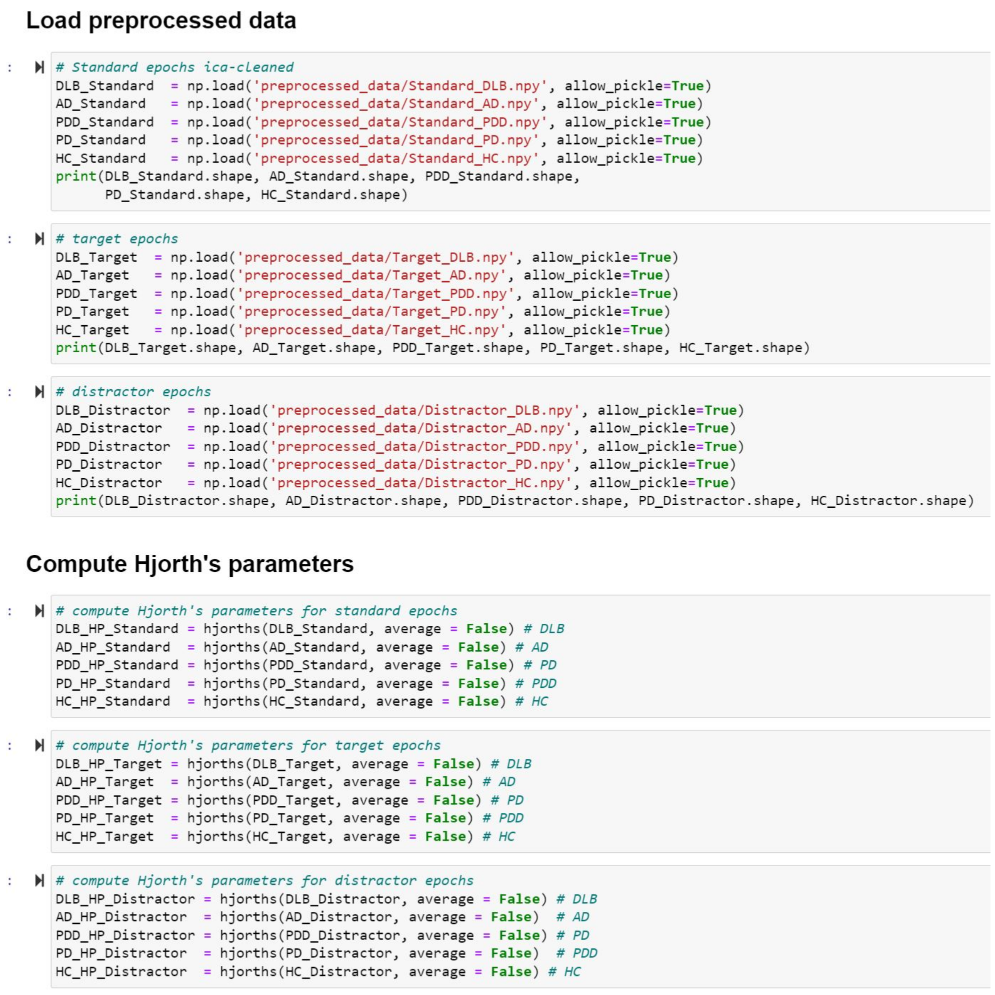
```
# save the data belonging to the all stimuli into file with the same name
cleaned_data.save_data(standard = False , target = False, distractor = False,
                       reaction_time = False, all_stimuli = True)
```

Figure 1: This screenshot illustrates the process of reading the path storing EEG signal data and subsequently splitting the data based on subject groups. Each subject group's data is then assigned to a separate list for easy organization and analysis. The lists are then fed into the second class in "preprocessing.py" for being preprocessed and saved into the "preprocessed_data" file.

## 2.5 Hjorths.py

The "Hjorths.py" file contains a Python script with two classes. The first class takes the preprocessed data as input and performs computations to derive Hjorth's parameters, which are descriptors of the EEG signal's mobility, complexity, and activity. The class then returns a dataframe containing the calculated parameters for the respective subject group.

The second class utilizes the results obtained from the first class. It assigns labels to the subject groups and their orresponding classes



**Load preprocessed data**

```
# Standard epochs ica-cleaned
DLB_Standard  = np.load('preprocessed_data/Standard_DLB.npy', allow_pickle=True)
AD_Standard   = np.load('preprocessed_data/Standard_AD.npy', allow_pickle=True)
PDD_Standard  = np.load('preprocessed_data/Standard_PDD.npy', allow_pickle=True)
PD_Standard   = np.load('preprocessed_data/Standard_PD.npy', allow_pickle=True)
HC_Standard   = np.load('preprocessed_data/Standard_HC.npy', allow_pickle=True)
print(DLB_Standard.shape, AD_Standard.shape, PDD_Standard.shape,
      PD_Standard.shape, HC_Standard.shape)
```

```
# target epochs
DLB_Target  = np.load('preprocessed_data/Target_DLB.npy', allow_pickle=True)
AD_Target   = np.load('preprocessed_data/Target_AD.npy', allow_pickle=True)
PDD_Target  = np.load('preprocessed_data/Target_PDD.npy', allow_pickle=True)
PD_Target   = np.load('preprocessed_data/Target_PD.npy', allow_pickle=True)
HC_Target   = np.load('preprocessed_data/Target_HC.npy', allow_pickle=True)
print(DLB_Target.shape, AD_Target.shape, PDD_Target.shape, PD_Target.shape, HC_Target.shape)
```

```
# distractor epochs
DLB_Distractor  = np.load('preprocessed_data/Distractor_DLB.npy', allow_pickle=True)
AD_Distractor   = np.load('preprocessed_data/Distractor_AD.npy', allow_pickle=True)
PDD_Distractor  = np.load('preprocessed_data/Distractor_PDD.npy', allow_pickle=True)
PD_Distractor   = np.load('preprocessed_data/Distractor_PD.npy', allow_pickle=True)
HC_Distractor   = np.load('preprocessed_data/Distractor_HC.npy', allow_pickle=True)
print(DLB_Distractor.shape, AD_Distractor.shape, PDD_Distractor.shape, PD_Distractor.shape, HC_Distractor.shape)
```

**Compute Hjorth's parameters**

```
# compute Hjorth's parameters for standard epochs
DLB_HP_Standard = hjorths(DLB_Standard, average = False) # DLB
AD_HP_Standard  = hjorths(AD_Standard, average = False) # AD
PDD_HP_Standard = hjorths(PDD_Standard, average = False) # PD
PD_HP_Standard  = hjorths(PD_Standard, average = False) # PDD
HC_HP_Standard  = hjorths(HC_Standard, average = False) # HC
```

```
# compute Hjorth's parameters for target epochs
DLB_HP_Target = hjorths(DLB_Target, average = False) # DLB
AD_HP_Target  = hjorths(AD_Target, average = False) # AD
PDD_HP_Target = hjorths(PDD_Target, average = False) # PD
PD_HP_Target  = hjorths(PD_Target, average = False) # PDD
HC_HP_Target  = hjorths(HC_Target, average = False) # HC
```

```
# compute Hjorth's parameters for distractor epochs
DLB_HP_Distractor = hjorths(DLB_Distractor, average = False) # DLB
AD_HP_Distractor  = hjorths(AD_Distractor, average = False)  # AD
PDD_HP_Distractor = hjorths(PDD_Distractor, average = False) # PD
PD_HP_Distractor  = hjorths(PD_Distractor, average = False)  # PDD
HC_HP_Distractor  = hjorths(HC_Distractor, average = False) # HC
```

Figure 2: The screenshot demonstrates the process of loading preprocessed data and feeding them into the "hjorths" class for feature extraction

## 2.6 ML.py

The "ML.py" file consists of five classes that handle various classification and result analysis tasks.

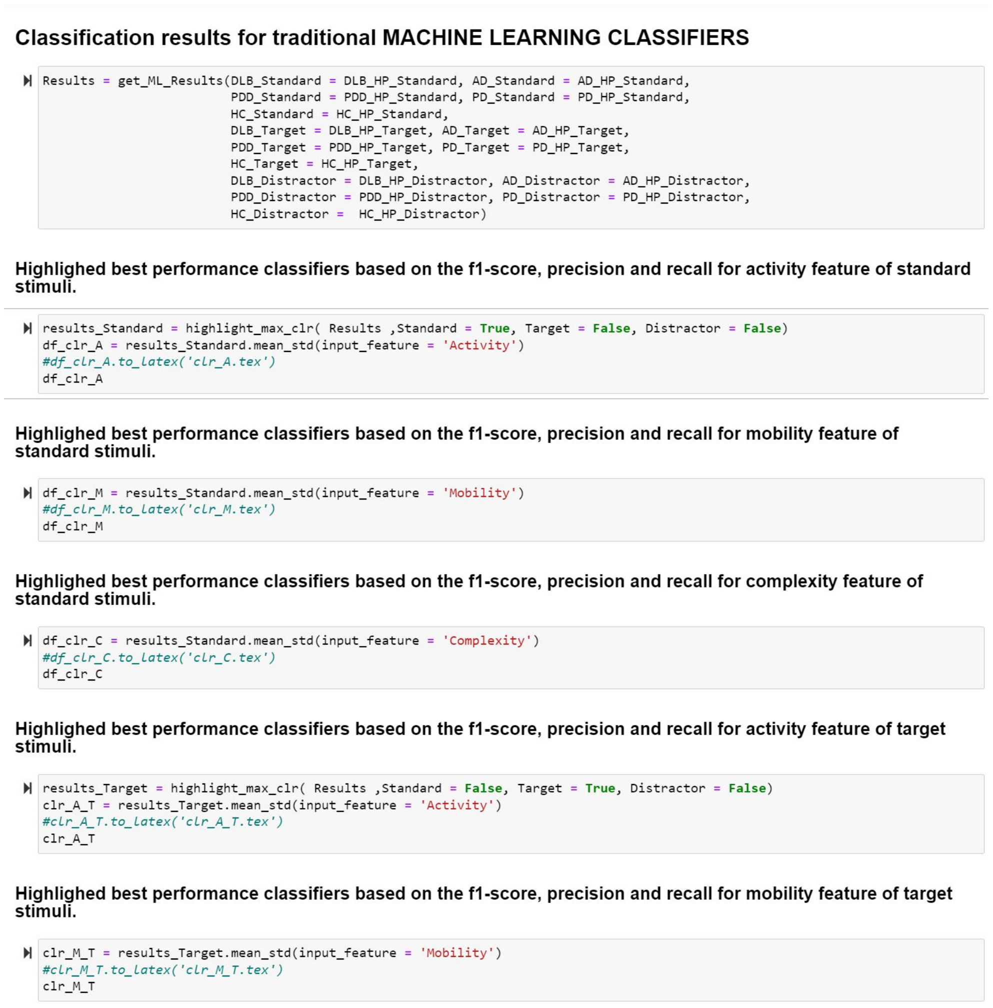
The first class is responsible for classifying the subjects into groups based on a given dataset. It takes the dataset as input and applies a classification algorithm to assign subjects to specific groups. The class then returns the performance results, such as accuracy, f1-score, precision, recall, and confusion matrix.

The second class iterates through the datasets, applies the classification algorithm using the first class, and collects the results for easy access.

The third class is responsible for identifying classifiers with the highest performance based on metrics like f1-score, precision, and recall.

The fourth class highlights the best performance by accuracy using the accuracies obtained from the second class.

The fifth class is used for plotting confusion matrices.


**Classification results for traditional MACHINE LEARNING CLASSIFIERS**

```
Results = get_ML_Results(DLB_Standard = DLB_HP_Standard, AD_Standard = AD_HP_Standard,
                         PDD_Standard = PDD_HP_Standard, PD_Standard = PD_HP_Standard,
                         HC_Standard = HC_HP_Standard,
                         DLB_Target = DLB_HP_Target, AD_Target = AD_HP_Target,
                         PDD_Target = PDD_HP_Target, PD_Target = PD_HP_Target,
                         HC_Target = HC_HP_Target,
                         DLB_Distractor = DLB_HP_Distractor, AD_Distractor = AD_HP_Distractor,
                         PDD_Distractor = PDD_HP_Distractor, PD_Distractor = PD_HP_Distractor,
                         HC_Distractor =  HC_HP_Distractor)
```

**Highlighed best performance classifiers based on the f1-score, precision and recall for activity feature of standard stimuli.**

```
results_Standard = highlight_max_clr( Results ,Standard = True, Target = False, Distractor = False)
df_clr_A = results_Standard.mean_std(input_feature = 'Activity')
#df_clr_A.to_latex('clr_A.tex')
df_clr_A
```

**Highlighed best performance classifiers based on the f1-score, precision and recall for mobility feature of standard stimuli.**

```
df_clr_M = results_Standard.mean_std(input_feature = 'Mobility')
#df_clr_M.to_latex('clr_M.tex')
df_clr_M
```

**Highlighed best performance classifiers based on the f1-score, precision and recall for complexity feature of standard stimuli.**

```
df_clr_C = results_Standard.mean_std(input_feature = 'Complexity')
#df_clr_C.to_latex('clr_C.tex')
df_clr_C
```

**Highlighed best performance classifiers based on the f1-score, precision and recall for activity feature of target stimuli.**

```
results_Target = highlight_max_clr( Results ,Standard = False, Target = True, Distractor = False)
clr_A_T = results_Target.mean_std(input_feature = 'Activity')
#clr_A_T.to_latex('clr_A_T.tex')
clr_A_T
```

**Highlighed best performance classifiers based on the f1-score, precision and recall for mobility feature of target stimuli.**

```
clr_M_T = results_Target.mean_std(input_feature = 'Mobility')
#clr_M_T.to_latex('clr_M_T.tex')
clr_M_T
```

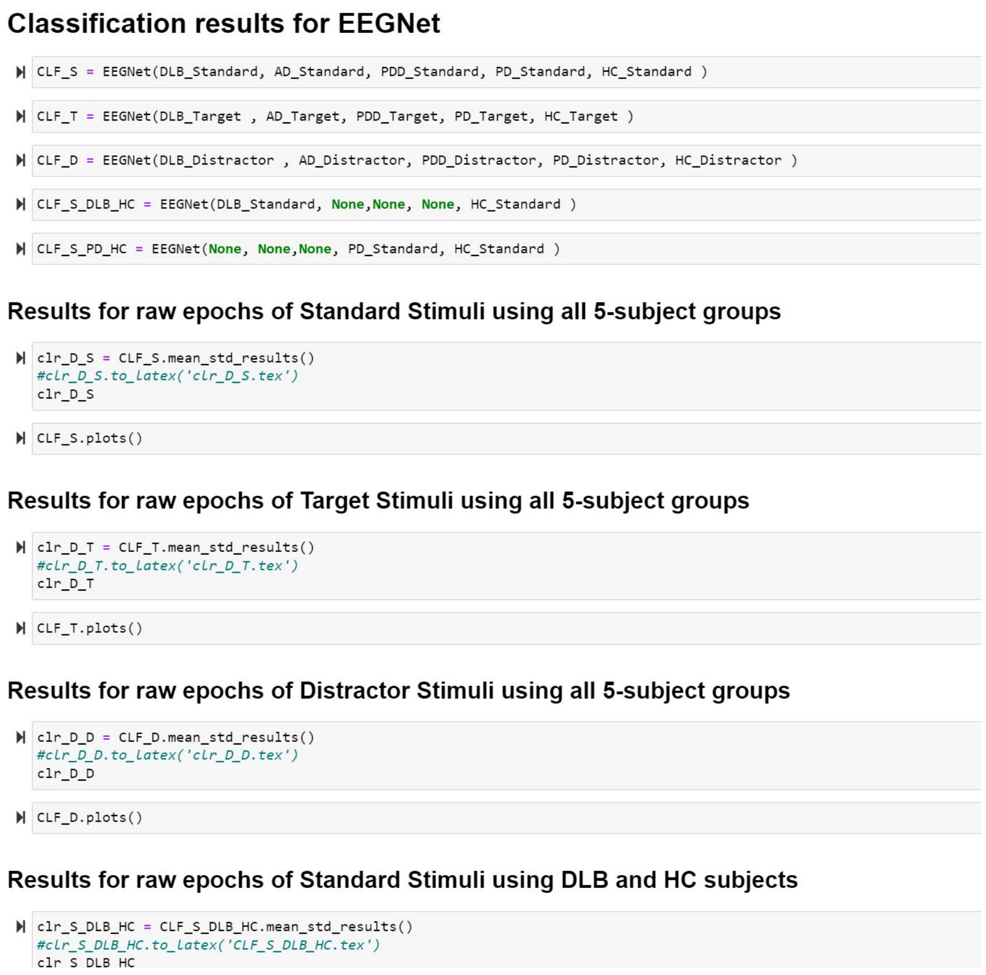
Figure 3: The screenshot illustrates the execution of classification and evaluation algorithms to obtain results. The second class in "ML.py" is employed to classify all datasets, enabling easy access and comparison of results. Subsequently, the third, fourth, and fifth classes are utilized to visualize the best performance and confusion matrices.

## 2.7 DL.py

The "DL.py" file is a class-based Python script that utilizes EEGNet for subject classification based on ERPs patterns. This class takes the epochs of raw

EEG data as input. It applies the EEGNet model to classify the subjects and returns various performance metrics, including accuracy, f1-score, precision, recall, and a confusion matrix.

In addition to performance metrics, the class also provides visualization capabilities. It can also generate plots for accuracy and validation accuracy, along with loss and validation loss.



## Classification results for EEGNet

```
CLF_S = EEGNet(DLB_Standard, AD_Standard, PDD_Standard, PD_Standard, HC_Standard )
```

```
CLF_T = EEGNet(DLB_Target , AD_Target, PDD_Target, PD_Target, HC_Target )
```

```
CLF_D = EEGNet(DLB_Distractor , AD_Distractor, PDD_Distractor, PD_Distractor, HC_Distractor )
```

```
CLF_S_DLB_HC = EEGNet(DLB_Standard, None,None, None, HC_Standard )
```

```
CLF_S_PD_HC = EEGNet(None, None,None, PD_Standard, HC_Standard )
```

### Results for raw epochs of Standard Stimuli using all 5-subject groups

```
clr_D_S = CLF_S.mean_std_results()
#clr_D_S.to_latex('clr_D_S.tex')
clr_D_S
```

```
CLF_S.plots()
```

### Results for raw epochs of Target Stimuli using all 5-subject groups

```
clr_D_T = CLF_T.mean_std_results()
#clr_D_T.to_latex('clr_D_T.tex')
clr_D_T
```

```
CLF_T.plots()
```

### Results for raw epochs of Distractor Stimuli using all 5-subject groups

```
clr_D_D = CLF_D.mean_std_results()
#clr_D_D.to_latex('clr_D_D.tex')
clr_D_D
```

```
CLF_D.plots()
```

### Results for raw epochs of Standard Stimuli using DLB and HC subjects

```
clr_S_DLB_HC = CLF_S_DLB_HC.mean_std_results()
#clr_S_DLB_HC.to_latex('CLF_S_DLB_HC.tex')
clr S DLB HC
```

Figure 4: The EEGNet can be executed as shown in the Figure.

## 2.8 Results.ipynb

By running the "Results.ipynb" Jupyter Notebook file, you can generate all the results presented in this thesis. The notebook contains the necessary code and instructions to load the data, apply preprocessing techniques, extract features, and perform classification using ML and DL algorithms.