UNIVERSITY OF GENOVA

PHD PROGRAM IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

Artificial Intelligence for the Edge Computing Paradigm

by

Ali Daher

Thesis submitted for the degree of *Doctor of Philosophy* (35° cycle)

November 15th, 2022

Prof. Daniele Caviglia	Supervisor
Prof. Hussein Chible	Supervisor
Dr. Marco Muselli	Supervisor
Dr. Enrico Ferrari	Supervisor
Prof. Maurizio Valle	Head of the PhD program

Department of Electrical, Electronics and Telecommunication Engineering
and Naval Architecture - DITEN

I would like to dedicate this thesis to my Parents and Sisters, for their moral support during my studies whether I was home visiting or working abroad. Whose emotional guidance helped me complete my research goals…

Declaration

I declare that except where citations are made referring to outside works from the literature, the contents of this thesis are original and have not been submitted and are not under consideration in any other research degree in another university. I have written this thesis as an original work which contains the results from projects done in collaboration with other researchers and supervisors.

Ali Daher
February 2023

Acknowledgements

I would like to thank Prof. Daniele Caviglia, Prof. Hussein Chible, Dr. Marco Muselli and Dr. Enrico Ferrari for their guidance throughout the research period where I worked on my thesis, who without their support, the projects undertaken would not have been successful.

Abstract

With modern technologies moving towards the internet of things where seemingly every financial, private, commercial and medical transaction being carried out by portable and intelligent devices; Machine Learning has found its way into every smart device and application possible. However, Machine Learning cannot be used on the edge directly due to the limited capabilities of small and battery-powered modules. Therefore, this thesis aims to provide light-weight automated Machine Learning models which are applied on a standard edge device, the Raspberry Pi, where one framework aims to limit parameter tuning while automating feature extraction and a second which can perform Machine Learning classification on the edge traditionally, and can be used additionally for image-based explainable Artificial Intelligence. Also, a commercial Artificial Intelligence software has been ported to work in Client/Server setups on the Raspberry Pi board where it was incorporated in all of the Machine Learning frameworks which will be presented in this thesis. This dissertation also introduces multiple algorithms that can convert images into Time-series for classification and explainability but also introduces novel Time-series feature extraction algorithms that are applied to biomedical data while introducing the concept of the *Activation Engine*, which is a post-processing block that tunes Neural Networks without the need of particular experience in Machine Leaning. Also, a tree-based method for multiclass classification has been introduced which outperforms the One-to-Many approach while being less complex that the One-to-One method.

The results presented in this thesis exhibit high accuracy when compared with the literature, while remaining efficient in terms of power consumption and the time of inference. Additionally the concepts, methods or algorithms that were introduced are particularly novel technically, where they include:

1. Feature extraction of professionally annotated, and poorly annotated time-series.
2. The introduction of the *Activation Engine* post-processing block.

3. A model for global image explainability with inference on the edge.

4. A tree-based algorithm for multiclass classification.

Table of contents

List of figures

List of tables

List of abbreviations

ADC Analog to Digital Converter

AI Artificial Intelligence

ANN Artificial Neural Network

CACAO Contour Assisted Convolutional Neural Networks

CCTV Closed Circuit Television

CNN Convolutional Neural Network

Cpp C++ (C-Plus-Plus)

CPU Central Processing Unit

CSV Comma Separated Values

DT Decision Trees

ECG Electrocardiogram

EEG Electroencephalogram

FE Feature Extraction

FFT Fast Fourier Transform

FMCW Continuous-Wave Frequency-Modulated

GPU Graphical Processing Unit

GUI Graphical User Interface

HDMI High-Definition Multimedia Interface

IoT Internet of Things

KNN K-Nearest-Neighbor

LAN Local Area Network

LCD Liquid Chrystal Display

LLM Logic Learning Machine

LSTM Long Short Term Memory

ML Machine Learning

MLP Multi-Layer Preceptron

NET Network

NN Neural Network

OS Operating System

PC Personal Computer

PPG Photoplethysmogram

RCS Radar Cross Section

RAM Random Access Memory

RGB Reg-Green-Blue based Images (Color Images)

RNN Recurrent Neural Networks

SSH Secure Shell

SD Secure Digital

STD Standard Deviation

SVM Support Vector Machine

TinyML Tiny machine learning

TS Timer-series

USB Universal Serial Bus

VAMPIRE Vectorized Automated Machine Learning Pre-Processing and Post-processing

WIN32 Windows 32-Bits

WIN64 Windows 64-Bits

XAI Explainable Artificial Intelligence

Part I

Introduction

Chapter 1

Overview

1.1 Thesis Overview

Artificial Intelligence is playing a prominent role in modern technology while changing our culture, means of entertainment, and accessing services online in an accurate unmanned and automated manner. In another case, which in its primitive form is unrelated, we find the adoption of Machine Learning (ML) in every edge device in the world of the Internet of Things, where limitless implementations in biomedical, industrial, and robotic applications can be observed. ML is employed on the Edge to identify and classify images, biomedical signals and monitor user behavior by means of the deployment of Artificial Intelligence (AI) models on embedded devices, which in terms, infer and act accordingly based on past training and inspection. This case requires lightweight models, high accuracy, and fast convergence or else the overall system cannot be utilized commercially in the real world.

In the field of the Internet of Things (IoT), AI techniques are more commonly used as the cognitive engine for decision-making and identification. These findings may include heart failure, seizure detection, gender recognition and satellite data classification.

Furthermore, AI is being used in every aspect of our lives from marketing, purchases, and banking transactions as well as being used in cutting-edge military crafts and monitoring systems, therefore, this thesis aims to follow this trend by aiming to always rely on ML to automate classification, interpret-ability and to solve engineering problems that may arise due the immense power of modern AI algorithms and workflows.

1.2 Motivation

The motivation behind this thesis is primarily to follow a trend which is the proliferation of AI in every engineering system realizable. The presented work consists of various ML models which classify and interpret targets accurately and efficiently, while remaining increasingly portable to other data sources or domains with minor tuning of the ML parameters. Furthermore, the different projects described in this thesis aim to classify pre-processed data, pre-process images and Time-series (TS), and automate the tuning of ML setups by means of novel light-weight post-processing algorithms. Additionally, another AI model was used to perform global image explainability on image-based data while competing with other classifiers reported in the literature that were taken from diverse application domains having a differing nature in data.

An additional motivation for the work presented in this thesis is related to the necessity of edge computing models in modern systems and services. As shown in Figure 1.1 in part (a), communication networks and devices have become interconnected within the same scope and often return to a central cloud server as a root for storage, processing and orchestration. Although this model has been successful during the time of its inception, the proliferation of smart devices, AI, and Big Data in every application imaginable has led to high computational costs and most importantly increased network traffic which leads to congestion and delay. Consequently, edge computing as shown in part (b), is a primary candidate as a solution for this problem. Instead of performing all data and computing operations on the cloud, when possible, specific tasks may be implemented on so-called edge devices, which are IoT devices themselves, where this limits the network traffic and the delay of sending and receiving data between all devices and the centralized cloud server.

Additionally, the edge computing paradigm works in synergy with ML algorithms in modern systems, therefore, in Cecilia et al. (2020); Lapegna et al. (2021), the authors implement light-weight ML programs on GPU-enable IoT boards which help in reducing the power consumption as well as reducing the real-time inference time on portable edge computing modules. Additionally, in Novac et al. (2021), ML algorithms are run on embedded microcontrollers by taking power consumption into consideration, while performing inference indoors where pre-processing is applied through radio frequencies. This demonstrates that IoT boards are involved in every phase of the ML workflow, from collection, training, inference and deployment.

In regards to pure software applications, the authors in Canedo and Skjellum (2016); Hodo et al. (2016) employ ML for improving the security of IoT system where NNs are used

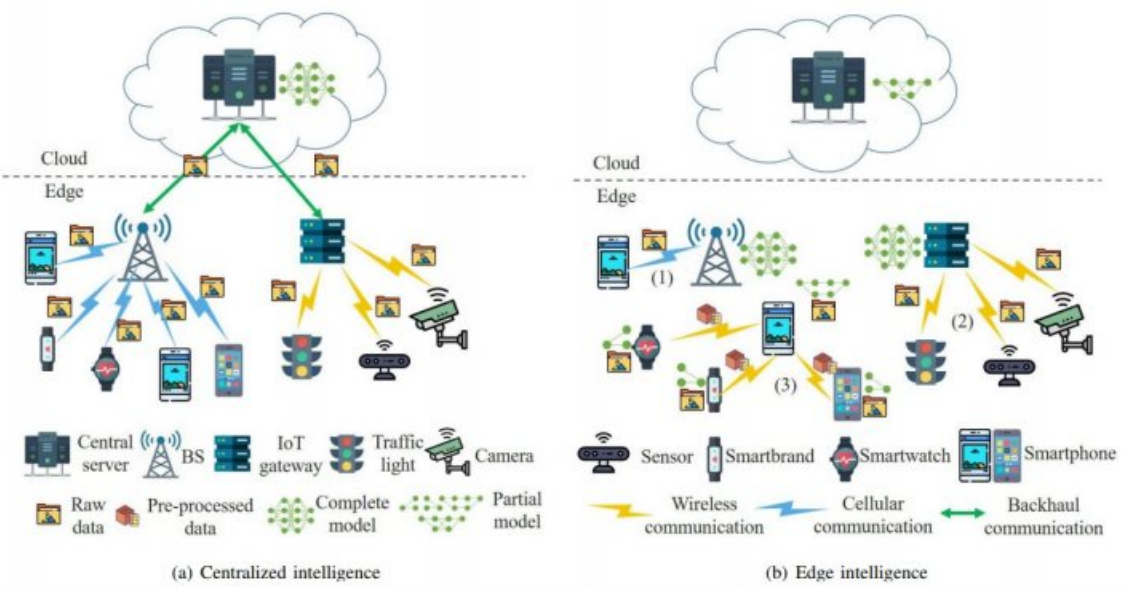

Fig. 1.1 (a) Classic cloud computing architecture with the processing and storage being computed solely on the cloud. (b) The edge computing model which offloads certain tasks onto IoT devices(Xu et al. (2020)).

as the ML model of choice. This applications takes into account information from software, networking, and cryptography data which expands the scope of applications of AI on the Edge.

1.3 Synthesis and Summary

The thesis implements ML on the edge with a general applications perspective while dealing with edge computing issues such as power consumption and the time of inference. Also, global image explainability was discussed by providing human-understandable statement in the field of gender classification. Moreover, the vectorized automation of ML workflows was presented which contains multiple pre-processing and post-processing techniques. Furthermore, the pure software aspect of porting a complete system from one platform to another has been explained in detail.

The remainder of this thesis continues as follows: Chapter 2 presents the three main projects undertaken in this thesis, Chapter 3 provides the related literature and fundamentals to the presented AI projects, Chapter 4 outlines the contributions and lists the publications, Chapter 5 deals with radar classification using ML, Chapter 6 experiments with additional datasets and contributes with a novel image classification algorithm, Chapter 7 presents the

CACAO-X framework which deals with Explainable AI (XAI) and inference on the edge and Chapter 8 presents the *VAMPIRE* framework which provides novel techniques for Feature Extraction (FE) and automation while achieving efficient and accurate performance. Chapter 9 concludes the thesis while two appendices are included at the end where Appendix A describes in detail an FE algorithm from Chapter 8, and Appendix B presents some practical implementations which were carried out in Chapter 7.

Chapter 2

Research Projects

2.1 Classic ML on the Edge and Tree-based Multi-class Classification

This thesis contains three main projects; the first major project in this thesis concerns ML classification of pre-processed data on the edge, as well as implementing an Image-to-TS conversion algorithm and a tree-based multi-classification approach.

The first task was to port the commercial software Rulex (Muselli (2005, 2012); Muselli and Ferrari (2009)) onto the Raspberry Pi (Vujović and Maksimović (2014)). This includes the compilation of internal and external libraries used in the Rulex source code in order to operate the tool in a client/server setup (Hajdarevic et al. (2014)) and perform ML classification in an IoT setting. The first dataset was split into four classes, where a tree-based approach was used for the classification methodology. In this method multiple forecasts were applied where the sub-classes were grouped as one class before splitting that parent class in the proceeding prediction. this method was able to surpass the common One-VS-All technique which is commonly used in multi-class classification by improving the performance metrics in addition to securing the ability for the designer to group the classes based on his own judgment where in the applications or data sources employed in this Thesis were classified with competitive results.

To test the Rulex/IoT system on larger datasets, forecasts were performed on pre-processed sources where the setup proved to be robust as it is portable to multiple domains. Also, an Image-to-TS conversion algorithm was applied on a random inconsistent image-based gender dataset, where Rulex achieved very good results while competing with forecasts

from the literature that mostly used mug-shot or forward-facing images. Moreover, in every case this arrangement exhibited low power consumption on the raspberry Pi.

2.2 Flexible Neural Networks and Explainability

A Flexible Neural Networks model name CACAO-NET is developed in this second project where it consists of arrangements of Multi-Layer Perceptrons (MLP), a Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) blocks which are used for general-purpose ML applications. The CACAO-X overall framework includes XAI for image data having a global set of output rules.

Regarding image classification using Flexible Neural Network (NN) models, a modified version of the Image-to-TS conversion algorithm, mentioned in Project 1, was used alongside landmarks features and an explicative labeling algorithm to achieve global image explainability. This consisted of a novel NN model, Rulex, and additional XAI tools employed in a complex setup. Moreover, this framework was used as a traditional image classifier on the Raspberry Pi where it achieved high accuracy without putting stress on the hardware and power resources. Moreover, the Image-to-TS algorithm which contains returns distance and image data of the each filtered image was implemented in the low-level C programming language, which is considerably faster than Python, and since the algorithm has reasonably high complexity. Furthermore, the compiled version of the code was interfaced into python and the conversion was applied per image and in real-time on the Raspberry IoT board for proven portability and deployment.

2.3 The AI Pre-processing and Post-processing Framework

The third major project that was developed in this thesis led to the design and implementation of the *VAMPIRE* framework, which is basically a set of pre-processing and post-processing algorithms targeted toward the edge. Firstly, regarding the set pre-processing algorithms, two versions have been presented which deal with professionally annotated datasets and with poorly annotated recordings. The FE algorithms *VAMPIRE FE1 and FE2* deal with TS, and in the published works, three types of biomedical TS were used as sources for FE and later training and testing on the Raspberry Pi.

In regards to the post-processing part, *VAMPIRE* presents the *Activation Engine* which is a new concept in the field of ML. A couple of *Activation Engines* are used to extract the

optimal threshold of the final layer on an NN to apply binary classification while using the tree-based approach discussed in Section 2.1 to perform multi-class classification.

The algorithms presented in this project are light-weight, converge quickly, and consume little power when compared with the literature. The framework introduces a novel representation of an input time-varying waveform to represent it in the D-Domain, which is a co-representation of the original signal as correlated with its frequency response thus having a resulting unit of V.Hz fluctuating over time.

Chapter 3

Literature and Fundamentals

The related work is outlined in this chapter by splitting the content into two main sections, where the first corresponds to the fundamental concepts used in this thesis and the second being a summary for the corresponding literature relating to each of the three projects that the thesis presents.

3.1 AI on the Edge: Fundamentals

In the following section, the basic ML algorithms which have been used in the experimental results of each project or chapter have been described fundamentally with the theoretical background as the operation of these AI models. These consist of Support Vector Machines (SVM), K-Nearest Neighbor (KNN), Multi-Layer Perceptron (MLP), Logistic Regression (LR), AdaBoost, Decision Trees (DT), and Random forest (RF) classifiers. Also, all of the described algorithms have been implemented on the low-resource Raspberry Pi IoT module. Additionally, the basic methods used to collect some of the recordings and images which have been employed for training and testing in the upcoming sections will be described in detail.

3.1.1 Classification and Regression Algorithms

Regarding the ML forecasts implemented in this thesis, there is a mixture of well-known fundamental algorithms which have been applied in addition to some novel designs which will be described in the upcoming chapters. This subsection provides the fundamental working of the models which have been used either to perform forecasts within the scope

of a project or to compare the classic ML models with the ML techniques which have been developed in this thesis.

Support Vector Machines

A SVM (Cervantes et al. (2020); Ghosh et al. (2019); Muthukrishnan et al. (2020); Okwuashi and Ndehedehe (2020)) is a mature ML algorithm which relies on hyperplanes as a factor of safety for robust and high-performance metrics across a wide range of application domains. In this subsection, the fundamental concepts behind these methods are outlined.

Considering n points $\{\mathbf{x}_i, y_i\}$ $i = 1, \ldots, n$ where $y_i \in \{-1, 1\}$ being a set of x_i.The classes may be expressed as: $y_i = 1$ and $y_i = -1$. Therefore, we need to determine aa hyperplane that maps x_i's in the form of a higher dimensional space to create a factor of safety in case of the classification. The said hyperplane must have a margin that can maximize the minimum distance of the hyperplane in the case of the closest data points. an illustration of a linear hyperplane is displayed in Figure 3.1

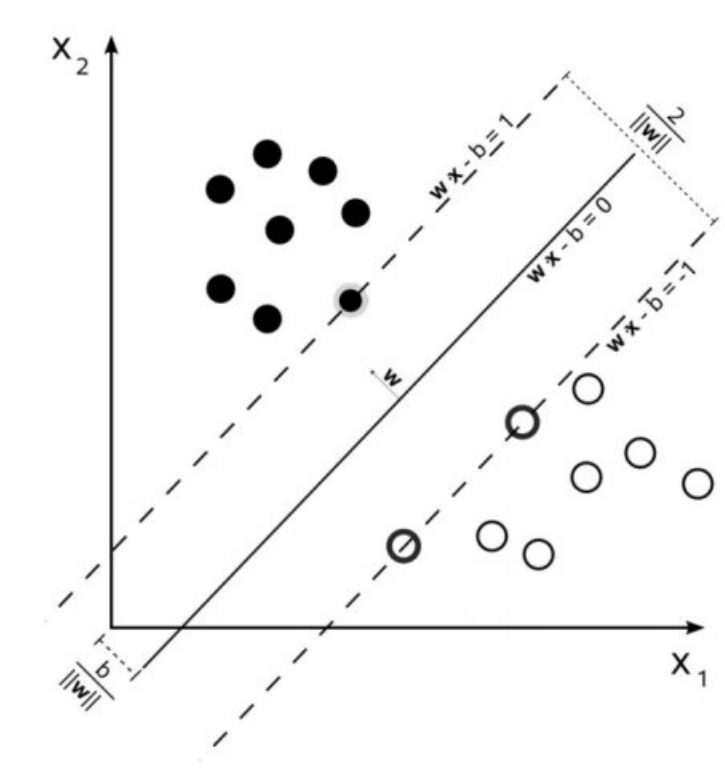

Fig. 3.1 Two SVM hyperplanes

In case of a linear function, the hyperplane may be applied as follows:

$$\mathbf{x}_i\mathbf{w} + b = 0$$

For two hyperplanes that create the factor of safety, their minimum distance must be as large as possible. this is presented in the following equation:

$$\mathbf{x}_i\mathbf{w} + b = +1$$

and

$$\mathbf{x}_i\mathbf{w} + b = -1$$

It can be proven that the distance between these two hyperplanes is equal to $\frac{2}{||w||}$:

$$d_+ + d_- = \frac{|1-b|}{\|w\|} + \frac{|-1-b|}{\|w\|} = \frac{2}{\|w\|} \tag{3.1}$$

Consequently, we can minimize $\|w\|$ which is always positive. Also, for every $i \in (1,n)$, x_i and y_i the following constraints arise:

$$\mathbf{x}_i\mathbf{w} + b \geq +1, \quad y_i = +1 \tag{3.2}$$

$$\mathbf{x}_i\mathbf{w} + b \leq -1, \quad y_i = -1 \tag{3.3}$$

$$\equiv \tag{3.4}$$

$$y_i(\mathbf{x}_i\mathbf{w} + b) - 1 \geq 0, \qquad \forall i \tag{3.5}$$

Consequently, in this thesis, SVM has been used in performing forecasts either as a proof of concept or in order to perform a comparison with the presented novel algorithms, due to its robustness.

K-Nearest Neighbor

KNN (Abu Alfeilat et al. (2019); Ayyad et al. (2019); Gou et al. (2019)) is an algorithm which relies on distance measurements between a target node and the k closest surrounding nodes to determine the class label of that single data point. An illustration of this setup is illustrated in 3.2 where the newly introduced yellow dot that is closer in Euclidean distance to the green dot class than the red dot class will be appointed to the green class. In case the surrounding dots are of mixed classes, a voting scheme of counting or summation may

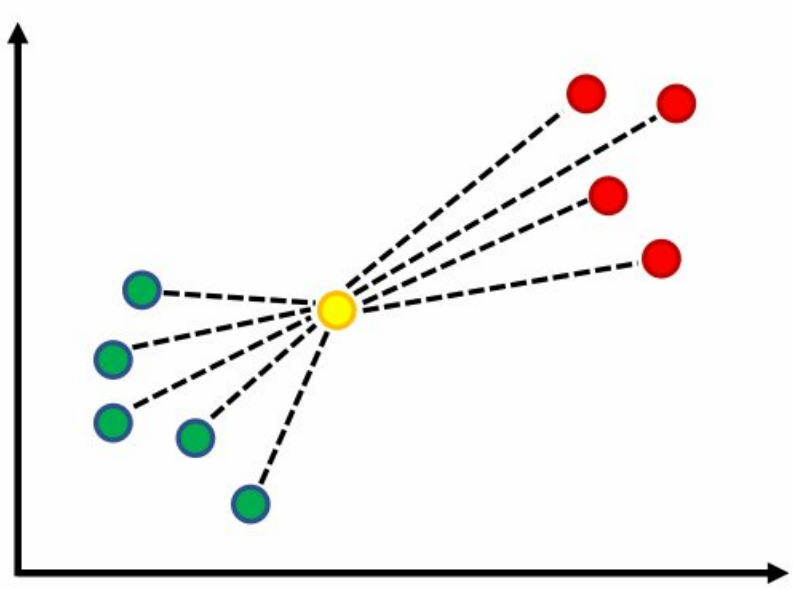

Fig. 3.2 A 2D map representing how KNN operates in case of a binary classification problem having two features.

be employed as a reliable metric for classification. Additionally, in case of ML regression, averaging can be employed to estimate the target value.

Algorithm (3.1) describes shows how KNN operates where a counting scheme is used for classification, as presented, and the Euclidean distances between the target point and all other points are computed, then to closest k points are extracted before finalizing the classification or regression task using the appropriate method.

Algorithm 3.1 *K-Nearest Neighbor operation*

Require: $\forall$ *(*X_i*,* Y_i*), k*
Ensure: *Classification and Regression task*
1: **for** every new point (X_, Y_) **do**
2: **for** *every points i of (*X_i*,* Y_i*)* **do**
3: *Append Euclidean Distance ((*X_i*,* Y_i*),(X_, Y_)) → Distances*
4: **end for**
5: *"j1, j2, j3, ..., jk" ← Argmin(Distances,k)*
6: **if** *Task is Regression* **then**
7: **Return** *Average(Distances[j1:jk])*
8: **end if**
9: **if** *Task is Classification* **then**
10: **Return)** *MAX(LabelCount[j1:jk])*
11: **end if**
12: **end for**

Decision Trees

A DT is a supervised ML model (Charbuty and Abdulazeez (2021); Wan et al. (2020); Yoo et al. (2020)), used in both classification and regression and consists of a hierarchical tree-based structure composed of one root node and multiple internal nodes and leaves where each leaf corresponds to a specific class label from the target dataset.

As shown in 3.3, the DT begins at the root, which does not have any incoming branches. The proceeding branches connect decision nodes which dictate (based on a condition) which path down the tree the algorithm will take in order to reach a decision, in the form of a leaf, where the leaves represent every possible result in the domain dataset.

A DT employs a greedy search while adopting a divide-and-conquer approach to identify the best diversion down the tree in order to reach a leaf. This process splits the tree from top to bottom recursively every time input is introduced where DT model metrics are usually evaluated with cross-validation.

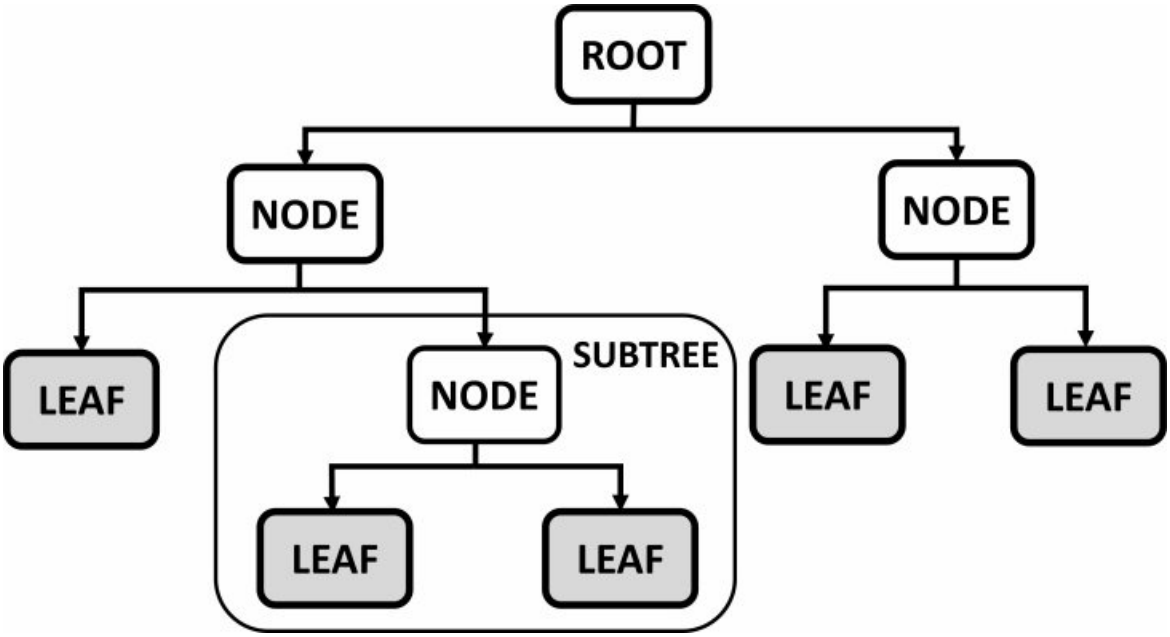

Fig. 3.3 Decision Tree composed of nodes, leaves and sub-trees.

Logistic Regression

The LR model (Boateng and Abaye (2019); Nusinovici et al. (2020); Rymarczyk et al. (2019)) is mostly implemented for classification and predictive applications. LR estimates the probability of an event may occur while relying on independent variables where the

overall probability is usually bounded between 0 (False) and 1 (True). A depiction of the LR classification for a binary example is shown in Figure 3.4. Therefore in LR, logit transformation is employed which infers the probability of a True label over the probability of False label. This probability in case of a dataset having n variables X_i may be extracted in the following expression:

$$Pr(Y_i = 1|X_i) = \frac{exp(\beta_0 + \beta_1 X_i + \beta_2 X_2 + .. + \beta_n X_n)}{1 + e.xp(\beta_0 + \beta_1 X_i + \beta_2 X_2 + ... + \beta_n X_n)} \tag{3.6}$$

Consequently, a more compact form of this activation function can be written as follows:

$$\sigma(z) = \frac{1}{1 + e^{-X\beta}} \tag{3.7}$$

It can be proven the log likely-hood function which is to be maximized in the optimization process during training, can be used as provided here:

$$l(\beta) = \sum_{i=1}^{n} [y_i X_i \beta - log(1 + e^{X_i \beta})] \tag{3.8}$$

In LR, the β variables are updated through iterations in order to achieve the best fit while relying on the log likely-hood. Consequently, the iterations maximize this function to extract the most feasible parameters. Afterwords, conditional probabilities are used per input to implement a reliable prediction.

Bayesian Classification: Naive Bayes Classifier

The NB ML algorithm (Balaji et al. (2020); Salmi and Rustam (2019)) is a probabilistic classifier based on fundamental probability theory which relies on robust assumptions. In some cases, these assumptions may not be based on real truth, which is why they are considered as naive.

the fundamental expression which the algorithm is based upon may be viewed in this equation:

$$P(Y|X) = P(Y)\frac{P(X|Y)}{P(X)} \tag{3.9}$$

In case of binary classification, and in practice, the above function may be employed in the following pair of expressions:

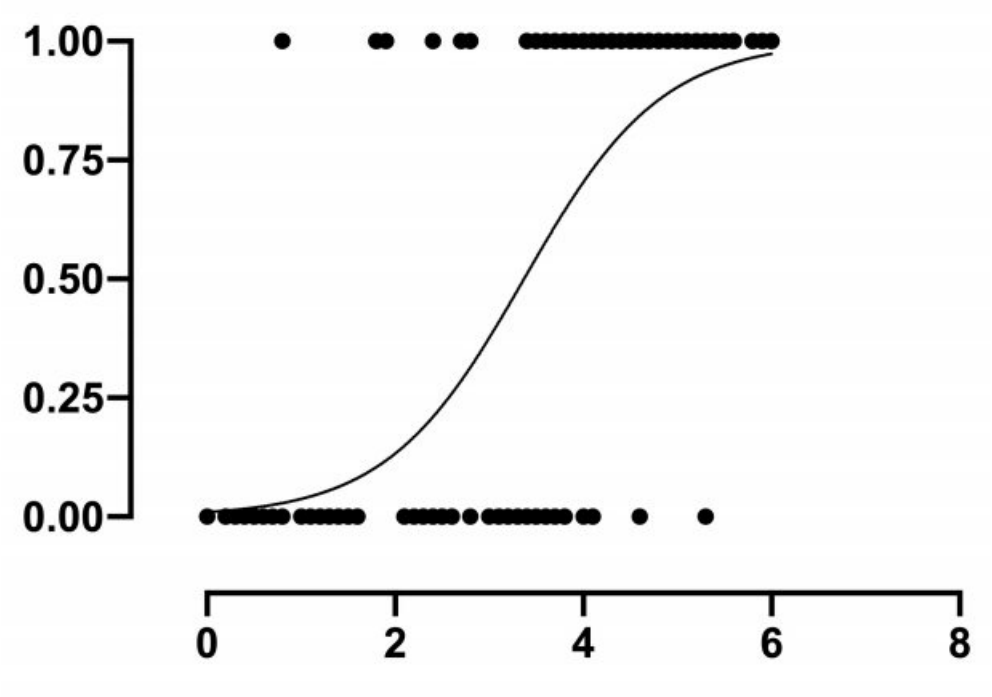

Fig. 3.4 Logistic regression operation with two class labels.

$$P(X = (a,b)|Y = 1) = P(X1 = a|Y = 1) * P(X2 = b|Y = 1) \tag{3.10}$$

$$P(X = (a,b)|Y = 0) = P(X1 = a|Y = 01) * P(X2 = b|Y = 0) \tag{3.11}$$

Finally, in order to predict any unseen data input, the maximum of the last two equations is used to reach an assumption:

$$Y = MAX(P(X = (a,b)|Y = 0), P(X = (a,b)|Y = 1)) \tag{3.12}$$

NB in employed in Chapter 7 for comparison purposes where the employed model in tuned for optimal performance.

Multilayer Perceptron

The most simple form of a NN is a MLP (Mirjalili and Mirjalili (2019); Singh and Banerjee (2019); Sreedharan et al. (2020)), as shown in Figure 3.5, where its units are grouped into layers, each layer having identical units. In a MLP, each unit in a layer is connected to every unit in the upcoming layer; hence the term fully connected, where the first layer is the input

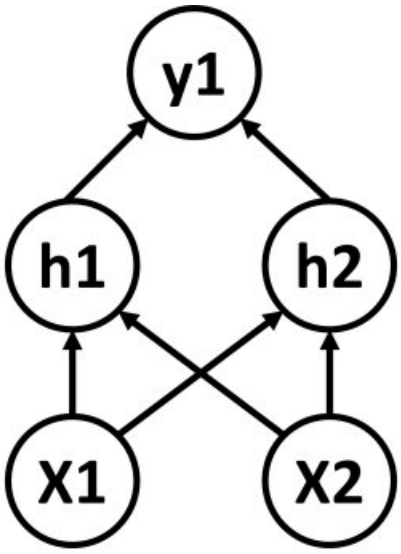

Fig. 3.5 Simple Multilayer Perceptron.

layer, and the last layer is the MLP output. finally, all the layers in between a referred to as hidden layers.

A simple MLP which computes the XOR function which is shown in Figure 3.5 having a hidden unit h1 that detects if at least one of the inputs is 1, and another unit h2 that detects if they are both 1. Therefore, the output unit will only activate in case h1 = 1 and h2 = 0 or if h1 = 0 and h2 = 1.

The set if equations which describe the operation of these layers can be observed as follows:

$$h^1 = \sigma^1(W^1 x + b^1) \tag{3.13}$$

$$h^2 = \sigma^2(W^2 h^1 + h^2) \tag{3.14}$$

$$y = \sigma^3(W^3 h^2 + h^3) \tag{3.15}$$

The σ function can be anyone of multiple types of activation functions which all have been used and studied in the literature, where there is no universal choice or conformity to which is best for a particular application or which combination yields better results. On the other hand, the choice of the activation function may be considered as just another hyper-parameter which is always tuned in the ML model development workflow.

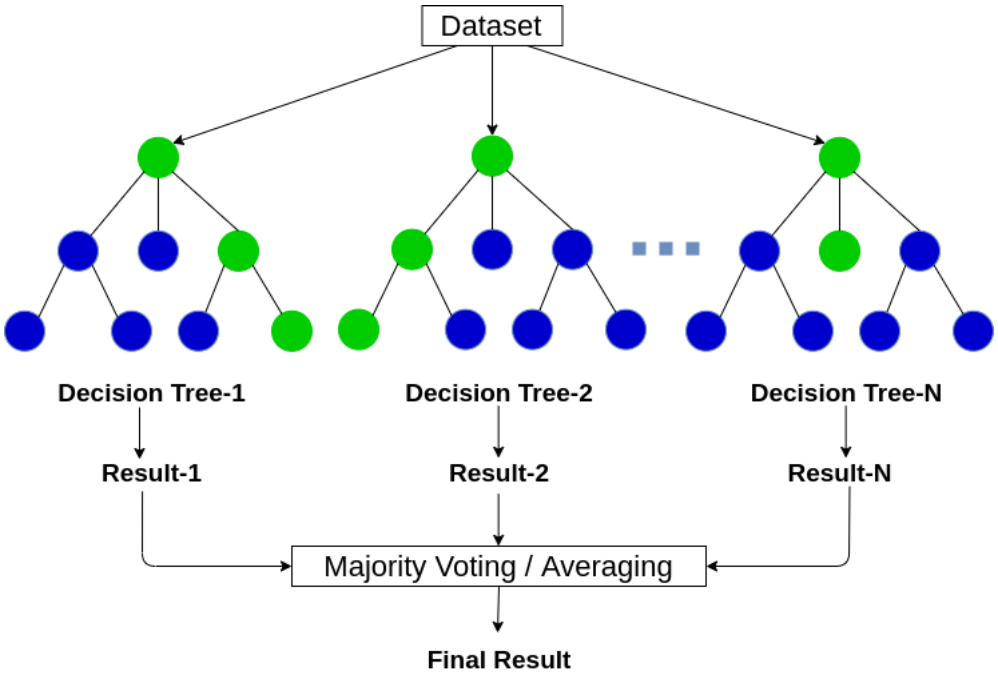

Fig. 3.6 Random Forest with multiple trees having different input combinations.

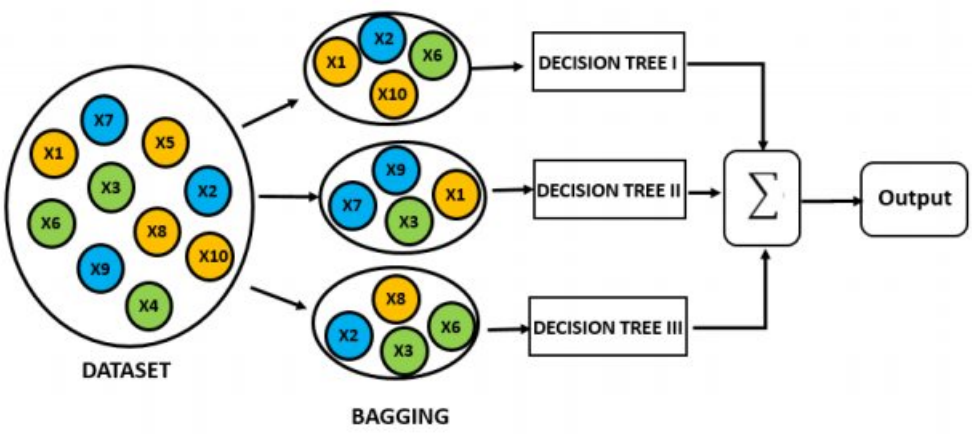

Fig. 3.7 Bootstrapping in Random Forest where a subset input is used in every tree.

Random Forest

A RF model (Schonlau and Zou (2020); Speiser et al. (2019); Tyralis et al. (2019)) consists of multiple DTs, where the program starts with a question, followed by a series of questions to infer probable classification. These questions are basically the nodes in the tree, acting as a means to split the data. Each question assists the overall model in reaching a decision, which is nothing but a leaf node.

Algorithm 3.2 *Adaptive Boosting: Adaboost*

Require: *Initialize Weights* $W_i = \frac{1}{N}$ *for every i*
Ensure: *Classification and Regression task*
1: *Start with NULL classifier* $f_o(x) = g_o(x) = 0$
2: **for** t = 0 till T-1 **do**
3: *Generate training Dataset using* W_i
4: *Fit a weak learner* g_t
5: *Set* $\lambda_t = \frac{1}{2}\frac{1-e_t}{e_t}$
6: $W_i \leftarrow W_i e^{\lambda t}$ *if wrongly classified*
7: $W_i \leftarrow W_i e^{-\lambda t}$ *if correctly classified*
8: **end for**
9: *Output Final Model* $f_T(x) = sgn(\sum_{t=0}^{T-1} \lambda_t . g_t\ )$

In RF, bootstrapping in usually implemented, which consists of employing a finite set of DTs which take as input a subset of the records from the dataset in addition to a subset of features from the same source. Consequently, these trees are trained using the sub-datasets before attempting to introduce unseen records.

After training, a new sample is used as input for the RF, where all the trees in the forest produce a resultant classification. As shown in Figure 3.6, a majority is applied to vote for the final classification. Also, in case of regression applications, an averaging scheme may be employed instead for a reliable output.

Adaptive boost: Adaboost

AdaBoost or Adaptive Boosting (Shahraki et al. (2020); Wang et al. (2019); Wang and Sun (2021)), is a ML Algorithm which is usually used with DTs to improve performance metrics better than other ML algorithms. This is shown in 3.8 where the outputs of the sub-algorithms, often called weak learners, are employed by relying a weighted sum which sets the classification prediction at the output of Adaboost. AdaBoost is sensitive to noise, however, it may be less susceptible to over-fitting than traditional learning algorithms.

The overall steps taken iteratively by Adaboost to reach a final model are provided by Algorithm 3.2. As shown, a weight is set for each record in the training dataset while training the model. After each iteration, this weight is updated from the previous model before summing all generated models in an attempt to achieve a strong classifier from multiple weak classifiers with improved performance metrics.

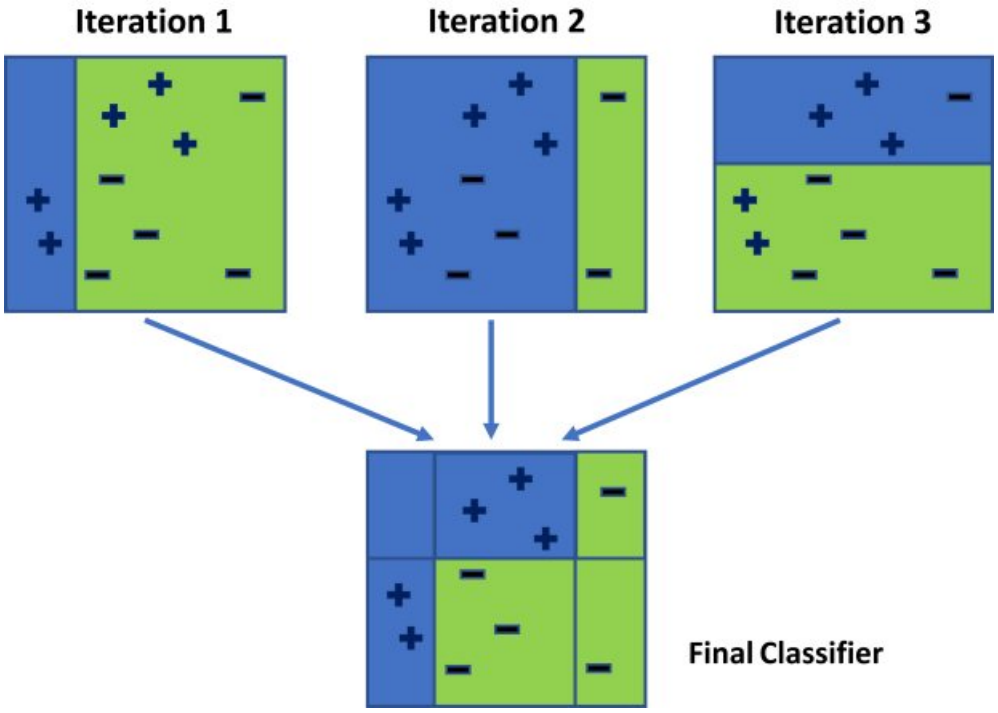

Fig. 3.8 Adaboost ML Algorithm which uses boosting and a weighted sum to improve performance.

Ensemble Learning: Stacking

Ensemble learning (Dong et al. (2020); Zhou et al. (2021)) is the process of employing multiple weak ML algorithms in parallel or a predefined order before fusing in diverse ways the outputs of this set of learning in order to reach a unified decision having improved performance metrics such as accuracy.

Three main types of ensemble exist:

1. Bagging which relates to bootstraping, is the method used in the RF classifier and consists of a majority decision scheme.

2. Boosting which is applied in Adaboost and relies an summing the weights of ordered classifiers.

3. Stacking which consists of applying mutliple diverse ML algorithms in parallel before applying a voting layer at the output for improved training and inference.

Figure 3.9 represent a stacking setup, which is generally dubbed ensemble learning, and is repeatedly used in the literature to reach very high-performance and benchmark results. The

final node is sometimes included in the training process which gives the ensemble method its superior performance.

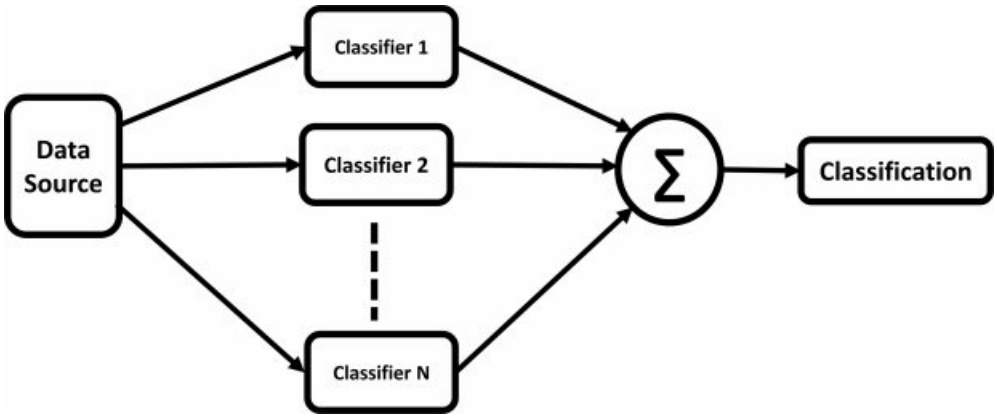

Fig. 3.9 Ensemble learning by means of stacking.

Meta-Learning

Meta-Learning (Hospedales et al. (2021); Wang (2021)) is an optimization approach to tuning ML models. Instead if manually tuning the hyperparameters or implementing a dedicated methodology, an Automated Machine Learning (AutoML) setup is implemented to optimize the parameters using a global optimizer which learns the internal ML model.

Figure 3.10, provides a high-level setup of how this architecture may be setup. the ML model to be tuned is an MLP for increased simplicity in this diagram. After training and evaluating the MLP model, performance metrics are compared with some criteria, which may a threshold or a settling points. Afterwords, a feedback control loop if formed in the Meta-Learning setup to continue tuning the hyperparameters or maybe the structure of the ML model until the target is reached or the performance criteria are satisfied.

3.1.2 Data Sources from Diverse Domains

Electroencephalogram: EEG

An Electroencephalogram (EEG) is used to measure and asses a brain's electrical activity (Aggarwal and Chugh (2022); Hosseini et al. (2020); Rasheed et al. (2020)). Even though imaging techniques have been developed for use in most medical applications, the EEG remains the goto tool for seizure prediction.

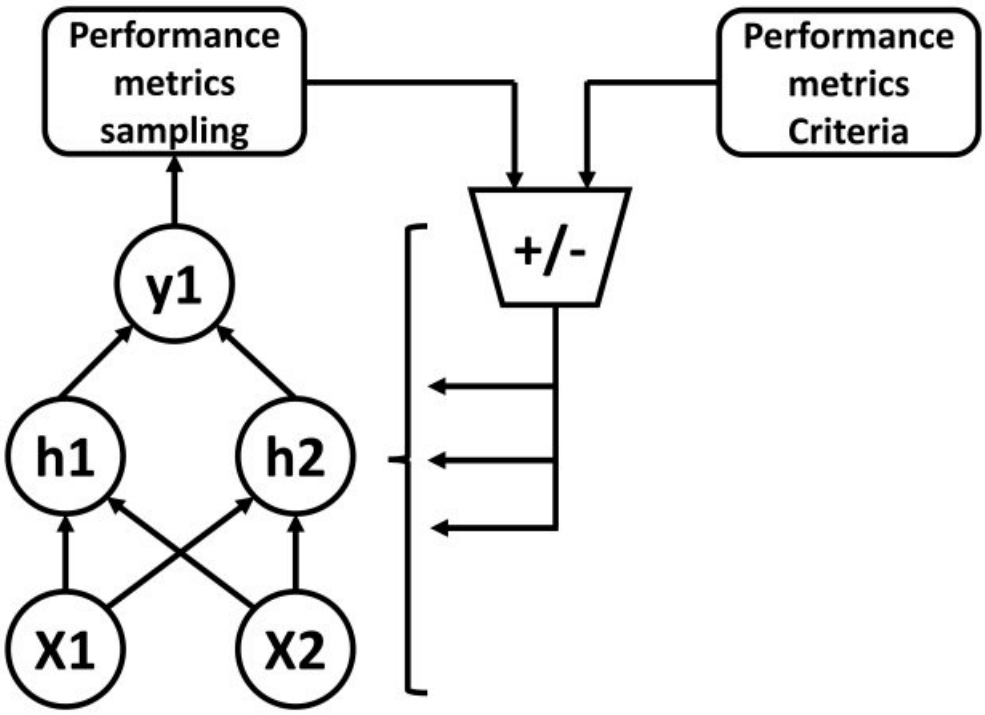

Fig. 3.10 Meta-learning by means of optimization and feedback.

EEG hardware includes electrodes, amplifiers, filters and digital processors, as shown in Figure 3.11. The electrodes are connected to the scalp to measure the electrical signal from the brain which is amplified and filtered before being sampled by a digital embedded system which connects to some form of display.

Brain waves may be assigned to one of four frequency bands. Namely: Beta, Alpha, Theta and Delta, mentioned from highest frequency till the lowest. After being filtered, these signals are provided in Figure 3.12 where the frequency ranges can also be inspected.

Electrocardiogram: ECG

An Electrocardiogram (ECG) collects recordings of the electrical activity from the heart (Sahoo et al. (2020); Wasimuddin et al. (2020)). It is an essential tool in the assessment and evaluation of cardiac-related symptoms for a patient. ECG is non-invasive and is generally used when investigating the severity of cardiovascular diseases. It is also employed to monitor patients undergoing surgery, and for observing people working in high-risk environments. Additionally, ECGs are used for research purposes in parallel with ML algorithms for automated medical diagnosis which can be more accurate and efficient than a medical expert's opinion.

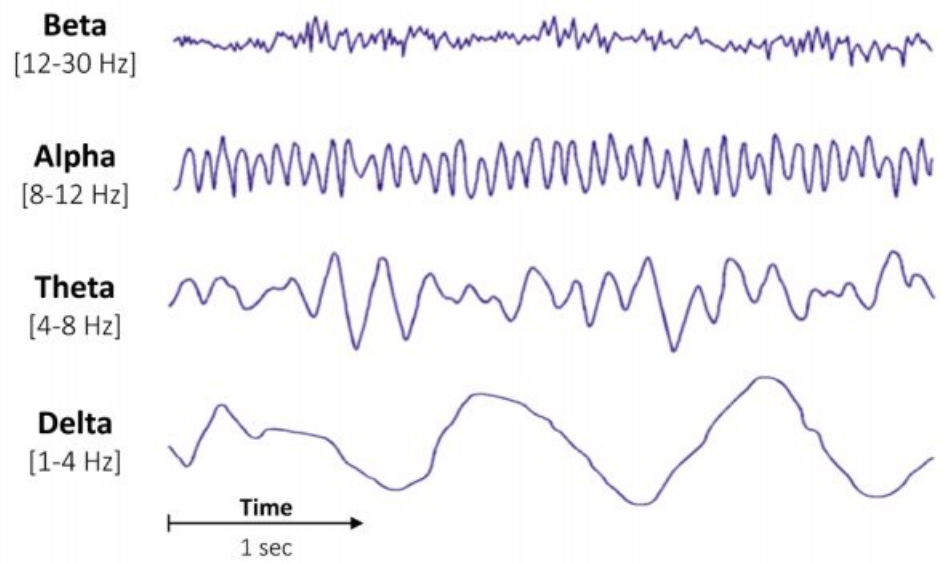

Fig. 3.11 EEG signals having different frequencies after filtering the original wave.

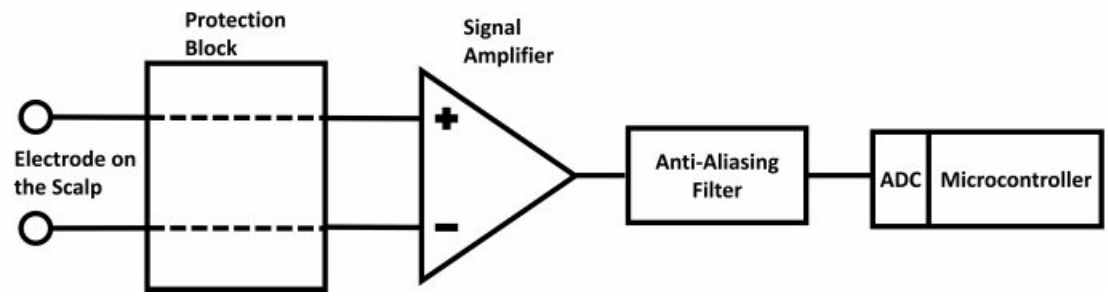

Fig. 3.12 EEG signal sampling using analog and digital electronics.

The shape of the measure ECG wave is in the shape shown in Figure 3.13 where it is defined by three steps: P, QRS and T. Although patients with arrhythmia may exhibit distorted ECG signals, which can be detected by AI Algorithms.

The hardware used to collect ECG waves is composed mainly of a virtual ground reference generator, an instrumentation amplifier, filters, an Analog to Digital Converter (ADC) and a microcontroller system for digital sampling. This can be inspected in Figure 3.14 where after amplification three filters are used to block any unwanted disruption from physical noise to power line sampling in the ECG electronic system.

Photoplethysmogram: PPG

A Photoplethysmogram (PPG) is a simple biomedical sampling device which relies on Light Emitting Diode (LED) and a photosensitive transistor for the purpose of monitoring a patient's

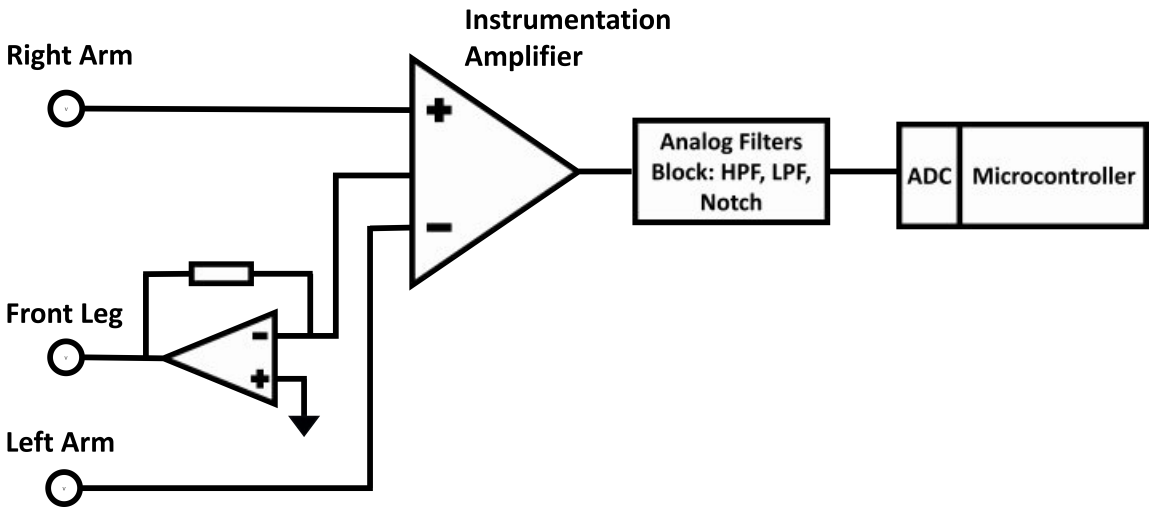

Fig. 3.13 ECG signal sampling using analog and digital filters and circuits.

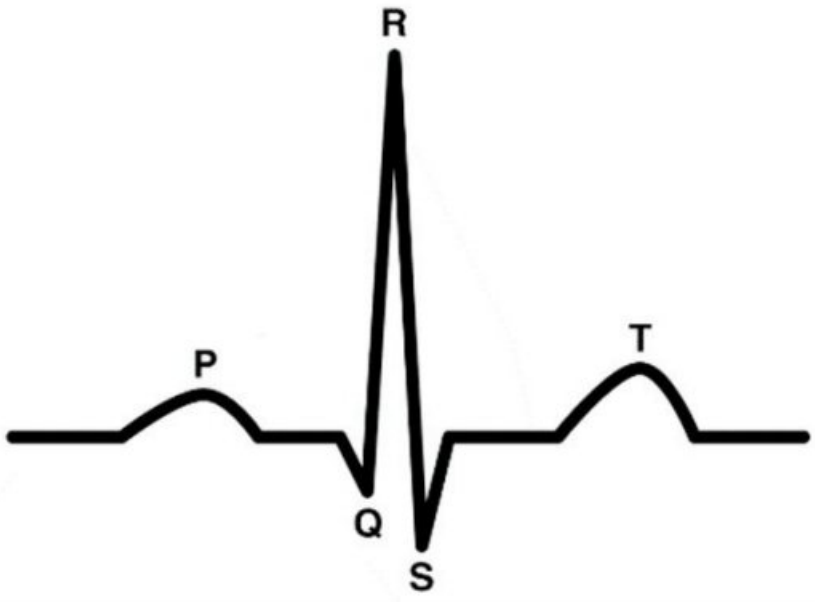

Fig. 3.14 An ECG signal having five key phases which are displayed in perfect operation.

heart rate (Chowdhury et al. (2020); El-Hajj and Kyriacou (2020); Roh and Shin (2021)). As shown in Figure 3.16 PPGs rely on non-invasive methods through emitting light into the body while a photo-detector is placed on the surface of the skin for reliable measurement. Similarly to ECG waveforms, PPG signals may be used in research for automated diagnosis and detection with the aid of ML techniques. These waveforms are illustrated in Figure 3.15 where a total of six heartbeats are present.

A PPG signal is not exclusively used in estimating heart rate, but also for pulse oxymetry readings. Additionally, the second derivative of these waveforms contains key information health-related symptoms where it assists in the early detection of cardiovascular illnesses prematurely, in addition to the detection of fatigue in a subject.

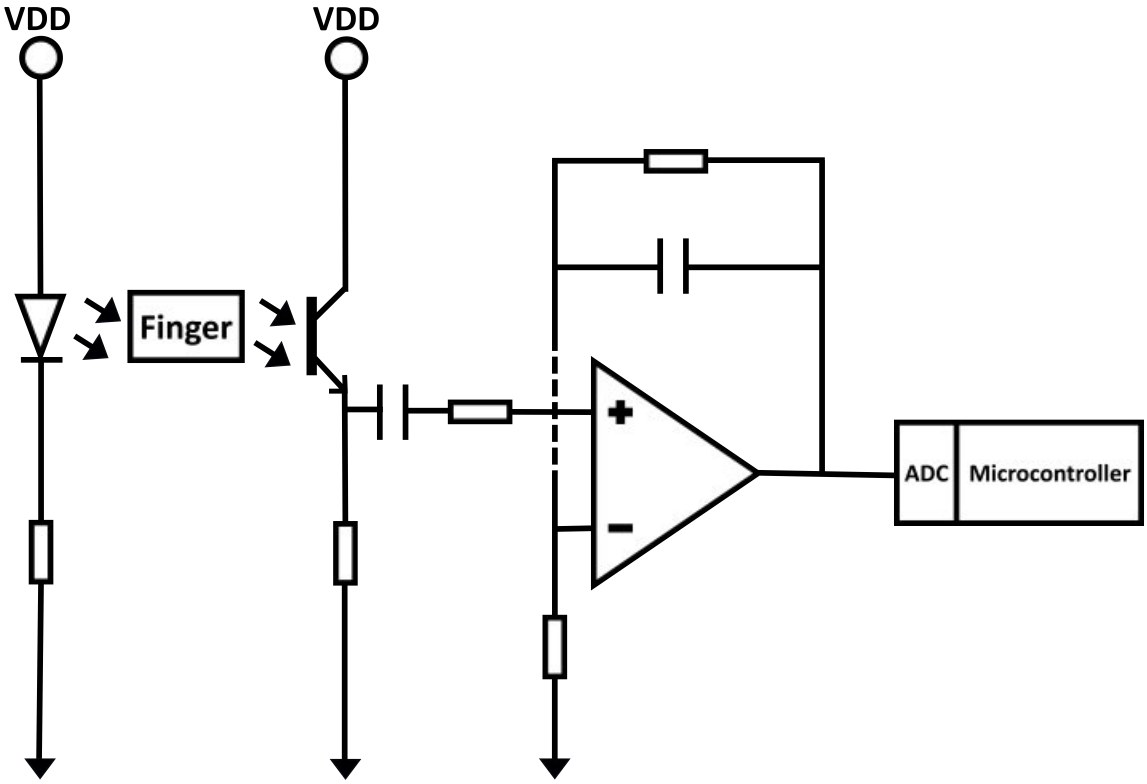

Fig. 3.15 PPG signal sampling using sampled using an analog pre-processing circuit.

Image Feature Extraction: Edge Detection

The Sobel filter (Chethan et al. (2019); Khlamov et al. (2022)) is an edge detection technique which is nothing but a mask and is used to detect edges in both the vertical and horizontal directions.

As shown in Figure 3.17, two matrices are used to apply the edge detection vertically and horizontally by means of convolution. The is evident in Figure 3.18 where the photo of a woman is passed through this filter resulting in the pencil drawing photo to the right.

The output pixel value in each image, when convolving the input, is determined by the following expression:

$$G = \sqrt{Gx^2 + Gy^2} \tag{3.16}$$

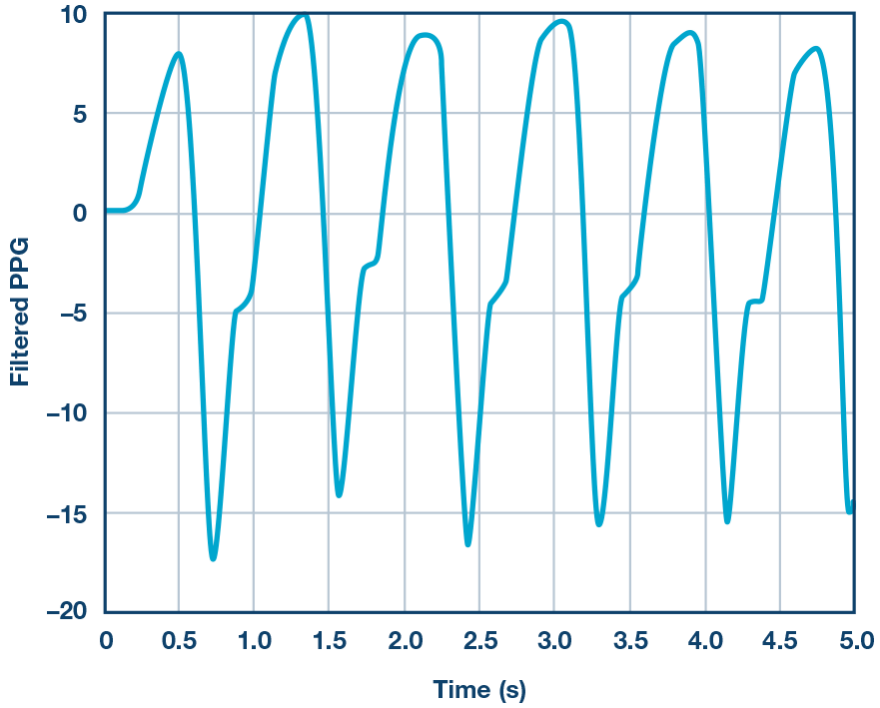

Fig. 3.16 A PPG signal sampled using a simple LED-based circuit.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Fig. 3.17 Sobel filters which are to be convolved with the input image to perform edge detection.

The angle or path of each edge is estimated as follows:

$$\Theta = atan(\frac{Gy}{Gx}) \tag{3.17}$$

Fig. 3.18 Example of edge detection using the Sobel filter.

The Sobel filter is used in Chapters 2 and 3 where its resulting output is employed with a novel FE algorithm that generates a corresponding TS applied in various ML setups related to image classifications and XAI.

3.2 State of the Art

3.2.1 General Machine Learning on the Edge-Related Work

When applying ML on Edge Computing devices, usually power consumption and resource management are crucial issues. Many works in the literature address these issues where applications spanning various fields have been reported.

Consequently, authors in Al-Khafajiy et al. (2018) present an edge computing framework for resource management optimization between IoT nodes to optimize the offloading of tasks in biomedical networks. They were able to optimize the load as well as job allocation. This architecture may be able to minimize resources management while maintaining a sustainable network paradigm for edge healthcare systems, thus stressing to the need developing software techniques for decision-making on the edge having a resource-aware perspective.

An additional key performance factor in ML on the edge is power consumption in IoT. Authors in Cecilia et al. (2020); Lapegna et al. (2021) discuss the use of light-weight AI algorithms to be applied on low-power GPU-enable boards on the edge, which is a relevant topic highlighting to low processing capabilities of traditional IoT boards and the demands for portable AI algorithms as well the ability to efficiently infer predictions. Authors inNovac et al. (2021), employ AI models on microcontrollers to address this issue by noting that

power consumption is rudimentary during ML classification on Edge Computing hardware, since their power sources may be insufficient due to packaging limitations. Moreover, in Novac et al. (2021) authors apply ML forecasts in an indoor scenario where features are collected from radio frequency measurements, which points out that the IoT boards are involved in every phase of the ML workflow, from data collection, training, inference and deployment. Forecasts in Kanawaday and Sane (2017) are applied on TS data for failure detection on a slitting machine where the data is sourced from multiple IoT sensors. This edge computing setup is based on industrial manufacturing data and hardware. this expands the scope of operation to mechanical processes and biomedical data, thus proliferating the range of applications to every other domain. Also, authors in Canedo and Skjellum (2016); Hodo et al. (2016) use AI to secure IoT network systems by employing NN, which adds the software, networking, and cryptography data as an application in ML on the Edge, thus stressing the requirement for employing AI on IoT boards, having a general applications perspective.

3.2.2 Tiny-ML Literature

Tiny Machine Learning (TinyML) is a way to compress trained ML models so they can operate and infer decisions on resource-constrained devices such as microcontrollers. Implementing AI algorithms is challenging due to limitations in power, memory, and computation. Usually, the training is performed on more powerful hardware and the classification of new data is applied after compressing the model offline.

In Ren et al. (2021) TinyOL is presented, which is a TinyML library with online learning capability. Due to the phenomenon of concept drift, a trained ML model may become less accurate over time if no online learning is performed using recent and relevant data mainly due to real-world changes. Therefore TinyOL introduced an unfrozen layer at the edge of a NN which is training by itself as a remedy to this problem.

A set of TinyML benchmarking tests have been performed and collected in Sudharsan et al. (2021) by using multiple datasets, NN models, and various IoT boards with the models being trained, compressed and then deployed on the edge. Furthermore, the experimental results and source code have been posted online as a benchmark reference for research purposes.

In Sanchez-Iborra and Skarmeta (2020) a survey describing the use of TinyML on smart and cheap embedded devices is discussed and the authors propose a multi-radio network

access architecture based on their findings. Additionally, the evaluate Tiny-ML frameworks being run on an Arduino Uno through various traditional ML models.

3.2.3 Flexible Neural Networks Literature

Flexible NNs as presented in this thesis are composed of a hybrid architecture where MLPs, a CNN and a LSTM are used in a complex setup for classification in various splits while having a global architecture which includes an architecture for XAI.

In regards to Flexible NN for classic inference and explainability, the same models are used for multiple applications generally and competitively, where three topics need to be studied:

1. Transfer Learning.
2. Application-specific ML.
3. Explainable AI.

Transfer Learning

Transfer learning is the act of using a pre-trained model, trained on original data, to be retrained partly on a different data source which is sometimes unrelated to the original application. Since numerous well-known architectures have been published with highly successful results, this approach has gained popularity where a from-the-scratch model is not available.

In Deepak and Ameer (2019), authors implement transfer learning using GoogLeNet on a Three-Class brain tumor dataset while using a k-fold setup with five folders. This setup includes the addition of a SVM at the output of the GoogLeNet which was included in the training. This setup achieves benchmark accuracy using this approach; however, it does not report the accuracy using a greater number of classes.

Skin Cancer classification using CNN is discussed in Brinker et al. (2018) where transfer learning is compared with a from-scratch setup. They conclude that CNN achieves the best performance when it is pre-trained by means of another larger dataset before optimizing the parameters of the final layers which are tuned to the target application. Authors in Kadhim and Abed (2019) implement multiple transfer learning setups using AlexNet (Lu et al. (2019)), VGG19 (Carvalho et al. (2017)), GoogLeNet (Tang et al. (2017)) and Resnet50 (Mukti and Biswas (2019)) for satellite image classification, where they conclude that Resnet50 achieves the best performance on the three different datasets. Even so, the CACAO-NET

model presented in this thesis outperforms transfer learning approaches in multiple cases and remains competitive in others, as will be presented in Chapter 7.

Application-specific features in ML Classification

FE is employed to reduce the dimensionality of datasets thus improving performance while at the same time increasing the accuracy of ML algorithms. This task is usually application-specific where it is beneficial for the specific application since the pre-processing steps should always be appropriate for the target data source. Consequently, ML algorithms are seldom preceded by FE, wherein Lu et al. (2014), a weather classification FE technique is discussed wherein addition to application-specific features, label-specific features are used. The features which have been extracted include sky and shadow detection, as well as the Sunny and Cloudy classes. Furthermore, authors in Roser and Moosmann (2008) introduce histogram-based features applied to weather data applied to driver assistance. A vector is applied to an SVM setup where the features used are related to brightness, contrast, sharpness, saturation...etc. Even though these features are not application-specific, they are only applicable to image data where the ambiance of a photo infers correct classification.

In Zhang et al. (2016), a multiple kernel learning algorithm for weather classification is presented that relies on category-specific dictionary learning. The following set of features are used: sky, shadow, rain, streak, snowflake, dark channel, contrast, and saturation. This means the sky is cropped out and that the remaining landscape and shadows are detected using a shadow detection tool taken from Sajjad et al. (2019).

Explainable AI for image classification

XAI is the process of clearly identifying the reason behind a ML classification. Usually ML programs are black-box systems. This means that the logic inferring why a specific target is classified as it was is not evident. XAI on the other hand aims to provide a way to explain to the operator the reason for every decision. This is done by adding and removing features while estimating it's effect in a mathematically intuitive fashion.

XAI is split into two disciplines: 1- local explainability, which deal with each instance in isolation, and 2- global explainability which provides rules that generalize for the entire dataset. The work in this project relates to global image explainability.

A domain-specific XAI technique is used in Stieler et al. (2021) to provide the user with reasons behind the classification of skin cancer images through the LIME tool, where the ABCD-rule is applied which is a method used by dermatologists for the diagnosis kin cancer

lesions. Scatter plots are used to display the local explainability features. In Vermeire et al. (2022) a counterfactual XAI model is provided where segments are cropped out to determine the local parts that lead to correct or incorrect classification.

Authors in Pintelas et al. (2020) implement an XAI framework for global image explainability that uses CNN as the reference model. A brain tumor dataset was chosen in the experiments undertaken where questions and answers are generated providing explicative rules. The framework outputs specific and humanistic sentences as global explainable results for the application. Firstly, in regards to global explainability, the cited work includes the tumor's size, sum and correlation scores, however, it does not provide any general information as extracted from the image dataset and model. Also, this output remains abstract as it cannot be applied to objects due to the variability in color, size, shape and settings where this model cannot provide useful information.

3.2.4 Feature Extraction and Automation Related Work

Firstly, pre-processing or FE is used to reduce the dimensionality of data and to convert measurements to a more compatible format to be used as input in a ML workflow. Secondly, post-processing is the act of applying some cognitive block or engine at the output of a ML classification for improved performance.

Feature extraction and pre-processing of TS

Raw data may not be applied directly to a ML model. Therefore, FE is performed on a TS which is mostly the case in biomedical measurements.

Authors in Fister et al. (2018) apply a Nature-Inspired Differential-Evolution algorithm is a pre-processing stage used with a Logistic Regression block. The features are selected based on an accuracy threshold which is taken from the output of the LR algorithm.

A framework for biomedical signal pre-processing (Jovic et al. (2017)) was applied to ECG time-series having examples diagnosed with myocardial ischemia, atrial fibrillation, and congestive heart failure. Since biomedical waveforms are noisy, therefore, filtering is almost always employed as a pre-processing step. The authors in Venkatesan et al. (2018) pre-process an ECG with the aid of an adaptive filter to remove the noise before using the discrete wavelet transform and extract features for classification using SVM. Wavelet transform is also applied to extract features from an EEG signal for classification using MLP in Kalayci and Ozdamar (1995).

In Khalid et al. (2014), a review on FE and feature selection in ML methods are presented. The review describes how FE is used to reduce the dimensionality in data sources through feature space transformation, and that feature selection reduces dimensionality since it points out the subset of features having the most crucial effect on classification accuracy.

A Python-based pre-processing Library which is based on nature-inspired algorithms has been developed in Karakatič (2020). The EvoPreprocess library is compatible with the ML Scikit-Learn Library and performs well compared to other frameworks.

A Bag-of-Words representation for TS is presented in Wang et al. (2013), where the presented techniques treat a TS as a text document and extract its segments as words. Ensemble learning was applied to biomedical TS in Jin and Dong (2016) where the Chinese cardiovascular disease database (Zhang et al. (2010)) was used as a data source.

Automated ML and post-processing

In the process of ML model development the hyper-parameters need to be tuned for the best performance. This also includes class-weights and feature-weights and ML model-specific parameters.

This requires expertise in ML in general and may demand ML experience in that particular application. Therefore, to facilitate the tuning process automated ML techniques have been reported in the literature and industry where these workflows may become time-consuming and require sufficient processing power and may not be applied on small IoT boards.

One way to avoid tuning is through post-processing techniques such as ensemble-learning which votes for the best results among a few others where each classifier corresponds to a standalone ML model. This method may avoid iterations but is not necessarily less compute-intensive than an optimization solution.

Meta-learning, is another automated ML method which consists of a global optimization that holds sub-problems within its sphere. The sub-problems are ML workflows which are sometimes of the same architecture but with different parameters wherein other cases they may be differing in both composition and hyper-parameters. In contrast to ensemble learning which is a parallel arrangement, this is and iterative process which may take more time depending on the hardware. It should be noted that is not uncommon to have both meta-learning and ensemble learning in the same optimization problem.

Authors in Rasp and Lerch (2018) employ an ensemble learning setup using MLP for weather forecasts by implementing diverse configurations. In Tanwani et al. (2009) the authors present guidelines for ML scheme selection for biomedical applications. They have

performed forecasts on 31 datasets and applied various ML algorithms to select the best configuration.

In (Thornton et al. (2013)), Auto-WEKA package is presented which is based on the original WEKA tool (Hall et al. (2009)), developed as a tool for easy-to-implement ML. Auto-WEKA applies Bayesian optimization to automatically select the ML algorithm with its optimized parameters. Auto-WEKA is designed for non-experts to achieve near-professional results. In Feurer et al. (2020) AUTO-SKLEARN is presented by the authors which is a Python module based on the Scikit-Learn Library. AUTO-SKLEARN implements both ensemble learning and meta-learning to automate the hyper-parameters for the most suitable ML algorithm.

The authors in Guo et al. (2017) claim that modern MLPs are poorly calibrated since increasing the depth of the network improves accuracy, however, they argue that this may affect the calibration negatively. This is due to the complexity of large MLPs where specialized architectures are a preferred solution.

3.2.5 Research Questions

From the previous sections in the present Chapter, many questions arise related to the motivation for the projects implemented in this Thesis. They encompass portability, explainability, and efficiency. The main questions are as follows:

1. Can general-purpose and commercial ML tools be deployed on limited resource boards while preserving accuracy and efficiency?

2. Is it possible to automate ML workflows by novices using a simple MLP model with competitive accuracy and with high efficiency on the Edge?

3. Can light-weight and general-purpose ML models with a competitive performance which are deploy-able on the Edge be used as reference models for Global Image explainability, which is itself a standalone task?

The projects presented in this thesis aim to respond to these questions by developing Automated FE and light-weight post-processing techniques that are light weight, yet still capable to be deployed on the edge, while remaining competent with traditional offline approaches and having the versatility to perform at a high level across AI disciplines.

Chapter 4

Thesis Contributions and Publications

4.1 Thesis Contributions

In this section, the contributions for each research project are presented and outlined and mostly specific to the fields of ML and automation. The contributions are arranged based on every project (grouping one or more publications) rather than specifying those for each publication mainly to avoid any overlap.

4.1.1 Contributions for the ML on the Edge and Tree-based Multi-class Classification Project

The main contributions performed in the first project consist of converting images into TS using statistical radial scanning for ML classification. Additionally, to the best of our knowledge, the tree-based multi-class approach is the first algorithm using this methodology to be published. These contributions can be viewed in the following list:

1. Multiclass classification using a tree-based methodology.
2. Image-to-TS FE based on radial scanning.
3. Porting Rulex onto the Raspberry Pi.
4. Classifying gender using inconsistent images with competitive accuracy while outperforming other techniques which rely on a controlled environment and consistent subject criteria.

4.1.2 Contributions for the Flexible Neural Networks and Explainability Framework (CACAO-X)

The main contributions performed in the second project consist of the development of a general-purpose ML model dubbed CACAO-NET meaning Contour Assisted Constitutional Neural Networks with Image Explainability and Inference on the Edge. The model achieved accurate results in terms of validation an test on the edge as well as being part of the overall CACAO-X framework which performs global image XAI with the aid of novel algorithms and a new architecture for this purpose.

1. Classifying images and their extracted TS with hybrid NN.
2. Using a model and its local explainable features for global explainability.
3. Using a model with application-specific landmarks for global explainability.
4. Using explicative labeling with an explainability algorithm to generate rules.
5. A callback function which tracks the mean and deviation of the validation accuracy.

4.1.3 Contributions for the AI Pre-processing and Post-processing Framework (*VAMPIRE*)

The framework which is called *VAMPIRE* consists of novel Pre-processing and Post-processing algorithms that extract features from TS and tune NN models without heavy structural and parameter manipulation. Additionally, the tree-based technique previously mentioned is written in the form of a complexity notation to provide a mathematical model for the algorithm. The summary of the contribution to be found in this Project/Publication can be viewed below:

1. *VAMPIRE FE1* for mundane Time-series.
2. *VAMPIRE FE2* for semi-annotated Time-series.
3. *VAMPIRE Activation Engines* for NN threshold extraction.
4. The Complexity of the multiclass tree-based approach.
5. The introduction of the D-Domain (V.Hz-Domain)

4.2 Publications listing

The work described in this thesis has allowed the publication of five scientific manuscripts confirming the validity of the results achieved and which report a summary of the experimental results obtained. They consist of one book chapter and four scientific articles:

The following are the publications corresponding to Project 1 entitled "Classic ML on the Edge and Tree-based Multi-class Classification":

- Daher, A.W., Rizik, A., Randazzo, A., Tavanti, E., Chible, H., Muselli, M. and Caviglia, D.D., 2020. Pedestrian and multi-class vehicle classification in radar systems using rulex software on the raspberry pi. Applied Sciences, 10(24), p.9113.

  https://doi.org/10.3390/app10249113

- Daher, A.W., Rizik, A., Muselli, M., Chible, H. and Caviglia, D.D., 2020, November. Porting Rulex Machine Learning Software to the Raspberry Pi as an Edge Computing Device. In International Conference on Applications in Electronics Pervading Industry, Environment and Society (pp. 273-279). Springer, Cham.

  https://doi.org/10.1007/978-3-030-66729-0_33

- Daher, A.W., Rizik, A., Muselli, M., Chible, H. and Caviglia, D.D., 2021. Porting Rulex Software to the Raspberry Pi for Machine Learning Applications on the Edge. Sensors, 21(19), p.6526.

  https://doi.org/10.3390/s21196526

The following publication corresponds to Project 2 entitled "Flexible Neural Networks and explainability":

- Daher, A.W., Ferrari, E., Verda, D., Chible, H., Muselli, M., and Caviglia, D.D., 2022. CACAO-X: Contour Assisted Convolutional Neural Networks with Global Image Explainability and Edge Computing Classification. Preprint.

  https://doi.org/10.2139/ssrn.4250119

The following publication corresponds to Project 3 entitled "The AI Pre-processing and Post-processing Framework":

- Daher, A.W., Ferrari, E., Muselli, M., Chible, H. and Caviglia, D.D., 2022. *VAMPIRE*: vectorized automated ML pre-processing and post-processing framework for edge applications. Computing, pp.1-35.

  https://doi.org/10.1007/s00607-022-01096-z

Part II

Artificial Intelligence for the Edge Computing Paradigm

Chapter 5

Radar Classification using ML on the Edge

5.1 Chapter Abstract

Nowadays, cities can be perceived as an increasingly dangerous place. Usually, Closed Circuit Television (CCTV) is one of the main technologies used in a modern security system. However, poor light situations or bad weather conditions (rain, fog, etc.) limit the detection capabilities of image-based systems. Microwave radar detection systems can be an answer to this limitation and take advantage of the results obtained by low-cost technologies for the automotive market. Transportation by car may be dangerous, and every year car accidents led to fatalities of many individuals. Humans require automated assistance when driving through detecting and correctly classifying approaching vehicles, and more importantly, pedestrians. In this thesis, we present the application of ML to data collected by a 24 GHz short-range radar for urban classification. The training and testing take place on a Raspberry Pi as an edge computing node operating in a Client/Server arrangement. The software of choice is Rulex, a high-performance ML package being controlled through a remote interface. Forecasts with a varying number of classes were performed with one, two, or three classes for vehicles and one for humans. Furthermore, we applied a single forecast for all four classes, as well as cascading forecasts in a tree-like structure while varying algorithms, cascaded blocks order, setting class weights, and varying the data splitting ratio for each forecast to improve prediction accuracy. The presented approach achieved an accuracy of 100% for human classification and 96.67% for vehicles overall, which is useful in practice. The proceeding subclasses were predicted with accuracies of 90.63% for Motorcycles, and the same accuracy of 77.34% for both the Cars and Trucks subclasses.

5.2 Introduction

As cities are getting smarter, and as the spread of intelligent surveillance technologies is gaining popularity within developed countries, making urban transport secure and efficient plays a key role in the safety of individuals as well as in affecting traffic flow, which in terms may negatively impact businesses within a city (Carter (2020)).

Unlike video cameras, the operation of short-range microwave radars is not much affected by the presence of adverse weather conditions. This fact makes them ideal, in addition to classical CCTV systems (Norris et al. (2004)), for operating round-the-clock automatic surveillance in an urban environment. Radars (Prophet et al. (2018); Rizik et al. (2019); Rohling et al. (2010)) can be used for vehicle and pedestrian classification by relying on FE from the range and Doppler profiles of each target. The data collected by radar measurements can be used as input to ML algorithms for classification. However, the hardware should be low-cost, lightweight while providing good performance. So, for this application, we have chosen a Raspberry Pi (Maksimović et al. (2014); Vujović and Maksimović (2014); Zhang et al. (2018)), a small portable edge computing device, which is a very effective platform for real-world scenarios as well as for educational and research purposes. Raspberry Pi is ideal for Edge Computing applications, where the node or embedded device possesses high processing capabilities and is required to have enough storage space to avoid cloud access. With the large number of wireless sensor nodes that are used over a wide range of applications, from wearable sensors to image processing and surveillance, along with the integration of AI and ML in their decision-making, they are required to be as smart as possible to avoid cloud access and reduce network traffic. Urban classification may not have cloud access and requires low latency, so it is the ideal example where both training and testing need to be applied on an edge computing node.

The main goal of the present work is the porting of the state-of-the-art ML package Rulex (Muselli (2012)) onto the Raspberry Pi computational platform. The dependencies were compiled to produce output binary files that are compatible with the target platforms, the first being the Windows 32 Bit Client and the second being the Raspberry Pi Server, which is where the Rulex engine runs.

The remainder of this thesis is organized as follows: Section 2 presents a review on Pedestrian–Vehicle Classification, Section 3 introduces the porting of Rulex on the Raspberry Pi, and Section 4 describes the adopted ML architecture. Also, Forecast results obtained with the present implementation are presented in Section 5. Finally, we draw some conclusions in Section 6.

5.3 Chapter Related Work

Radar systems can be used for detecting and classifying different targets, such as pedestrians and vehicles (Prophet et al. (2018); Rizik et al. (2019); Rohling et al. (2010)). Indeed, such systems produce, through a proper antenna, an electromagnetic wave that propagates to the objects eventually located in the inspected scenario. The targets interact with the impinging radiation employing the well-known scattering mechanism, generating a scattered field that partly returns to the radar receiver. Specifically, the reflected waves contain information about the characteristics of the objects that generated them. In this work, a 24 GHz Continuous-Wave Frequency-Modulated (FMCW) radar system has been used (Rizik et al. (2019)). Consequently, the main information that can be obtained is related to the range and radial speed of the objects. In particular, such quantities are embedded in the frequency shift due to the propagation delay and to the Doppler effect (which is always present when dealing with moving targets) and are extracted from the range-Doppler maps obtained by performing a double Fast Fourier Transform (FFT) of the acquired data (Rizik et al. (2019); Wojtkiewicz et al. (1997)).

From these measurements, it is possible to derive features to be used for ML classification. Specifically, the ML features used for training are the extension of the range and velocity profiles, as well as the standard deviation, mean, and variance for the same variables, in addition to the radar cross-section and the estimated target velocity (Rizik et al. (2019)). However, in vehicle classification, there is the issue of lateral moving vehicles (i.e. along a direction perpendicular to the radar axis), which may be mistaken for pedestrians (Heuel and Rohling (2012)). Indeed, longitudinal moving vehicles (i.e. traveling along the direction of the radar axis) have a large range profile and a point-shape velocity profile on the range-Doppler diagram. The opposite is true for pedestrians, due to multiple velocities caused by the movement of limbs. As for lateral moving vehicles, their range profile is comparable to the range profile of pedestrians and their velocity profile may approach that of longitudinal moving vehicles, so it is needed to calculate the lateral and longitudinal velocities, and with these additional features, we attempt to avoid misclassification. Another feature is the Radar Cross section (RCS), which is the equivalent scattering surface of the target seen by the radar and is related to the amount of power that is reflected by the object (Liaqat et al. (2011); Skolnik (2001)). In the rest of this work, we will refer to data obtained with the system described in Rizik et al. (2019) that have been made available to us courtesy of the authors.

5.4 Porting Rulex to Raspberry Pi

5.4.1 AI on the Edge Fundamentals

Due to increase use of online data and the rise of portable smart devices and the IoT, online traffic congestion costs and resource demands are being exhausted. Additionally, ML models are becoming ubiquitous in every engineering system or device, from smartphones to transportation, manufacturing and all consumer electronics. consequently, Edge Computing has been introduced, where the data traffic and processing are performed on IoT devices which are distant from the concerned server or data center, hence the term: Edge.

However, Edge Computing devices remain less powerful when compare with servers or even larger personal devices, therefore, the algorithms being run on IoT devices need to take into account their processing power, their storage and speed (or Latency).

When it comes to ML predictions on the Edge, especially with large datasets and image data, specific pre-processing or FE needs to be performed on the input that for practical operation. Also, the ML models themselves have to be lightweight in order to accommodate for the memory and engine of the IoT device.

In this project, the target is to prove that the employed ML models and the newly introduced conversion algorithms are able to run on small general-purpose IoT board with competitive results while maintaining tolerable inference speed and power consumption-

5.4.2 System Setup

Rulex is a ML Software that supports various ML algorithms that can be easily applied in a user-friendly environment (Muselli (2012)). The Rulex Graphical User Interface (GUI) provides a means of importing training data and manipulating it before applying ML Algorithms. The main proprietary algorithm for Rulex is the Logic Learning Machine (LLM) (Muselli (2005); Muselli and Ferrari (2009)) which implements XAI. LLM was the main algorithm used for most of the classifications, where a tree-based structure, that combines vehicle classes to achieve more accurate results, was adopted. Then, these new combined classes are split recursively until all vehicles have been classified in their respective sub-classes.

The Raspberry Pi is a credit-card-sized Personal Computer (PC), which can perform application-specific tasks as well as performing general-purpose everyday computing, where it can connect to most PC Input/output hardware. The Raspberry Pi has multiple Digital Input/Output pins which can be used in embedded system applications such as motor control,

serial communications, Liquid Crystal Display (LCD), and interfacing with a practically infinite variety of sensors.

Nowadays, IoT devices are becoming more intelligent because they support AI software and algorithms, so in this work, we have deployed Rulex to operate on the Raspberry Pi which is one of the most popular IoT hardware platforms. In order to port a software package from one platform to another, all of its internal and external dependencies should be compiled on the target platform. After compilation with a specific tool, binaries or executables are generated, which are a formatted version of the code to be linked to succeeding layers of the source code. Furthermore, before porting software from 64 Bits to 32 Bits, all of its dependencies should be compiled in 32 Bits. Visual Studio may be used to compile the libraries and code when porting to Windows 32 Bits. However, when porting to Linux, we used CMake (Clemencic and Mato (2012); Fober et al. (2018)), which is a cross-platform application for generating executables or libraries.

Rulex external libraries were ported to 32 Bits as the first step before compiling the entire code. We ported the source code on Windows 32 Bits which is the interface and Raspbian 32 Bits which is where the engine runs. During porting, one of the issues we faced was the incompatibility of some data types with 32-Bit hardware and software, so they were either changed or cast into a compatible data type. Another issue is the inability to generate a larger number of threads, so we developed two software tests, where one was written in Python and the other in C-plus-plus (Cpp) as an attempt to find out what was the maximum number of threads that could be generated. Moreover, the source code was modified accordingly to optimize the maximum number of threads. After compiling locally on Windows 32 Bits, we proceeded to remotely compile the source code. Rulex was also debugged remotely and made compatible with both Operation Systems (OS) by using various Macros and correctly setting variable types.

5.5 Classification architecture

5.5.1 Classification methodology

In urban classification, there are usually four classes: a pedestrian class and three vehicle classes. We can go about and run forecasts by relying on multiple algorithms in one overall simulation. Rulex possesses various algorithms to choose from, such as NN, KNN, DR, SVM, and LLM, all of which can be used for classification, however, since the adopted methodology applies multiple ML algorithms in a cascaded setup and tests multiple arrangements, which

can have a large number of forecasts, we have included just NN and LLM. This has been adopted since NN is a widely used ML algorithm, and also LLM for being the commercial algorithm of Rulex. Since the vehicle class can be split into 3 sub-classes, namely cars, trucks, and motorcycles, and since there is no need to differentiate between them in real-world scenarios, we have taken advantage of this fact by applying a sub-class-based tree structure, where the classes are nodes and the algorithms are branches. The forecasts of the tree-based method can lead to improved validation results by setting different weights in each forecast for every class, and by choosing a convenient split ratio between training and testing data, also separately for each forecast, where the split method used is holdout validation. It is also possible to use different classification algorithms in each forecast on each branch. This is useful in case the adopted algorithm is not generating the expected results for the dataset at hand. Figure 5.1 presents such a tree structure, where the leaves are the final classification outputs. As presented, different types of algorithms were applied to the final branches of the tree since they possess the lowest success rate. So, the prediction accuracy is optimized by varying weights, data splitting, and more critically the algorithm used for classification. The further we go down the tree the harder it gets to differentiate between classes.

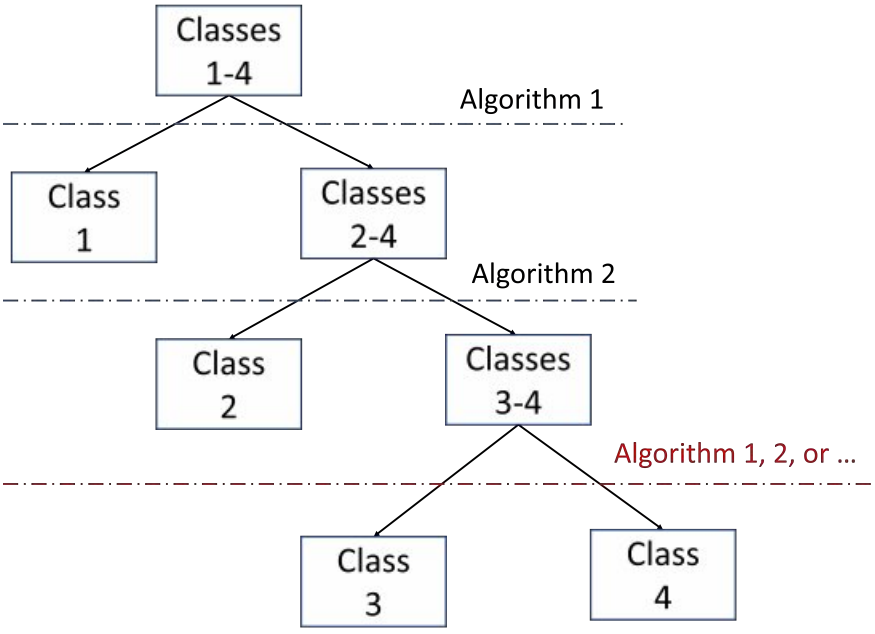

Fig. 5.1 Sub-Class based Tree Structure.

5.5.2 System set-up

The ML system used consists of the Rulex Engine running on the Raspberry Pi as an application server, which is where forecasts are applied. This Engine is accessed through a graphical Interface running on a windows Client, while a PostgreSQL Server is used as the common storage point between both nodes.

The Data acquisition system has been described in Rizik et al. (2019). It is a standalone system dedicated to FE developed and operated separately concerning the Client/Server Raspberry Pi arrangement used to run Rulex for target classification. It consists of a Distance2Go radar board developed by Infineon technologies (Infineon (2022)), a Raspberry Pi 3 B+, and a video camera oriented in the same direction as the radar beam. The system supply is provided by a 5 V, 10 Ah power bank that is sufficient for operating the whole system for several hours: for this wok, it was installed on an internal road of the DITEN department of the University of Genoa. The system collects the raw data from the radar board through a Universal Serial Bus (USB) connection and extracts the features which are then forwarded to the processing chain of the Rulex software for classification.

5.6 Results and Discussion

The dataset collected by the data acquisition system described consists of 120 rows equally distributed into 4 classes with 30 patterns for every class. The features consist of the mean, variance, and standard deviation of the range and Doppler profiles, along with their reflectivity and the estimated velocity of the target.

In order to maximize forecast accuracy, we have applied multiple tree-based subclass arrangements to simulate using Rulex in a Client/Server setup (Hajdarevic et al. (2014)). As stated earlier, multiple cascaded simulations were applied with a varying number of classes as well as cascaded order. So, a summary of all the applied forecasts is presented in Figure 5.2, and described in detail in this Section. In Figure 5.2, the red labels stand for the Cases and ML algorithms used in that particular simulation, and the green labels represent classes that will be split into subclasses in the upcoming simulation. In Case 1, we split the data into two classes: Humans, and vehicles where LLM was used. The accuracy is shown in Table 5.1.

In Case 1, the ML algorithm used is LLM for classification. However, in Case 2 we only consider vehicle sub-classes. The simulation was applied using LLM where the prediction accuracies are found in Table 5.2: In Case 3 we apply forecast using LLM for vehicle classes

Table 5.1 Humans and Vehicles training and testing prediction accuracy.

Class	Training	Testing
Humans	100%	100%
Vehicles	95%	100%

Table 5.2 Vehicles, 3 Classes training.

Class	Training	Testing
Cars	96%	75%
Motorcycles	94%	91%
Trucks	100%	72%

Table 5.3 Cars/Trucks and Motorcycles training and testing prediction accuracy.

Class	Training	Testing
Motorcycles	100%	91%
Cars/Trucks	98%	100%

by splitting the data into 2 classes as shown in Table 5.3. In Cases 4 and 5, the Cars/Trucks class has been split into two sub-classes, cars and trucks. In Case 4 we use NN, whereas LLM has been used in Case 5. The results of these can be found in Tables 5.4 and 5.5 respectively.

Table 5.4 Cars and Trucks with NN training and testing prediction accuracy.

Class	Training	Testing
Cars	95%	87%
Trucks	100%	70%

Cases 1 to 5 were processed separately to get a glimpse of how LLM would perform with this given dataset. From the outputs generated in Tables 5.1 to 5.5, we can estimate the overall prediction accuracy for a cascaded setup. Furthermore, it should be noted that misclassified records in preceding forecasts will be treated as correctly classified in upcoming forecasts, which leads to the overall accuracy of the cascaded system being incorrectly estimated. The preceding forecasts were all done with a 70%/30% split for training and testing data respectively, and with all weights being set to unity. Furthermore, we can apply multiple cascaded setups which are based on the previous forecasts. If we cascade Cases 1 and 2, the projected output is presented in Table 5.6. In Case we cascade Cases 1, 3, and then 4, the projected output is provided in Table 5.7.

Table 5.5 Cars and Trucks with LLM training and testing prediction accuracy.

Class	Training	Testing
Cars	95%	100%
Trucks	100%	66%

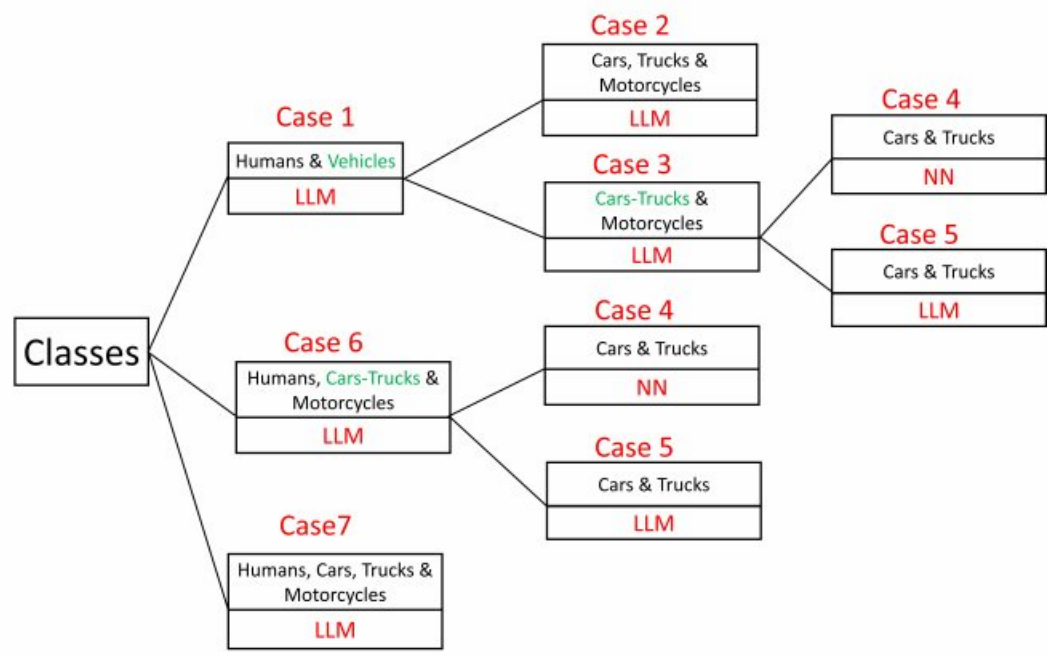

Fig. 5.2 Cases 1 – 7 featuring all applied forecasts presented in this article.

Table 5.6 Cases 1 then 2 prediction accuracy.

Class	Forecast
Humans	100%
Motorcycles	91%
Cars	75%
Trucks	72%

Table 5.7 Cases 1, 3, and then 4 prediction accuracy.

Class	Forecast
Humans	100%
Motorcycles	91%
Cars	87%
Trucks	70%

If we cascade Cases 1, 3, and 5 we get the results shown in Table 5.8. Other variations of initializing the cascaded system with LLM can be found in Table 5.9, which is Case 6 where one class for humans along with 2 classes for vehicles are taken.

Finally, a single forecast for all 4 classes which is applied using LLM is presented in Table 5.10, namely Case 7, that consists of forecasting all classes in a single block. With the variation added in Tables 5.9 and 5.10, we can apply two additional combinations to cascade. So, we can cascade Case 6 followed by Case 4 which employs NN or we can cascade it with Case 5 which uses LLM. These last two combinations include a situation where the previous prediction was not 100% accurate, so we need to take that into account when theoretically estimating the overall accuracy. When combining Case 6 followed by Case 4, the Cars/Trucks class has an accuracy of 94%, so naturally, the Neural Network predictions in Case 4 will be multiplied by 0.94. The same can be said for Case 5, where the Cars and Trucks classes' success rates are multiplied by the same factor. Tables 5.11 and 12 provide the projected output forecast accuracy for the last two scenarios.

Table 5.8 Cases 1, 3, and then 5 prediction accuracy.

Class	Forecast
Humans	100%
Motorcycles	91%
Cars	100%
Trucks	66%

Table 5.9 Humans and Vehicles 2-Classes training and testing prediction accuracy.

Class	Training	Testing
Humans	94%	69%
Motorcycles	94%	100%
Cars/Trucks	97%	94%

Table 5.10 Default LLM forecast training and testing prediction accuracy.

Class	Training	Testing
Humans	100%	73%
Cars	100%	100%
Motorcycles	84%	91%
Trucks	90%	72%

Table 5.11 Cases 6 then 4 prediction accuracy.

Class	Forecast
Humans	100%
Motorcycles	69%
Cars	0.94*87 = 82%
Trucks	0.94*70 = 66%

Table 5.12 Cases 6 then 5 prediction accuracy.

Class	Forecast
Humans	100%
Motorcycles	69%
Cars	0.94*100 = 94%
Trucks	0.94*66 = 62%

All the preceding simulations only provide an estimation of actual results, when cascading multiple engines. This is due to not considering the false-positive Cases of the forecast. We took into consideration the entire dataset for each algorithm block and ignoring some of the rows which were correctly classified in the abandoned class. In Case two algorithms are cascaded, the first block should be followed by a Rulex Data management block which will filter out the True and False positives in the abandoned class and remove them from the table. However, we still have to multiply the proceeding blocks with their parent class's success rate to calculate the overall accuracy.

For Case 1, we split that data 70/30, with 70% used for training and 30% being used for testing. The same was applied for Case 3. However, for Case 4, with the reduced number of rows, the data was split 65/35 with 65% used for training and the remaining 35% being used for testing. The main reason for changing the split ratio in Case 4 is due to the fact the prediction is applied to half of the dataset, and we found that increasing the size of the test-set can lead to higher accuracy for the given data.

As for weights, the only way to set them and optimize results is by trial and error, and intuition. There is no universal method to select weights accordingly. Unity gain in the Case 1 block already should provide very good results, so there is no need to change the weights. With a unity gain, in Case 3, the Cars-Trucks class, which will be used in the proceeding block, should be accurate while keeping the Motorcycles class forecast precise enough. A gain of 1.5 was chosen for the Cars-Trucks field and 1.0 for Motorcycles. As for the final

block, which is Case 4, both Trucks and Cars classes which originate in Case 3, have an equal True positive rate of 80% in testing. So, weights are left at unity.

Table 5.15 represent the accuracy for training and testing of Cases 1, 3, and 5 respectively, and as predicted using Rulex. All the forecasts present good results for testing. Humans were detected with a rate of 100%, and vehicles overall at a rate of 96.67%. In Case 3, which is block 2, the Cars-Trucks class has a true positive rate of 93.75% and motorcycles at 90%. As for the Cars and trucks block, which is Case 5, the success rate is 80% for both trucks and cars. Tables 5.13 and 5.14 present the overall output true and false-positive rates for the chosen All-LLM forecast. Humans are detected without any errors for the test dataset. The overall forecasts of the motorcycles, cars, and trucks have been calculated based on the preceding forecasts to become 90.63% for Motorcycles and 77.34% for both cars and trucks. Furthermore, the overall prediction of vehicles is 96.67%, which can be useful in practice.

Table 5.13 Main forecast.

Class	Forecast
Cars	100%
Humans	96.67%

Table 5.14 Forecast for Vehicles.

Class	Forecast
Motorcycles	0.9667*93.75=90.63%
Cars	0.9667*80 = 77.336%
Trucks	0.9667*80 = 77.336%

Table 5.15 Forecast for Vehicles.

Case	Training		Testing	
Case 1	Humans	Vehicles	Humans	Vehicles
	100%	95.82%	100%	96.67%
Case 2	Cars/Trucks	Motorcycles	Cars/Trucks	Motorcycles
	100%	100%	100%	93.75%
Case 3	Cars	Trucks	Cars	Trucks
	95%	100%	80%	80%

5.7 Chapter Conclusion

The ML software Rulex has been ported to the Raspberry Pi in a Client/Server setup, to apply for edge computing applications. The device was used to make forecasts on a pedestrian and vehicle classification dataset for urban security applications. Multiple forecasts were cascaded in a tree-like structure while tuning the parameters of every forecast. Classes were split into subclasses and single process simulations were applied, where we've estimated the overall accuracy for various cascaded setups. After, exhausting all the possible arrangements, the setup with the best-projected output was simulated in a cascaded configuration, which provides an improved prediction outcome. This approach achieves higher accuracy over the classical approach of applying a single forecast for all classes. Also, combining classes into a parent class can be useful in practice such is the case with the vehicles parent-class. However, this approach is exhaustive and time-consuming and may require setting parameters for different forecasts and various ML algorithms. Moreover, the tree-based method for improving ML forecasts can be used in various configurations. After applying the proposed method, humans were classified with an accuracy of 100% and vehicles with an accuracy 96.67%. The final vehicles subclasses forecast accuracies are 90.63% for Motorcycles and 77.34% for the Cars and Trucks classes. In regards the this present radar dataset, using Rulex LLM on the edge, the overall accuracy is 86.32% which outperforms the results reached in Rizik et al. (2021) by 1.3% mainly due to the tree-based setup, newly introduced in this thesis.

Chapter 6

Porting ML Software on the Raspberry Pi and Image to TS Feature Extraction

6.1 Chapter Abstract

Edge Computing enables to perform measurements and cognitive decisions outside a central server by performing data storage, manipulation, and processing on the IoT node. Also, AI and ML applications have become a rudimentary procedure in virtually every industrial or preliminary system. Consequently, the Raspberry Pi is adopted, which is a low-cost computing platform that is profitably applied in the field of IoT. As for the software part, among the plethora of ML paradigms reported in the literature, we identified Rulex (Muselli (2012)), which can be found online in Muselli (2022), as a good ML platform, suitable to be implemented on the Raspberry Pi. In this thesis, we present the porting of the Rulex ML platform on the board to perform ML forecasts in an IoT setup. Specifically, we explain the porting Rulex's libraries on Windows 32-Bits (WIN32), Ubuntu 64 Bits, and Raspbian 32 Bits. Therefore, with the aim of carrying out an in-depth verification of the application possibilities, we propose to perform forecasts on five unrelated datasets from five different applications, having varying sizes in terms of the number of records, skewness, and dimensionality. These include a small Urban Classification dataset, three larger datasets concerning Human Activity detection, a Biomedical dataset related to mental state, and a Vehicle Activity Recognition dataset. The overall accuracies for the forecasts performed are: 84.13%, 99.29% (for SVM), 95.47% (for SVM), and 95.27% (For KNN) respectively. Finally, an image-based gender classification dataset is employed to perform image classification on the Edge. Moreover, a novel image pre-processing algorithm was developed that converts images into TS by relying on statistical contour-based detection techniques. Even though the dataset contains

inconsistent and random images, in terms of subjects and settings, Rulex achieves an overall accuracy of 96.47% while competing with the literature which is dominated by forward-facing and mugshot images. Additionally, power consumption for the Raspberry Pi in a Client/Server setup was compared with an HP laptop, where the board takes more time, but consumes less energy for the same ML task.

6.2 Introduction

The IoT paradigm is rapidly extending to many sectors of society because it allows to substantially improve the monitoring or control of complex and extensive processes, offering an innovative approach for multiple fields of application, such as quality of life, urban challenges, logistics, agriculture and livestock, climate change, mass production, health, energy and water production and distribution, and many more.

The huge amount of data produced by the nodes of the networks of which the IoT is made up must be processed in an efficient and effective way, and ML techniques are certainly among the most suitable for this purpose. Therefore, it is straightforward that ML tools can play a key role in further expanding the scope of applications, as well as their effectiveness. In past implementations, ML forecasts have been performed on a remote server before delivering results on the IoT Computing Node to limit network traffic, Edge Computing (Premsankar et al. (2018); Yu et al. (2017)), setups can be employed to avoid intensive cloud access and keep that data storage and processing on the IoT device as much as possible.

Our work is placed in this perspective. Notably, this thesis reports an extended version of the work previously presented at ApplePies 2020 (Daher et al. (2020a)). The system we report is based on the Raspberry Pi platform , which is a low-cost, low-power credit-card-sized board that is used for embedded system and general-purpose computing applications. As for ML software, we have adopted for this investigation Rulex (Muselli (2012)), an AI environment intended for non-domain experts, and we have ported it to the Raspberry Pi platform.

Rulex was ported to three different OS, namely to WIN32, Ubuntu 64 Bits, and on Raspbian 32 Bits which is the official OS of the Raspberry Pi. All external and internal dependencies have been compiled (Daher et al. (2020a)) and verified. Moreover, the Client/Server setup has been used to perform forecasts on the edge after debugging the software through its source code.

To explore the application possibilities of this operating environment, in addition to the Radar Classification dataset already reported in (Daher et al. (2020a)), three new pre-

processed datasets taken from diverse domains were also implemented using Rulex on the Edge. These encompass a Human Activity Detection dataset using Smartphones, a Brainwave Mental State Classification dataset, and an Activity Recognition dataset for Dumpers in earth moving sites that are also recorded using Smartphones. We also investigated the problem of performing gender detection. To this end, a new pre-processing Algorithm for image classification was developed to convert facial images into TS using a contour-based approach.

The contributions presented in this thesis include the porting of a high-performance ML package on the Raspberry Pi in a Client/Server setup, as well as the development of a novel pre-processing Algorithm that converts images into Time-series using statistical measurements, that are directly applicable to any ML configuration. The methodology in this chapter consists of compiling all the required libraries on the target platforms. The correct version of the libraries should be chosen where all libraries should be compatible with their predecessor since libraries may be built on top of each other. Also, in the linking process, the code needs to be changed to satisfy all the target platforms Windows 64-bits (WIN64), WIN32, Raspbian 32-bits, and Ubuntu 64-bits). Finally, the fully featured Rulex software package can be run on open-source hardware. Additionally, the pre-processing contribution and the ML tests on the system using multiple datasets and, in a Client/Server arrangement demonstrate the system effectiveness.

In the rest of this chapter Section 2 presents the literature review, Section 3 describes the actions taken to bring Rulex on Raspberry Pi in a Client/Server arrangement (Hajdarevic et al. (2014)), Section 4 presents in detail the image pre-processing algorithm, and reports the results of all the ML forecasts and energy consumption achieved. Finally, Section 5 draws the conclusions.

6.3 Chapter Related Work

6.3.1 Hardware platform

The hardware platform adopted in this work is the multi-purpose Raspberry Pi (Maksimović et al. (2014); Vujović and Maksimović (2014)). It was conceived to handle application-specific tasks, as well as being used for everyday computing operations. It supports USB, High Definition Multimedia Interface (HDMI), and Secure Digital (SD) Card connections, in addition to having standard digital Input/Output pins controlled through the on-board ARM microcontroller.

Regarding Communication protocols, the Raspberry Pi ports Local Area Network (LAN) and Wi-Fi connectivity. Furthermore, the Raspberry Pi's digital pins can be used for interfacing with a wide variety of peripherals, for applications ranging from motor control, LCD Display functions, and many more, and can be interfaced with compatible smart sensor modules.

6.3.2 Machine Learning platform

The AI suite named Rulex (an acronym for Rule Extraction) has been created specifically for the management, the visualization, and the analysis of data: it consists in an integrated visual platform which allows to perform any operation in a simple and direct way, freeing the user from the necessity of knowing implementation details about memorization and execution (Muselli (2012)). It actually implements many ML Algorithms (both supervised and unsupervised) such as LLM (Parodi et al. (2018)), NN, DT, KNN, and others through an easy-to-use GUI. Specifically, the LLM is a method of supervised analysis based on an efficient implementation of the Switching NN (Muselli (2005)) and Monotone Boolean Reconstruction (Muselli and Quarati (2005)) through the Shadow Clustering technique (Muselli and Ferrari (2009)) and is able to extract intelligible rules from data.

Rulex, through its GUI, also allows you to choose and apply other standard ML algorithms to perform predictions. Through the interface, it is also possible to manipulate and filter data before applying forecasts using the same software package. Rulex (Muselli (2012)) is natively supported on WIN64 Bits PCs, where it operates in a standalone setup, storing its workflows on a local database.

6.3.3 Machine Learning and IoT systems

As mentioned in the previous section, IoT is being applied to many applications and systems. For example, authors in Al-Khafajiy et al. (2018) propose and edge computing framework for collaboration among nodes with the aim to improve resources management and achieve optimal offloading directed toward healthcare systems. Also, energy consumption on the edge and the used of ML to improve its performance is addressed in Cecilia et al. (2020); Lapegna et al. (2021); Novac et al. (2021), since energy consumption is essential during ML forecasts due to the limited power supplies available for light-weight IoT devices. Furthermore, authors in Novac et al. (2021) apply ML algorithms for an indoor classification applications which uses features collected from radio frequency measurements. Also, ML forecasts are applied on TS in Kanawaday and Sane (2017) to predict failure on a slitting machine by relying on

data collected by IoT sensors. Additionally, ML is used to secure IoT networks in Canedo and Skjellum (2016); Hodo et al. (2016) while improving systems security using NN.

Therefore, due to the vast scope of application of ML on the edge including its possible use in energy, healthcare, security, and resource allocation, the main purpose of this chapter was to deploy a fully featured ML package on the edge to expand its services to the field of IoT.

6.4 Porting Techniques and Tools

The fundamental task of this project is the porting of the Rulex from 64 bit to 32-bit platforms. In the case of Windows, the Visual Studio environment to accomplish this task, however, in the case of Raspbian 32-bits and Ubuntu 64-bits, CMake (Clemencic and Mato (2012); Fober et al. (2018)) was employed to compile the external and internal dependencies and port Rulex.

In Figure 6.1, a tree-based file structure is shown, which presents a set of libraries or dependencies that exist in a porting process. The header files contain functions that are called in Cpp files. These header files may depend on other header files, however, Cpp files cannot call a function unless the corresponding header file is included. The Cpp files generate their output files which produce an overall output binary file. A binary file contains a compiled or encrypted version of the functions found in a header file. Consequently, binary files could depend on other binary files, where in general, the final target is an executable. Rulex on

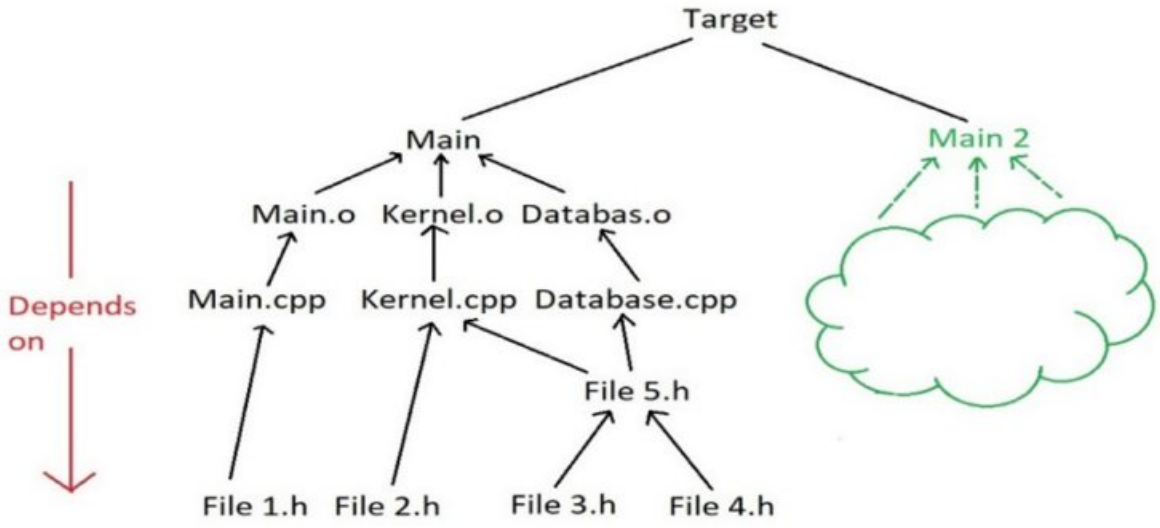

Fig. 6.1 Dependencies file structure (Daher et al. (2020a).

the Raspberry Pi can operate in Standalone mode or Client/Server mode. In the latter case, a WIN32 system is used as a client, where the Rulex GUI is running, and the Raspberry Pi functions as a server or ML engine. A Secure Shell (SSH) connection (Allen (2003)) is used in case the connection is over a public or private network. After connecting to the Rulex Engine on the Raspberry Pi, remote development was used to debug the code so it can operate on both WIN32 as well as Linux 32 Bit/ 64 Bit such that it runs without any manual modification.

In the process of testing, the software runs showed that some of the original C/Cpp variables from the WIN64 version are not compatible with 32 Bit systems. Therefore, it was necessary to make the source redundant concerning this issue. So, the flow of the code was diverted to a path or snippet specific to the running OS, which is implemented using Macros. For example, a Macro such as _WIN32 was used to detect a WIN32 OS and _WIN64 for detecting WIN64. Also, to detect ARM, the __arm__ Macro was executed.

Rulex GUI source code was debugged through the SSH connection where a PC acts as the Client, and the Raspberry Pi forming the ML Server. In the Client/Server setup, a Docker-based PostgreSQL container (Bellavista and Zanni (2017)) is placed as the common storage point between the Client and server nodes.

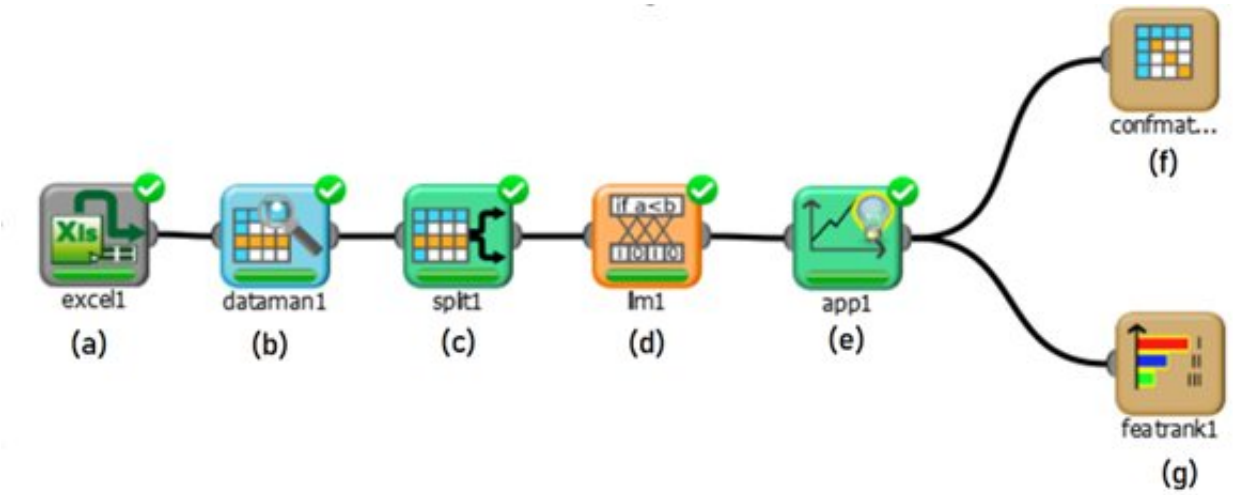

Fig. 6.2 Rulex processing blocks: consisting of (a) excel1 which imports data, (b) dataman1 for viewing and data mining, (c) split1 for splitting dataset, (d) lm1 which performs the machine learning algorithm, (e) app1 to apply the model, and finally (f) confmatrix1 and (g) featrank1 which visually display results (Daher et al. (2020a)).

6.5 Forecast Results

To verify the effectiveness of the proposed solution, after porting Rulex to WIN32 as a Client, and Raspberry Pi as an application server, we tested Rulex on multiple datasets from five diverse applications having a different number of samples and with varying dimensions. Additionally, we've tested the accuracy of the Image-to-TS pre-processing Algorithm on a gender classification dataset.

6.5.1 Radar Classification

The Urban Classification dataset was recorded by a short-range 24 GHz radar, based on the Infineon BGT24MTR11 RF transceiver (Infineonradar (2022)), and particularly, the Distance2Go development kit by Infineon (Infineon (2022)).

For ML forecasts, four classes were considered: One for Humans and three vehicle classes. Namely, Car, Truck, and Motorcycle where the dataset contains 120 records. In Daher et al. (2020b), a multiclass tree-based classification technique was implemented to improve the prediction accuracy using this dataset.

The radar data has been recorded using another separate system dedicated to FE. This standalone system is composed of three parts. Firstly, A 24 GHz radar, a second Raspberry PI 3B+, and a PC running MATLAB. This second Raspberry PI was employed to connect the MATLAB station to the radar board. The Raspberry PI collects the data from the radar, and then it sends it to MATLAB running on a PC where FE is applied (Rizik et al. (2019, 2021)). Below is a list of the features extracted using the radar measurements:

1. R: The spread in the range-FFT spectrum caused by the target.
2. R1: Variance of the range-FFT spectrum.
3. R2: Standard deviation of the range-FFT spectrum.
4. R3: Average of the range-FFT spectrum.
5. V: The spread in the Doppler-FFT spectrum caused by the target movement.
6. V1: Variance of the Doppler-FFT spectrum.
7. V2: Standard deviation of the Doppler-FFT spectrum.
8. V3: Average of the Doppler-FFT spectrum.

9. RCS: Radar Cross-Section, that gives a measure for the reflectivity of the target.

10. Vest: The estimated speed of the target.

After extracting features using the second system, the first Client/Server setup which consists of Rulex running on a Raspberry PI is used for ML predictions. An example of a workflow in Rulex GUI running on the Raspberry is presented in Figure 6.2, where there are data processing blocks followed by an LLM, and finally a confusion matrix. Moreover, a block that splits data for training and testing is shown, where the ratio is 65% for training and 35% for testing.

Figures 6.3 and 6.4 show the training and testing accuracies using LLM. In Figure 6.3, Cars and Humans are classified with a rate of 100%. As for Motorcycles and Trucks, they are 84.2% and 89.5% respectively. Figure 6.4 shows the testing accuracies where Cars are detected with a rate of 73.3%, Humans at 100%, Trucks at 72.7%, and Motorcycles were recognized with a rate of 90.9%.

6.5.2 Human Activity Detection using Smartphones

The Activity dataset from Anguita et al. (2013) consists of features that have been recorded using the Accelerometer and Gyroscope that are included in a Smartphone. The dataset is pre-processed such that it can be directly applied in Rulex since the features are composed of

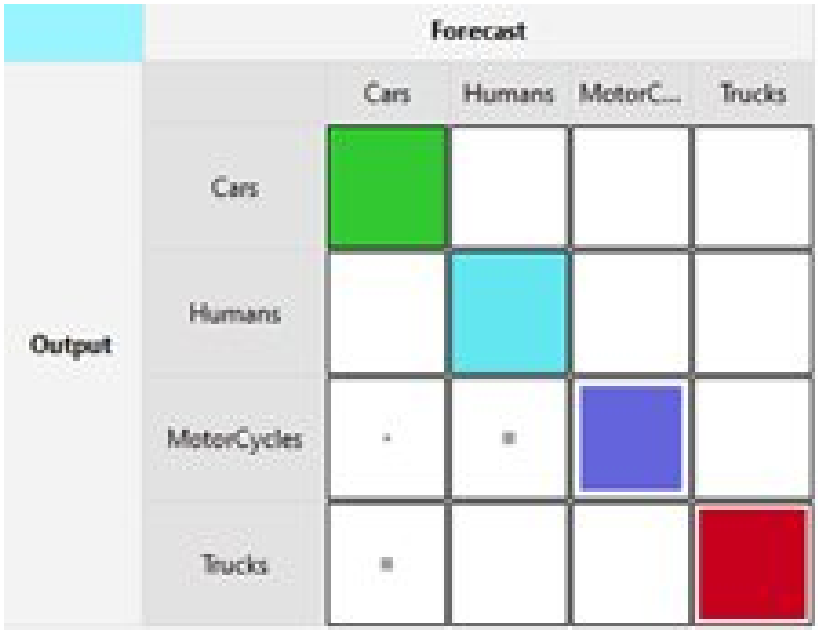

Fig. 6.3 Radar training forecast (Daher et al. (2020a)).

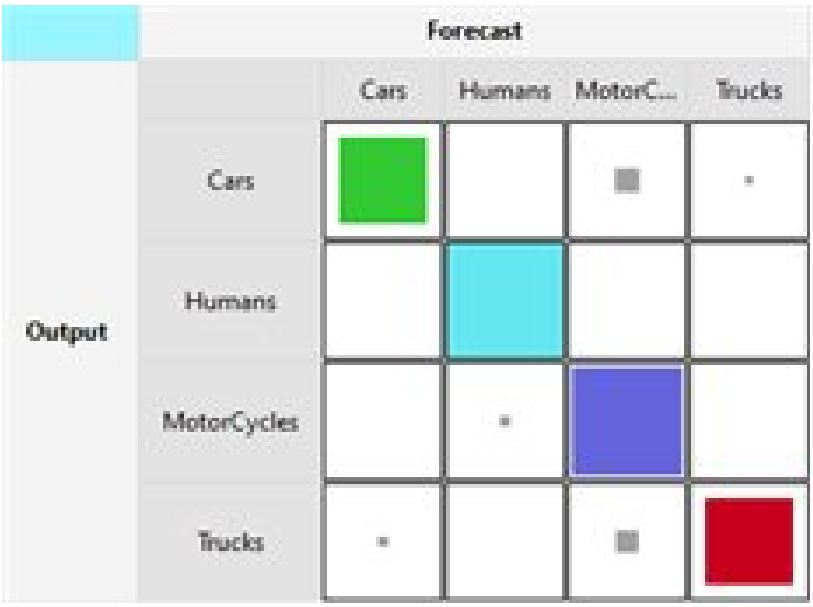

Fig. 6.4 Radar testing forecast (Daher et al. (2020a)).

basic statistical operations based on the measurements. These consist of the mean, Standard Deviation (STD), variance, and others. The activity performed by a subject is divided into six classes: Laying, Standing, Sitting, Walking, Walking Upstairs, and Walking Downstairs.

A subset of the original dataset was used which is reduced to 7530 samples and 560 features, excluding the subject field which identifies the person or test subject specifically. When this field is included, the accuracy is increased considerably, however, it does not consider that the Smartphone can be carried by different people. This has been done to test the robustness of the ML forecasts and investigate the effectiveness of the present experiment. Furthermore, the dataset labels are distributed equally as shown in the histogram from Figure 6.5.

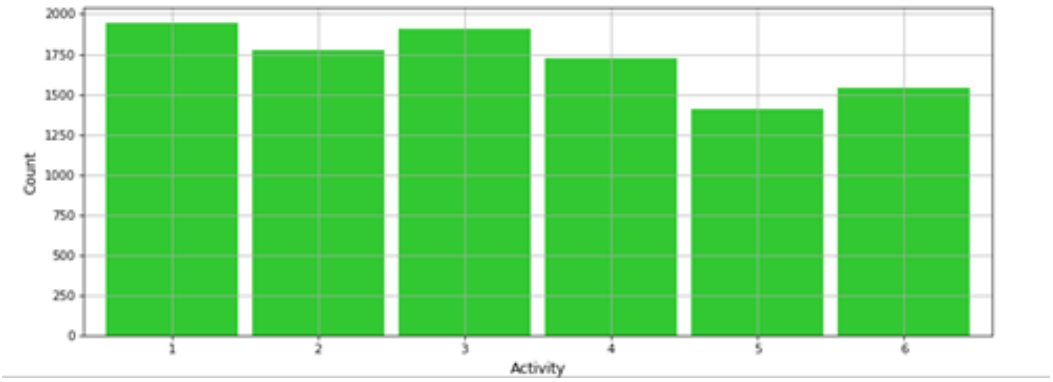

Fig. 6.5 Low class skewness for the Human Activity Detection dataset.

The ML forecasts for the Smartphone Activity Detection dataset are provided in Table 6.1, wherein the Rulex Software, three basic ML Algorithms are implemented: LLM, KNN, and SVM. All tests have been applied in a Client/Server arrangement with the Rulex Engine running on the Raspberry Pi. As shown in Table 6.1, high testing accuracy was reached in every forecast, where most notably, in the case where SVM is applied, a near-perfect accuracy is achieved.

Table 6.1 Forecast accuracy for the Smartphone Activity Detection application.

Human Activity Detection			
Algorithm	LLM	KNN	SVM
Laying	100%	100%	100%
Standing	86.8%	94.4%	98.4%
Sitting	88.48%	91.86%	97.63%
Walking	94.74%	100%	100%
Walking Upstairs	89.89%	99.47%	100%
Walking Downstairs	100%	100%	100%

6.5.3 Brainwave Mental State Classification

In Anguita et al. (2013), EEG recordings were used to predict the mental state of a subject through ML techniques. In this chapter, the features from the concerned dataset are used in multiple forecasts to classify the mental into three given classes: Relaxed, Neural, and Concentrating states. Multiple Algorithms have been implemented consisting of the three basic ML Algorithms: SVM, NN and KNN, where SVM achieved the best performance. Figure 6.6 presents a plot of one frequency-based feature value vs. class labels, where this feature exhibits a lower value for one of the labels.

Additionally, Figure 6.7 presents the same plot for another mean-based feature from the dataset, where results show that this feature has varying levels for each of the three classes. This illustrates the influence on every feature on each of the labels.

In this chapter, we applied forecasts using Rulex running on the Raspberry Pi and in a Client/Server setup. Regarding the dataset, a total of 2480 samples and 988 features were employed for both training and testing in a 70/30 split. The accuracies for each forecast are presented in Table 6.2.

As shown, LLM, KNN, SVM have high testing accuracies thus demonstrating the robustness of the experiment, where the most accurate forecasts were achieved using SVM with an overall accuracy of 95.47%.

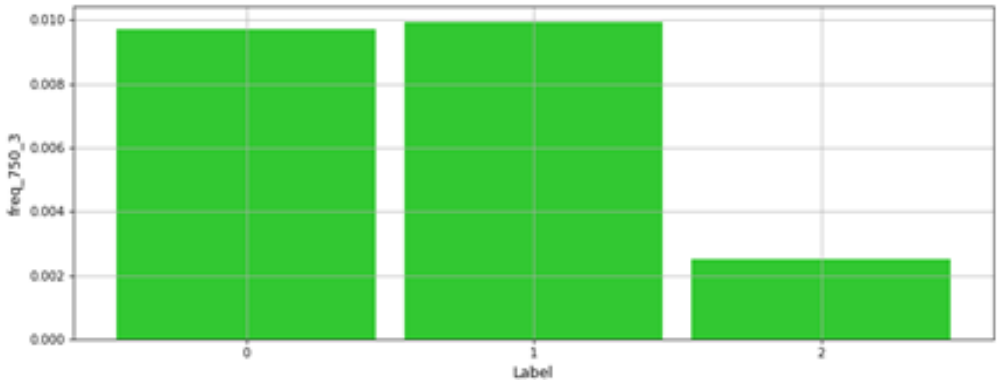

Fig. 6.6 Distribution of frequency-based feature values vs. class labels.

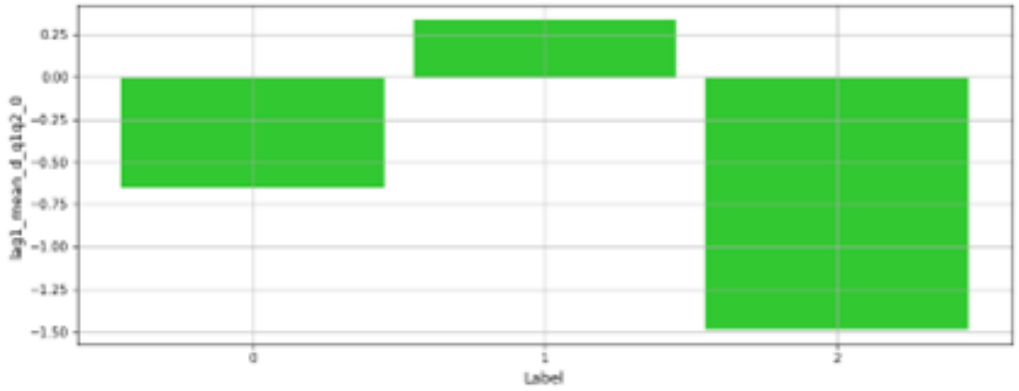

Fig. 6.7 Distribution of mean-based feature value vs. class labels.

Table 6.2 Mental State Classification with Rulex.

Human Activity Detection			
Algorithm	LLM	KNN	SVM
Relaxed	92.77%	94.78%	96.39%
Neutral	76.02%	83.74%	89.84%
Concentrating	97.19%	100%	99.60%

Table 6.3 Power consumption for Mental State Classification for HP laptop.

Mental State Classification with Rulex Power consumption		
Station	LLM	SVM
Time taken	0.022 sec./Task	0.018 sec./Task
Energy (4 Watts)	1.5 J./Task	8.37 J./Task

Table 6.4 Power consumption for Mental State Classification for Raspberry Pi.

Mental State Classification with Rulex Power consumption		
Station	LLM	SVM
Time taken	0.37 sec./Task	0.08 sec./Task
Energy (4 Watts)	1.08 J./Task	0.48 J./Task

In addition to applying forecasts on the edge for this dataset, we've studied the power consumption of the Raspberry Pi board by recording the elapsed time and estimating the power consumption, while comparing it with the energy consumed by an HP Laptop. The results can be viewed in Tables 6.3 and 6.4, whereas shown for LLM and SVM, the Raspberry Pi achieves more time but less energy to perform the same task.

6.5.4 Vehicle Activity Recognition

A dataset for Activity Recognition of Dumpers in earth-moving sites is presented in Axelsson and Wass (2019). It utilizes data taken from Smartphone sensors such as Gyroscopes and Accelerometers to record signals (while the Dumper is working) for FE. An illustration of the data collection phase is presented in Figure 6.8, with the Smartphone installed inside the Dumper for taking measurements.

The pre-processed dataset was used to classify the state of the vehicle, whether a Dumper is in one of six states, namely: idle, driving, loading, dumping, engine-off, and unknown. Figure 6.9 illustrates the dispersal of the labels in the dataset where there is clear skewness in the class distribution. A subset of around 216,000 samples having just 8 features were used as a data source to apply ML training and testing with a 70/30 split. Forecasts were applied through Rulex running on the Raspberry Pi in a Client/Server arrangement using the KNN Algorithm. The testing accuracy in confusion matrix format is presented in Figure 6.10, where the overall forecast accuracy using KNN on the Edge is 95.27%. Furthermore, with regards to power consumption, as employed in the previous dataset, the time elapsed to perform an ML task is recorded to estimate the consumed energy. In the case of the HP

Fig. 6.8 Taking sensor measurements using Smartphones to classify the state of a Dumper in an earth-moving site (Axelsson and Wass (2019)).

laptop which consumes around 70 W, as shown in Table 6.3, the time taken for KNN is 11.82 min and the energy is 49.63 KJ. As for the Raspberry Pi, where the power is 4 W, the time taken is 142.4 min and the energy consumed is 34.18 KJ.

6.5.5 Gender Classification

Gender classification can be useful for performing studies if implemented in an automated manner. In Karimi et al. (2016), gender detection using names, countries, and facial images is discussed where authors found that the accuracy is strongly dependent on the country, meaning that ethnicity can play a role in prediction accuracy.

Classification of Gender that is based on forward-facing images is reported in Yang et al. (2006), where a minimum error rate of 2.85% is achieved. A Classifier implemented using SVM that detects gender with a minimum error of 3.4% is presented by Moghaddam and Yang (2000), where mug-shot images are used for training and testing.

In practice, when implementing gender classification, facial images can be misaligned in contrast to some forward-facing datasets that are seldom used in the literature. Therefore, authors in Eidinger et al. (2014) apply the dropout technique and SVM to tackle this issue and classify age and gender under “in the wild ” conditions.

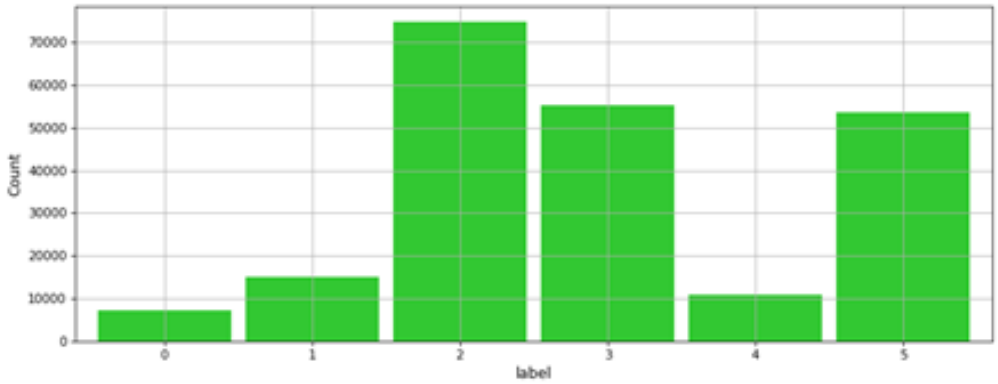

Fig. 6.9 High class skewness for the Vehicle Activity Recognition dataset.

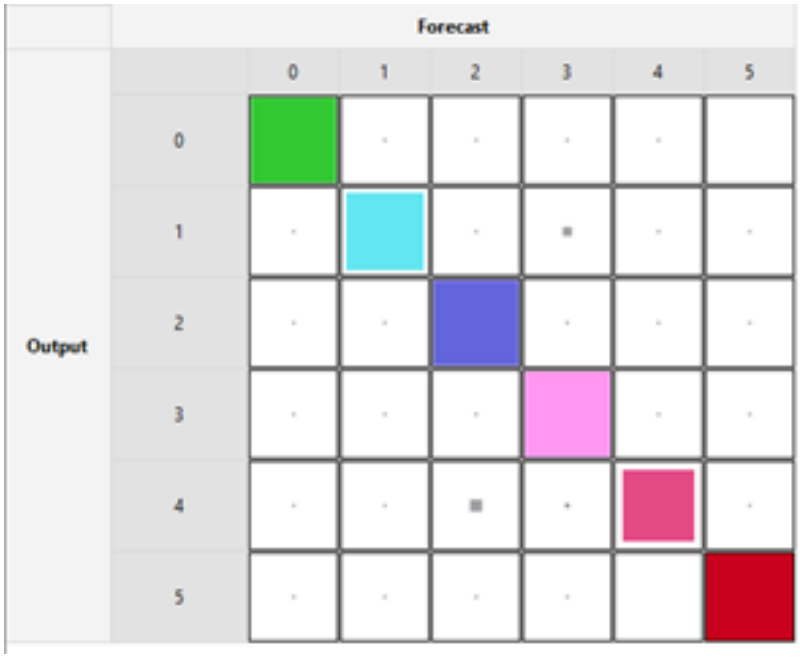

Fig. 6.10 Testing Accuracy for Vehicle Activity Recognition using KNN.

In this chapter, we've implemented a novel Algorithm that converts facial images from various ethnicities, poses, zoom levels, and ages into Time-series that are generated using a radial scanning-based approach Izbicki (2011). Moreover, a Sobel filter Bora (2017) is used to detect the edges of an image and with the 360° radial scan, the distance from the center (which is also determined in Algorithm 5.1) till every visibly detected pixel is calculated. statistical functions of this distance for every degree in the scan are computed, before generating a Time-series for every function. Finally, the TS are grouped to form a dataset that is used in an ML forecast.

Image-to-Time Series using Statistical Radial Scanning and Sobel Filters

Image Classification usually requires specialized ML Algorithms such as CNN which are used for applications like object detection and face recognition Levi and Hassner (2015). Most applications using CNNs are dependent on the color of an image and its distribution.

Algorithm 5.1 *Extracting center points from sobel filter output*

Require: *image dataset*
Ensure: *arrays sx, sy, centerx,centery*

```
for i in all images do
    sobel = sobel(current image)
    sx[i], sy[i] = shape of sobel
    for positions of all horizontal points: do
        sumx = sum of points where (pixel > threshold)l
        if sumx > 0 then
            meanx = meanx + position
            countx++
        end if
    end for
    centerx[i] = meanx/countx
    for positions of all vertical points: do
        sumy = sum of points where (pixel > threshold)
        if sumy > 0 then
            meany = meany + position
            county++
        end if
    end for
    centery[i] = meany/county
end for
Return sx, sy, centerx, centery
```

Although, some applications may not be so much reliant on color but rather the shape or contour of an image.

With a different approach, we developed a novel Algorithm aimed to perform FE for these types of applications. Specifically, it has been designed to reduce the execution time, requiring less processing power compared to CNN which demands sufficient resources Albawi et al. (2017)Furthermore, using this approach it is possible to apply the created Time-series to any ML Algorithm thus increasing the number of possible setups.

Firstly, as shown in the pseudo-code reported in Algorithm 1, a Sobel filter extracts the edges of each input image before determining the center points of Sobel output. Initially, for every horizontal level, the sum of pixels having brightness greater than a predefined threshold

Algorithm 5.2 *Statistical radial scanning of images*

Require: *image dataset and arrays sx, sy, centerx, centery*
Ensure: *pre-processed dataset*

for *i in all images* **do**
  for *every angle in Sobel(0 - 90^o)* **do**
    if *2nd iteration in loop or more* **then**
      Rotate by 90^o
    end if
    if $y \lesssim 45^o$ **then**
      ratio = y/45
    end if
    if $y \geq 45^o$ **then**
      ratio = 45 / (90.01 - y)
    end if
    for k = *centery[i];* $k \lesssim$ *sy[i];* $k = k + 1$ **do**
      for j = *centerx[i];* $j \lesssim$ *sx[i];* $j = j + 1$ **do**
        if $k \geq$ *(j * ratio) * 0.9* **and** $k \lesssim$ *(j * ratio) * 1.1* **then**
          if *sobel (j, k)* $\geq$ *threshold* **then**
            $D \Leftarrow \sqrt{(j - centerx[i])^2} + \sqrt{(j - centery[j])^2}$
          end if
        end if
      end for
    end for
    Add mean, median, STD, variance, maximum, and minimum
    of D and Ang to a Times-series
  end for
end for
task: *Interpolate all twelve Time-series such that every unit (record) is expanded till the max length before grouping them into a dataset to apply ML forecasts.*

is computed. Then, for each horizontal level, the mean (meanx) of the positions where the threshold was exceeded is computed. The value meanx corresponds to centrex, which is the center of filtered image. The same steps are followed to extract centery which is the vertical center of the same image.

In summary, Algorithm 5.1 calculates the vertical and horizontal means for every level to determine the image center's coordinates. Subsequently, As shown in Algorithm 5.2, radial scanning is performed on the Sobel filter's output. In Algorithm 5.2, the outer loop is used to scan the entire 360° around the center with coordinates (centerx, centery), and in case a pixel is brighter than a Threshold, the distance from the center is calculated, where statistical functions (mean, median, STD, variance, maximum, and minimum) are computed for the distances of every angle. Therefore, multiple corresponding Time-series are generated, however, with different lengths for each record. Finally, these TS are interpolated (To make each set of Time-series equal, and to have the same number of features for every record) and grouped in one file to form a dataset applicable for ML predictions. The pseudo-code from Algorithms 5.1 and 5.2 has been implemented in the Python programming language to perform the Image-to-TS pre-processing before performing ML forecasts on the Edge using Rulex.

Gender Classification Experimental Results

To test the novel image pre-processing algorithm described in the previous section, we considered a gender image classification dataset taken from [41]. It consists of two classes: Male and Female. The images are random where no specific angle is adopted as a reference, but rather the images of both classes have been taken from various inconsistent conditions in terms of angle, dimensions, zoom level, and background (cfr. Figure 6.11).

Also, some of the photos contain part of the body of the subject. Furthermore, the people in the photos have very wide ranges of age, ranging from youthful to elderly, as well as belonging to different ethnicities while taking different poses. This is demonstrated in Figure 6.11, where an example of six random female subjects is shown.

The dataset consists of 2700 female photos and 2720 male photos. These photos were pre-processed using the novel algorithm developed in this chapter to generate the corresponding dataset that consists of sets of Time-series.

The generated dataset has been used for ML forecasts using Rulex on the Raspberry Pi. KNN and SVM were used to perform training and testing with an 80/20 split, where the results for the forecasts performed on the Edge are presented in Table 6.5.

Fig. 6.11 Six random images of women with varying ages, ethnicities, image background, dimensions, poses, and zoom level taken from the dataset found in Izbicki (2011).

In Yang et al. (2006), the images used for gender classification were composed of people belonging to a single ethnicity using that are all forward-facing to the camera. A minimum error rate of 2.84% (accuracy of 97.16%) was achieved using SVM. Another case where SVM is used is reported in Moghaddam and Yang (2000), where low-resolution forward-facing images are used to reach an error rate of 3.4% (96.% accuracy). However, mug-shots or forward-facing photos may not be available in real-time practical situations, therefore, authors in Eidinger et al. (2014) address this issue and attempt to determine the age and gender of random inconsistent images where the highest gender classification accuracy is 88.4%, (which is significantly lower than the publications where forward-facing photos were used) taken as the best from numerous forecasts.

As shown in Table 6.5, the Image-to-TS conversion algorithm is capable of pre-processing random images (Where color is not an issue) and can be applied to various ML Algorithms to achieve high gender detection accuracy. Consequently, it can compete with previously published techniques that rely on impractical (consistent) mug-shot images for classification.

Table 6.5 Forecast accuracy for Gender Image Classification.

Gender Detection with Rulex		
Station	KNN	SVM
Female	98.72%	95.60%
Male	94.24%	88.29%

6.6 Chapter Conclusion

In this thesis, we refer to the porting of Rulex, a ML software that natively runs on WIN64, on the Raspberry Pi, for Edge Computing applications. We also reported the results obtained in different application domains, namely with five unrelated datasets, which we used to test the performance of our implementation in real IoT environments.

The main result obtained is that Rulex now operates in a Client/Server setup with the interface being operated on a PC as a WIN32 application, with the ML algorithms being run on the Raspberry Pi ARM 32-Bit microcontroller. The overall accuracy for the urban dataset was 84.2% and 99.3% for the Activity Detection dataset. The Brainwave dataset resulted with an accuracy of 95.47% using Rulex, and the Vehical Activity recognition dataset was forecast with an overall accuracy of 95.27% using KNN. Finally, inc case of the gender dataset where an Image-to-TS conversion was performed, where this approach led an accuracy of 97%.

As regards the performance obtainable through this implementation, first we considered a dataset related to the classification of pedestrians and vehicles using a high-frequency radar: An ML workflow was implemented on our platform and good results were achieved. However, as such dataset is relatively small, we considered four additional datasets from diverse application fields. Three pre-processed datasets having a larger number of samples and with low and high dimensionality and varying skewness were adopted. The pre-processed datasets include a Human activity Recognition dataset using Smartphones, a vehicle Activity Detection dataset, and an EEG Classification dataset related to mental state.

Furthermore, a novel pre-processing Algorithm was developed and implemented in Python, that converts images into TS to pre-process a gender detection dataset that contains inconsistent facial images in terms of the background and dimensions of the image itself and the age, ethnicity, and pose of the subject. Also, the accuracy achieved for gender classification is considerably competitive with the literature, where the competition is dominated by mug-shot and forward-facing images. Moreover, in every forecast, the training and testing were performed using Rulex on the Edge, where in general, each experiment achieved high

classification accuracy through the Client/Server interface. Also, the power consumption was investigated by comparing the Raspberry Pi as an edge computing node with an HP laptop, where for the same ML algorithm and dataset, the Raspberry Pi consumes less energy.

Chapter 7

CACAO-X: Contour Assisted Convolutional Neural Networks with Global Image Explainability and Edge Computing Classification

7.1 Chapter Abstract

Image classification and interpretability is a basic procedure in modern AI systems as well as being employed in the fields of security, instrumentation, and image classification. XAI also plays a crucial role in providing ML techniques with a white-box output that allows the practitioner to follow or interpret the rules. Therefore, we present CACAO-X which is a framework for ML forecasts that utilizes an algorithm which converts images into TS and employs CNN, LSTM, and MLP on the original images and converted data in a mixed input approach achieving accurate forecasts on multiple applications. Furthermore, Shapley values are employed on the proposed ML models to extract the local explainable features of such images. These features are again used with the same conversion algorithm to generate time-series and application specific landmarks using LLM to determine the explainable rules. In addition to being generated automatically, explicative labeling is adopted in the dataset which gives insight to the operator as to which landmarks, positions and angles contribute to the classification of each record. The system is robust as it achieves benchmark results in the gender classification application while remaining competitive in all others where it is used as a reference model for global image explainability. Additionally, the ML inference for every application is performed according to the edge-computing paradigm, by deploying the models on a limited resources Raspberry Pi platform through SSH access. Moreover, a novel

callback function has been developed which is able to stop the training of the CACAO-NET model at a stable state based on the level of oscillation around the most recent mean accuracy.

7.2 Introduction

Image classification is used in various applications ranging from security, clinical decision support systems, and object recognition where AI techniques aid to generalize and provide accurate performance in the identification process. Moreover, to perform training and inference many options are available. Usually, a practitioner will use one of many AI algorithms to achieve his goals where the most used by far is NN and more specifically CNN (Li et al. (2021)). CNN aims to extract features automatically and has a wide range of operations wherein the literature CNN proved to achieve benchmark results in multiple classification applications where it does not require any application specific or hand-crafted features. Additionally, due to the wide range of successful existing architectures, some practitioners may choose to implement transfer learning, which consists of using a model that has been trained on an unrelated dataset. Data science specialists rely on these architectures by using a pre-trained model and fine-tuning the layers near the output by swapping the final fully-connected layers and freezing the remaining layers (Weiss et al. (2016)). However, the reported architectures (Dhillon and Verma (2020)) are usually overly complex and are much larger then conventional NN and transfer learning is application-sensitive, in a sense that it is related to the domains of the pre-trained dataset and not the target dataset. Therefore, if the pre-trained model does not perform well on another dataset, the practitioner may choose to implement another approach. Consequently, in this perspective, to overcome these problems we propose a Contour Assisted CNN with Image explainability, or expressed as CACAO-X, which uses a fixed structure applicable in diverse application and with minor tuning. This is obtained through the combination of the automated feature generation using CNN and the extracted TS which provide general shape information that assists to improve the overall performance.

This thesis introduces a mixed Data NN architecture which can be used in general-purpose image classification. The model is implemented so that it is robust and can achieve accurate ML classification in a practical workflow. The main input is a set of Red-Green-Blue (RGB) or grayscale images which are used to derive a set of TS based on the Sobel output of each image. In order to demonstrate the generality of this framework, training and testing were done on a PC using six datasets and the same NN architecture or by applying inference in an edge computing setup on the Raspberry Pi (Vujović and Maksimović (2014)).

Furthermore, the local explainable features are extracted from the two inputs—the image and its TS representation—after the model has been trained. This is done to give insight to which pixels contribute to each classification. Also, the local explainers are used alongside application specific landmark measurements in an automatically generated and annotated dataset to be applied using an XAI algorithm (Gunning et al. (2019)) to present global explainability of image-based classification models.

The rest of this project is organized as follows: Sections 2 conducts the literature review regarding various ML techniques and works which utilize similar datasets; section 3 presents the fundamental blocks and the CACAO-X framework including the global explainability part as well as an image classifier we named CACAO-Net which is used for general-purpose image classification. Section 4 describes the six diverse datasets used in the project which are different enough to prove the general nature of the provided approach. Also, section 5 provides and methodology used in the implementation and section 6 presents the experimental results and comparison, and finally, section 7 concludes the paper.

7.3 Chapter Related Work

In this section, we will present works from the literature which implement AI models for the same application domains provided in this chapter. Most works either implement transfer learning or derive application specific features which are limited to that application. Therefore, a motivation for the purpose of CACAO-X is outlined at the end of this section to clarify the advantages and generality of the framework.

7.3.1 Transfer Learning

In Deepak and Ameer (2019), authors implement transfer learning using GoogLeNet on a Three-Class brain tumor dataset while using a k-fold setup with five folders. This setup includes the addition of a SVM at the output of the GoogLeNet which was included in the training. This setup achieves benchmark accuracy using this approach; however, it does not report the accuracy using a greater number of classes.

Skin Cancer classification using CNN is discussed in Brinker et al. (2018) where transfer learning is compared with a from-scratch setup. They conclude that CNN achieves the best performance when it is pre-trained by means of another larger dataset before optimizing the parameters of the final layers which are tuned to the target application. Authors in Kadhim and Abed (2019) implement multiple transfer learning setups using AlexNet (Lu et al. (2019)),

VGG19 (Carvalho et al. (2017)), GoogLeNet (Tang et al. (2017)) and Resnet50 (Mukti and Biswas (2019)) for satellite image classification, where they conclude that Resnet50 achieves the best performance on the three different datasets. Even so, the CACAO-Net model presented in this project outperforms transfer learning approaches in multiple cases and remains competitive in others, as will be presented in section 6.

7.3.2 Application-specific features in ML

In Lu et al. (2014) binary weather classification is performed using application specific as well as label-specific feature which not only limit the model to that application but also to each class. The features extracted from this image dataset include sky and shadow detection for the Sunny Cloudy classes. Also in Roser and Moosmann (2008), histogram-based features are extracted from a three-class weather dataset for driver assistance. The vector which is used as input for an SVM classifier is composed of features related to brightness, contrast, sharpness, saturation...etc. Although these features are not application specific, however, they may not be applied in most applications wherein these features are irrelevant and the overall ambience of the image does not contribute to correct classification such as the case of object detection in contrast to weather classification datasets and others.

Authors in Zhang et al. (2016) use multiple kernel learning algorithm for multi-class weather classification using category-specific dictionary learning. The features used in this work include sky, shadow, rain, streak, snowflake, dark channel, contrast, and saturation. For example, the sky is detected by cropping out the remaining landscape and the shadow is detected using a shadow detection tool first implemented in Sajjad et al. (2019).

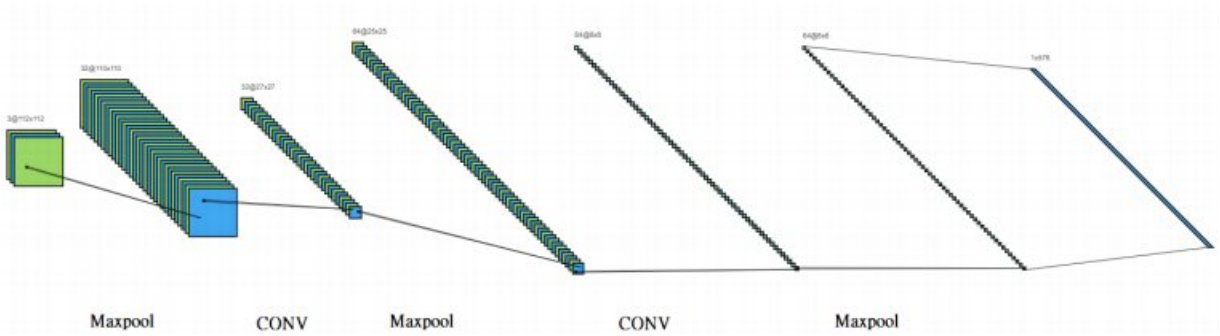

Fig. 7.1 The CNN architecture used in CACAO-NET Model1 1 and 2.

7.3.3 Conventional ML

In Sajjad et al. (2019), CNN is used for multi-grade brain tumor classification where the authors implement data augmentation to train the model. They provide experimental results for both original and augmented data where the data augmentation on the training data improved the inference. In Das et al. (2019) CNN is again used for brain tumor classification using a three-class dataset after pre-processing the images. Authors in Abiwinanda et al. (2019), implement a CNN which does not rely on segmentation but rather a direct approach, whereas reported, competes with complex region-based segmentation methods using the same data source. Although the three preceding models (Das et al. (2019); Sajjad et al. (2019); Zhang et al. (2016)) perform well, they remain application specific and therefore cannot translate to other applications without any heavy structural modifications of the CNN model.

In Hekler et al. (2019), a hybrid multiclass skin cancer detection method is presented that combines Resnet50 with human intelligence to achieve superior classification results. In this work, data was down-sampled and augmented appropriately to avoid any skewness in the class labels. The model achieves accurate results for both binary and multiclass cases, however as shown in section 6, it is possible to achieve better results without human intelligence while using a fairly general model.

7.3.4 Explainable AI for image classification

In Stieler et al. (2021), a domain-specific XAI approach is used to explain the classification of skin cancer images using the LIME tool, and the ABCD-rule which is an approach used by dermatologists for systematic diagnosis of skin cancer lesions. The work consists of drawing a final scatter plot as a visual aid for local explainability. Authors in Vermeire et al. (2022) present a counterfactual XAI approach where the segment of each instance is cropped out in order to infer to which part of an image contributed to its classification or misclassification to a particular class.

In Pintelas et al. (2020) the authors presents an XAI framework for global image explainability relying on CNN models as the reference for the rule extraction. They chose brain tumor data as a case study for their experiments where they manage to produce questions and answers that summarize the conditions that are followed by CNN for classification. They provide both specific and humanistic sentences to state the portability of the framework. Regarding global explainability, the addressed work only includes the tumor's size, sum and correlation scores and does not provide any position information regarding the complete

analysis. Also, this approach may only be applicable to abstract images such as tumors or patterns and cannot be translated to objects or subjects in conceptual domains.

7.3.5 CNN Basics

CNNs are usually used in image classification and automated FE where the network is composed of convolution and max-pooling layers. The convolutional layer convolves its input with a filter before down-sampling through a max-pooling layer to reach better generalization and faster convergence. This process is repeated to reduce the size of one or more dimensions and increase others knowing that the final CNN output is applied to a fully-connected layer which is coupled to a standard NN.

An illustration of the used CNN as applied is shown in Figure 7.1 where the actual dimensions are provided. Also, the convolution details can be viewed in (1) where I is the input, K is a filter, and $m\ x\ n$ represents the filter's dimensions.

$$F(i,j) = (I \circledast K)(i,j) = \sum\sum(i+m,j+n)k(m,n) \tag{7.1}$$

Each output from each of the stacked convolutional layers is applied to a *Relu* activation function which is shown in (2):

$$Relu_{i,j} = max(0,F(i,j)) \tag{7.2}$$

The equation for each max-pooling layer succeeding a *Relu* function can be found below:

$$y_{i,j} = max_{i,j=1}^{h,w} Relu_{i,j} \tag{7.3}$$

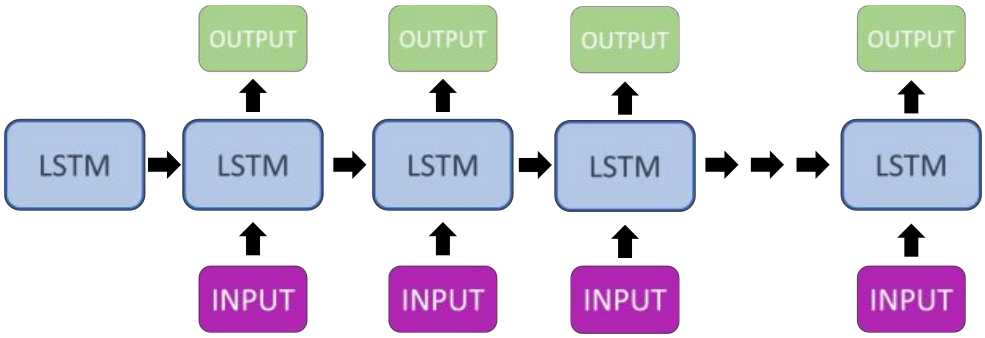

Fig. 7.2 Basic LSTM sequence as used in CACAO-NET model 2.

7.3.6 Long Short-Term Memory Networks

LSTM networks are used to build Recurrent Neural Networks (RNN) to deal with sequences or TS and are used in this present work alongside CNN models to generalize the architecture and to give two flavors, namely, Model 1 and Model 2. An abstract depiction of an RNN is displayed in Fig. 2, where each blue block can be decomposed as an LSTM block as shown in Fig. 3.

An LSTM cell takes a current input x_t and the precious information a_{t-1}. Every cell has a state c_t from (5), which behaves as a memory that retains past information. $c^{\sim t}$ from (4) is a candidate generated by the LSTM cell that may replace c_t, where W_a^c and W_x^c are the weights used to generate the candidate cell. c_t can be computed using a_{t-1} and c_{t-1} as shown in (6):

$$c^{\sim t} = tanh(W_a^c a^{t-1} + W_x^c x_t) \tag{7.4}$$

$$c^t = f^t \otimes c^{t-1} + u^t \otimes c^{\sim t} \tag{7.5}$$

$$a^t = o^t \otimes tanh(c^t) \tag{7.6}$$

Furthermore, the forget, update, and output gates of each cell may be generated using equations(7-9).

$$f^t = \sigma(W_a^f a^{t-1} + W_x^f x_t) \tag{7.7}$$

$$u^t = \sigma(W_a^u a^{t-1} + W_x^u x_t) \tag{7.8}$$

$$o^t = \sigma(W_a^o a^{t-1} + W_x^o x_t) \tag{7.9}$$

7.3.7 Motivation for CACAO-X

ML models are usually used for domain-specific tasks where they are tuned to achieve the highest accuracy possible. ML practitioners are resorting to transfer learning to meet their goals in terms of accuracy while saving design time. Although, this approach does not usually achieve the desired performance since the pre-trained model (even after tuning) may not fit to the target application. Furthermore, in terms of image explainability, most works implement local interpretability while not providing general human understandable statements. Regarding global explainability, the statements are mostly abstract and do not provide detailed information, and yet, there still is a lack of a unified model which if mildly tuned can have enough generality and perform competitively in most image classification scenarios while providing an explainer for its predictions. As a whole, the model proposed in

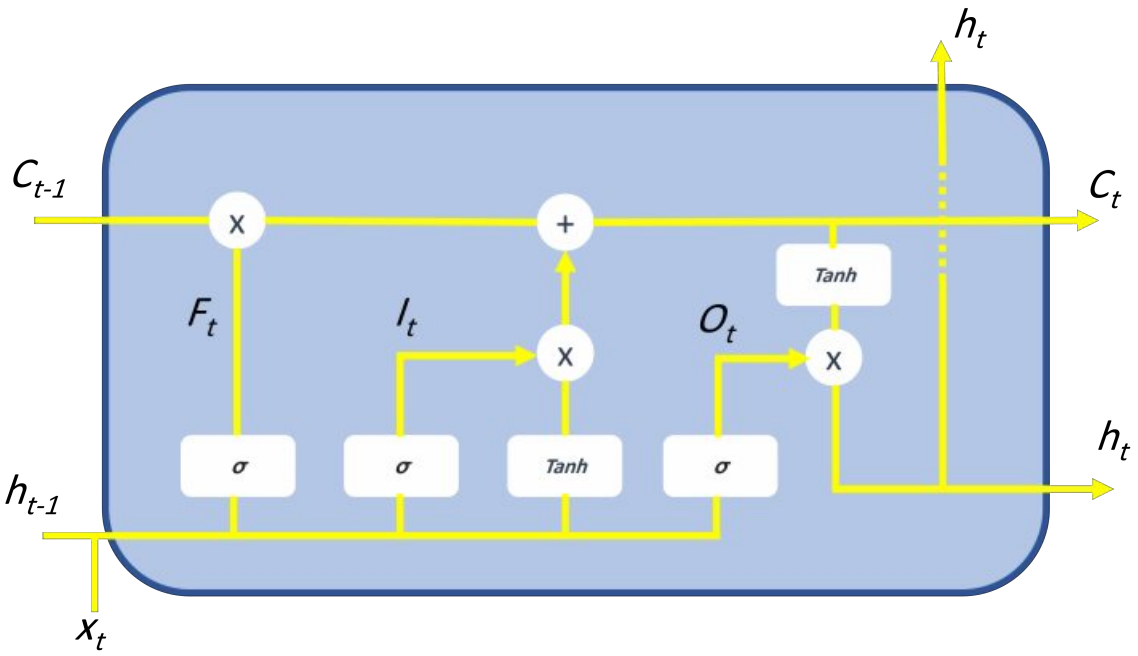

Fig. 7.3 Basic LSTM block used in CACAO-NET model 2.

this project is able achieve such generality and performance while the collective framework introduces the concept of global image explainability (Using the CACAO-Net model). To the best of our knowledge this is the first semi-domain-specific XAI model with human understandable rules that can perform these tasks, where explicative labeling is introduced to interpret the global conditions behind image classification. Although this approach is not completely domain-independent and mainly due to the landmark features, it still remains an impressive step towards true global image explainability with understandable features.

7.4 CACAO-X Framework

In the ML forecasts performed using CACAO-Net, the first set consists of the raw RGB images which correspond to the class labels. The second type of input consists of sets of TS which are the pre-processed versions of the same images. These contain the statistical contour or edge information of the 3D data in 1D format.

Algorithm (7.1) which was first introduced by Daher et al. (2021) is used to achieve this task. The algorithm is split into three main parts which is displayed in Figure 7.4, whereas shown, each image is passed through a Sobel filter to generate a pencil drawing of the same image before applying Algorithm (1) which is implemented using lower level C C-Code

that accessed by Python using the Ctypes module (Ctypes (2023)) for optimal performance, the TS generated in C-Code are applied during the inference per image on a Raspberry Pi to accurately assess the performance, deployment and portability of the CACAO-Net classification model. These tasks may be summarized as follows:

1. The first task is to apply a Sobel filter (Edress et al. (2021)) on each image and to extract the centroid having coordinates (centerx and centery). After applying the filter, supposing the filtered image has a dimension of sx*sx. The employed centerx variable may be computed as shown in the following expressions:

$$\begin{aligned} &For\ x \leq sx:\ [\ Hitx = \Sigma_{x=1}^{sx}\ Mask(Sobel[x,:] \gtrsim \gamma)\ ] \\ &\quad if\ Hitx \gtrsim 0 \rightarrow sumx\ +=\ x,\ countx\ +=\ 1 \\ &centerx = sumx/countx \end{aligned} \tag{7.10}$$

Algorithm 7.1 *Statistical radial scanning of images*

Require: *image dataset and arrays sx, sy, centerx, centery*
Ensure: *pre-processed dataset*
for *i in all images* **do**
  for *every angle in Sobel(0 - 90^o)* **do**
    if *2nd iteration in loop or more* **then**
      Rotate by 90^o
    end if
    if *y $\lesssim$ 45^o* **then**
      ratio = y/45
    end if
    if *y $\geq$ 45^o* **then**
      ratio = 45 / (90.01 - y)
    end if
    for *k = centery[i]; k $\lesssim$ sy[i]; k = k + 1* **do**
      for *j = centerx[i]; j $\lesssim$ sx[i]; j = j + 1* **do**
        if *k $\geq$ (j * ratio) * 0.9* **and** *k $\leq$ (j * ratio) * 1.1* **then**
          if *sobel (j, k) $\geq$ threshold* **then**
            D $\Leftarrow$ $\sqrt{(j - centerx[i])^2} + \sqrt{(j - centery[j])^2}$
            lineA = ((centerx, centery), (centerx, sx))
            lineB = ((centerx, centery), (j, k))
            L1 = lineA[1,1] - lineA[0,1]
            L2 = lineA[1,0] - lineA[0,0]
            slope1=L1/L2
            L3 = lineB [1,1] - lineB [0,1]
            L4 = lineB [1,0] - lineB [0,0]
            slope2 = L3/L4
            angle = **atan***((slope2-slope1)/(1+(slope2*slope1)))*
            Append angle to Ang
          end if
        end if
      end for
    end for
    Add mean, median, STD, variance, maximum, and minimum
    of D and Ang to a Times-series
  end for
end for
task: *Interpolate all twelve Time-series such that every unit (record) is expanded till the max length before grouping them into a dataset to apply ML forecasts.*

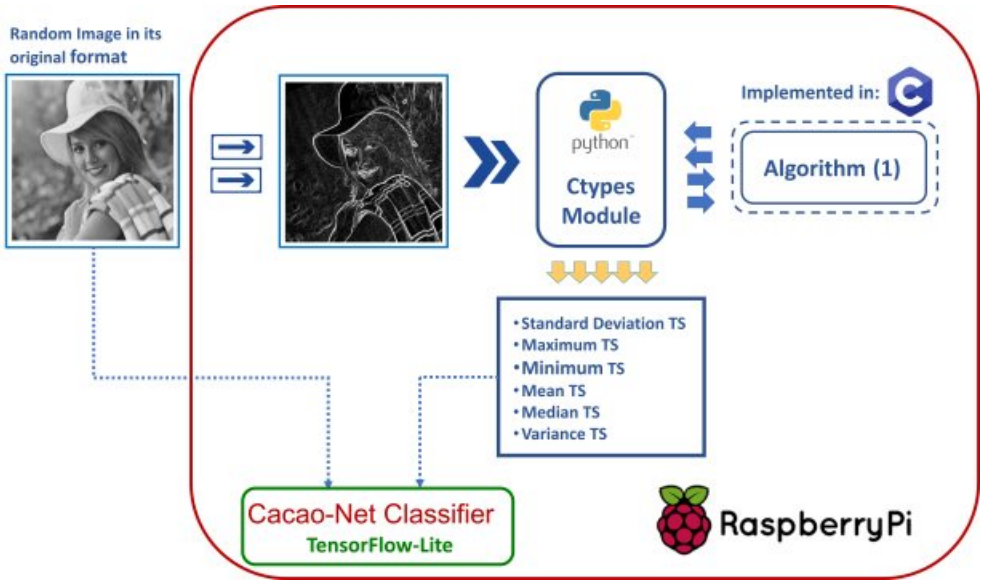

Fig. 7.4 ML Inference workflow for the CACAO-Net classifier on the Raspberry Pi.

As for centery, it can consequently be extracted using the techniques as shown in (11):

$$\begin{aligned} &For\ y \leq sy\text{: }[\ Hity = \Sigma_{y=1}^{sy}\ Mask(Sobel[:,y] \gtrsim \gamma)\ ] \\ &\quad if\ Hity \gtrsim 0 \rightarrow sumy\ +\!=\ y,\ county\ +\!=\ 1 \\ &centery = sumy/county \end{aligned} \tag{7.11}$$

2. Secondly, a radial scan (Izbicki (2011)) is applied from the centroid with coordinates (centerx, centery) obtained in step 1 to convert the Sobel output into six TS which are related to six statistical functions (Standard Deviation, Mean, Variance, Maximum, Minimum, and Median).

3. After extracting the TS for all records, since for each record the length of TS may not be consistent, therefore, each set is interpolated till the maximum length of all TS to end up with a dataset that has the same number of features for every record. However, the Angle array *Ang* is introduced in addition to the Distance variable *D* in this work to improve the explainability in the following sections. Moreover, the angle related features expressed in the variable *Ang* are not utilized in the NN classification architecture proposed since the *D* vector can provide enough information for accurate image classification. The process of interpolation may be implemented through the proposed Python command using the OpenCV library (OpenCV (2023)):

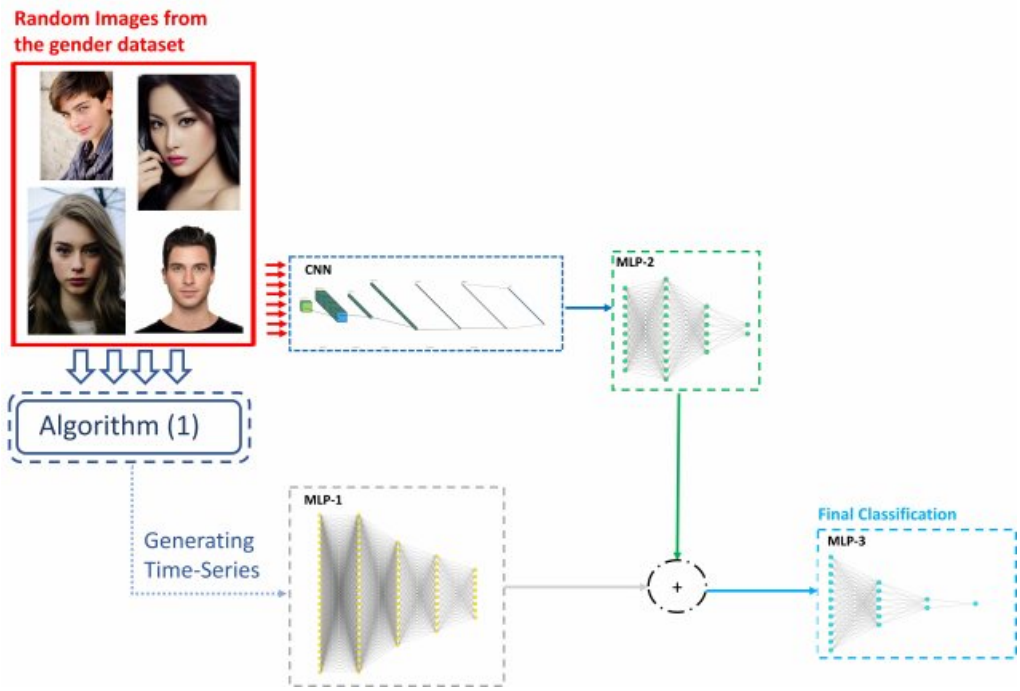

Fig. 7.5 The Hybrid CACAO-Net (Model 1) mixed-data Neural Network for image classification. Algorithm (7.1) which is used for pre-processing was implemented offline before applying inference on the edge.

$$Ts = np.array(cv2.resize(Ts, (1, maximum\_length))).flatten() \tag{7.12}$$

Where the TS "Ts" is resized till the maximum length available using the OpenCV library before being converted into a Numpy array (Numpy (2023)) which is consequently turned into a 1D object using the flatten command.

The CACAO-Net model 1 that has been adopted is presented in Figure 7.5, whereas illustrated, the image data is used as input to a CNN which feeds a Multi-Layer Perceptron (MLP) block. In parallel, the extracted TS is processed through another MLP (Yu et al. (2019)) before fusing the two sub-networks followed with a second MLP to apply ML classification. Furthermore, the pre-processing title Algorithm (1) which generates a TS (Daher et al. (2022)) is applied in real-time during the inference stage which is performed on a Raspberry Pi remotely. A more complete description of CACAO-Net models architecture is provided in 7.1 where the size of every layer in the model is extracted as found in Keras (Ramasubramanian and Singh (2019)).

Additionally, experiments using specialized XAI techniques have been performed using the model to interpret and extract which of the physical parameters of an image led to accurate classification. One way to implement local explainability is through Shapley values (Merrick and Taly (2019)), where the marginal contribution *MC* of each feature *fi* is estimated based on the weighted average of each combination and additions of all features, as shown in Equation (13-15):

$$SHAP_{fi} = w_1.MC_{fi,(fi)} + w_2.MC_{fi,(fi,fi+1)} + w_3.MC_{fi,(fi,fi+1,fi+2)} + ... \tag{7.13}$$

Table 7.1 Layer hierarchy of the CACAO-NET model 1: The summary for a *Z*-class dataset.

Layer	Shape	Filters
Proposed model		
MLP-1		
DENSE	1186	
DENSE	1220	
DENSE	800	
DENSE	600	
DENSE	400	
CNN & MLP-2		
CONV2D	(112x112)x3	(3x3)x32
MAX-Pool		(4x4)x32
CONV2D	(27x27)x32	(3x3)x64
MAX-Pool		(3x3)x64
CONV2D	(8x8)x64	(3x3)x64
MAX-Pool		(2x2)x64
DENSE	576	
DENSE	600	
DENSE	300	
DENSE	100	
DENSE	50	
MLP-3		
DENSE	450	
DENSE	300	
DENSE	50	
DENSE	20	
DENSE	*Z*	

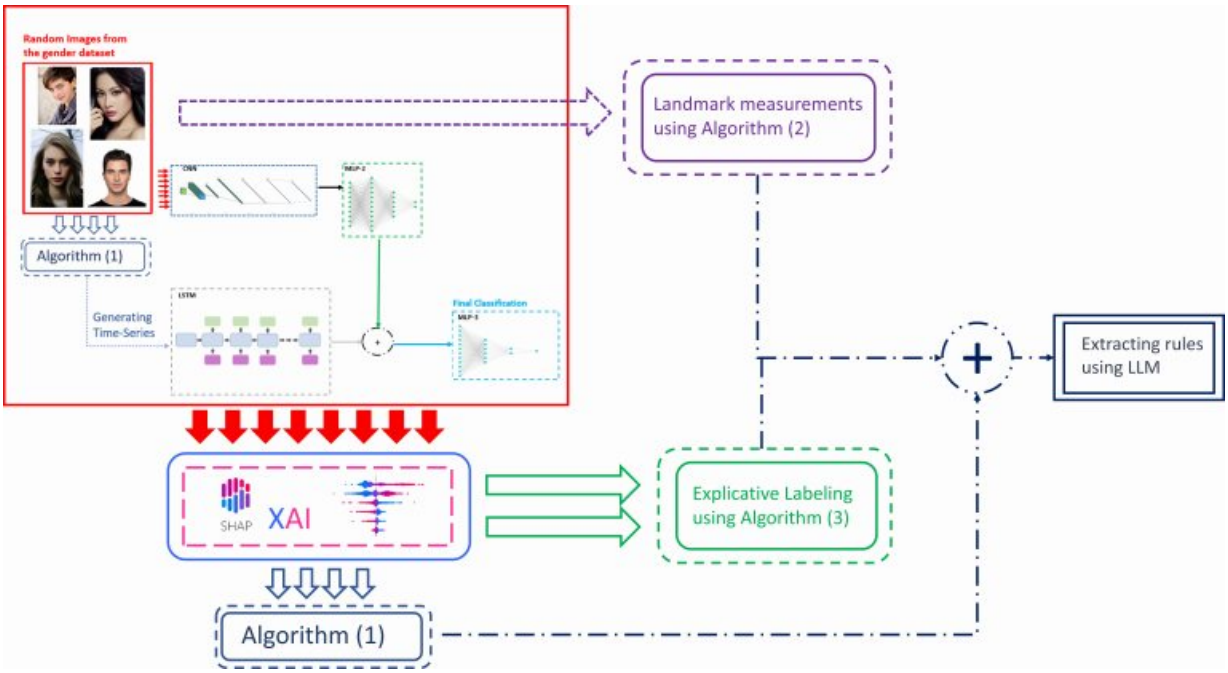

Fig. 7.6 CACAO-X XAI framework using CACAO-NET model 2.

$$Where:\ MC_{fi,fi,\ldots} = Prediction_{+fi} + Prediction_{-fi} \tag{7.14}$$

$$w_i = i - \binom{F}{i} \tag{7.15}$$

These Shapley values correspond to local XAI, where the pixels that correspond to the correct classification for each single input images are generated visually to give insight to the practitioner to which human understandable features contributed to the classification. On the other hand, global explainability which is the topic of this chapter deals with giving human understandable rules and conditions that contribute to the classification of each class for the entire dataset. The same principle is applied for RGB images by computing the Shapley values of each pixel and visually displaying the image while highlighting the pixels which locally contributed to the target classification of that instance. An example of the Shapley output on one image can be viewed in Figure 7.1. As presented in Figure 7.1, in the original image to the left is applied to the explainer which is based on a mixed-data NN. The image at the center provides the pixels which classify this subject as a female.

In addition to the Shap-values-based TS which include angle and distance information from each centroid, application specific landmarks may be used to add measurement to the XAI model. Therefore, for the gender classification dataset, the facial landmarks which were located using the Python Face Recognition library (Geitgey (2020)) are illustrated in Figure

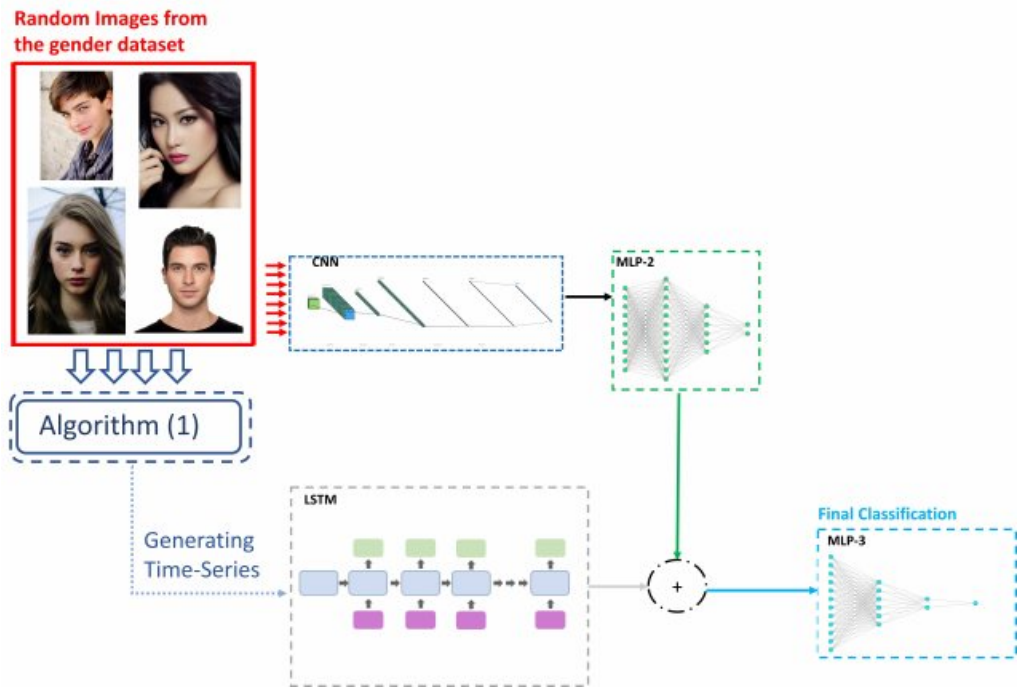

Fig. 7.7 The Hybrid CACAO-Net (Model 2) mixed-data Neural Network for image classification. Algorithm (7.1) which is used for pre-processing was implemented offline before applying inference on the edge.

7.8 to the right, where the facial landmarks for each subject are clearly shown in the form of red dots.

Consequently, using the coordinates of the extracted landmarks, the distance measurements between the combinations of each landmark pairs are computed and stored as features to be used for XAI implementation. Algorithm (7.2) explains how these features heat can be extracted in a systematic pseudocode.

The final experiment deals with image explainability and relies on the Shapley heat-map as well as the *Ang* and *D* variables as inputs where data labeling is crucial for interpretability, therefore, the labeling of the landmark distances, angles and the TS data are meticulously annotated to clarify why each class was classified or misclassified. Therefore, for the XAI Algorithm, the Logic Learning Machine (LLM) was adopted to perform the analysis using the Rulex machine learning package (Muselli (2012)) which has explainability as its prime feature. Algorithm (7.1) and the LLM engine are operating as C/Cpp vectors for maximal and parallel performance.

The LLM algorithm relies on switching neural networks and shadow clustering (Muselli and Ferrari (2009)) to derive explainable rules related to a particular dataset. In a brief description, explainability is achieved by a mapping positive Boolean function $f: 0, 1^n \rightarrow 0,$

Algorithm 7.2 *Facial Landmark FE*

Require: *Landmark coordinates*
Ensure: *Combinational distance features*
1: **for** *all landmarks in every photo* **do**
2: **for** *k in landmark array* **do**
3: **for** *j in landmark array* **do**
4: *Append Euclidean distance*$((x_k, y_k),(x_j,y_j))$
5: *to Features[n] where n is the number of landmarks*
6: **end for**
7: **end for**
8: $n{+}{+}$
9: **end for**
10: **Return** *Features*

1 with input *xi , i = 1, . . . , d*, to approximate an unknown function *g*. The methodology consists of determining if $Bi \in Xi$ to find *g'*, which approximates *g* on *X* where:

$$\textit{for every set } B \in B', \textit{ where } \prod_{i=1}^{d} Bi\text{: } Bi \in B'i,\ i = 1,...,d \tag{7.16}$$

$$\textit{Consider } Ql_n^1 \subset \{0,1\}^n \textit{ contains n bits having exactly łvalues 1} \tag{7.17}$$

Using ϕ, *X* is mapped into the strings of Ql_n^1 , so that $\phi)(x) \neq \phi(x')$ only if $x \in B_2$ and $x' \in B_1$, such that $B_1 \subset B$ and $B_2 \subset B$. Then the positive Boolean function is selected such that $f(\phi(x)) = g'(x)$ for every $x \in X$.

Furthermore, to allow for better explainability with LLM, explicative labeling is added to the automated dataset which is created using Algorithm (7.1). Consequently, annotating the measurement range, statistical operation, and its value for both angles and distances can be highly informative to an operator who is experimenting with XAI. Therefore, in addition to the angle and distance quantities, also an explicative label is added to the landmark pair distances through iterating over the Python data structure which contains the feature values. Also, the second portion of the pseudo-code explicatively outlines the angle or distance of each feature along with statistical quantities. Algorithm (7.3) describes how to annotate the features in the dataset used for ML explainability where the first part consists of a first set of two embedded loops which describes each landmark pairs used to measure the Euclidean distances as measured in Algorithm (7.2). Also, the second portion of the pseudo-code explicatively outlines Algorithm 3.1 which was first introduced by Daher et al. (2021) that is used to achieve this task.

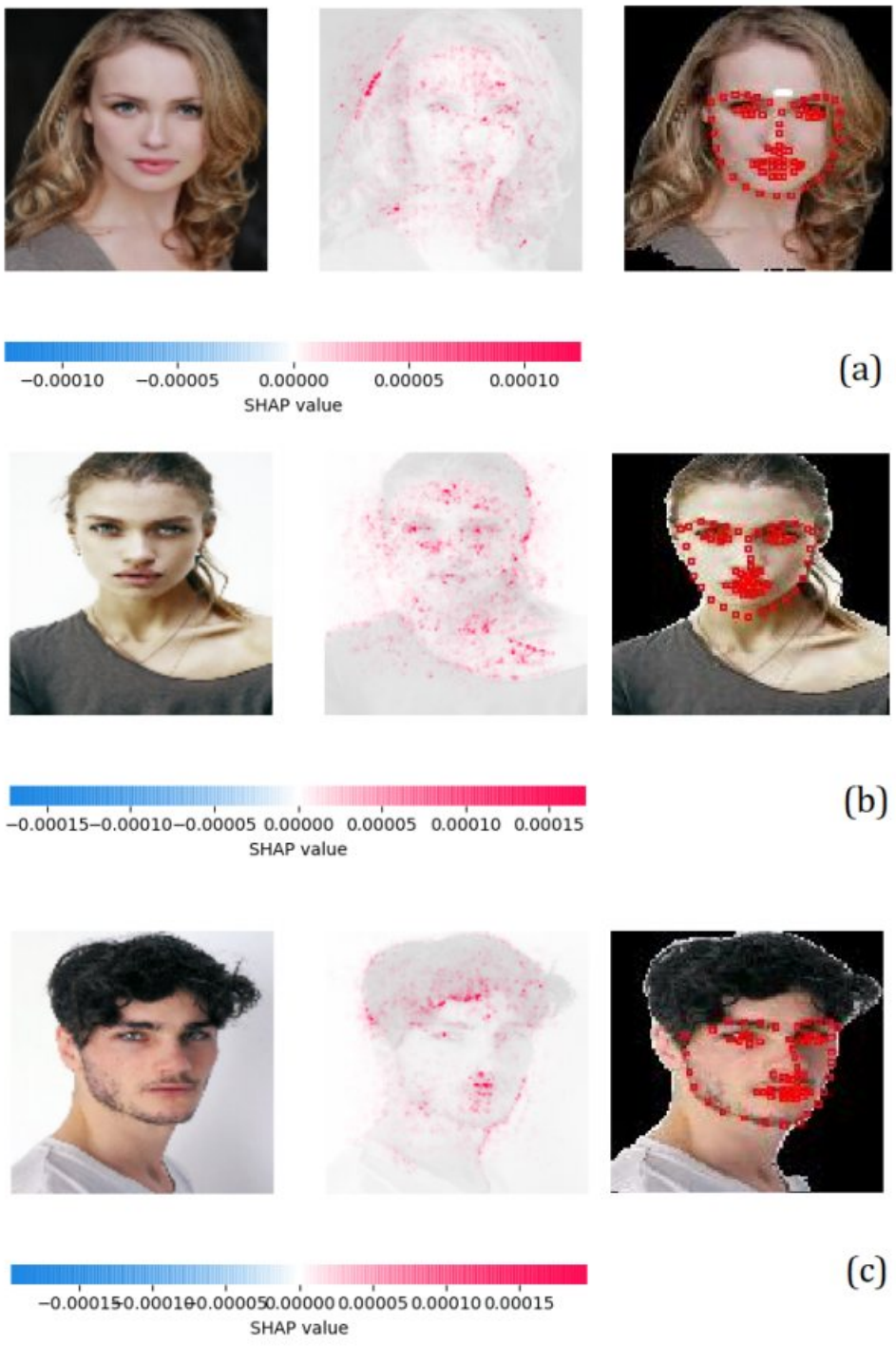

Fig. 7.8 The Shap-values of three subjects of images taken from the gender classification dataset, with the original image being placed to the left and the local explainable pixels at the center. To the right, application-specific landmarks of each subject using the Python Face Recognition Library are illustrated.

Algorithm 7.3 *Explicative feature labels for time-series*

Require: *One of the twelve Timeseries generated using Algorithm (1)*
Ensure: *Statistical time-series feature labels*
for *every coordinate1 and key1 in landmarks* **do**
  Count1++
  for *every coordinate2 and key2 in landmarks* **do**
    Count2++
    if *(Count2 ≥ Count1 and key1 == key2) or key2 ≥ key1* **then**
      Append key1, count1, "Static", key2, count2 to Label1
      Append key1, count1, "Static", key2, count2 to Label2
      Append Label1, Label2 to Labels
    end if
  end for
end for
for *Type in D and Ang time-series* **do**
  for *i in time-series* **do**
    if *i ≤ len(time-series)* **then**
      Tvalue ="0-90"
    else if *i ≤ len(time-series)/2* **then**
      Tvalue ="90-180"
    else if *i ≤ 3.len(time-series)/4* **then**
      Tvalue ="180-270"
    else if *i ≥ 3.len(time-series)/4* **then**
      Tvalue ="270-360"
    end if
    if *modulo(i,6) == 0* **then**
      Statvalue =" Variance"
      Len = i/(len(time-series)/360)
    else if *modulo(i,5) == 0* **then**
      Statvalue =" Min"
    else if *modulo(i,4) == 0* **then**
      Statvalue =" Max"
    else if *modulo(i,3) == 0* **then**
      Statvalue =" Median"
    else if *modulo(i,2) == 0* **then**
      Statvalue =" STD"
    else
      Statvalue =" Mean"
    end if
  end for
end for
Append Type, Statvalue, Len, TValue to Desc
Return*Labels, Desc*

In order to fully understand and interpret image classification in an explainable way, XAI may be implemented on image forecasting models, such as Keras (Ramasubramanian and Singh (2019)), to find out which physical parts of instances of a class led to correct classification. Therefore, we have applied the Python Library for XAI namely SHAP (Lundberg (2018)) to extract the pixels for each image which contribute to the classification and apply Algorithm 7.1 in its complete version on each of the so-called heat maps. After applying the contour-based Image-to-TS conversion algorithm both angle and distance statistical patterns are used as features for an upcoming forecast using LLM running on Rulex (Muselli (2012)). Therefore, to supply LLM with useful information to implement XAI for global interpretation, Algorithm (7.3) is used to label each feature in a clear description. Additionally, since the gender classification dataset is employed as a reference for explainability, the landmark-based

Table 7.2 Layer hierarchy of the CACAO-NET model 2: The summary for a Z-class dataset.

Layer	Shape	Filters
Proposed model		
LSTM		
LSTM	1237	
LSTM	100	
LSTM	80	
DENSE	40	
CNN & MLP-2		
CONV2D	(112x112)x3	(3x3)x32
MAX-Pool		(4x4)x32
CONV2D	(27x27)x32	(3x3)x64
MAX-Pool		(3x3)x64
CONV2D	(8x8)x64	(3x3)x64
MAX-Pool		(2x2)x64
DENSE	576	
DENSE	600	
DENSE	300	
DENSE	100	
DENSE	50	
MLP-3		
DENSE	90	
DENSE	300	
DENSE	50	
DENSE	20	
DENSE	Z	

features computed using Algorithm (7.2) are used along with the interpolated TS as features for the LLM. Furthermore, Algorithm (7.3) in its first part provides explicative labeling for these landmark positions. Consequently, Figure 7.6, presents an overall illustration of the CACAO-X framework for global explainability using the gender detection application. As shown, this setup utilizes CACAO-Net model 2, as shown in 7.7, which relies on an LSTM rather than an MLP coupled to the TS input (Although an MLP could be used) and a complete dissection of this architecture can be viewed in Table 7.2.

7.5 Datasets

To assess the effectiveness of the CACAO-X framework for image classification, five diverse datasets were adopted to evaluate the robustness of the proposed model. Therefore, the same architecture was used for each dataset while non-structure-related parameters like batch size, learning rate, epoch count, and decay that were mildly tuned (Mainly due to different numbers of classes and numbers of records). The training for every dataset was performed on an MSI laptop whereas the testing was applied on Raspberry Pi Boards remotely. In an upcoming section, various ML works which rely on similar datasets are discussed and later compared.

7.5.1 Gender classification dataset

A subset of the dataset found in Gupta (2011) is used which consists of 5400 images grouped according to gender. Both male and female images consist of randomly collected shots with varying ethnicities, ages, poses, zoom levels, and background. We have implemented ML classification on this dataset previously in Daher et al. (2021) using the described Image-to-TS algorithm, where an overall accuracy of 96.5% was achieved. Therefore, with the attempt to expand the scope of operation and robustness of this algorithm we have implemented two hybrid models in Fig. 5 and Fig. 6 to improve existing accuracy and explore other application fields.

7.5.2 Satellite images dataset

This remote sensing dataset provided in Reda (2021) images grouped into 4 classes: Cloudy, Desert, Green area, and Water area. This dataset is also used as a data source for ML classification using the proposed model. In this chapter a total of 1525 images were used in ML operations.

7.5.3 Skin Cancer dataset

In Fanconi (2020), a skin cancer dataset (with 3290 samples) is presented which is divided into Malignant and Benign classes. Due to the nature of the images, being differentiable by color, contour detection on its own as discussed in Daher et al. (2021) cannot be effective in achieving high classification accuracy. Consequently, the addition of raw 3-D RGB data can

add color information to the contour-based shape information while spinning up the described hybrid CACAO-Net) architecture, with the goal of diagnosing skin images accurately.

7.5.4 Weather images dataset

The dataset in Parteek (2020) consists of images of weather images distributed over 4 classes: Cloudy, Rain, Shine, and Sunrise. It consists of 300 cloudy, 215 rain, 253 shine, and 357 sunrise images.

7.5.5 Shadow dataset

The dataset in Mohanna et al. (2022), CNN is used to discriminate between single human targets and shadowed targets in a binary approach using a low-cost radar as a sensor while relying on various Mobile-Net architectures as the classifier. The dataset consists of 1200 samples equally split between two classes wherein each subset of the records, the distance between the targets and the sensor varies.

7.6 Methodology using oscillation-based early-stopping

In order to achieve the highest validation accuracy while avoiding overfitting and insuring the stability of the ML model, a novel callback function based on STD-based early-stopping has been developed. Algorithm (7.4) demonstrates the method for early-stopping where the five previous validation accuracies are stored in a queue where their standard deviation including the last accuracy is compared with a threshold to stop training at a stable state. Concerning the thresholds, they are application specific and can be set experimentally by running training a few times and easily noticing the ranges of the validation metrics before setting the threshold parameters. Although, the task is simple it can be sometimes an iterative process, however, much simpler and more straightforward than ML parameter tuning. Furthermore, the algorithm has a safety mechanism wherein case the target accuracy is not reached, it will update the validation threshold by a less stringent value.

Regarding the data splitting, two arrangements were employed to validate the CACAO-Net classifier. The consists of a K-folder setups with k = 5 and in a Train/Validate split where the early function was used to stop the training for each run. Additionally, a three-set arrangement was also used to train the classifier while stopping at a state based on the validation performance before applying the inference for a third set on a Raspberry Pi

Algorithm 7.4 *STD-Based Early-Stopping Call-back Function*

Require: *Validation accuracy,* $\beta 1$*,* $\beta 2$*, QueueSize,* γ*, Ep1, Ep2, Start*
Ensure: *Stable validation accuracy*

```
for i in every epoch do
    if i ≥ Start then
        Append val-acc[i] to QUEUE
    end if
    if size(QUEUE) ≥ QueueSize then
        Delete(QUEUE, 0)
    end if
    if Val-acc[i] ≥ β1 then
        if STD(QUEUE) ≤ γ then
            if i ≥ Ep1 then
                STOP training
            end if
        end if
    end if
    if Val-acc[i] ≥ β2 then
        if STD(QUEUE) ≤ γ then
            if i ≥ Ep2 then
                STOP training
            end if
        end if
    end if
end for
```

board. For all classification experiments the inference was performed on two Raspberry Pi's remotely using a condensed Tensorflow Lite (TF-Lite) model.

The hyper-parameters of the CACAO-Net models were tuned based on the K-fold validation accuracy metric by setting the Learning rate, weight decay, batch size, and the number of epochs. A relatively short amount of time was dedicated for this phase where the very mild tuning was applied to reach the desired results mainly due to the robustness of the CACAO-Net model. The actual values or the parameters are shown in Table 3.

7.7 Experimental results

The final goal of this work is to achieve competitive results in all five datasets having five diverse application domains. High accuracies were achieved in the preliminary forecasts, such as 98.4% for the gender detection dataset, and 96.8% for the skin cancer dataset. As for

the satellite imaging dataset an accuracy of 97.39% was attained and overall accuracies of 99.4% and 91.1% for the shadowed targets and weather classification dataset, respectively. Additionally, a novel early-stopping function has been implemented which tracks the current validation accuracy and the *N* previous ones, while computing the *STD* of the set along with checking if the current value surpasses a predefined threshold (which is updated as the number epochs are increases) to insure performance and stability. Moreover, XAI was used to interpret the image-based gender detection dataset while using multiple Python libraries for the task, where results show good correlation as to which physical parts of the image determines gender.

7.7.1 Forecast results

A K-fold setup (with five folds) was adopted by relying five datasets where the implementation is being conducted on an MSI laptop while the inference for the unseen data is applied using TF-Lite and on two Raspberry Pi platforms. This has been performed on all five datasets to evaluate the robustness of the CACAO-X framework. In every experiment, a three-way split was applied, as described in Table 7.3, for remote inference on the Raspberry Pi.

Table 7.3 Summary of CACAO-Net ML classification hyper-parameters.

Dataset	**Learning rate**	**Batch size**	**Decay**	**Train/Validate/Test**	**Epochs**
Gender	5e-4	500	5e-6	0.72/0.18/0.10	100
Satellite	5e-4	30	5e-6	0.72/0.18/0.10	200
Shadow	5e-4	30	5e-6	0.81/0.09/0.15	100
Skin Cancer	5e-4	300	5e-6	0.76/0.14/0.10	100
Weather	1e-4	50	1e-6	0.66/0.29/0.05	100

Table 7.4 Summary of CACAO-Net ML classification results on the edge.

Dataset	**Validation MSI** (Acc)	**Inference RP4** (Acc)	**Time/Task RP4 (FE)** (msec.)	**Time/Task RP4 (Inf)** (msec.)	**Time/Task RP3 (Inf)** (msec.)
Gender	98.40 ± 0.5 %	98.15 %	52.54	14.66	139.4
Satellite	97.28 ± 0.5 %	94.08 %	104.02	9.20	145.7
Shadow	99.41 ± 1.1 %	98.89 %	109.57	8.72	144.9
Skin Cancer	96.84 ± 0.38 %	95.50 %	124.72	9.24	155.8
Weather	91.20 ± 2.0 %	92.86 %	117.32	9.45	129.7

The first inference setup consists of a Raspberry Pi 3B accessed remotely where the TS FE using Algorithm (7.1) has been implemented offline and in Python, therefore only the inference time/task has been recorded. Secondly, a Raspberry Pi 4B with 4GB of Random Access Memory (RAM) was used with the same Image-to-TS conversion which is implemented per image and in real-time. In this setup, the low-level C version of the code is compiled as a shared Linux ".SO" library for proven deployment on limited resources devices.

$$Accuracy_{ij} = \frac{TP_{ij} + TN_{ij}}{TP_{ij} + TN_{ij} + FP_{ij} + FN_{ij}} \tag{7.18}$$

$$Accuracy_{i(avg)} = \frac{1}{N}\Sigma_{j=1}^{N}\alpha_j \frac{TP_{ij} + TN_{ij}}{TP_{ij} + TN_{ij} + FP_{ij} + FN_{ij}} \tag{7.19}$$

$$Accuracy_{STD}{}^2 = \frac{1}{K.N}\Sigma_{i=1}^{K}\Sigma_{j=1}^{N}(Accuracy_{ij} \text{ - } Accuracy_{i(avg)})^2 \tag{7.20}$$

A summary of the experimental results can be viewed in Table 7.4, where it shows the validation and testing accuracies for every dataset, where CACAO-Net achieves robust performance on the edge in terms of accuracy, reliability regarding new data, as well as being light-weight with good performance metrics. Moreover, the functions used to determine

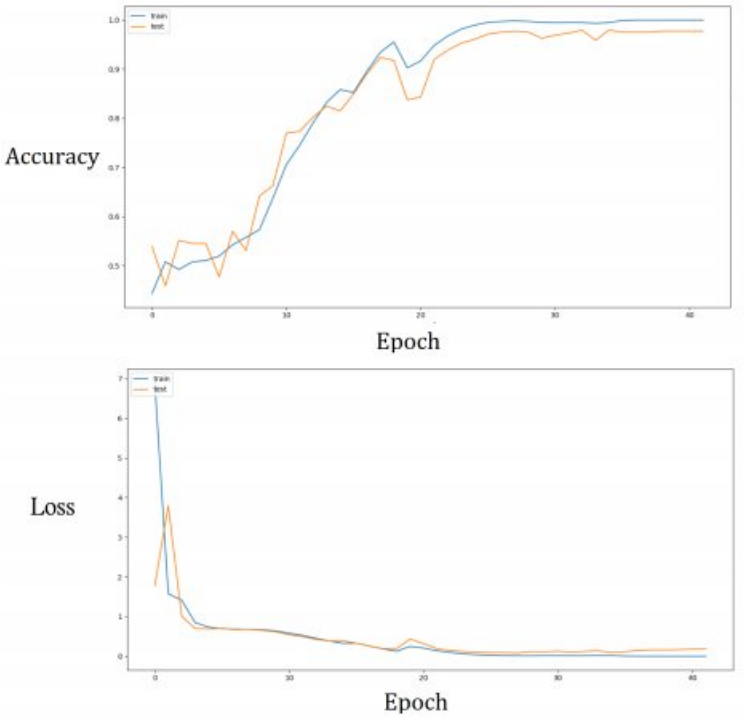

Fig. 7.9 Gender classification accuracy and loss curves.

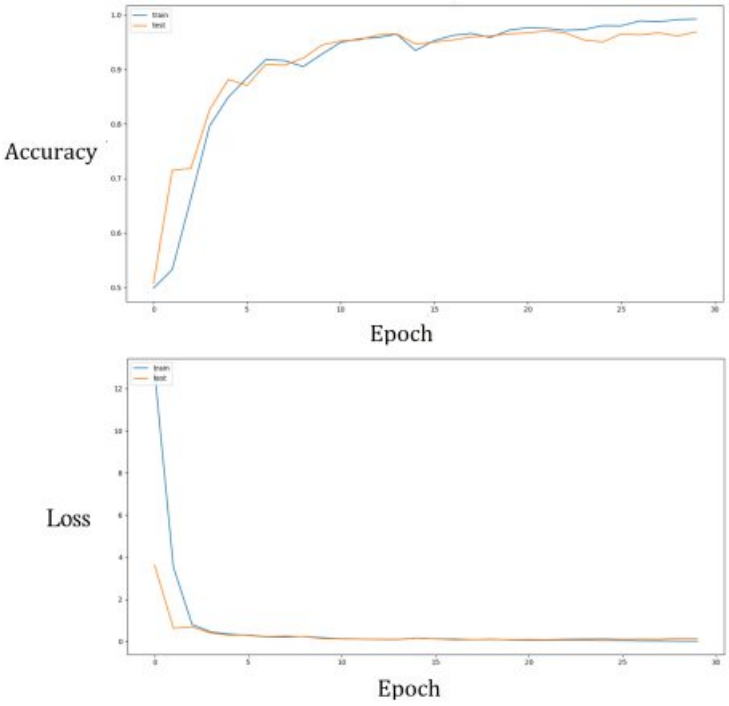

Fig. 7.10 Skin cancer classification accuracy and loss curves.

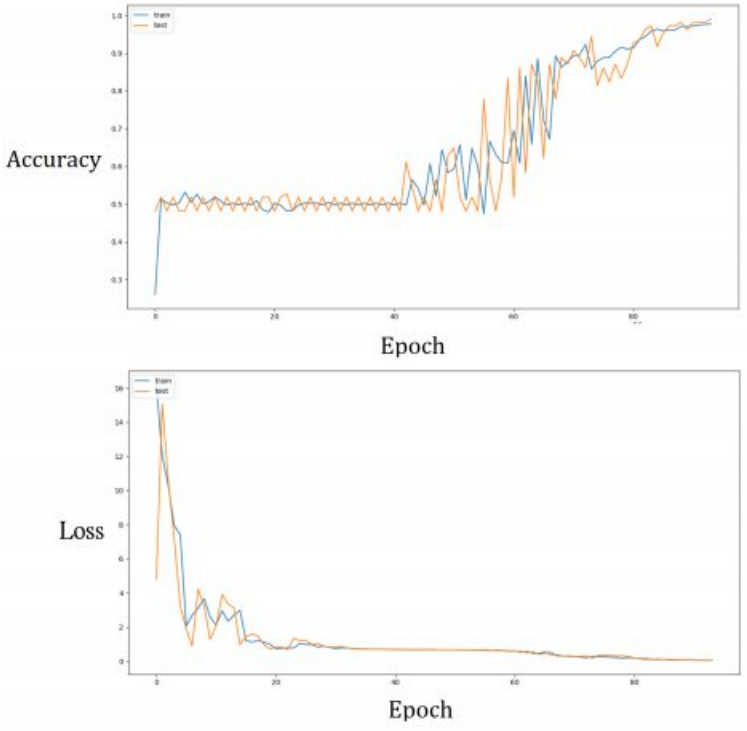

Fig. 7.11 Shadow classification accuracy and loss curves.

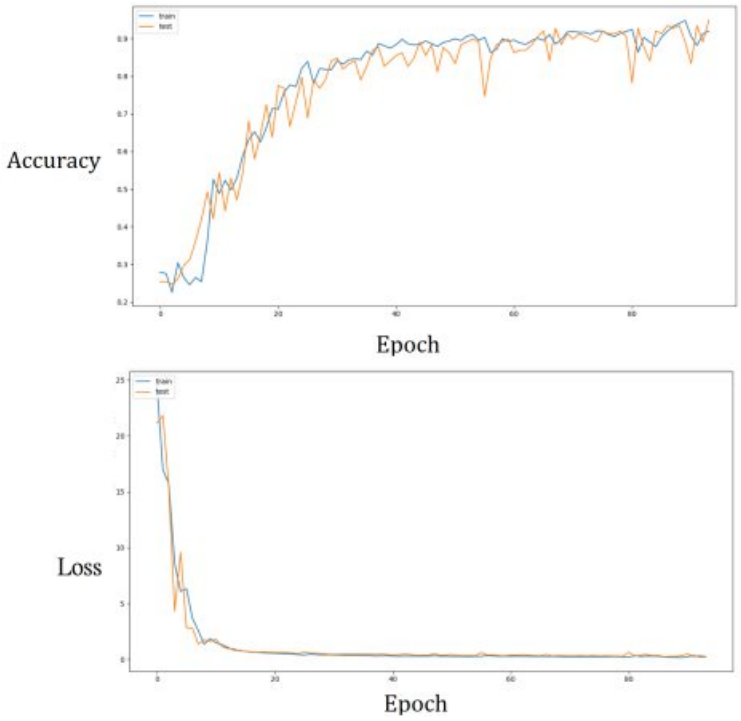

Fig. 7.12 Satellite classification accuracy and loss curves.

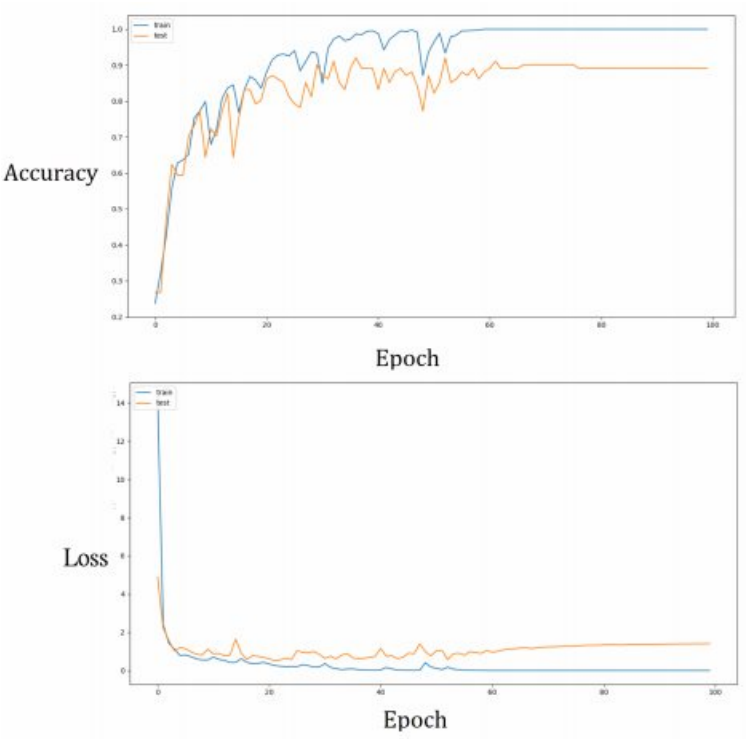

Fig. 7.13 Weather classification accuracy and loss curves.

the values in Table 4 are shown in equations (18-20). As shown in the equations TP is the true-positive rate, TN is the true-negative rate, FP is the false-positive rate, and FN is the false-negative rate. K represents the number of forecasts performed and N corresponds to the number of class labels. Finally, α represents the weight for each class label from its particular dataset.

Moreover, the accuracy and loss curves can be viewed in 7.9-7.13 where the performance is robust in terms of settling stability for all five datasets where for each the upper curve represents the Train/Validate accuracy and the lower curve corresponding to the loss. In these results, the early stopping callback function, discussed in Section 5, is responsible for the halting of the training at the desired target validation accuracy which based on its most recent stable (Low deviation) mean. Additionally, the inference confusion matrices are provided in Figure 7.14, where the results summary shows good performance in five experiments, where through visual inspection, it is shown that the FP's and FN's are kept to a tolerable minimum as shown in the heat-map representations.

7.7.2 CACAO-NET comparisons

In Yang et al. (2006), forward-facing images of people of the same ethnicity were used. In this setup a maximum accuracy of 97.16% was achieved using a SVM. Also, SVM was also used in Moghaddam and Yang (2000), where the dataset contains low-resolution and forward-facing images where an accuracy 96.6% was attained. In Eidinger et al. (2014) authors attempt to detect gender using inconsistent images and in their experiments, the highest classification accuracy reached is 88.4%. In contrast to being a setup used exclusively for gender classification, CACAO-Net outperforms the literature in both averaged validation and testing forecasts while being able to transition to other application fields and remains competent. Regarding the shadowed targets dataset, CACAO-Net outperforms the ML workflows applied in Mohanna et al. (2022) by a maximum of 17.9% and a minimum 7.2% where the algorithm presented in this project was not exclusively developed for any particular application.

Regarding weather classification, in Zhang et al. (2016) a multiclass dataset of weather images is presented for four classes where an accuracy of 71.4% is achieved. A maximum overall accuracy of 94.8% is achieved in Roser and Moosmann (2008) for a three-class dataset, however, a single forecast was performed without averaging multiple runs, therefore, the results may have overfit to that particular split. A binary weather classification setup with application-specific features is discussed in Lu et al. (2014) where an averaged accuracy of

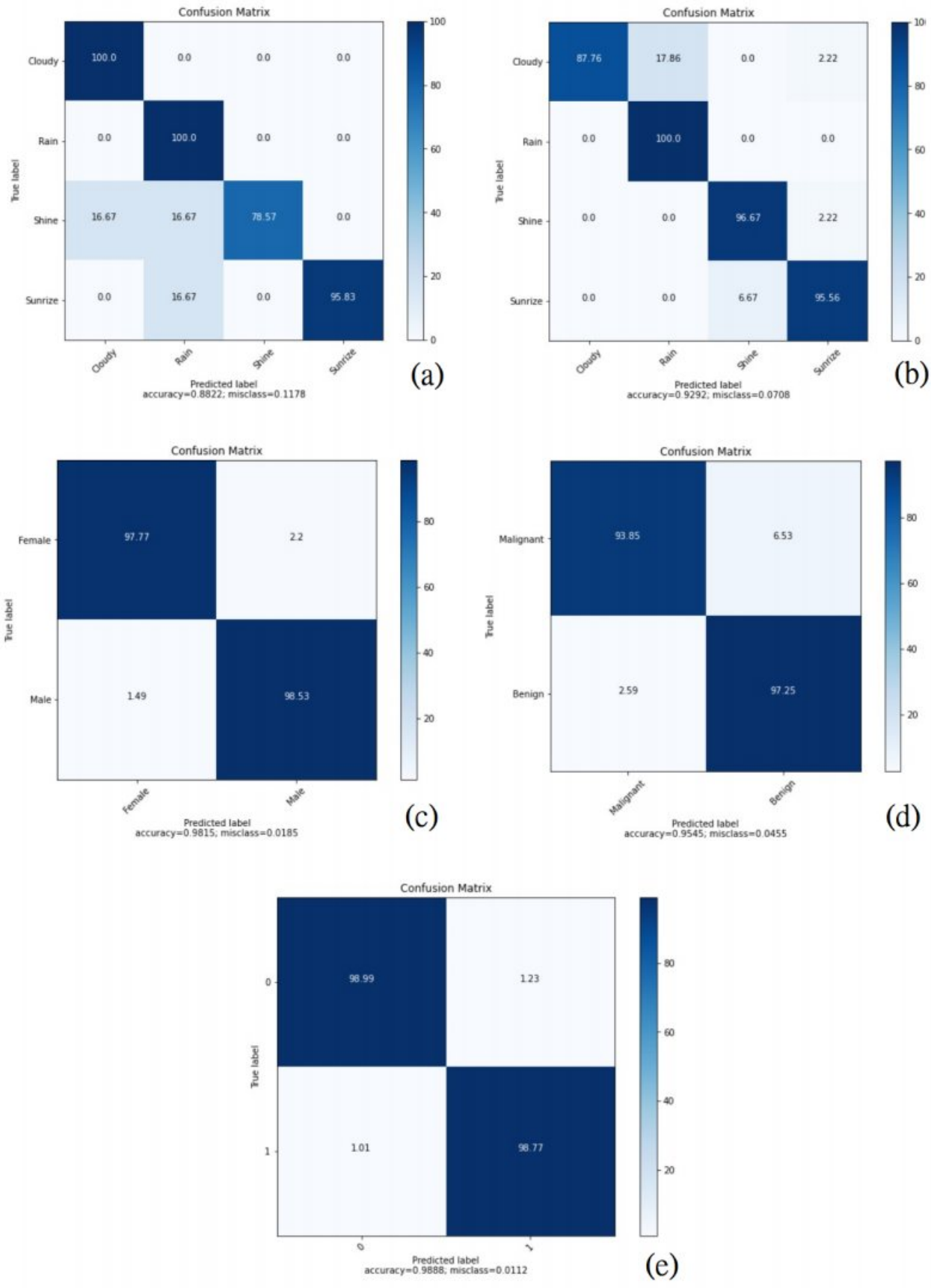

Fig. 7.14 Confusion matrices from the inference performed using TF-Lite on the Raspberry Pi for all five datasets: (a) Weather dataset. (b) Satellite images dataset. (c) gender dataset. (d) Skin cancer dataset. (e) Shadowed target dataset.

53.1% is achieved. Regarding the skin cancer classification in Hekler et al. (2019) it is carried out using CNN while relying on a questionnaire of instance selection where the methods used achieve a maximum accuracy of 83%.

Satellite classification of forest/non-forest images is discussed in Cihlar (2000) where the overall accuracy using SVM varies between 92.1% and 98.9% having an average of 96.3%. In fact, CACAO-Net achieves a much smaller range based on the STD field as shown in Table 4.

As shown in Table 4, for the Raspberry Pi 3B+, the average time taken for each inference task is shown where the FE is implemented offline. As for the Raspberry Pi 4B (4GB), the FE is implemented on the blind just before the inference, and therefore, the average times taken for both inference and FE are displayed. Consequently, due to more recent versions of TF-Lite on the Raspberry Pi 4B and due to its more advanced architecture, an improvement of 11x to 15x is visible regarding the inference time taken per task. Additionally, even after adding the FE to the inference process, since Algorithm 3.1 is compiled in low-level C-Code, the Raspberry Pi 4B's overall performance remains superior to its predecessor.

Consequently, due to the nature of the CACAO-Net model and its flexibility, the accuracy compares well with the literature where similar datasets are used in addition to being portable to apply edge inference while relying on Algorithm (7.4) to deploy a stable ML model after training has halted.

7.7.3 CACAO-Net more reliable comparisons with classic ML models

In section 6.1 multiple comparisons with other works from the literature are discussed in addition to providing performance metrics such as accuracy, inference speed on the edge, and the FE time-taken per task while relying on TF-Lite models for classification. Regarding the comparisons, most of the examples relate only to accuracy and its deviation by applying multiple forecasts in various splits to prove the robustness of CACAO-Net model 1. However to further validate the performance of the CACAO-X framework, additional comparisons using the exact same datasets and splitting content with classic ML models is presented while relying on more reliable metrics.

In regards to the comparison seven tuned AI algorithms taken from the Python Pycaret Automated ML (AutoML) library (Pycaret (2023)) were used, namely:

1. Naive Bayes (NB) which is a probabilistic ML model that relies on the Bayes theorem.

2. Logistic Regression (LR) is a statistical method that finds a relationship between dependent and independent features.

3. K-Nearest Neighbor (KNN) is a ML algorithms which attempts to create boundaries by grouping records and assigning them appropriately to a certain class label by means of iteration.

4. Multi-Layer Perceptron (MLP) is a very simple or shallow NN which is the basis for the CNN and LSTM networks discussed in Section 3.

5. Decision Trees (DT) is an AI algorithm that splits the dataset iteratively in order to isolate all points that correspond to every class label.

6. Random Forest (RF) uses forests or groups of DTs to achieve very accurate performance by means of running multiple DTs and reaching a final decision by means of a voting scheme.

7. Adaboost (Ada) is a weighted AutoML algorithm which relies on meta-learning to optimize its training based on the performance criteria.

Pycaret is a significant automated AI library which provides tuned ML models capable of competing with published works, since it uses AutoML techniques to reach good performance in all of the described metrics in this thesis.

Regarding the metrics which are employed, other than accuracy, precision, recall and specificity are used to assess the performance of ML model where precision evaluates how precise the classification of a class is from a pool of samples assigned to that label by prediction. Recall on the other hand evaluates from the actual labels of each sample, how probable it is to reach a correct classification. Additionally, specificity is evaluated by dividing the number of TN results by the total number of negatives. Equations (21 - 23) present the expressions for precision, recall and specificity respectively:

$$Precision_{ij} = \frac{TP_{ij}}{TP_{ij} + FP_{ij}} \tag{7.21}$$

$$Recall_{ij} = \frac{TP_{ij}}{TP_{ij} + FN_{ij}} \tag{7.22}$$

$$Specificity_{ij} = \frac{TN_{ij}}{TN_{ij} + FP_{ij}} \tag{7.23}$$

The averages of accuracy, precision and recall taken from a series of forecasts in terms of each i^{th} forecast can be written as follows:

$$Percision_{i(avg)} = \frac{1}{N}\Sigma_{j=1}^{N}\alpha_j \frac{TP_{ij}}{TP_{ij} + FP_{ij}} \tag{7.24}$$

$$Recall_{i(avg)} = \frac{1}{N}\Sigma_{j=1}^{N}\alpha_j \frac{TP_{ij}}{TP_{ij}+FN_{ij}} \tag{7.25}$$

$$Specificity_{i(avg)} = \frac{1}{N}\Sigma_{j=1}^{N}\alpha_j \frac{TN_{ij}}{TN_{ij}+FP_{ij}} \tag{7.26}$$

The standard deviations of this series of forecasts may be computed using equations (27 and 29):

$$Precision_{STD}{}^2 = \frac{1}{K.N}\Sigma_{i=1}^{K}\Sigma_{j=1}^{N}(Precision_{ij} - Precision_{i(avg)})^2 \tag{7.27}$$

$$Recall_{STD}{}^2 = \frac{1}{K.N}\Sigma_{i=1}^{K}\Sigma_{j=1}^{N}(Recall_{ij} - Recall_{i(avg)})^2 \tag{7.28}$$

$$Specificity_{STD}{}^2 = \frac{1}{K.N}\Sigma_{i=1}^{K}\Sigma_{j=1}^{N}(Specificity_{ij} - Specificity_{i(avg)})^2 \tag{7.29}$$

In some datasets the precision and recall may provide with conflicting results mainly due to skewness in the dataset and may be the cause of a non-robust model or improper tuning or data splitting. Therefore, there exists metrics that combine these two fundamental measurements in order to provide a more general and fair assessment of the ML workflow. These consist of the F1-score and the Mathews Correlation Coefficient (MCC), as shown in equations(30-31):

$$F1\text{-}score_{ij} = \frac{2.Precision_{ij}.Recall_{ij}}{Precision_{ij}+Recall_{ij}} \tag{7.30}$$

$$MCC_{ij} = \frac{TP_{ij}.TN_{ij} - TP_{ij}.TN_{ij}}{(TP_{ij}+FP_{ij}).(TP_{ij}+FN_{ij}).(TN_{ij}+FP_{ij}).(TN_{ij}+FN_{ij})} \tag{7.31}$$

The averages for the F1-score and the MCC taken from a series of forecasts in terms of every ith forecast can be written as follows:

$$F1\text{-}score_{i(avg)} = \frac{1}{N}\Sigma_{j=1}^{N}\alpha_j \frac{2.Precision_{ij}.Recall_{ij}}{Precision_{ij}+Recall_{ij}} \tag{7.32}$$

$$MCC_{i(avg)} = \frac{1}{N}\Sigma_{j=1}^{N}\alpha_j \frac{TP_{ij}.TN_{ij} - TP_{ij}.TN_{ij}}{(TP_{ij}+FP_{ij}).(TP_{ij}+FN_{ij}).(TN_{ij}+FP_{ij}).(TN_{ij}+FN_{ij})} \tag{7.33}$$

$$F1\text{-}score_{STD}{}^2 = \frac{1}{K.N}\Sigma_{i=1}^{K}\Sigma_{j=1}^{N}(F1\text{-}score_{ij} - F1\text{-}score_{i(avg)})^2 \tag{7.34}$$

$$MCC_{STD}{}^2 = \frac{1}{K.N}\Sigma_{i=1}^{K}\Sigma_{j=1}^{N}(MCC_{ij} - MCC_{i(avg)})^2 \tag{7.35}$$

To further determine the performance reliability of our ML model another metric may also be used which is balanced accuracy that signifies the average of the precision and the specificity where it may be employed using the following equations:

$$Balanced\text{-}acc_{ij} = \frac{Precision_{ij} + Specificity_{ij}}{2} \tag{7.36}$$

$$Balanced\text{-}acc_{i(avg)} = \frac{1}{N}\Sigma_{j=1}^{N}\alpha_j\frac{Precision_{ij} + Specificity_{ij}}{2} \tag{7.37}$$

$$Balanced\text{-}acc_{STD}{}^2 = \frac{1}{K.N}\Sigma_{i=1}^{K}\Sigma_{j=1}^{N}(Balanced\text{-}acc_{ij} - Balanced\text{-}acc_{i(avg)})^2 \tag{7.38}$$

Dataset metrics discussion

Regarding the classification comparisons with the traditional classifiers and using the same data source used in every split, CACAO-Net was able to outperform each ML model in virtually every metric (Excluding one dataset) related to accuracy, precision, recall, F1-score, MCC and Balanced accuracy proving the advantage of fusing both image and TS data in the training and classification stages. Although some models came close to reaching the performance of the CACAO-Net model 1 classifier in four datasets, when applied on other data sources, the CACAO-Net had better stability and robustness that validate its portability onto various unrelated domains.

the results for the extensive set of automated forecasts conducted can be inspected in Tables 7.5-7.9 for a comparison between CACAO-Net model 1 and the seven AI models which were tuned using Pycaret (Pycaret (2023)) whereas mentioned the CACAO-Net multi input approach outperforms its AutoML competition in every metric 97% of the time (in terms of metric means).

Regarding the gender, weather, satellite and skin cancer data sources, CACAO-Net outperforms the tuned ML models in every metric to sufficiently prove it portability onto various ML domains, however, in case of the shadowed target dataset, LR achieves a 100% validation score in every metric overcoming CACAO-Net and the application-specific ML model from Mohanna et al. (2022), which was also surpassed Ada, RF, KNN and DT, solidifying Pycaret as a reliable choice for the metrics comparison mainly due to outperforming a subset of the published results mentioned in this thesis. Nevertheless, CACAO-Net outperforms LR in every other domain substantially which demonstrates its generality.

Table 7.5 CACAO-Net gender classification testing results with comparisons using tuned ML models.

Algorithm	Accuracy	Precision	Recall	F1-score	MCC	Balanced-acc
CACAO-mean	**98.40 %**	**98.40 %**	**98.40 %**	**98.40 %**	**96.80 %**	**98.39 %**
CACAO-STD	±0.5 %	±0.5 %	±0.5 %	±0.5 %	±1.0%	±0.5%
LR	93.14 %	93.14 %	93.15 %	93.146 %	86.83 %	93.17 %
Ada	85.04 %	85.09 %	85.04 %	85.04 %	70.12 %	85.08 %
RF	95.80 %	95.82 %	95.80 %	95.80 %	91.61 %	95.82 %
KNN	81.66 %	81.73 %	81.66 %	81.66 %	63.38 %	81.70 %
MLP	48.46 %	23.87 %	48.46 %	31.64 %	00.00 %	00.50 %
DT	91.50 %	94.87 %	91.62 %	91.5’ %	83.11 %	91.57 %
NB	67.52 %	67.55 %	67.52 %	67.53 %	35.03 %	67.52 %

Table 7.6 CACAO-Net Shadowed targets images classification testing results with comparisons using tuned ML models.

Algorithm	Accuracy	Precision	Recall	F1-score	MCC	Balanced-acc
CACAO-mean	**99.41 %**	**99.42 %**	**99.41 %**	**99.41 %**	**98.83 %**	**99.38 %**
CACAO-STD	±0.4 %	±0.5 %	±0.5 %	±0.5 %	±1.0 %	±0.5%
LR	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**
Ada	96.57 %	96.57 %	95.57 %	95.57 %	93.14 %	95.58 %
RF	96.57 %	96.69 %	96.57 %	96.57 %	93.25 %	96.64 %
KNN	96.08 %	96.24 %	96.08 %	96.08 %	92.32 %	96.16 %
MLP	51.47 %	26.49 %	51.47 %	34.98 %	00.00 %	00.50 %
DT	95.10 %	95.16 %	95.10 %	95.10 %	90.25 %	95.04 %
NB	55.88 %	55.88 %	55.88 %	50.13 %	13.28 %	54.89 %

7.7.4 CACAO explainability results

Algorithm (7.1) was used to convert all images into TS in terms of statistical measurements of the distances (without Angles) to be applied in the CACAO-Net. The full version of Algorithm 3.1 on the other hand is employed to provide explication labeling based on the Shapley features (which were extracted from the Python SHAP library) and landmark distribution through Algorithms (7.2) and (7.31), respectively. This is clearly demonstrated in Figure 7.8. A sample of the global image explainability results as performed using LLM running in Rulex using the said explicative labels as shown in Tables 7.10 and 7.11.

Table 7.7 CACAO-Net Satellite images classification testing results with comparisons using tuned ML models.

Algorithm	Accuracy	Precision	Recall	F1-score	MCC	Balanced-acc
CACAO-mean	**97.28 %**	**97.32 %**	**97.28 %**	**97.28 %**	**96.34 %**	**97.43 %**
CACAO-STD	±0.5 %	±0.5 %	±0.5 %	±0.5 %	±0.7 %	±0.5%
LR	72.63 %	76.55 %	72.63 %	72.47 %	65.06 %	73.91 %
Ada	55.11 %	47.15 %	55.11 %	45.31 %	46.63 %	58.08 %
RF	91.61 %	91.67 %	91.61 %	91.60%	88.69 %	91.44 %
KNN	90.88 %	91.25 %	90.88 %	90.89 %	878.81 %	91.76 %
MLP	73.36 %	82.43 %	73.36 %	69.13 %	68.62	75.65 %
DT	86.13%	88.32 %	86.13 %	86.16 %	81.33 %	86.00 %
NB	79.93 %	81.46 %	79.93 %	79.39 %	73.90 %	81.07 %

Table 7.8 CACAO-Net Skin cancer images classification testing results with comparisons using tuned ML models.

Algorithm	Accuracy	Precision	Recall	F1-score	MCC	Balanced-acc
CACAO-mean	**96.84 %**	**96.85 %**	**96.84 %**	**96.84 %**	**93.68 %**	**96.83 %**
CACAO-STD	±0.4 %	±0.4 %	±0.4 %	±0.4 %	±0.8 %	±0.4%
LR	92.13 %	92.17 %	92.13 %	92.13 %	84.30 %	92.14 %
Ada	93.52 %	93.52 %	93.52 %	93.52 %	87.04 %	93.52 %
RF	94.54 %	94.46 %	94.54 %	94.54 %	89.09 %	94.54 %
KNN	92.59 %	93.17 %	92.59 %	92.57 %	85.77 %	92.65 %
MLP	56.76 %	74.72 %	56.76 %	46.82 %	25.18 %	56.56 %
DT	89.07 %	89.09 %	89.07 %	89.07 %	78.16 %	89.06 %
NB	73.52 %	75.54 %	73.52 %	73.03 %	49.09 %	73.64 %

In Table 7.10, a subset of 8 conditions taken from a single rule are shown which contribute to correct classification as a Female photo. Rule 1.1 (Condition 1.1) relates to the distance between the chin at the 4^{th} position till the 5^{th} position on the left eye, wherein case the measurement is greater than a defined threshold infers to correct detection. Rules 1.2 and 1.3 present the same concept where the landmarks distances relate to the left eyebrow, left eye and bottom lip using the same principle.

Rules 1.4 and 1.8 in Table 7.10, correspond to the same set of conditions, however, they correspond to TS statistical distances. In rule 1.4, the minimum distance of hits in the Sobel filter is greater than 52 for 112x112 image at an angle of 183 degrees infers the image is that

Table 7.9 CACAO-Net Weather images classification testing results with comparisons using tuned ML models.

Algorithm	**Accuracy**	**Precision**	**Recall**	**F1-score**	**MCC**	**Balanced-acc**
CACAO-mean	**91.20 %**	**91.42 %**	**91.20 %**	**91.20 %**	**88.16 %**	**90.73 %**
CACAO-STD	±2.0 %	±2.0 %	±2.0 %	±2.0 %	±2.6 %	±1.9%
LR	81.04 %	81.69 %	81.04 %	81.01 %	74.49 %	79.26 %
Ada	72.04 %	77.86 %	72.04 %	73.11 %	63.67 %	70.51 %
RF	83.41 %	83.75 %	83.41 %	83.51 %	77.61 %	82.94 %
KNN	74.41 %	79.48 %	74.41 %	75.14 %	66.78 %	74.07 %
MLP	80.94 %	82.13 %	80.09 %	79.54 %	73.49 %	77.15 %
DT	69.19 %	70.61 %	69.19 %	69.73 %	58.44 %	67.13 %
NB	73.46 %	73.44 %	74.46%	73.1 %	64.17 %	72.64 %

Table 7.10 A sample of image explainability results taken for a single rule having a set of conditions as performed using LLM running in Rulex relating to the Female class.

Rule #	**Rules for classification as a Female**
1.1	*'chin4-S-left-eye5'* $\geq$ *8.3*
1.2	*'left-eyebrow5-S-left-eye5'* $\geq$ *37.1*
1.3	*'bottom-lip5-S-bottom-lip9'* $\geq$ *0.5*
1.4	*27.44* $\leq$ *'TS-STD-est-43-R-0-90'* $\leq$ *49.32*
1.5	*'TS-min-est-183-R-180-270'* $\geq$ *52.64*
1.6	*38.4* $\leq$ *'TS-min-est-193-R-180-270'* $\leq$ *55.93*
1.7	*'Ang-STD-est-6-R-0-90'* $\geq$ *-45.55*
1.8	*-83.82*$\leq$ *'Ang-var-est-46-R-0-90'* $\leq$ *-68.72*

of a Female. In Rule 1.8, in case the variance of the angel of hits at an angle of 46 varies between -83.82 and -68.72 means that the photo is more likely to be for a Female.

In Table 7.11, a subset of 3 sets conditions belonging to 3 rules are presented. Rule 1.1 (Condition 1.1) infers that in case the distance from the 5^{th} position on the chin till the 6^{th} position on the top lip is greater than 44.48 the photo is more likely to be that of a man. From rule 3.2, we can conclude that in case the mean distance at an angel of 76 degree that lies between the centroid till each hit is less than 10.53 means that the current photo is that of a male.

Table 7.11 A sample of image explainability results taken for three rules having three sets of conditions as performed using LLM running in Rulex relating to the Male class.

Rule #	**Rules for classification as a Male**
1.1	*'chin5-S-top-lip6' $\geq$ 44.48*
1.2	*'bottom-lip3-S-bottom-lip8' $\geq$ 13.38*
1.3	*'Ang-STD-est-156-R-90-180' $\leq$ -29.54*
1.4	*'Ang-max-est-246-R-180-270' $\leq$ -51.27*
2.1	*'chin11-S-bottom-lip12' $\geq$ 6.04*
2.2	*27.83 < 'chin13-S-left-eye2' $\geq$ 47.1*
3.1	*124.23 $\leq$ 'TS-median-est-316-R-270-360' $\leq$ 185.8*
3.2	*'TS-MEAN-est-76-R-0-90' $\leq$ 10.53*

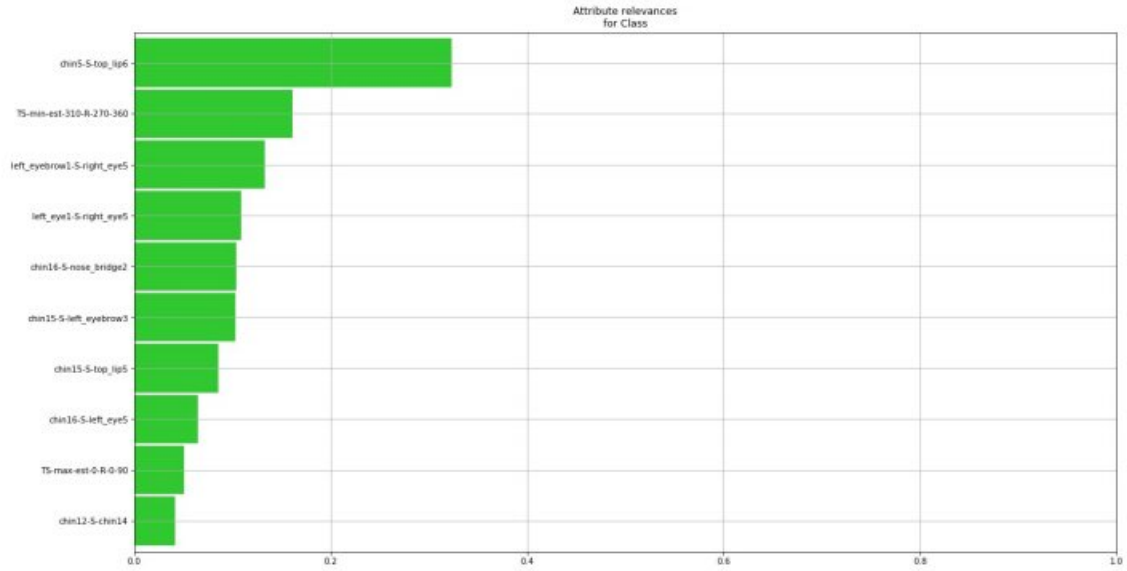

Fig. 7.15 Feature ranking for both Female (0) and Male (1) classes of the gender classification dataset in absolute format.

Figure 7.15 presents the absolute feature ranking of the LLM used as the explainability engine within the CACAO-X framework. The most prominent metric corresponding to the first feature in this figure which relates to distance between the 5^{th} position of the chin and the 5^{th} position of the top lip which we may conclude for the first time in the literature corresponds to the most contributing factor for gender classification when facial landmarks are taken into account.

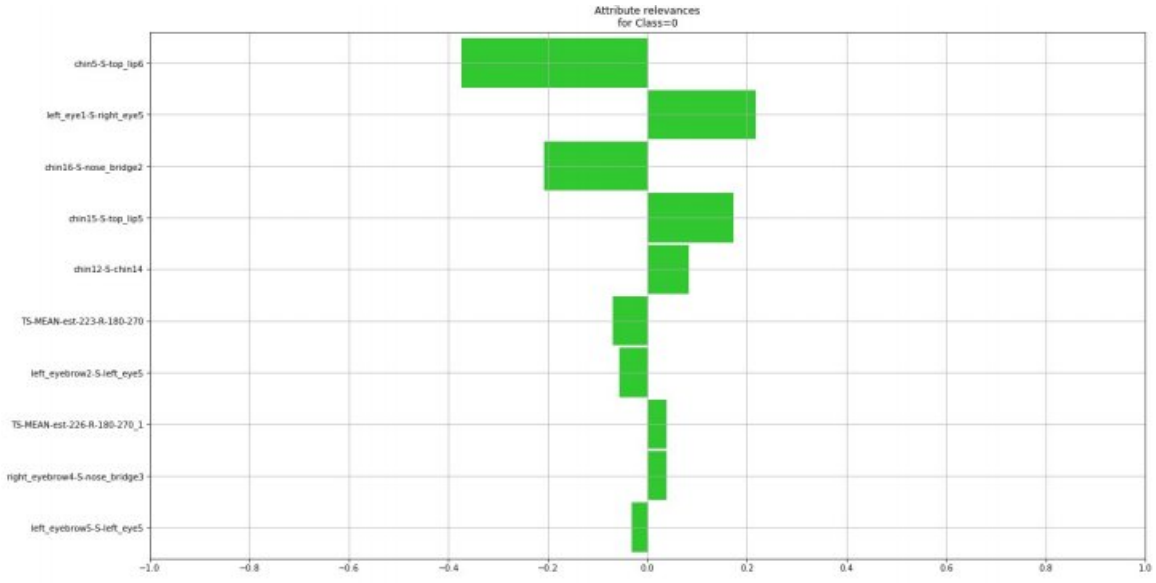

Fig. 7.16 Feature ranking for the Female (0) class from the gender classification dataset.

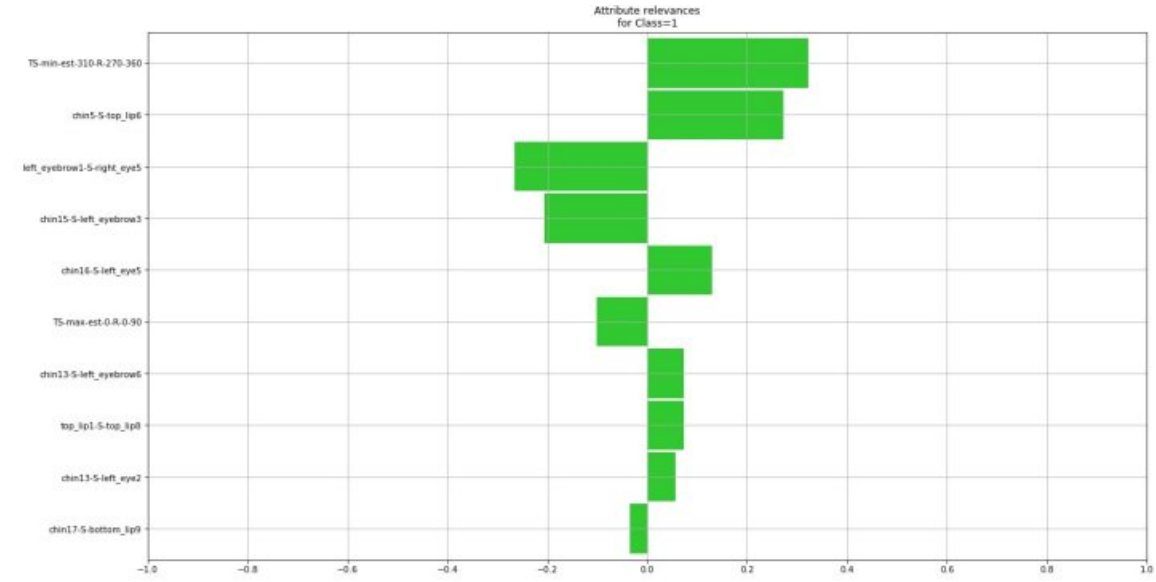

Fig. 7.17 Feature ranking for the Male (1) class from the gender classification dataset.

In Figures 7.16 and 7.17, it is shown that the top feature from Fig. 14 play a more prominent role in the classification Females than Males since Fig. 15 is mainly concerned with the Female class and Fig. 16 deal with the Male class. The same concept may also be applied to all of the remaining features when comparing the absolute ranking with the label-sensitive ranking as discussed.

7.8 Chapter Conclusion

This thesis presents a unified model for global image expandability and general-purpose image inference having a real-world FE approach for deployment on resource-constrained devices. The explainability is implemented with the use landmark-detectable applications and polar TS information which provides explicative conditions through which a novel XAI architecture is employed. This model is implemented using the described methods which are derived by extracting features from the Shapley output and finally performing the explainability stage using LLM. Actually, the presented CACAO-X framework is robust enough that it can be implemented for traditional ML classification with little tuning of the hyperparameters where CACAO-Net takes an image and its TS transform as input. Moreover, the CACAO-X Image-to-TS conversion algorithm was used during ML Classification in a low-level C implementation which was applied on the Raspberry Pi 4B before inference in real-time. Forecast results prove that after applying various ML experiments from diverse datasets the CACAO-Net achieved high accuracy in most applications while remaining increasingly competitive in each prediction presented. K-Fold cross-validation was applied on an MSI laptop which used the validation part of a Train/Validate/Test split. Then, the remaining testing data is used to perform inference remotely on the Raspberry Pi through a TF-Lite classifier where high accuracy was achieved and the overall inference Time/Task with FE included validates the portability and deployment of the CACAO-Net Classifier. The proposed function ensures that the training halts at a stable and that all results show competitive performance in terms accuracy as well as inference time on the edge. Additionally, CACAO-Net was compared with standard ML models from the Pycaret AutoML library and using the same data split content, where CACAO-Net model 1 outperformed its competition in virtually all five datasets and in 97% of recorded performance metrics.

After applying K-fold validation on all five datasets, the gender dataset was forecast with an accuracy of 98.4% and tested with an accuracy of 91.5% with an impressive inference time on the Raspberry Pi 4 with the pre-processing being taken into account of the Satellite dataset resulted with an average accuracy of 97.31% and a testing accuracy of 94.1%. Regarding the Shadows dataset, the average accuracy was an impressive 99.41% and the testing accuracy reached 98.9%. Regarding the skin cancer and weather datasets, the averaged accuracies were 96.8% and 91.21% respectively, and for the testing, 95.5% and 91.9% respectively. In every case the inference on the Raspberry Pi (with FE included) outperformed the Raspberry Pi 3 without the real-time FE.

Regarding the K-fold setups, the validation and loss curves for training as controlled by Algorithm 7.4 stopped at stable state based on the oscillation around the most recent mean either on the first of second set of thresholds or until the final epoch.

When comparing CACAO-X with Pycaret, CACAO-Net outperformed the AutoML library, which produces tuned models that outperform published works substantially. CACAO-Net outperformed Pycaret in every metric 97% of the time in accuracy, precision, recall, F1-score, MCC, and balanced accuracy. In case of the shadow dataset, Pycaret outperformed the original source in accuracy with a range ranging between 8.5% and 18.5% demonstrating the reliability of comparing CACAO-Net with this Python library.

Although, CACAO-X does not contribute to true general-purpose image explainability which at this point still abstract in the literature and since global image-explainability requires proper alignment and landmark detection, this thesis remains the first to introduce a semi-domain-specific XAI model with human understandable rules since it utilizes explicative labeling to interpret the global conditions behind image classification.

Chapter 8

VAMPIRE: Vectorized Automated ML Pre-processing and Post-processing Algorithms for the Internet of Things

8.1 Chapter Abstract

ML techniques aim to mimic the human ability to automatically learn how to perform tasks through training examples. They have proven capable of tasks such as prediction, learning and adaptation based on experience and can be used in virtually any scientific application, ranging from biomedical, robotic, to business decision applications, and others. However, the lack of domain knowledge for a particular application can make FE ineffective or even unattainable. Furthermore, even in the presence of pre-processed datasets, the iterative process of optimizing ML parameters, which do not translate from one domain to another, maybe difficult for inexperienced practitioners. To address these issues, we present in this thesis a Vectorized Automated ML Pre-processing and Post-processing Framework, approximately named (*VAMPIRE*), which implements FE algorithms capable of converting large TS recordings into datasets. Also, it introduces a new concept, the *Activation Engine*, which is attached to the output of a MLP and extracts the optimal threshold to apply binary classification. Moreover, a tree-based algorithm is used to achieve multiclass classification using the *Activation Engine*. Furthermore, the IoT gives rise to new applications such as remote sensing and communications, so consequently applying ML to improve operation accuracy, latency, and reliability is beneficial in such systems. Therefore, all classifications in this project were performed on the edge in order to reach high accuracy with limited resources. Moreover, forecasts were applied on three unrelated biomedical datasets, and on

two other pre-processed urban and activity detection datasets. Features were extracted when required, and training and testing were performed on the Raspberry Pi remotely, where high accuracy and inference speed were achieved in every experiment. Additionally, the board remained competitive in terms of power consumption when compared with a laptop which was optimized using a Graphical Processing Unit (GPU).

8.2 Introduction

ML has become a key technique that is used in many modern applications in many fields, such as business, engineering, and healthcare. It is considered a form of AI, and because it possesses the ability to learn, adapt, and remember, it proved superior to classical hard-coded programming methods, which are application-specific in nature. Many ML algorithms can make predictions in multiple fields while generalizing and have a relatively high accuracy regarding a specific application. However, a ML process requires a workflow that usually consists of data collection, FE, setting parameters, and applying the forecast on both training and test-sets. This process is iterative and, relying upon the application, some phases may require more tuning than others, depending on whether accuracy is reduced due to high bias or high variance.

ML algorithms usually require that the practitioner possesses domain knowledge in the field associated with the dataset that he is using, and most notably when FE is required. FE is the process of conversion of data into a better format to use in ML setups. However, currently, many pre-processed datasets are available, where ML algorithm can be applied directly to the data without any domain knowledge. This can be the case in many medical, physical, and business applications.

However, even after acquiring pre-processed data, the ML algorithm hyper-parameters still need to be tuned to optimize forecast accuracy. Moreover, this parameter tuning process can require a notable effort and sufficient experience in ML in general, and even ML domain experience in that specific application. This is dominantly the case, since hyper-parameter tuning setups does not translate well from one domain to another.

In this thesis, we present *VAMPIRE*, a vectorized automated ML pre-processing and post-processing framework. It consists of novel pre-processing and post-processing algorithms that we developed in the Python programming language. The first facilitates the FE phase in case the user is dealing with TS data, where the algorithm can be applied to fully annotated and semi-annotated datasets. Furthermore, a post-processing algorithm has been developed

that implements *Activation Engine*s, which are applied to the outputs of a MLP and extracts the optimal threshold relevant to the test-set based on training accuracy.

The *VAMPIRE* framework achieves robust performance, since it can be applied to multiple types of TS for FE and can be implemented to all applications performed using MLP through the *Activation Engine* concept. Also, the main computational blocks have been implemented using Python Numpy vectors to considerably increase the speed of FE, dataset generation, and post-processing substantially. Moreover, a subset of the extracted features from the TS has been applied using the Rulex software in an edge computing arrangement. Rulex (Muselli (2005)) is a general-purpose ML platform that operates through a GUI; It can operate in a client/server setup where the ML forecasts are applied on the general-purpose Raspberry Pi IoT device (Daher et al. (2021)). Rulex applies Switching NN (Muselli (2005)) and Positive Boolean Function Reconstruction (Muselli and Ferrari (2009)) to implement XAI which provide a white-box learning approach to ML.

Multiple classification forecasts were applied using various datasets taken from various fields in a mesh setup, where some datasets were pre-processed using the algorithms developed in this thesis, and others were already pre-processed using different techniques. Also, for the forecasts, Rulex was applied to a subset of the applications, while MLP with and without the new post-processing block was applied to the remaining datasets. Additionally, some forecasts were performed using just one of the ML setups. Also for four of the used datasets, a comparison with the literature is outlined and discussed since there exists viable related work suitable for comparison. Moreover, a complete description is provided in the experimental part of this project. All ML training and testing were applied in an edge computing setup with limited resources by applying training and testing outside a cloud server. Therefore, ML workflows were either applied using Rulex running on the Raspberry Pi and in a Client/Server arrangement, or by running a MLP remotely on the Raspberry Pi with *VAMPIRE*'s *Activation Engine* placed at its output. However, since the *Activation Engine* is a basic binary classification block, a multi-class tree-based technique which was developed in Daher et al. (2020b) was incorporated into the *VAMPIRE* Framework to implement multi-class classification. Furthermore, the inference time of the framework was compared to other edge setups and so was its power consumption with a laptop having an onboard GPU.

This chapter is organized as follows: Section 2 reviews the literature while discussing various automated ML techniques, related pre-processing and post-processing algorithms, and introduces the Rulex (Muselli (2012)) and Raspberry Pi (Daher et al. (2021)) platforms used in this project. The new *VAMPIRE* pre-processing algorithms are presented in detail

in section 3 and *VAMPIRE*'s *Activation Engine* block is described in section 4. Section 5 presents the data sources used for FE and classification to test the *VAMPIRE* framework's robustness. All experimental results achieved using the data sources and the performance comparisons are provided in section 6. Also, we draw a conclusion in section 7, and finally an appendix explaining version two of the pre-processing algorithm in detail concludes the chapter.

8.3 Chapter Related Work

FE or pre-processing is used to convert raw data into a more convenient format to apply various ML algorithms. On the other hand, post-processing is the act of applying an optimization block at the output of a ML workflow to improve accuracy.

8.3.1 Feature Extraction and pre-processing of TS

A certain dataset in its available format may not be directly applicable to a ML algorithm. Therefore, FE is usually applied to TS data, and most notably biomedical datasets.

In Fister et al. (2018) authors propose a Nature-Inspired Differential-Evolution for feature selection which was applied as a pre-processing method using the LR algorithm. The algorithm selects features based on the accuracy threshold applied taken from the output of LR.

A web platform for biomedical TS pre-processing (Jovic et al. (2017)) was applied to ECG signals for cases of myocardial ischemia, atrial fibrillation, and congestive heart failure. Biomedical signals are usually noisy and so filtering is an important issue in TS pre-processing. In Venkatesan et al. (2018) an ECG signal is pre-processed using an adaptive filter to remove noise before applying discrete wavelet transform to extract features for classification using SVM. In Kalayci and Ozdamar (1995) authors describe the process of applying wavelet transform to extract features from an EEG signal for classification using MLP.

In Khalid et al. (2014) authors review FE and feature selection in ML techniques. They describe how FE is used to reduce the dimensionality of a dataset through the transformation of the feature space, and also discuss how feature selection reduces the dimensionality as it points out the subset of features that possess the most significant effect on forecast accuracy.

A FE Python pre-processing Library based on nature-inspired optimization algorithms has been developed in Karakatič (2020). The EvoPreprocess library is compatible with the ML Scikit-Learn Library and performs well compared to other frameworks.

A TS is a waveform that corresponds to physical, biological, or business data, that changes over time and is found in various ML applications. In Wang et al. (2013), a Bag-of-Words representation for biomedical TS is presented. The methods presented treat the TS as text documents and extract segments as words. In that paper, the Bag-of-Word's methods were applied on two ECG datasets taken from Moody and Mark (2001) and an EEG dataset from Goldberger et al. (2000). An ensemble learning approach is also applied to biomedical datasets in Jin and Dong (2016) which use the Chinese cardiovascular disease database (Zhang et al. (2010)). Shortfuse is presented in Fiterau et al. (2017), which is a biomedical TS FE method that implements LSTM to outperform competing models by 3%, in terms of accuracy.

In the case of biomedical applications, ECG, EEG, and PPG are the most common measurements. ECG consists of measuring the electrical activity of the heart, PPG consists of the measurement of blood pressure and heart rate using light emitting diode, and an EEG provides the electrical signals taken from the scalp and usually have a much wider frequency spectrum than ECG and PPG signals. These three signal categories are used in this thesis as data sources for FE using the *VAMPIRE* framework.

8.3.2 Automated ML and post-processing

A ML workflow includes also a hyper-parameter tuning, such as setting class-weights and feature-weights, which are used as parameters in a loss function to optimize classification accuracy regarding the training set. In some applications, expert knowledge may not be available to tune the ML parameters, and so, automated ML techniques and software packages have been developed to facilitate the process of implementing forecasts. However, this can be a time-consuming process and requires much more processing power compared to classical workflows that are performed by experts in the field.

Post-processing is the process of applying a cognitive block at the output of a ML algorithm such as MLP or LR. This is applied for each sample and may consist of applying multiple ML forecasts in parallel before combining all the outputs to optimize the overall accuracy.

In Rasp and Lerch (2018) authors apply ensemble learning using MLP on weather forecasts in different configurations. Ensemble learning is a technique that applies multiple

ML forecasts in parallel and on the same dataset. Then, the outputs are brought together, and an optimal output is selected using a voting scheme.

Guidelines are provided by authors in Tanwani et al. (2009) to select the best ML scheme in the case of biomedical application. They have performed forecasts on 31 datasets and applied various ML algorithms to select the best configuration.

Another approach to ML automation is Meta-learning, which consists of iterating not just parameter tuning but also ML algorithms in the quest for optimization. This can be perceived as an optimization problem per algorithm and a collective optimization problem which attempts to iterate between different ML techniques. According to this approach, the Auto-WEKA package (Thornton et al. (2013)) has been presented as a modification of the original WEKA workbench (Hall et al. (2009)), conceived as a toolset for data analysis and predictive modeling. Auto-WEKA applies Bayesian optimization to automatically select the algorithm to be used, along with its tuned parameters. It is a tool dedicated to non-experts to apply ML forecasts and to achieve good performance. In Feurer et al. (2020) authors present the AUTO-SKLEARN framework which is a Python implementation based on the Scikit-Learn Library. The framework applies ensemble learning at the output of a Meta-learning configuration to automate parameter tuning while choosing the most suitable ML algorithm.

A situation where expert knowledge is not always available is presented in Zhang et al. (2020) which consists of the prediction of the performance of a tunnel boring machine. So, a Bayesian optimization scheme is applied for hyper-parameter tuning and algorithm selection, and a neural architecture search for MLP optimization.

In Guo et al. (2017) authors claim that modern MLP is poorly calibrated since increasing the depth improves accuracy, however, may affect the calibration negatively. Furthermore, a domain-specific ML framework dedicated to predicting the properties of inorganic materials has been presented in Ward et al. (2016).

8.3.3 Platforms used on the edge

IoT encompasses many fields of application which demand the monitoring and management of power, bandwidth, and reliability. For example, authors in Al-Khafajiy et al. (2018) propose an IoT framework for resource management with the goal of optimal task offloading, where the framework is intended for healthcare systems. Power consumption during ML forecasts on edge is a critical issue due to the limited power availability for IoT nodes. Therefore, power consumption optimization on the edge using ML is investigated in Cecilia et al. (2020); Lapegna et al. (2021); Novac et al. (2021). Also, in Kanawaday and Sane

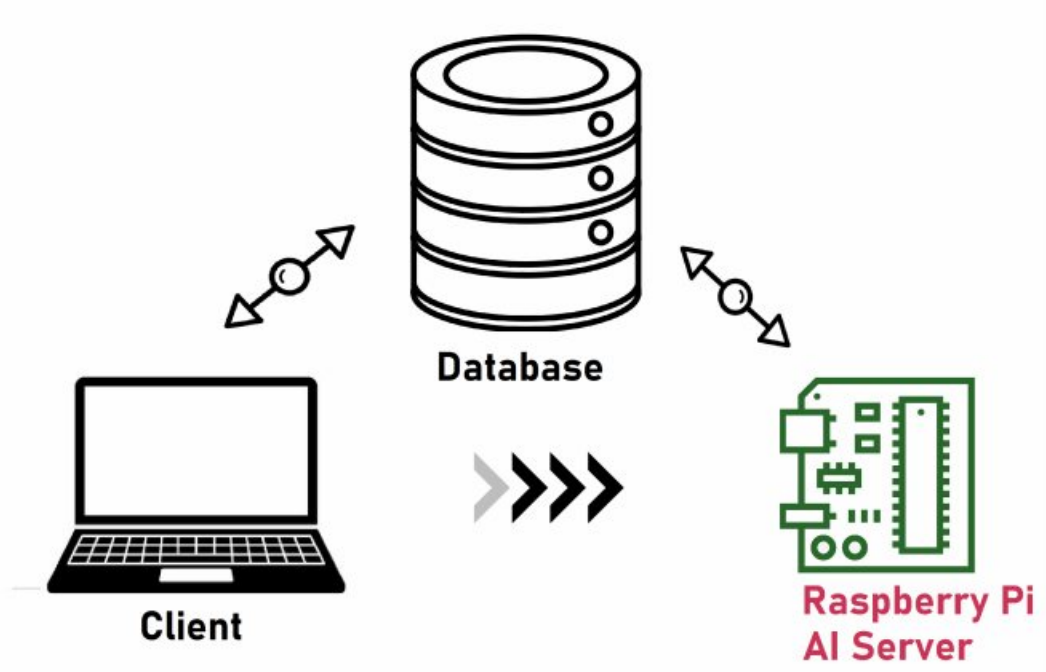

Fig. 8.1 ML forecasts applied using Rulex software over the public network in a client/server setup.

(2017) ML is applied on TS data from IoT sensors in order to predict failure in a slitting machine. Additionally, in Canedo and Skjellum (2016); Hodo et al. (2016) MLPs are used to improve security in IoT networks.

Therefore, with the vast scope of ML applications in IoT along with the existing benefits of ML solutions, we've developed the present framework to perform ML pre-processing and prediction using limited resources. Consequently, the hardware platform chosen in this thesis is the multi-purpose and widely popular Raspberry Pi (Vujović and Maksimović (2014)). The Raspberry Pi is used as an edge computing node where the act of storing and making a cognitive decision on a local node rather than making computation and exchanging big data with a cloud server (Zhang et al. (2018)).

The Raspberry Pi is ideal for edge computing applications since it possesses adequate processing power and is able to manage enough data for most ML applications. As for the software part, The ML platform used is Rulex (Muselli (2012)) which is a ML package directed towards non-domain-experts and which has been ported to the Raspberry Pi as reported in Daher et al. (2021). The user does not need to write any code to carry on ML forecasts thanks to an easy-to-use GUI. It allows to easily manipulate the data, apply one among many ML algorithms, and visualize the output. Following this approach, a subset of the forecasts applied in this project were performed using Rulex in an edge computing

environment. The Client/Server arrangement that has been implemented with Rulex is presented in Figure 8.1, whereas shown, the Rulex engine runs on the Raspberry Pi and is operated remotely while relying on a database server as a common storage point. Also, forecasts were performed using *VAMPIRE*'s post-processing algorithm, where an SSH connection (Cecilia et al. (2020)) across the public network was established through the interface of the VSCode editor (without a third-party database), which was used for remote development.

In case of using Rulex as classifier, the client consists of an HP laptop with an Intel code-i7 1.8 GHz processor having 16GB of RAM running Windows 10 and without a GPU. The AI server is a Raspberry Pi 3B+ model with an ARM Cortex-A53 1.4GHz processor and running Raspbian as its operating system. The database used is a PostgreSQL 12 docker deployed server operated through a dedicated ODBC driver while running on Azure Cloud. The Client consists of a graphical interface developed in Python 2.7, and the AI Engine is written in C/Cpp vectors for optimal performance. The connection between client and server is an encrypted SSH connection over the public or private network. In case of operating *VAMPIRE* remotely, again, an SSH connection locally or through the local or public network is applied where the code is written in Python 3.8 and the ML models and post-processing are developed using Tensorflow and Numpy libraries respectively. As for the dataset generation and FE algorithms, they are also written in the Python 3.8 Numpy library and are performed outside the Raspberry Pi on an MSI laptop with an Intel 7-10750H Central Processing Unit (CPU) 2.60GHz processor with 16GB of RAM and a 6GB 2060 Nvidia GPU and on the same HP laptop used as a client. Consequently, the datasets were generated simultaneously on both computers with roughly the same time and efficiency.

8.3.4 Motivation for ML edge computing frameworks and automated dataset generation

Applying ML on edge devices and under performance constraints requires the development of light data processing programs which are not much of a burden under limited resources. However, specification restrictions are not the only issue faced in edge paradigms. Usually, training takes place on a remote cloud server (Murshed et al. (2021)) and so are the online training updates due to continuously changing data from the real world. The former demands bandwidth resources while compute-heavy ML tasks can cause the edge node to lag which results in undesired downtime. Consequently, both of these issues are common design goals in an industrial ML system deploy-able on the edge.

Therefore in Sudharsan et al. (2020a) Edge2Train is presented, which is a an edge computing setup that allows for online training on the edge with automatic dataset generation capability. The framework uses an instrumentation feedback system which monitors changes in the real world in order to update and validate the model through training on the edge. Even so, this arrangement does not attempt to tune the parameters of ML model but rather takes them as inputs from an external application. Furthermore, currently this setup only deals with binary classification applications. Also in Sudharsan et al. (2020b) authors implement a pipeline that executes CNN on edge devices. The pipeline aims to reduce the size and structure of the network and the volume of data such that they can be deployed on a unit with limited resources, while preserving accuracy. However, this method trains the model outside an edge node before deployment and does not include FE which is missing from the framework.

For final or industrial implementation of edge-based AI systems, one fundamental characteristic is automated feature generation regardless if training is applied in one shot or whether online updates are applied. TS forecasting setups on the other hand usually include FE in order to reduce the dimensionality of data and since it may improve prediction accuracy. Therefore, automating this process is desirable considering the large amounts of data that can be collected by biomedical edge devices and from multiple patients. Authors in Meisenbacher et al. (2022) review the automation of TS ML processes which encompass: Pre-processing, feature engineering, hyperparameter optimization, model selection and ensembling. They reveal that the majority of publications only cover three out of the five pipelines mentioned. Also, based on their report, ensembling and parameter optimization are not suitable to be applied on the edge due to high computation requirements and since multiple complete training runs are required twice for each block. EEG automated FE is reported in Martone et al. (2019) where a GUI named Training Builder was developed which generates a dataset automatically based on predefined computations and relies on a windowing technique to split an unbounded TS into smaller finite sets. However, the length of sets and the overlap times needs to be defined manually. Consequently, a practitioner may need to experiment with these time values since they affect the forecast accuracy which limits the automation attribute.

The *VAMPIRE* framework described in this thesis avoids hyperparameter tuning and AutoML which are increasingly compute-expensive but relies on a post-processing block that can run to a finite number of times where the runtime is adjustable for a loss in accuracy, where this loss is usually very small or improbable. Furthermore, the framework adds a multiclass classification feature using the tree-based method which is described in section

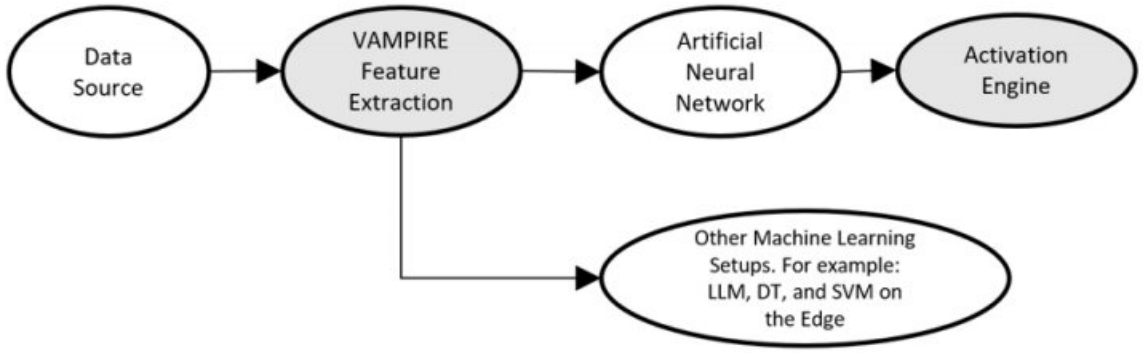

Fig. 8.2 Workflow of the *VAMPIRE* framework with various setups.

6. As for the automated dataset generation part of the *VAMPIRE* framework, the process is completely automated since in the case of dealing with streams of data, the algorithm will set a variable length for each finite set which in continuously checked and updated without any predefined parameters and variables. The FE algorithms provided in this framework may be implemented in an online learning approach due to their automated nature since they do not require parameters from an operator or from any additional programs. Also, the same statement can be declared regarding the post-processing technique which is adopted.

8.4 Novel *VAMPIRE* pre-processing algorithms

The thesis presents a new FE method which takes a TS as an input, and can automatically generate a dataset in Comma Seperated Values (CSV) format that can be processed by any ML algorithm. This process is automated and may only require the user to apply some filtering technique to the input signal to reduce noise. Furthermore, the key computational blocks in the code were implemented using Python Numpy to increase processing speed by a considerable factor.

The pre-processing algorithm is applied in two arrangements. The first operates directly by applying multiple algorithms in case the signals are annotated by experts in the field. The second version has been developed to deal with poorly annotated datasets.

Preliminary operations consist in subdividing the whole TS in portions, each one with start and end clearly defined, and to annotate each portion according to a classification provided by an expert in the field. A semi-annotated dataset consists of a TS that is classified over a longer period of time, meaning larger portion size, and may be classified by non-domain-experts with limited accuracy, such as a patient annotating her or his own state. The *VAMPIRE* pre-processing algorithms are implemented in two setups, being capable of extracting datasets

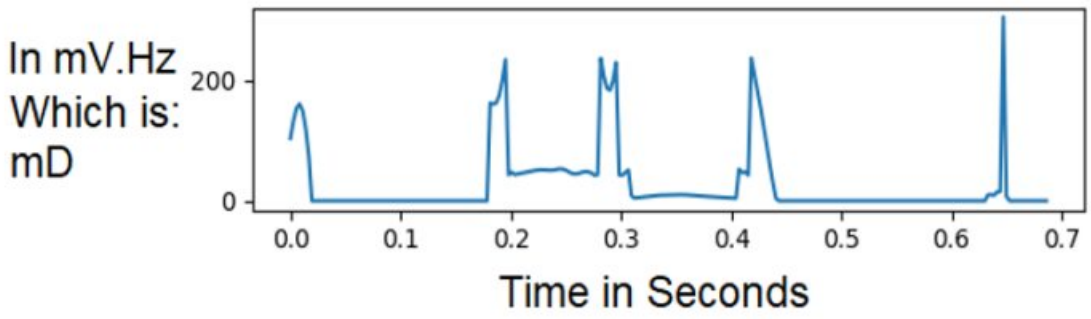

Fig. 8.3 A waveform used for FE after being converted to the D-Domain. This is the result of applying the FE method describes in section 3, which correlates between time-varying signals and their frequency response.

without any excessive intervention by the ML practitioner. An illustration of the overall workflow of the *VAMPIRE* framework is provided in Figure 8.2, where the shaded areas represent to parts where the *VAMPIRE* framework algorithms are being implemented.

In case of TS data, the *VAMPIRE* pre-processing algorithm converts the recordings into datasets, which can be applied to a MLP with a post-processing block (or any other ML setup). Furthermore, a MLP with an *Activation Engine* at the output may be applied to any pre-processed datasets.

8.4.1 *VAMPIRE* pre-processing algorithm: *VAMPIRE FE1*

The novel FE algorithm *VAMPIRE FE1* applies FFT (Takahashi (2019)) to every record in the waveform recordings and stores frequency response information. The FFT input and the output signal are normalized on a scale of *0-N* where *N* is maximum value of the normalized signal. Then, the frequencies from the FFT, and the voltages from the original signal are correlated and multiplied, to form the novel V.Hz curve, which is a new unit of representation proposed in this thesis, where for every biomedical signal the curve is extracted and plotted versus the time axis as shown in Figure 8.3 The process of generating the curve presented in Figure 8.3, is illustrated in Figure 8.4. As shown, the voltages from the original TS are correlated with the maximum frequency value taken from the FFT output, in order to generate the V.Hz waveform. **Algorithm (8.1)** presents the coding steps needed to convert a time-varying voltage into the D-Domain in order to further process the signal and extract features. It consists of taking *FFT* of signal x and normalizing its voltages to the same range, and then multiplying each maximum frequency with the identified voltage to convert x into the V.Hz Domain, which we will refer to as the D-Domain for simplicity.

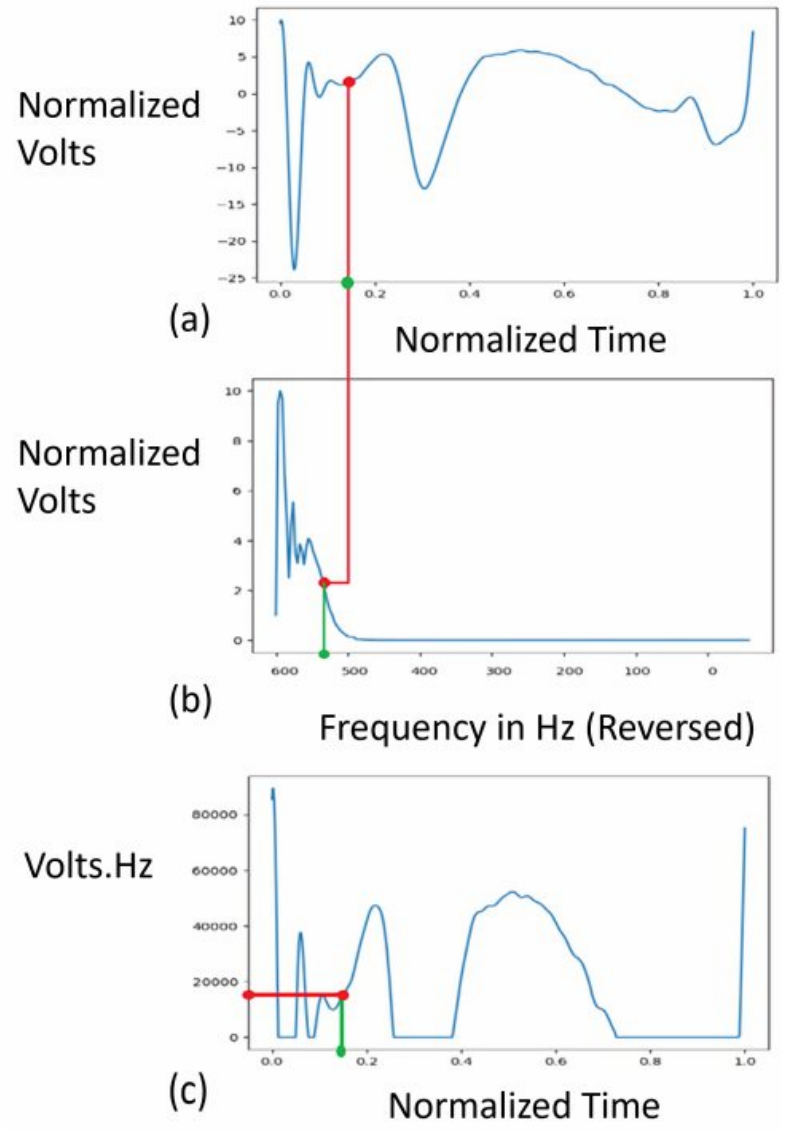

Fig. 8.4 (a) The original TS from the recording with normalized voltages. (b) The FFT output for the TS in (a). (c) After extracting the maximum frequency from (b) related to a particular voltage in (a), the normalized voltages and frequencies are multiplied generating the curve in (c).

After extracting the signal in the D-Domain which is a V.Hz curve, the spikes from this signal are detected and two variables *a1* and *b1* are computed. The *a1* variable represents the time that is taken from the first zero-crossing of the V.Hz signal till the peak of the same signal. Variable *b1* represents the time taken from the peak till the second crossing. Their difference *Diff1* and their summation *Sum1* are computed. *mean*, *STD*, *variance*, *median*, *range*, *maximum* and *minimum* values are derived for *a1*, *b1*, *Sum1*, and *Diff1*, and are saved

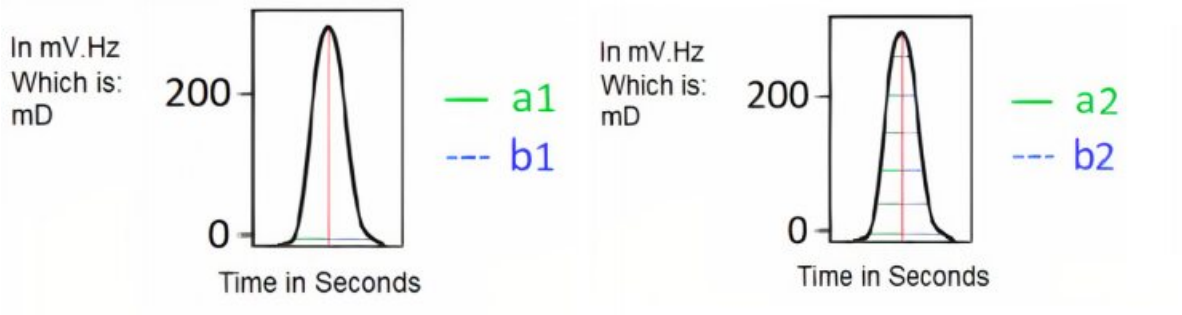

Fig. 8.5 (a) Base variable a1 and b1 used to derive the first set features. (b) Variable a2 and b2 used to derive second set of features. These variables are used to derive the features used in ML training, by implementing basic statistical operations.

Algorithm 8.1 Algorithm that converts a time-varying voltage signals into the D-Domain

Require: Waveform x indexed by i
Ensure: VHz which is the D-Domain representation of x

$\alpha 1 \Leftarrow lowerthreshold$
$\alpha 1 \Leftarrow upperthreshold$
$x \Leftarrow input\ waveform$
$x \Leftarrow \text{normalize}(x)$
for *i for every waveform in x from the dataset* **do**
    $FFT \Leftarrow \text{frequency response}(x[i])$
    $FFT \Leftarrow \text{normalize}(FFT)$
    $VHz \Leftarrow \text{empty array of size}(x[i])$
    for $j \leq size(x[i])$ **do**
        $temp \Leftarrow x[i,j]$
        for $k \leq \text{size}(FFT)$ **do**
            if $FFT[k] \geq \alpha 1 * temp$ **then**
                if $FFT[k] \leq \alpha 2 * temp$ **then**
                $Freq.\text{insert}(k)$
                end if
            end if
        end for
        if $\text{size}(Freq) \geq 0$ **then**
            $max \Leftarrow \max(Freq)$
            $VHz[i] \Leftarrow max * x[i,j]$
            $Freq \Leftarrow 0$
        end if
    end for
end for
Return VHz

Algorithm 8.2 Algorithm that extracts static base variables from the D-Domain signals

Require: D-Domain signals and VHzcross indexes
Ensure: Arrayss a1, b1, Sum1, Diff1, IntegVHz, PeakVhz used for FE
  for every *s crossings* in *VHz* **do**
    $VHza \Leftarrow VHz[s]$
    while $k \leq$ size(*VHzcross*) **do**
      for *j* from *VHzcross*[*s,k*] $\Rightarrow$ *VHzcross*[*s,k+1*] **do**
        if $VHza[j] == peak$ **then**
          ipeak[*s*].insert(*j*)
        end if
      end for
      peak $\Leftarrow$ max(*VHza*[*VHzcross*[*s,k*]:*VHzcross*[*s,k+1*]])
      PeakVHz[*s*].insert(*peak*)
      Area $\Leftarrow$ integral(*VHza*[*VHzcross*[*s,k*]:*VHzcross*[*s,k+1*]])
      IntegVHz[*s*].insert(*Area*)
      k=k+2
    end while
    while $d \leq$ size(*VHzcross*) **do**
      a1[*s*].insert(*ipeak*[*s,d/2*] - *VHzcross*[s,d])
      b1[*s*].insert(*VHzcross*[*s,d+1*] - *ipeak*[*s,d/2*])
      Diff1[*s*].insert(tail(*b1*[*s*])-tail(*a1*[*s*]))
      Sum1[*s*].insert(*VHzcross*[*s,d+1*] - *VHzcross*[*s,d*])
      d = d + 2
    end while
  end for
  Return $a1, b1, Sum1, Diff1$, *IntegVHz*, *PeakVHz*

as features for that record. Variables *a1* and *b1* are presented in Figure 8.5(a). Also, for every spike, the *peak*, as well as the *area* or *integral* values can be used to generate the same set of feature. **Algorithm (8.2)** describes the procedure that is used to extract *a1*, *b1*, *Sum1*, and *Diff1* by detecting the peak of each spike along with the zero-crossing pairs needed to calculate the base variables along with the *peak* and *area* values. The algorithm requires as input the V.Hz waveform along with the indexes determining the start and end of each spike.

Another class of variables is based on variables *a2* and *b2* presented in Figure 8.5 (b), which consist of the of sets of the averaged triangular function from zero-crossings to the maximum of the spike. Also, corresponding values for *Sum2* and *Diff2* are computed. Afterwards, statistical function of arrays *a2*, *b2*, *Sum2*, and *Diff2* are computed, where again, they consist of the *range*, *mean*, *variance*, *STD*, *median*, *maximum*, and *minimum* values. **Algorithm (8.3)** outlines the process used to append all measurements taken for every spike to arrays relating to the reported base variables, where it takes the D-Domain representation

Algorithm 8.3 Algorithm that extracts dynamic base variables from the D-Domain signals

Require: D-Domain signals and VHzcross indexes
Ensure: Base arrays a2, b2, Sum2, Diff2 used for FE

```
1: for every s crossings in VHz do
2:     VHza ⇐ VHz[s]
3:     for w till size(VHzcross) with step-size = 2 do
4:         h ⇐ w/2
5:         v0 ⇐ VHzcross[s,w]
6:         v1 ⇐ VHzcross[s,w+1]
7:         for n from vo till ipeak[s,h] do
8:             index1 ⇐ index(VHza[n])
9:             difference1 ⇐ (ipeak[s,h] - index1)
10:            a2[s,h].insert(difference1)
11:        end for
12:        for n1 from ipeak[s,h] till v1 do
13:            index2 ⇐ index(VHza[n1])
14:            difference2 ⇐ (index2 - ipeak[s,h])
15:            b2[s,h].insert(difference2)
16:            Sum2[s,h].insert(difference1+difference2)
17:            Diff2[s,h].insert(absolute(difference1-difference2))
18:        end for
19:    end for
20: end for
21: Return a2, b2, Sum2, Diff2
```

and the spike crossings as input, in order to generate the feature arrays that in turn are used to build the dataset in CSV format.

8.4.2 *VAMPIRE* pre-processing algorithm: *VAMPIRE FE2*

The goal of the second version of the pre-processing algorithm *VAMPIRE FE2* is to extract features from a semi-annotated dataset as in the case of the dataset. It allows to detect the status of TS in real-time, in case the classes are distributed over long periods of time such as the case in some biomedical applications. The classes in the PPG dataset are taken over an hour and are not period-specific such is the case with the ECG and EEG datasets. So, we could take a fixed period length distributed over an hour and classify the periods with their overall hourly class. However, a different approach is adopted, consisting in the detection of the uniformity of the positively rectified signal and to sense a re-occurrence in the voltages or peaks. This is performed while determining the window size that is continuously updated. So, child records that are encompassed by a parent record are assigned, but with different window

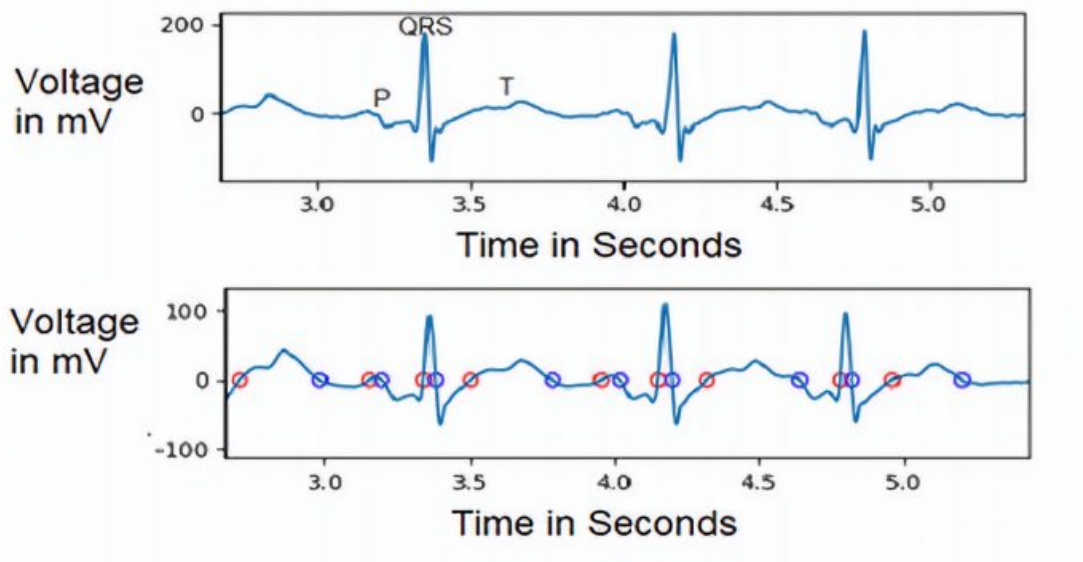

Fig. 8.6 ECG Waveform before and after passing through the novel moving average algorithm, which generates a more defined signal where it is easier to extract features in the case of a semi-annotated data source.

Algorithm 8.4 Modified moving average

Require: Waveform x
Ensure: avgs which is the moving average results from input x

```
avgs ⇐ x
step ⇐ step size
j ⇐ step
while j ≤ size(x) do
    k ⇐ j
    while k ≥ j − step do
        avgs[j] ⇐ avgs[j] + avgs[k − 1]
        k−−
    end while
    avgs[j] ⇐ avgs[j]/step
    j++
end while
Return avgs
```

sizes which are equal to the number of the child record's zero-crossing pairs (corresponding to the rectified signal). Therefore, to correctly rectify the input signal, a novel moving average algorithm has been developed which defines the TS signal as it can be easily detected as shown in **Algorithm (8.4)**.

Algorithm 8.5 *VAMPIRE* pre-processing algorithm for semi-annotated data

Require: Signal x
Ensure: Crosspx
$\beta 1 \Leftarrow starting\,window$
$\beta 2 \Leftarrow last\,window$
for *every M portions of x* **do**
    Add *N-to-P* and *P-to-N crossing index pairs of x* to *Cross*
    for *j* in *Cross* **do**
        Patt[*j*] ⇐ max(*x*[*Cross*[*2j*],*Cross*[*2j+1*]])
    end for
    for *Window* from $\beta 1$ till $\beta 2$ **do**
        for *k* in *Window* till end of *Patt* **do**
            Mean[*k*] ⇐ Mean(*Patt*[*k*] + . . . + *Patt*[*k* – *Window* + *1*])
            if STD(*Mean*[*k*]) ≤ *Threshold* **then**
                Set *Window*
                Break
            end if
        end for
    end for
    while *i* with *every index in Cross* **do**
        if *Last index in Cross* – *i* ≥ *Window***2* **then**
            Crosspx.insert(*crossing pairs from Cross*)
            i = *i* + *Window***2*
        end if
    end while
    Shift *x* by *M*
end for
Return *Crosspx*

Figure 8.3 presents two example signals: The upper signal is the raw ECG signal which is difficult to process as it is. The novel moving average algorithm was applied to the signal while updating the voltages with the average of the current mean and the *n-1* previous means with a window of width *n*. In case a signal crosses zero the averaged output will remain in its vicinity for a longer time, thus allowing for zero-crossing detection more easily. **Algorithm (8.4)** describes the code used to implement this algorithm whereas presented returns the mean-based moving average. Figure 8.3, presents all the zero-crossings of an ECG signal after it has passed through Algorithm (8.4): the red dots represent zero crossings with positive slope and the blue dots represent zero crossings with negative slope.

The pre-processing algorithm presented provides a novel technique to generate a waveform from both frequency-dominant and voltage-dominant signals, such as EEG and ECG

signals respectively. This output waveform exists in the new D-Domain where for every record in the used dataset the bounds of each record is clearly defined. However, the generalized version of the algorithm, which can be applied to poorly annotated datasets provides novel methods that expand the scope of operation into a wider range of applications. This technique is presented in **Algorithm (8.5)** whereas demonstrated the waveform is subdivided into *M* portions whose window size is based on the maximum voltage of each Negative-to-Positive *(N-to-P)* and Positive-to-Negative *(P-to-N)* crossing pairs. The moving means of multiple maximum voltages are collected before calculating their *STD* which is compared to a predefined *Threshold* in order to set the current *Window* size. this *Window* size is continuously refreshed in order to subdivide the waveform *x* using *Crosspx* in real-time. Appendix A at the end of this thesis presents practical details of the code execution of **Algorithm (8.5)**, by posting data taken directly from the Python interpreter to further explain the operation used to pre-process a TS, in case it is not meticulously annotated.

8.5 *VAMPIRE* post-processing algorithms - *Activation Engines*

In this thesis, in addition to the pre-processing algorithms presented, we further propose a new post-processing technique we name the *Activation Engine*. It consists of applying an optimal thresholding technique at the output of a classification algorithm, aimed to significantly improve prediction accuracy. It is simple, yet capable of improving the accuracy without requiring too much effort on hyper-parameter tuning. This approach can lead to more accurate classification results, using a single ML setup, and without resorting to automated techniques which demand processing power, are time-consuming, and are challenging to implement. Additionally, the *Activation Engine* was implemented using Numpy vectors to allow for efficient post-processing on the edge.

The *Activation Engine* has been developed to improve the testing accuracy of a MLP with a binary *Softmax* output. **Algorithm (8.6)** determines the *Ref* variable which presents the factor between the occurrence for the two classes in the training set. **Algorithm (8.7)** may run the *For* Loop **(Bordered in bold)** from **Algorithm (8.6)** twice since there might be two activation functions at the output.

In this case, the algorithm swaps the first and second *Softmax* outputs for each iteration. Finally, in case there are two *Activation Engines*, the cluster that has the highest accuracy will be voted based on overall training accuracy and is then applied to the testing data.

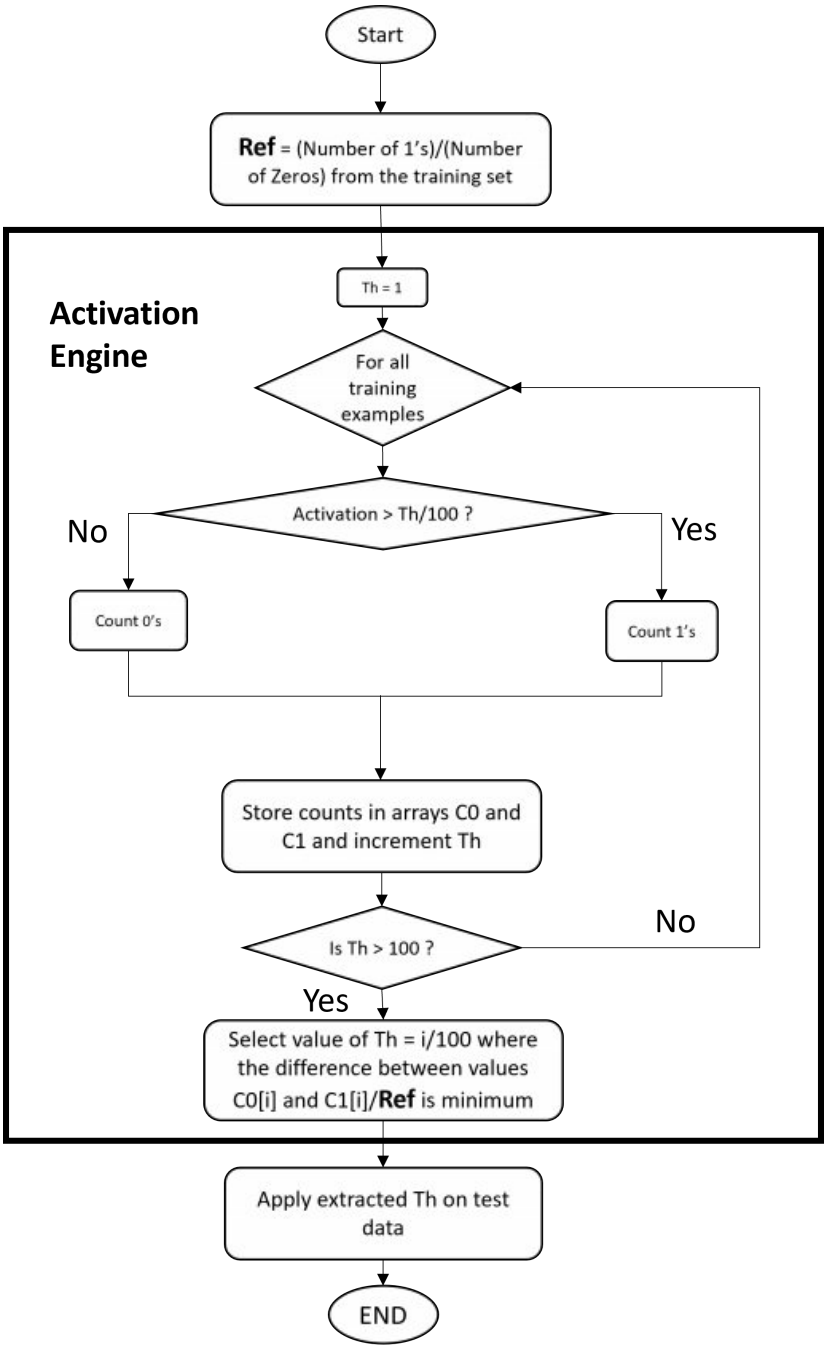

Fig. 8.7 Algorithm 8.6: *VAMPIRE Activation Engine*.

Furthermore, since there are two activation functions, there should be two possible clusters per function. So, there can be four values from two pairs that have to be considered for voting.

After determining values for array *Th*, the best version of the two thresholds arrays is extracted from the training data, and two indices are computed which point to the better

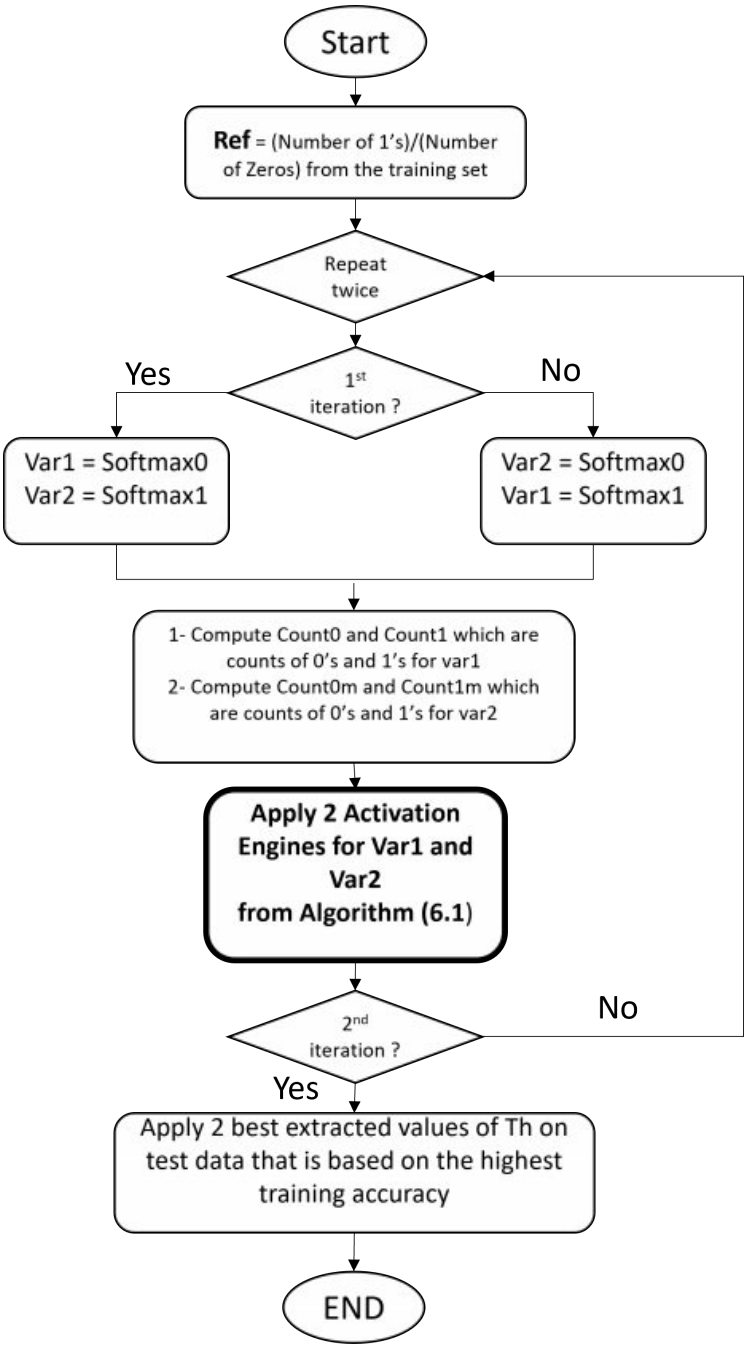

Fig. 8.8 Algorithm 8.7: Combining two *Activation Engines*.

version of the thresholds. Finally, the best threshold version is applied on the testing date resulting in improved prediction accuracy.

Algorithm (8.6) and **Algorithm (8.7)** as shown in Figure 8.7 and 8 respectively, describe the steps and iterations taken to perform the binary classification versions of the *Activation*

Engine. Moreover, a tree-based technique is used to perform multi-class classification using this approach. However, it is important to point out that since an output of two *Softmax* functions are used, the variable *Th* is a 2x2 matrix since it has two possible values for every *Softmax* threshold.

The FE algorithms presented have been developed in order to convert TS of various characteristics and notations into datasets directly applicable to various ML setups, and without resorting to any advanced and stressful pre-processing techniques, which are in terms difficult for inexperienced practitioners. Furthermore, since beginners in ML might find parameter tuning unreachable, the post-processing algorithm presented: The *Activation Engine*, may be used along with a MLP in order to optimize forecast performance without turning to automated ML methods, such as meta-learning or ensemble learning, which require considerable processing power and are time-consuming in nature.

The complexity of the *Activation Engine* can be expressed as shown in *Equation 1* where the factor 2 corresponds to the 2x2 matrix representing the two possible threshold pairs of each *Softmax* activation function. Also the factor 4 corresponds the counting of *True* and *False* values of each binary classification for the two cases of that threshold. the factor 6 represents he extraction of the indices and the evaluation of the *True* and *False* for the optimal threshold extraction. *T1* and *T2* represent the sizes of the training set and test-set respectively. Consequently, *N* corresponds to the value 100 found in **Algorithm (8.6)** and **Algorithm (8.7)** from Figure 8.7 and 8.8 which sets the resolution for the threshold used by the *Activation Engine* inference. Additionally, *C* corresponds to the number of class labels used in the dataset which is included since the post-processing algorithm uses a tree-based approach to achieve multi-class classification. Although, setting *C=2* implies that the tree-based method is not used and so *Equation 1* is valid also for binary classification:

$$Complexity = \sum_{0<=i<C-1} (1-\frac{i}{C}).(2.(N.4.T1+2.T1)+2.6.T1+2.6.T2) \tag{8.1}$$

Furthermore ,f *N.T1* is large enough, which represents the product of the training set size and threshold precision, *Equation 1* can be approximated in terms of *Equation 2*:

$$Complexity = \sum_{0<=i<C-1} (1-\frac{i}{C}).8.N.T1 \tag{8.2}$$

8.6 Data sources

In order to test the performance of the *VAMPIRE* Framework, five data sources taken from five unrelated applications were adopted to perform forecasts using its pre-processing algorithms and *Activation Engines*. Three biomedical TS datasets (Related to ECG, EEG, and PPG data) were used to implement *VAMPIRE FE1* and *FE2*. Therefore, the biomedical datasets along with the pre-processed datasets were employed to test Rulex and the *Activation Engine* on data that was pre-processed using *VAMPIRE FE1 and FE2* and using different FE techniques from the literature. All tests were performed through a general-purpose laptop as a client and the Raspberry Pi as an AI applications server.

8.6.1 ECG dataset

For the ECG application, the database used is the MIT-BIH arrhythmia database, (Goldberger et al. (2000); Moody and Mark (2001)). It consists of 48 30 min ECG recordings which pair with 48 annotation files that contain the classification of each sample. Therefore, the *VAMPIRE FE1* was used which deals with fully annotated datasets. In this dataset, there are 20 arrhythmia classes, however, we have chosen to consider five that have enough samples for feasible ML training and testing. Also, we have applied multiple forecasts for the three most dominant classes and with four most dominant classes to demonstrate that the algorithm is more accurate in case a larger number of samples is available. Therefore, a collective subset of 35989 samples was used in the forecasts having 73 features.

8.6.2 EEG dataset

The EEG recordings were taken from the CHB-MIT scalp EEG database (Goldberger et al. (2000)) which consists of records from pediatric subjects with intractable seizures. Moreover, in accordance with the previous section, *VAMPIRE FE1* was implemented to generate the features. The dataset analyzed in this work contains records from 22 subjects, sampled at a rate of 256 samples per second with 16-bit resolution. Most files contain 23, 24, or 26 EEG signals. As a whole, a random subset from all patient was used with 17978 records having 180 features for the train/test split.

8.6.3 PPG dataset

The PPG data was taken from part of the predicting cognitive fatigue with Photoplethysmography project (Finean and Dillon (2021)) and is taken from the Kaggle dataset repository (Finean (2021)). The recordings consist of participants taking a 22 hour-shifts of gaming. These are used to classify the Stanford Sleepiness Scale (1-7). However, due to the ambiguity of the data, since it is a self-assessment and not a clinical diagnosis, we have split the classes into two class groups *True* and *False*, which signify *Sleepy* or *Not Sleepy*. Furthermore, the classification is taken over long periods of time, so, in order to classify the gamer's fatigue level in real-time, *VAMPIRE FE2* was employed. Two gamers records were chosen for the present ML workflow with a total of 31355 instances with 64 features.

8.6.4 Radar dataset

The radar dataset from Rizik et al. (2019) consists of features extracted from a double FFT applied to radar measurements. These include statistical quantities of this signal such as *mean*, *STD*, *variance*, *range*, and others.

There are four classes, where there is one for *Humans* and three others for *Vehicles*. A MLP with an *Activation Engine* at its output with K-Folder cross-validation splits was implemented, which achieves high forecast accuracy. This dataset consists of 120 records equally split over its labels having 10 features.

8.6.5 Activity dataset

The activity detection dataset from Anguita et al. (2021) consists of features taken from the accelerometer and gyroscope embedded in a smartphone. These include statistical quantities of this signal such as *mean*, *STD*, *variance*, and others. Six classes describe the activity performed by a subject such as *Laying*, *Standing*, *Sitting*, *Walking*, *Walking Upstairs*, and *Walking Downstairs*. This dataset consists of 561 pre-processed features including a user field while containing 7352 samples.

8.7 Experimental results

All related experimental results will be presented in detail in this section. The biomedical TS described in the previous section were used to test *VAMPIRE* framework's performance where the recordings were pre-processed to generate corresponding datasets, and were either

applied using Rulex in a Client/Server arrangement on the Raspberry Pi, or by remotely employing a MLP with an *Activation Engine* on the same board. In case of the EEG and PPG datasets, both of *VAMPIRE*'s pre-processing algorithms and *Activation Engines* were employed, in addition to applying the EEG and ECG datasets using Rulex on the Raspberry Pi.

Concerning the pre-processed datasets, both the activity detection and radar classification datasets were implemented with a MLP having an *Activation Engine* at its output. Furthermore, the activity detection dataset was applied using Rulex on the edge. In every case where ML was applied on the edge using Rulex running on the Raspberry Pi, holdout validation was applied while implementing a K-folder setup for the EEG, PPG, and radar forecasts. Additionally, the power consumption and inference time will be evaluated by comparing with the performance of other ML setups.

8.7.1 Results for ECG forecasts

In the case of the ECG application, we used 73 Features for the generated dataset, which were extracted using *VAMPIRE FE1* from the MIT BIH arrhythmia database. The features consist of the *mean*, *variance*, *STD*, *median*, *minimum*, and a *maximum* of the variables *a1*, *b1*, *Sum1*, *Diff1*, the *integral* of the spike, the *area* of the spikes, and the same statistical functions for the sets of *a2*, *b2*, *Sum2*, and *Diff2* as presented in Figure 8.5.

We have applied ML algorithms DT, and SVM in their default format using the Rulex ML platform running on the Raspberry Pi in a client/server setup (Hajdarevic et al. (2014)). Three arrangements were adopted with 3, 4, and 5 classes which take the more dominant classes in the dataset into consideration. Just five classes have been used in the simulations which are the forward-slash */* which represents a Paced Beat, *N* for Normal, *L* and *R* which are the left and right bundle branch respectively, and *V* which means Premature Ventricular Contraction.

VAMPIRE FE1 requires a large TS dataset as input, mainly since it transforms the input data from one domain to another, which is then used for training and testing. However, we were still able to achieve highly accurate forecasts using classical ML algorithms.

Figures 8.9(a) and 8.9(b) represent the accuraies for the training and testing using DT respectively, however, due to the lack of samples in classes A and V, the accuracy is less than for the other classes in the testing. In the case of 5 classes, the accuracies for DT and SVM in testing are 95.55% and 96.24% respectively.

Table 8.1 Five classes for ECG classification using Rulex on the edge.

Classes:	/	L	N	R	V
SVM:	99.57%	94.61%	98.80%	96.88%	89.11%
Decision Trees:	98.19%	93.82%	97.10%	95.12%	89.98%

ECG Classification using Rulex running on the Raspberry Pi with 5 classes

Table 8.2 Four classes for ECG classification using Rulex on the edge.

Classes:	/	L	N	R
SVM:	99.24%	94.77%	98.93%	97.14%
Decision Trees:	99.24%	94.45%	97.20%	96.40%

ECG Classification using Rulex running on the Raspberry Pi with 4 classes

In Figures 8.9(c) and 8.9(d) the accuracies for both training and testing using DT with 4 classes is presented, respectively. DT and SVM perform more accurately in the case of 4 classes. class V which has the worst performance has been omitted. The overall accuracies in the case of DT and SVM testing are 96.60% and 97.62% respectively.

Figures 8.9(e) and 8.9(f) present the training and testing forecast accuracy for SVM using a 3-class arrangement. For all algorithms using the same datasets, the performance improved in terms of accuracy as the fewer dominant classes were omitted. This is due to the lack of samples where the algorithm does not generate enough information in the features for accurate prediction. The overall accuracies in the case of 3 classes for DT and SVM are 97.99% and 98.73% respectively.

In all the forecasts, the data which were used for prediction were taken from the files which are dominated by the targeted classes. For instance, if there are files that predict classes *F* or *S* in a large enough number, these files were omitted to focus on the targeted classes without adding any excessive data. So, even classes containing labels such as *N* which represent a normal beat and exist in almost all files, the samples are only taken from the files which are rich in the target classes.

Table 8.3 Three classes for ECG classification using Rulex on the edge.

Classes:	/	L	N
SVM:	99.86%	99.82%	95.93%
Decision Trees:	99.57%	98.73%	95.80%

ECG Classification using Rulex running on the Raspberry Pi with 3 classes

This methodology proved to be fair since it generates very good results consistently as the number of classes varied. This is also valid for various ML algorithms. A complete set of results for forecast accuracy using the described FE method is presented in Tables 1, 2, and 3.

In Ahamed et al. (2020) and ensemble learning approach was applied to the same ECG dataset which is used in this project and another much smaller data source with the aim of improving prediction accuracy. The system described is not fully automated since it requires careful tuning while it relies on grid-search to tune an SVM from the ensemble. The best accuracy achieved was realized by an MLP which is 98.06% while the overall ensemble accuracy was recorded at 97.78%. However, the best accuracy was taken without considering additional forecasts in a K-folder setup which provides more reliable and unbiased results. In Sree et al. (2021) a ML framework dedicated to ECG classification is presented which partly used the dataset used in this project found in Moody and Mark (2001). Even though an overall accuracy of 98.2% was achieved, this is due to the addition of synthetic data which might have caused the ML model to over-fit to the test-set. Furthermore, when using real world testing data, the accuracy dropped to 81% using Random Forest and with a two-second window 85% using a five-second window for a four-class forecast.

8.7.2 Results for EEG forecasts with *VAMPIRE FE1* and Rulex

In the second forecast case, we have applied *VAMPIRE FE1* on TS related to epileptic seizure Detection using EEG measurements, which are taken from subjects with intractable seizures. Forecasts were applied using Rulex ML software running on the Raspberry Pi in an edge computing setup. However, unlike the previous set of ECG Features, the total number of features for the EEG applications was multiplied by three, since brainwaves are frequency-dominant signals unlike voltage-dominant signals occurring in ECG waveforms.

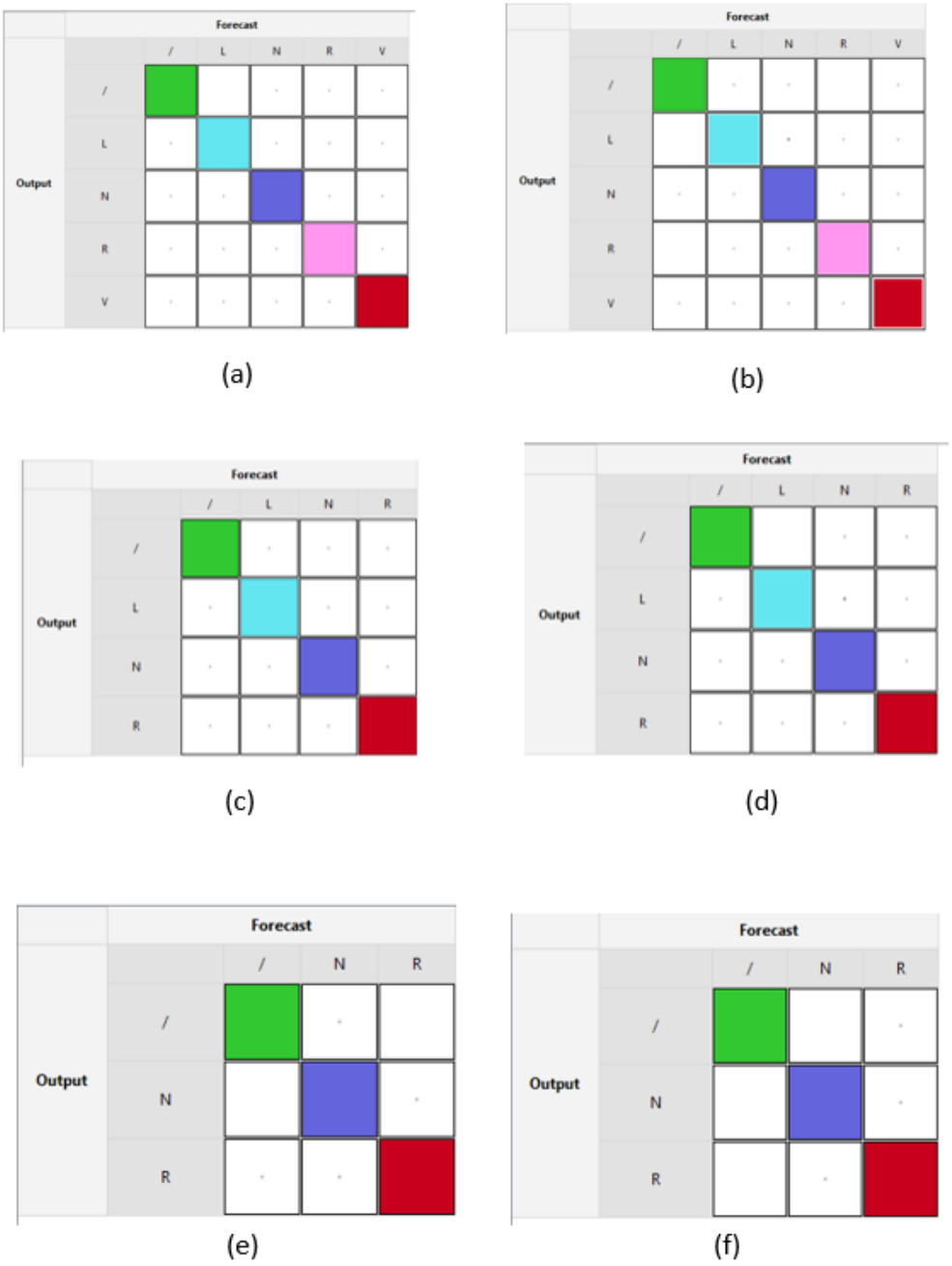

Fig. 8.9 Training and testing accuracies for ECG forecasts:(a) and (b) DT training and testing accuracy with 5 classes respectively. (c) and (d) DT training and testing accuracy with 4 classes respectively. (e) and (f) SVM training and testing accuracy with 3 classes respectively.

So, the EEG recording was split into three frequency bands using digital band-pass filters and the same number of features was generated for each frequency band.

Table 8.4 EEG forecast using Rulex on the edge.

Classes:	Non-Seizure	Seizure
LLM:	85.087%	85.992%
SVM:	88.289%	81.493%
DT:	84.355%	84.05%

EEG Classification using Rulex running on the Raspberry Pi

The forecast accuracies using the Rulex implementation are presented in Table 4, where SVM, DT, and LLM algorithms were applied to the extracted features which provide accurate results across all the adopted algorithms.

8.7.3 EEG forecasts with *VAMPIRE FE1* and *Activation Engines*

As for the prediction results on the EEG data using *VAMPIRE*'s *Activation Engine* on the Raspberry Pi, a Tensorflow implementation of a MLP was built using Keras. It consists of a MLP trained using back-propagation. The MLP has an input layer with 219 inputs and a Rectified Linear Unit (*Relu*) activation function. The is proceeded by four hidden layers alternating between *Sigmoid* and *Relu* until it reaches 2 *Softmax* functions for a binary output. The corresponding transfer functions for the *Sigmoid*, *Relu*, and *Softmax* functions can be found in *Equations 3 - 5*:

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{8.3}$$

$$Relu(x) = \max(0,1) \tag{8.4}$$

$$Softmax(xi) = \frac{e^{-xi}}{\sum_{j=1}^{k} e^{xj}} \tag{8.5}$$

The training using Tensorflow was run for 60 epochs with a K-folder split with K = 5. As for the proposed MLP with *Activation Engines*, the overall testing accuracy was 85.32% with the 2 output classes being predicted with accuracies of 87.58% and 83.07%. In Qureshi et al. (2021) forecasts were applied partly on the same EEG dataset used in this project using classical ML algorithms and fuzzy-based techniques where an accuracy of 85.6% was achieved using a deep CNN (which is computationally expensive) and 90%

using Sparse Extreme Learning Machine (Bai et al. (2014)). However, the authors use a subset of five patients from the CHB-MIT dataset whereas in this project a completely random subset from all patients is used which helps to avoid over-fitting and allows for more reliable generalization, while remaining competitive. Furthermore, the MLP was tested after removing the *Activation Engines* and relying on a one-vs-all approach where the new method outperforms the classical technique by 3% which has an overall accuracy of 82.2%.

8.7.4 PPG forecasts with *VAMPIRE FE2* and *Activation Engines*

Regarding the Sleepiness PPG recording dataset, we considered two subjects out of the existing five, specifically gamers 1 and 3. The 7 sleepiness classes were grouped into parent classes by equally distributing them into *Sleepy* and *Not Sleepy*. So, in this binary classification arrangement, we have made forecasts using the Python Tensorflow library through a MLP. Also, as in the EEG application, the *Activation Engine* technique was applied at the output of the MLP to improve testing performance.

VAMPIRE FE2 described in **Algorithm (8.5)** was used to extract the features from the PPG dataset, considering that the TS is patient-annotated with no expert medical assessment present. Moreover, the *Activation Engine* described in **Algorithm (8.6)** and **Algorithm (8.7)** from Figures 8.7 and 8.8 was applied at the output of a MLP to further improve accuracy on the edge.

The MLP implemented using Keras has an input layer with 63 inputs and a *Relu* activation function. The is proceeded by three hidden layers having a *Relu* as an activation function and has two *Softmax* binary outputs. We have used the above MLP arrangement with hidden layers having a size 70, and the algorithm was run for 20 epochs using Tensorflow.

For gamer 3, after 5 epochs, the testing accuracy as reported by the *Activation Engines* with a k-folfer setup having $K = 5$ was 89.23% and 79.77% for *Sleepy* and *Not Sleepy* classes, respectively. As for the overall accuracy, it was equal to 83.59% in general.

As for gamers 1 and 3 combined, after 5 epochs, the testing accuracy as reported by the *Activation Engine* arrangement with a K-folder setup having $k = 5$ was 85.43% and 76.76% for *Sleepy* and *Not Sleepy* classes, respectively. As for the overall accuracy, it was equal to 78.46% in general. Additionally, the MLP was tested with no *Activation Engines* as its output where the performance degraded without the post-processing block having an unbalanced output testing accuracy of 90.93% and 38.58% for the two classes.

8.7.5 Urban classification using *Activation Engines*

A multi-class dataset for urban classification via radar was used to test the *Activation Engine* setup in Python in the case of four available classes. These are *Humans*, *Cars*, *Trucks*, and *Motorcycles*.

The tree-based structure used in **Algorithm (8.8)** which is shown in Figure 8.10, works similarly to the one-vs-one method for multi-class classification except that it combines child classes into parent classes, and once a child class is reached, its accuracy is multiplied with the prediction accuracies of the parent classes. With this technique, each forecast accuracy at a given level should be multiplied by the accuracy of their preceding parent forecasts.

This tree as presented in **Algorithm (8.8)** from Figure 8.10, applies four forecasts using MLP with the *Activation Engines* method described in **Algorithm (8.6)** and **Algorithm (8.7)** from Figures 8.7 and 8.8. The MLP consists of four *Relu* layers and one *Softmax* output layer. After running the algorithms in K-folder arrangement with $K = 15$, the following results were obtained in Table 5 where they were compared with results from the literature where the dataset was first tested. Although the previous results don't use K-folders to generate reliable

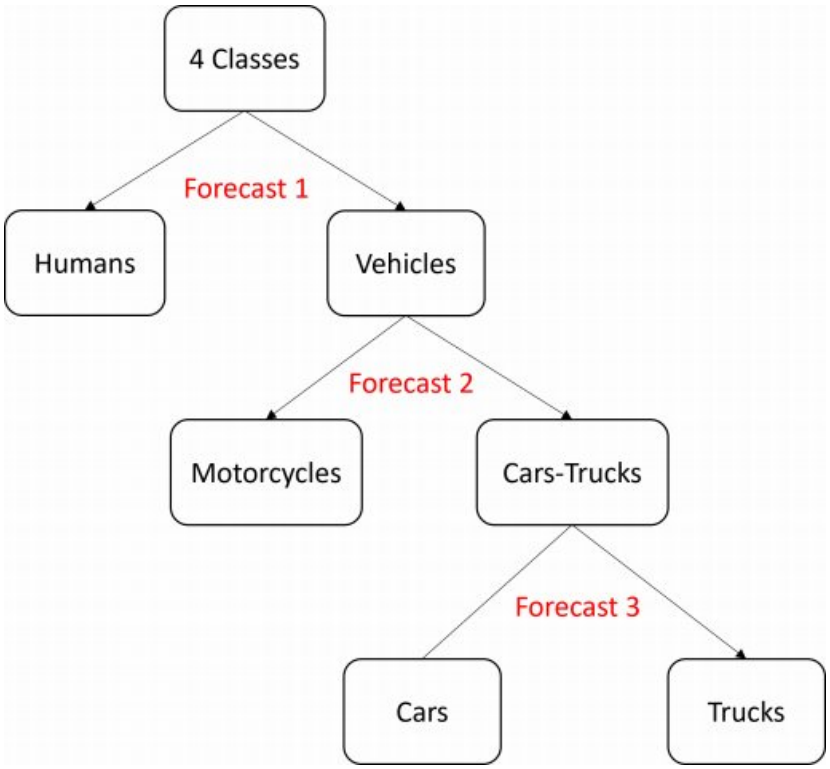

Fig. 8.10 Algorithm 8.8: Applying tree-based structure to perform classification on urban dataset.

results in terms of test-set-variability, *VAMPIRE* was able to achieve an overall classification accuracy with an 8.8% improvement over previous techniques.

In the results from *VAMPIRE*, the *Motorcycles* detection accuracy shown is multiplied by that of *Vehicles*. Also, the accuracy for *Cars* and *Trucks* classes was multiplied with the

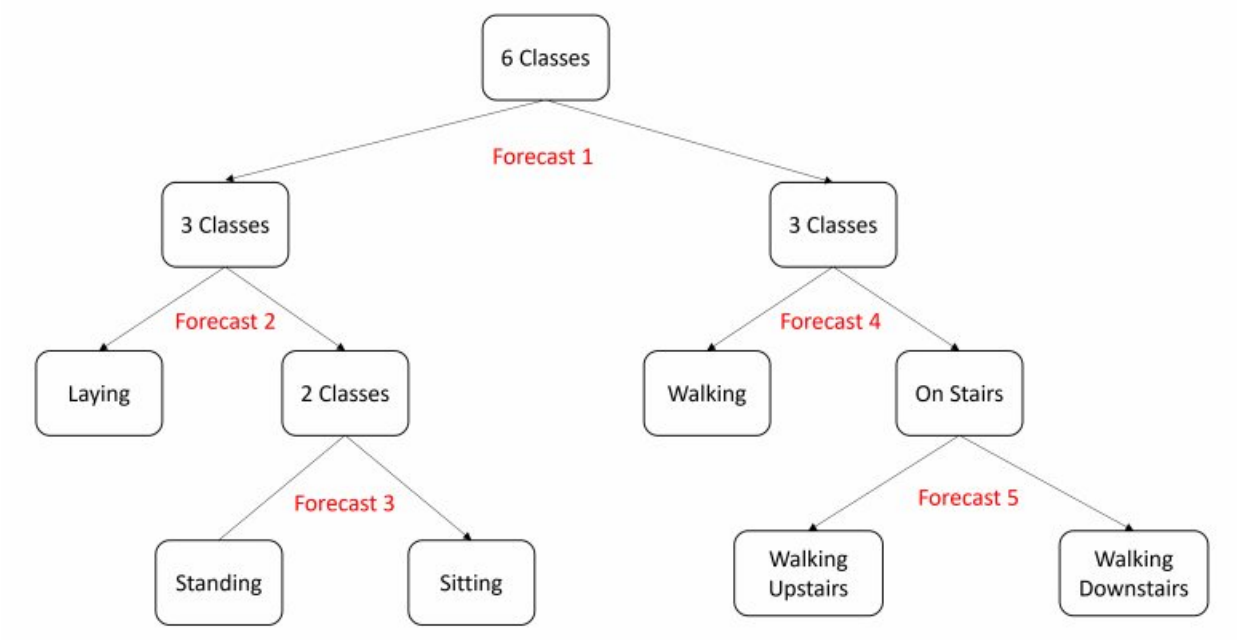

Fig. 8.11 Algorithm 8.9: Applying tree-based structure to perform classification on the human activity detection dataset.

Table 8.5 Radar classification accuracy of pedestrians and vehicles using *VAMPIRE* on the edge with comparisons.

Algorithms:	Classic MLP	*VAMPIRE*	Rizik et al. (2019)	Rizik et al. (2021)
K-folds:	Yes	Yes	No	No
Humans:	94.78%	93.60%	100%	100%
Vehicles:	-	99.40%	80%	82.24%
Motorcycles:	41.6%	99.40%	83.33%	75%
Cars & Trucks:	-	94.85%	80%	88.89%
Cars:	97.22%	83.23%	75%	80%
Trucks:	22.71%	93.75%	83.33%	100%

Urban classification using *Activation Engines*.

Table 8.6 Human activity forecasts using Rulex and *VAMPIRE* on the edge.

Algorithm:	LLM	KNN	SVM	Bulbul et al. (2018)	Classic MLP	*VAMPIRE*
User field:	No	No	No	Yes	No	No
Laying:	100%	100%	100%	100%	89.11%	100%
Standing:	86.80%	94.40%	98.40%	96%	93.82%	99.81%
Sitting:	88.48%	91.86%	97.63%	97%	99.92%	99.81%
Walking:	94.74%	100%	100%	99%	98.57%	100%
Walking Upstairs:	89.89%	99.47%	100%	100%	99.36%	100%
Walking Downstairs:	82.93%	100%	100%	99%	89.01%	100%

Human activity classification using Rulex.

Vehicles class accuracy, in addition to being multiplied with the accuracy of the combined *Cars-Trucks* parent class. Also, after removing the post-processing block and applying the MLP model using a one-vs-all approach, the performance degraded with the classic MLP which highlights the significance of the *Activation Engine* applied at the output.

8.7.6 Human activity classification using *Activation Engines*

A multi-class dataset for human activity detection using smartphone sensors was used which has six classes that were also predicted using the same approach. As a first step, the classes were allocated to two parent groups, *Sitting*, *Laying*, and *Standing* as the first, and the three classes related to *Walking* in the other. Moreover, these two classes were applied using a ML algorithm, and were subsequently split into sub-classes following the same procedure as in the radar classification application, and are similarly presented in Figure 8.11 Again, as pointed out in the previous application, each accuracy achieved through a child forecast is multiplied with the preceding parent class accuracy. Forecasts were performed using a MLP with of four *Relu* layers and two *Softmax* outputs functions having two *Activation Engines* connected at the MLP output. Furthermore, the human activity detection dataset was applied on the edge using Rulex with multiple ML algorithms where the accuracies for LLM, KNN, SVM and VAMPIRE are provided in Table 6. Moreover, the MLP was tested in a one-vs-all setup without an *Activation Engines* (with K equal to 5) to validate its effectiveness in tuning

Table 8.7 Inference and specifications using *VAMPIRE* and other IoT setups on the edge.

Board	Cores	Frequency	RAM	GPU	ML setup	Inference/Task (msec.)
Rasp-Pi 3B+:	4	1.4GHz	1GB	NO	PPG	**3.1**
Rasp-Pi 3B+:	4	1.4GHz	1GB	NO	EEG	**6.1**
Rasp-Pi 3B+:	4	1.4GHz	1GB	NO	Activity	25
MacBook:	4	2.7GHz	8GB	NO	AlexNet	29
Jetson TX2:	6	2GHz	8GB	YES	AlexNet	13.5
FogNode:	4	3.2GHz	32GB	NO	AlexNet	27
Rasp-Pi (Zhang et al. (2018)):	4	0.9GHz	1GB	NO	SqueezeNet	2080

Inference time for IoT boards.

the model. Moreover, the forecasts applied in Anguita et al. (2013) are included in Table 6 to compare the accuracies achieved in every case.

In Bulbul et al. (2018), a KNN ensemble classifier with 30 instances was used to classify human activity using the same dataset where an overall accuracy of 98.6% was achieved. This accuracy is substantially less than that achieved using both SVM in the Rulex client/server forecast, and when using a MLP with a *VAMPIRE Activation Engine* edge setup, as shown in Table 6. In the original dataset, a user field was included which identifies the person who is holding the smartphone. However, in this experiment the field was removed for more generality where *VAMPIRE* outperforms the literature with a lesser degree of information.

8.7.7 Latency and power consumption on the edge

To evaluate the performance of *VAMPIRE* on the Raspberry Pi and to ensure its efficiency in addition to providing accurate predictions, a comparison has been made with a laptop (using the same data sources) and other edge setups described in the literature. Therefore, in Zhang et al. (2018), various ML libraries have been tested on multiple IoT modules including the Raspberry Pi. Consequently, Training was applied outside the edge node and inference was performed on each board. Table 7 presents a comparison between *VAMPIRE* on the Raspberry Pi and the additional setups. As shown, *VAMPIRE*'s optimized *Activation Engine* post-processing block outperforms the other setups presented in the literature where the inference/task is presented along with the specifications of all hardware platforms.

Table 8.8 Power consumption comparison between Raspberry Pi 3B+ and a MSI laptop using *VAMPIRE*'s *Activation Engine*

Platform	Dataset	Kfolds	GPU	DC Power (Wh)	AC Power (Wh)	inference/Task (msec.)
Rasp-Pi:	Activity	1	NO	1.28	1.63	25
Rasp-Pi :	EEG	5	NO	7.1	10.43	6.1
Rasp-PI:	PPG	5	NO	0.76	1.06	0.1
MSI-PC:	Activity	1	NO	-	3.2	3.1
MSI-PC:	EEG	5	NO	-	13.78	1.4
MSI-PC:	PPG	5	NO	-	5.2	1.5
MSI-PC:	Activity	1	Yes	-	0.92	4
MSI-PC:	EEG	5	Yes	-	8.38	0.24
MSI-PC:	PPG	5	Yes	-	2.29	0.2

Power consumption of the Raspberry Pi and MSI laptop running *VAMPIRE*.

Furthermore, in Yazici et al. (2018) authors present performance measurements taken from the Raspberry Pi on various classification and regression datasets using different ML classifiers. During the classifications, the inference time on the board depends on the dataset where these inferences are compared with those applied on a PC. the ratio between the Raspberry Pi's inference divided by the PCs inference varies between 175 and 306 depending on the dataset. However, using *VAMPIRE* on the Raspberry pi, this ratio varies only between 2 and 8 without a GPU, and between 6.25 and 25.4 with a GPU, as shown in Table 8. Additionally, the Raspberry pi's performance using *VAMPIRE* is compared with that of the MSI laptop (described in section 2.3) where the same datasets were used for the comparison in terms of power consumption (For training and testing combined) and inference time. Also, when using the laptop two forecasts were applied per dataset where the first was performed without including the onboard GPU and the second where a GPU was used to optimize the latency and power consumption. Table 8 illustrates the results for the EEG, activity detection and PPG datasets using *VAMPIRE*. As illustrated, when running training and inference, the Raspberry Pi outperformed the case with the laptop running without GPU optimization by 3-4 orders of magnitude regarding the energy consumption. Moreover, the board remains competitive with the PC even when it is optimized using the 2060 Nvidia GPU which is designed to increase the power and decrease the overall energy in ML workflows.

8.8 Chapter Conclusion

To summarize, this thesis presents the *VAMPIRE* framework, which implements novel pre-processing algorithms and introduces *Activation Engines* that can extract features from any TS data and can be applied to any other type of application, through the *Activation Engine* concept. The pre-processing algorithms consist of converting time-varying signals into the proposed D-Domain, which is a V.Hz co-representation the original signal and its FFT. The novel waveform is used to derive statistical information obtained from the shape of the rectified TS. Furthermore, the pre-processing can be applied in two modes: either using an annotated TS where a classification exists for every event in the signal, or when the TS is poorly annotated where the classification is taken for a large span of time. In the second case, an adaptive algorithm was implemented to detect an event in real-time by continuously classifying the current segment of a TS before producing the V.Hz waveform and extracting statistical features. Also, the thesis introduces the *Activation Engine* which is a post-processing block which improves testing accuracy by optimizing the classification threshold at the output of an MLP based on the training performance. The *Activation Engine* achieves high testing accuracy by relying on an optimal threshold extraction method that is based on the training accuracy by employing a pair of *Activation Engines* and relying on a clustering approach. Additionally, *VAMPIRE*'s core pre-processing and post-processing operations were implemented in Python Numpy vectors in order to the improve performance considerably. Also, every ML forecast was carried out in an IoT setting, either using Rulex running on the Raspberry Pi and in a client/server setup, or by performing forecasts using a MLP with an *Activation Engine* running on the Raspberry Pi remotely through an SSH tunnel.

In regards to the pre-processing algorithms developed in this thesis, experiments were carried out by generating three datasets from biomedical TS automatically before applying ML forecasts. The forecasts were performed on these datasets in an edge computing setup where our results confirm that high accuracy was achieved in every experiment. Furthermore, in the cases of the EEG and PPG datasets, in addition to *VAMPIRE FE1* and *VAMPIRE FE2*, *Activation Engines* were employed. In these two cases, the novel post-processing method led to better testing accuracy than in training due to the influence of the threshold clustering technique of **Algorithms 4.6.1 and 4.6.2**. Also, the forecast accuracy for the two already pre-processed datasets is significantly higher than the accuracies achieved in the original publications. This is the case, again, thanks to the addition of *Activation Engines* at the *Softmax* outputs of the MLP.

FE was performed on biomedical TS such as ECG, EEG, and PPG signals where a dataset can be generated in an automated manner and the operator only needs to remove noise and offset from the input signal, before applying ML algorithms on the Raspberry Pi. Also, two pre-processed datasets related to urban classification and human activity detection were also used to perform ML forecasts on the edge.

For the ECG dataset, after applying *VAMPIRE FE1* on the recordings, the resulting dataset being generated automatically, a series of six forecasts were performed. when predicting for five labels, the SVM accuracy was 96.2% and the DT accuracy was 95.6%. when predicting four classes, the accuracy metrics were 96.6% and 97.6% for DT and SVM respectively. Naturally, in case of three classes, higher accuracies of 98% and 98.7% were achieved for DT and SVM respectively.

When forecasting the EEG dataset using Rulex, accuracies approaching 85% and 86% were reached using LLM, SVM and DT. However, when applying *VAMPIRE FE1* and the *Activation Engine* on the EEG data, an overall accuracy was reached of around 85.21%.

Regarding the PPG dataset, where *VAMPIRE FE2* and the *Activation Engine* were employed, for a single gamer an accuracy of 83.6% was achieved and for two gamers, an accuracy of 78.5% was reached where the fatigue can be detected in real-time in contrast to the original dataset which provided measurements spanning a relatively long time when labeled.

As for the urban and activity detection dataset, very competitive accuracies were achieved with the aid of the tree-based classification method in parallel with the *Activation Engine* where competitive accuracies were reached when compared with the source datasets substantially.

When comparing power consumption between the Raspberry Pi 4B+ for the PPG, EEG and Activity dataset when inference was applied using *VAMPIRE*, inference times of 3.1 ms/task, 6.1 ms/task, and 25 ms/task outperformed a MacBook, Jetson TX2 and an additional cited Raspberry Pi considerable mainly due to using a classic MLP on the Raspberry 3b+ along with vectorizing the *Activation Engine* which to lightweight ML model.

Furthermore, the framework's accuracy, latency and power consumption on the Raspberry Pi was also evaluated where the *Activation Engine* outperforms most previously published results in terms of inference speed and accuracy. Moreover, regarding the power consumption of the Raspberry Pi, a comparison with a PC was employed where the board consumes less energy than the laptop in case the GPU is not included in the training and remains competitive in comparison with the GPU being used to optimize the ML workflow. In the end, accurate results across all experiments prove that *VAMPIRE* is easy to implement and demonstrate

the robustness of the framework by competing with different techniques from the literature (Which are compute-expensive in most cases) while maintaining generality and so it can be used virtually in any ML application.

Part III

Conclusion

Chapter 9

Summary and Conclusions

The following chapter presents a summary of the projects undertaken in this thesis by describing the contents and results of each project individually before making a general statement and providing a general and detailed outcome of the collective effort.

9.1 Project 1: Summary and Conclusions

In this thesis which stresses on ML on the Edge for lightweight operation, efficiency and high-performance, is outlined by multiple software contributions in AI on the Edge and in ML in general. Primarily, the task consisted of the porting of the Rulex Software on the Raspberry Pi IoT board. This was realized by compiling every internal and external dependency in Raspbian 32-Bits running on the portable board which itself is a challenging and stressful task. Post-porting, Rulex was tested on multiple datasets with varying depth and dimensionality where a tree-based classification algorithm was developed. Additionally, an Image-to-TS conversion algorithm was used to classify gender with a very good accuracy while scanning random images with varying conditions in terms of the subject and background environment.

Regarding the tree-based method for improved ML performance, the proposed method was applied on the urban dataset where after a series of forecasts: Humans were classified with an accuracy of 100% and vehicles with an accuracy 96.67%. The final vehicles subclasses forecast accuracies are 90.63% for Motorcycles and 77.34% for the Cars and Trucks classes. Overall, by employing LLM running in Rulex and on the Raspberry Pi, an accuracy is 86.32% was reached that outperforms the results reached in Rizik et al. (2021) by 1.3% mainly due to the robustness tree-based technique.

The overall accuracy for the urban dataset using the traditional One-VS-All method in Rulex was 84.2%, and an accuracy 99.3% was achieved for the human activity detection

application. As for the brainwave dataset the overall accuracy was 95.47%. Regarding the vehicle activity recognition dataset, the overall accuracy was 95.27% using KNN running in Rulex.

The results provided in this project infer that general-purpose commercial ML software may be deployed on limited-performance IoT nodes while preserving accuracy and efficiency.

9.2 Project 2: Summary and Conclusions

The thesis includes the CACAO-X framework which consists primarily of two CACAO-Net models 1 and 2 which are CNN/MLP and CNN/MLP/LSTM hybrids respectively. These models take two inputs: regular images and their TS equivalent using a modified version of the algorithm mentioned in section 7.1. This setup is used alongside landmark and local explainable information to achieve global image explainability with the aid of Rulex and explicative labeling (which is also introduced). Moreover, TF-Lite models are used to deploy the CACAO-Net onto the Raspberry Pi where high accuracy was reached for multiple image datasets while keeping the power consumption substantially low.

After applying a K-fold setup on all five datasets, the gender dataset forecast accuracy was 98.4% and the testing accuracy was 91.5% having a competitive inference time on the Raspberry Pi 4 while applying the pre-processing in real-time. Regarding the Satellite dataset, an average accuracy of 97.31% was reached with a testing accuracy of 94.1%. As for the Shadow dataset, the average accuracy reached 99.41% and the testing accuracy was 98.9%. Regarding the skin cancer and weather datasets, the averaged accuracies were 96.8% and 91.21% respectively, and for the testing accuracies, they were 95.5% and 91.9% respectively. In every case the inference on the Raspberry Pi 4, which included the FE during inference, outperformed the Raspberry Pi 3 (without the real-time FE).

Regarding the K-fold setups, the validation and loss curves for training as monitored by Algorithm 7.4 halted the training when the oscillation around the most recent mean was below two predefined thresholds, either on the first or second point, or until the final epoch.

When comparing CACAO-X with the Pycaret library, Cacao-Net Model 1 outperformed the Pycaret, which in terms tunes various ML models which actually outperform published works substantially. However CACAO-Net beat Pycaret 97% of the time in accuracy, precision, recall, F1-score, MCC, and balanced accuracy. However, regarding the shadow dataset, Pycaret outperformed the original source in terms of accuracy ranging from 8.5% and 18.5% proving that it is an accepted and reliable comparison with the Cacao-Net.

This project presents results which prove that lightweight and general-purpose ML architecture can be used as reference models for global image explainability while achieving competitive accuracy and efficiency on edge devices.

9.3 Project 3: Summary and Conclusions

The thesis also introduced the *VAMPIRE* framework which is a set lightweight pre-processing and post-processing algorithms capable of achieving high accuracy efficiently on the Raspberry Pi. Mainly it includes two versions of the pre-processing algorithm which convert biomedical TS into much smaller datasets in an automated setup. As for the post-processing part, *Activation Engines* were introduced which are applied in pairs and detect the optimal classification threshold on the test-set based on training behavior. Even though this approach is being applied to binary data, the tree-based approach discussed in this thesis was used to classify multiclass datasets very accurately using the *Activation engine*. Furthermore, this post-processing method may be used to avoid extensive hyperparameter tuning due to the addition of this novel concept.

After training ECG dataset, when applying *VAMPIRE FE1* on the recordings, using the dataset which was generated automatically, multiple forecasts were performed. when training for five labels, SVM reached an accuracy of 96.2% and for DT an accuracy of 95.6%. As for four classes, the accuracies were 96.6% and 97.6% for DT and SVM respectively. In case of three classes, the accuracies achieved were 98% and 98.7% for DT and SVM respectively.

After training using the EEG dataset using Rulex, accuracies approaching 85% and 86% were reached using LLM, SVM and DT. However, after employing *VAMPIRE FE1* and the *Activation Engine* on the same data, an overall accuracy was reached of around 85.21% for the EEG application.

Concerning the PPG dataset, where *VAMPIRE FE2* and the *Activation Engine* were applied, when using data for a single gamer an accuracy of 83.6% was achieved. For two gamers, an accuracy of 78.5% was reached where the fatigue can be classified in real-time using the dataset generate whereas when using the original dataset, the classification corresponded for a long time per label.

Regarding the urban and activity detection datasets, high-performance was achieved after employing the tree-based classification method in addition to using the *Activation Engine* where competitive accuracies were reached when compared with the source forecasts substantially.

When comparing power consumption between the Raspberry Pi 4B+ for the PPG, EEG and Activity detection datasets when applied *VAMPIRE*, inference times of 3.1 ms/task, 6.1 ms/task, and 25 ms/task outperformed a MacBook, Jetson TX2 and an additional cited Raspberry Pi substantially. This is the case because the classic MLP used on the Raspberry 3B+ was lightweight in addition to using Numpy vectors to implement the *Activation Engine*.

The results outlined in this project confirm that simple ML workflows can be automated by users with little experience in ML, while achieving high accuracy and efficiency when compared with works in the literature discussing ML on the edge.

9.4 Final Statements

The overall theme of this dissertation has been the implementation of ML on the edge with a general application perspective while dealing with edge computing issues such as efficiency, meaning inference and power consumption. Also, the topic of global image explainability was touched upon by providing an efficient model for the task by providing the reader with human understandable statements in the field of gender classification. Moreover, the vectorized automation of ML workflow was presented which contains multiple pre-processing and post-processing techniques.

Specifically, the thesis was split into three main projects or topics:

1. Classic ML on the Edge and Tree-based Multi-class Classification:

   This project included the porting of a commercial software: Rulex, onto a portable IoT board which is accessed remotely to perform both ML training and inference while maintaining power consumption and latency efficiently. It also introduced for the first time a tree-based multi-class classification algorithm where all the possible configurations were exhausted till the end, to find the logical reasoning for grouping and splitting exhibits the best fit for improving the performance metrics.

2. Flexible Neural Networks and Explainability:

   This project introduced a new NN architecture which was composed of a CNN, with three MLPs where one MLP could be replaced with an LSTM. The variable block was used to deal with TS, mainly since this architecture performs the classification of images and their TS equivalent for robust and high-performance inference. Also, this ML model was used in a much larger architecture which contains additional extracted information and two established XAI methods to attain the global explainability of

images classified in terms of gender. The project reveals for the first time the actual measurements and angles that assist computers in recognizing gender.

3. The AI Pre-processing and Post-processing Framework:

   The *VAMPIRE Framework* which is the final project in this thesis brings three new concepts into discussion. The first being the D-Domain representation of TS which fuses both time and frequency domain measurements into a single data stream where *VAMPIRE FE1* may be implemented which is nothing but a set of statistical expressions that describe the waveform. Secondly, the *VAMPIRE FE2* algorithm is introduced which deals with poorly annotated data and their pre-processing such that they can be employed in real-time classification. Finally, the *Activation Engine* when applied with the tree-based method from the first project, which is a Numpy-optimized post-processing algorithm that provides shallow NN, which do not perform well on abstract data, to outperform more advanced models while allowing its implementation for training and testing with impressive and efficient performance.

In the end, the projects and publications that have been discussed possess high-performance a low power consumption where various novel techniques have been noted in the field of Artificial Intelligence.

References

Abiwinanda, N., Hanif, M., Hesaputra, S.T., Handayani, A., Mengko, T.R., 2019. Brain tumor classification using convolutional neural network, in: World congress on medical physics and biomedical engineering 2018, Springer. pp. 183–189. doi:https://doi.org/10.1007/978-981-10-9035-6_33.

Abu Alfeilat, H.A., Hassanat, A.B., Lasassmeh, O., Tarawneh, A.S., Alhasanat, M.B., Eyal Salman, H.S., Prasath, V.S., 2019. Effects of distance measure choice on k-nearest neighbor classifier performance: a review. Big data 7, 221–248.

Aggarwal, S., Chugh, N., 2022. Review of machine learning techniques for eeg based brain computer interface. Archives of Computational Methods in Engineering , 1–20.

Ahamed, M.A., Hasan, K.A., Monowar, K.F., Mashnoor, N., Hossain, M.A., 2020. Ecg heartbeat classification using ensemble of efficient machine learning approaches on imbalanced datasets, in: 2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT), IEEE. pp. 140–145. doi:https://doi.org/10.1109/ICAICT51780.2020.9333534.

Al-Khafajiy, M., Webster, L., Baker, T., Waraich, A., 2018. Towards fog driven iot healthcare: challenges and framework of fog computing in healthcare, in: Proceedings of the 2nd international conference on future networks and distributed systems, pp. 1–7. doi:https://doi.org/10.1145/3231053.3231062.

Albawi, S., Mohammed, T.A., Al-Zawi, S., 2017. Understanding of a convolutional neural network, in: 2017 international conference on engineering and technology (ICET), Ieee. pp. 1–6. doi:https://doi.org/10.1109/ICEngTechnol.2017.8308186.

Allen, D.R., 2003. Eleven ssh tricks. Linux Journal 2003, 5.

Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L., 2021. Human activity recognition with smartphones. URL: https://www.kaggle.com/datasets/uciml/human-activity-recognition-with-smartphones.

Anguita, D., Ghio, A., Oneto, L., Parra Perez, X., Reyes Ortiz, J.L., 2013. A public domain dataset for human activity recognition using smartphones, in: Proceedings of the 21th international European symposium on artificial neural networks, computational intelligence and machine learning, pp. 437–442.

Axelsson, H., Wass, D., 2019. Machine learning for activity recognition of dumpers.

Ayyad, S.M., Saleh, A.I., Labib, L.M., 2019. Gene expression cancer classification using modified k-nearest neighbors technique. Biosystems 176, 41–51.

Bai, Z., Huang, G.B., Wang, D., Wang, H., Westover, M.B., 2014. Sparse extreme learning machine for classification. IEEE transactions on cybernetics 44, 1858–1870. doi:https://doi.org/10.1109/TCYB.2014.2298235.

Balaji, V., Suganthi, S., Rajadevi, R., Kumar, V.K., Balaji, B.S., Pandiyan, S., 2020. Skin disease detection and segmentation using dynamic graph cut algorithm and classification through naive bayes classifier. Measurement 163, 107922.

Bellavista, P., Zanni, A., 2017. Feasibility of fog computing deployment based on docker containerization over raspberrypi, in: Proceedings of the 18th international conference on distributed computing and networking, pp. 1–10. doi:https://doi.org/10.1145/3007748.3007777.

Boateng, E.Y., Abaye, D.A., 2019. A review of the logistic regression model with emphasis on medical research. Journal of data analysis and information processing 7, 190–207.

Bora, D.J., 2017. A novel approach for color image edge detection using multidirectional sobel filter on hsv color space. Int. J. Comput. Sci. Eng 5, 154–159.

Brinker, T.J., Hekler, A., Utikal, J.S., Grabe, N., Schadendorf, D., Klode, J., Berking, C., Steeb, T., Enk, A.H., Von Kalle, C., 2018. Skin cancer classification using convolutional neural networks: systematic review. Journal of medical Internet research 20, e11936. doi:https://doi.org/10.2196/11936.

Bulbul, E., Cetin, A., Dogru, I.A., 2018. Human activity recognition using smartphones, in: 2018 2nd international symposium on multidisciplinary studies and innovative technologies (ismsit), IEEE. pp. 1–6. doi:https://doi.org/10.1109/ISMSIT.2018.8567275.

Canedo, J., Skjellum, A., 2016. Using machine learning to secure iot systems, in: 2016 14th annual conference on privacy, security and trust (PST), IEEE. pp. 219–222. doi:https://doi.org/10.1109/PST.2016.7906930.

Carter, B., 2020. Making cities smarter - security today. Https://securitytoday.com/Articles/2020/04/09/Making-Cities-Smarter.aspx.

Carvalho, T., De Rezende, E.R., Alves, M.T., Balieiro, F.K., Sovat, R.B., 2017. Exposing computer generated images by eye's region classification via transfer learning of vgg19 cnn, in: 2017 16th IEEE international conference on machine learning and applications (ICMLA), IEEE. pp. 866–870. doi:https://doi.org/10.1109/ICMLA.2017.00-47.

Cecilia, J.M., Cano, J.C., Morales-García, J., Llanes, A., Imbernón, B., 2020. Evaluation of clustering algorithms on gpu-based edge computing platforms. Sensors 20, 6335. doi:https://doi.org/10.3390/s20216335.

Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., Lopez, A., 2020. A comprehensive survey on support vector machine classification: Applications, challenges and trends. Neurocomputing 408, 189–215.

Charbuty, B., Abdulazeez, A., 2021. Classification based on decision tree algorithm for machine learning. Journal of Applied Science and Technology Trends 2, 20–28.

Chethan, K., Sinchana, G., Nataraj, K., Choodarathnakara, A., 2019. Analysis of image quality using sobel filter, in: 2019 Third International Conference on Inventive Systems and Control (ICISC), IEEE. pp. 526–531.

Chowdhury, M.H., Shuzan, M.N.I., Chowdhury, M.E., Mahbub, Z.B., Uddin, M.M., Khandakar, A., Reaz, M.B.I., 2020. Estimating blood pressure from the photoplethysmogram signal and demographic features using machine learning techniques. Sensors 20, 3127.

Cihlar, J., 2000. Land cover mapping of large areas from satellites: status and research priorities. International journal of remote sensing 21, 1093–1114. doi:https://doi.org/10.1080/014311600210092.

Clemencic, M., Mato, P., 2012. A cmake-based build and configuration framework, in: Journal of Physics: Conference Series, IOP Publishing. p. 052021. doi:https://doi.org/10.1088/1742-6596/396/5/052021.

Ctypes, 2023. ctypes-python — pypi.org. https://docs.python.org/3/library/ctypes.html. Accessed 25-Jan-2023.

Daher, A.W., Ferrari, E., Muselli, M., Chible, H., Caviglia, D.D., 2022. Vampire: vectorized automated ml pre-processing and post-processing framework for edge applications. Computing , 1–35doi:https://doi.org/10.1007/s00607-022-01096-z.

Daher, A.W., Rizik, A., Muselli, M., Chible, H., Caviglia, D.D., 2020a. Porting rulex machine learning software to the raspberry pi as an edge computing device, in: International Conference on Applications in Electronics Pervading Industry, Environment and Society, Springer. pp. 273–279. doi:https://doi.org/10.1007/978-3-030-66729-0_33.

Daher, A.W., Rizik, A., Muselli, M., Chible, H., Caviglia, D.D., 2021. Porting rulex software to the raspberry pi for machine learning applications on the edge. Sensors 21, 6526. doi:https://doi.org/10.3390/s21196526.

Daher, A.W., Rizik, A., Randazzo, A., Tavanti, E., Chible, H., Muselli, M., Caviglia, D.D., 2020b. Pedestrian and multi-class vehicle classification in radar systems using rulex software on the raspberry pi. Applied Sciences 10, 9113. doi:https://doi.org/10.3390/app10249113.

Das, S., Aranya, O.R.R., Labiba, N.N., 2019. Brain tumor classification using convolutional neural network, in: 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), IEEE. pp. 1–5.

Deepak, S., Ameer, P., 2019. Brain tumor classification using deep cnn features via transfer learning. Computers in biology and medicine 111, 103345. doi:https://doi.org/10.1016/j.compbiomed.2019.103345.

Dhillon, A., Verma, G.K., 2020. Convolutional neural network: a review of models, methodologies and applications to object detection. Progress in Artificial Intelligence 9, 85–112. doi:https://doi.org/10.1007/s13748-019-00203-0.

Dong, X., Yu, Z., Cao, W., Shi, Y., Ma, Q., 2020. A survey on ensemble learning. Frontiers of Computer Science 14, 241–258.

Edress, I., Al-Sabawi, E.A., Younus, M.D., 2021. Design of fractional-order sobel filters for edge detections, in: IOP Conference Series: Materials Science and Engineering, IOP Publishing. p. 012028. doi:https://doi.org/10.1088/1757-899X/1152/1/012028.

Eidinger, E., Enbar, R., Hassner, T., 2014. Age and gender estimation of unfiltered faces. IEEE Transactions on information forensics and security 9, 2170–2179. doi:https://doi.org/10.1109/TIFS.2014.2359646.

El-Hajj, C., Kyriacou, P.A., 2020. A review of machine learning techniques in photoplethysmography for the non-invasive cuff-less measurement of blood pressure. Biomedical Signal Processing and Control 58, 101870.

Fanconi, C., 2020. Skin Cancer: Malignant vs. Benign. URL: https://kaggle.com/fanconic/skin-cancer-malignant-vs-benign.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., Hutter, F., 2020. Auto-sklearn 2.0: The next generation. arXiv preprint arXiv:2007.04074 24.

Finean, R., 2021. Ppg heart beat for cognitive fatigue prediction. URL: https://www.kaggle.com/datasets/canaria/5-gamers.

Finean, R., Dillon, A., 2021. Predicting cognitive fatigue with photoplethysmography (ppg). URL: https://www.researchgate.net/project/Predicting-Cognitive-Fatigue-with-Photoplethysmography-PPG.

Fister, D., Fister, I., Jagric, T., Brest, J., 2018. A novel self-adaptive differential evolution for feature selection using threshold mechanism, in: 2018 IEEE symposium series on computational intelligence (SSCI), IEEE. pp. 17–24. doi:https://doi.org/10.1109/SSCI.2018.8628715.

Fiterau, M., Bhooshan, S., Fries, J., Bournhonesque, C., Hicks, J., Halilaj, E., Ré, C., Delp, S., 2017. Shortfuse: Biomedical time series representations in the presence of structured information, in: Machine Learning for Healthcare Conference, PMLR. pp. 59–74.

Fober, D., Letz, S., et al., 2018. Building faust with cmake, in: International Faust Conference.

Geitgey, A., 2020. face-recognition — pypi.org. https://pypi.org/project/face-recognition. [Accessed 01-Aug-2022].

Ghosh, S., Dasgupta, A., Swetapadma, A., 2019. A study on support vector machine based linear and non-linear pattern classification, in: 2019 International Conference on Intelligent Sustainable Systems (ICISS), IEEE. pp. 24–28.

Goldberger, A.L., Amaral, L.A., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E., 2000. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. circulation 101, e215–e220.

Gou, J., Ma, H., Ou, W., Zeng, S., Rao, Y., Yang, H., 2019. A generalized mean distance-based k-nearest neighbor classifier. Expert Systems with Applications 115, 356–372.

Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S., Yang, G.Z., 2019. Xai—explainable artificial intelligence. Science robotics 4, eaay7120. doi:https://doi.org/10.1126/scirobotics.aay712.

Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q., 2017. On calibration of modern neural networks, in: International conference on machine learning, PMLR. pp. 1321–1330.

Gupta, A., 2011. Male and female faces dataset. URL: https://www.kaggle.com/datasets/ashwingupta3012/male-and-female-faces-dataset.

Hajdarevic, K., Konjicija, S., Subasi, A., 2014. A low energy aprs-is client-server infrastructure implementation using raspberry pi, in: 2014 22nd Telecommunications Forum Telfor (TELFOR), IEEE. pp. 296–299. doi:https://doi.org/10.1109/TELFOR.2014.7034409.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The weka data mining software: an update. ACM SIGKDD explorations newsletter 11, 10–18. doi:https://doi.org/10.1145/1656274.1656278.

Hekler, A., Utikal, J.S., Enk, A.H., Hauschild, A., Weichenthal, M., Maron, R.C., Berking, C., Haferkamp, S., Klode, J., Schadendorf, D., et al., 2019. Superior skin cancer classification by the combination of human and artificial intelligence. European Journal of Cancer 120, 114–121. doi:https://doi.org/10.1016/j.ejca.2019.07.019.

Heuel, S., Rohling, H., 2012. Pedestrian classification in automotive radar systems, in: 2012 13th international radar symposium, IEEE. pp. 39–44. doi:https://doi.org/10.1109/IRS.2012.6233285.

Hodo, E., Bellekens, X., Hamilton, A., Dubouilh, P.L., Iorkyase, E., Tachtatzis, C., Atkinson, R., 2016. Threat analysis of iot networks using artificial neural network intrusion detection system, in: 2016 International Symposium on Networks, Computers and Communications (ISNCC), IEEE. pp. 1–6. doi:https://doi.org/10.1109/ISNCC.2016.7746067.

Hospedales, T., Antoniou, A., Micaelli, P., Storkey, A., 2021. Meta-learning in neural networks: A survey. IEEE transactions on pattern analysis and machine intelligence 44, 5149–5169.

Hosseini, M.P., Hosseini, A., Ahi, K., 2020. A review on machine learning for eeg signal processing in bioengineering. IEEE reviews in biomedical engineering 14, 204–218.

Infineon, 2022. Demo distance2go. URL: https://www.infineon.com/cms/en/product/evaluation-boards/demo-distance2go/.

Infineonradar, 2022. 24ghz radar. URL: https://www.infineon.com/cms/en/product/sensor/radar-sensors/radar-sensors-for-iot/24ghz-radar/bgt24mtr11/.

Izbicki, M., 2011. Converting images into time series for data mining. URL: https://izbicki.me/blog/converting-images-into-time-series-for-data-mining.html.

Jin, L.p., Dong, J., 2016. Ensemble deep learning for biomedical time series classification. Computational intelligence and neuroscience 2016. doi:https://doi.org/10.1155/2016/6212684.

Jovic, A., Kukolja, D., Friganovic, K., Jozic, K., Car, S., 2017. Biomedical time series preprocessing and expert-system based feature extraction in multisab platform, in: 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), IEEE. pp. 330–335. doi:https://doi.org/10.23919/MIPRO.2017.7973444.

Kadhim, M.A., Abed, M.H., 2019. Convolutional neural network for satellite image classification, in: Asian Conference on Intelligent Information and Database Systems, Springer. pp. 165–178. doi:https://doi.org/10.1007/978-3-030-14132-5_13.

Kalayci, T., Ozdamar, O., 1995. Wavelet preprocessing for automated neural network detection of eeg spikes. IEEE engineering in medicine and biology magazine 14, 160–166. doi:https://doi.org/10.1109/51.376754.

Kanawaday, A., Sane, A., 2017. Machine learning for predictive maintenance of industrial machines using iot sensor data, in: 2017 8th IEEE international conference on software engineering and service science (ICSESS), IEEE. pp. 87–90. doi:https://doi.org/10.1109/ICSESS.2017.8342870.

Karakatič, S., 2020. Evopreprocess—data preprocessing framework with nature-inspired optimization algorithms. Mathematics 8, 900. doi:https://doi.org/10.3390/math8060900.

Karimi, F., Wagner, C., Lemmerich, F., Jadidi, M., Strohmaier, M., 2016. Inferring gender from names on the web: A comparative evaluation of gender detection methods, in: Proceedings of the 25th International conference companion on World Wide Web, pp. 53–54. doi:https://doi.org/10.1145/2872518.2889385.

Khalid, S., Khalil, T., Nasreen, S., 2014. A survey of feature selection and feature extraction techniques in machine learning, in: 2014 science and information conference, IEEE. pp. 372–378. doi:https://doi.org/10.1109/SAI.2014.6918213.

Khlamov, S., Tabakova, I., Trunova, T., 2022. Recognition of the astronomical images using the sobel filter, in: 2022 29th International Conference on Systems, Signals and Image Processing (IWSSIP), IEEE. pp. 1–4.

Lapegna, M., Balzano, W., Meyer, N., Romano, D., 2021. Clustering algorithms on low-power and high-performance devices for edge computing environments. Sensors 21, 5395. doi:https://doi.org/10.3390/s21165395.

Levi, G., Hassner, T., 2015. Age and gender classification using convolutional neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp. 34–42. doi:https://doi.org/10.1109/CVPRW.2015.7301352.

Li, Z., Liu, F., Yang, W., Peng, S., Zhou, J., 2021. A survey of convolutional neural networks: analysis, applications, and prospects. IEEE transactions on neural networks and learning systems doi:https://doi.org/10.1109/TNNLS.2021.3084827.

Liaqat, S., Khan, S., Ihsan, M., Asghar, S., Ejaz, A., Bhatti, A., 2011. Automatic recognition of ground radar targets based on target rcs and short time spectrum variance, in: 2011 International Symposium on Innovations in Intelligent Systems and Applications, IEEE. pp. 164–167. doi:https://doi.org/10.1109/INISTA.2011.5946127.

Lu, C., Lin, D., Jia, J., Tang, C.K., 2014. Two-class weather classification, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3718–3725.

Lu, S., Lu, Z., Zhang, Y.D., 2019. Pathological brain detection based on alexnet and transfer learning. Journal of computational science 30, 41–47. doi:https://doi.org/10.1016/j.jocs.2018.11.008.

Lundberg, S., 2018. Welcome to the shap. URL: https://shap.readthedocs.io/en/latest/index.html.

Maksimović, M., Vujović, V., Davidović, N., Milošević, V., Perišić, B., 2014. Raspberry pi as internet of things hardware: performances and constraints. design issues 3, 1–6.

Martone, A., Zazzaro, G., Pavone, L., 2019. A feature extraction framework for time series analysis. ALLDATA 2019 , 13.

Meisenbacher, S., Turowski, M., Phipps, K., Rätz, M., Müller, D., Hagenmeyer, V., Mikut, R., 2022. Review of automated time series forecasting pipelines. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery , e1475doi:https://doi.org/10.1002/widm.1475.

Merrick, L., Taly, A., 2019. The explanation game: Explaining machine learning models with cooperative game theory. arXiv preprint arXiv:1909.08128 doi:https://doi.org/10.48550/arXiv.1909.08128.

Mirjalili, S., Mirjalili, S., 2019. Evolutionary multi-layer perceptron. Evolutionary Algorithms and Neural Networks: Theory and Applications , 87–104.

Moghaddam, B., Yang, M.H., 2000. Gender classification with support vector machines, in: Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580), IEEE. pp. 306–311. doi:https://doi.org/10.1109/AFGR.2000.840651.

Mohanna, A., Gianoglio, C., Rizik, A., Valle, M., 2022. A convolutional neural network-based method for discriminating shadowed targets in frequency-modulated continuous-wave radar systems. Sensors 22, 1048. doi:https://doi.org/10.3390/s22031048.

Moody, G.B., Mark, R.G., 2001. The impact of the mit-bih arrhythmia database. IEEE Engineering in Medicine and Biology Magazine 20, 45–50. doi:httpsL://doi.org/10.1109/51.932724.

Mukti, I.Z., Biswas, D., 2019. Transfer learning based plant diseases detection using resnet50, in: 2019 4th International conference on electrical information and communication technology (EICT), IEEE. pp. 1–6. doi:https://doi.org/10.1109/EICT48899.2019.9068805.

Murshed, M.S., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., Hussain, F., 2021. Machine learning at the network edge: A survey. ACM Computing Surveys (CSUR) 54, 1–37. doi:https://doi.org/10.1145/3469029.

Muselli, M., 2005. Switching neural networks: A new connectionist model for classification, in: Neural Nets. Springer, pp. 23–30. doi:https://doi.org/10.1007/11731177_4.

Muselli, M., 2012. Extracting knowledge from biomedical data through logic learning machines and rulex. EMBnet. journal 18, 56–58. doi:https://doi.org/10.14806/ej.18.B.549.

Muselli, M., 2022. Rulex ai. URL: https://www.rulex.ai/.

Muselli, M., Ferrari, E., 2009. Coupling logical analysis of data and shadow clustering for partially defined positive boolean function reconstruction. IEEE Transactions on Knowledge and Data Engineering 23, 37–50. doi:https://doi.org/10.1109/TKDE.2009.206.

Muselli, M., Quarati, A., 2005. Reconstructing positive boolean functions with shadow clustering, in: Proceedings of the 2005 European Conference on Circuit Theory and Design, 2005., IEEE. pp. III–377. doi:https://doi.org/10.1109/ECCTD.2005.1523139.

Muthukrishnan, S., Krishnaswamy, H., Thanikodi, S., Sundaresan, D., Venkatraman, V., 2020. Support vector machine for modelling and simulation of heat exchangers. Thermal Science 24, 499–503.

Norris, C., McCahill, M., Wood, D., 2004. The growth of cctv: a global perspective on the international diffusion of video surveillance in publicly accessible space. Surveillance & Society 2. doi:https://doi.org/10.24908/ss.v2i2/3.3369.

Novac, P.E., Boukli Hacene, G., Pegatoquet, A., Miramond, B., Gripon, V., 2021. Quantization and deployment of deep neural networks on microcontrollers. Sensors 21, 2984. doi:https://doi.org/10.3390/s21092984.

Numpy, 2023. numpy-python — pypi.org. https://pypi.org/project/numpy/. Accessed 25-Jan-2023.

Nusinovici, S., Tham, Y.C., Yan, M.Y.C., Ting, D.S.W., Li, J., Sabanayagam, C., Wong, T.Y., Cheng, C.Y., 2020. Logistic regression was as good as machine learning for predicting major chronic diseases. Journal of clinical epidemiology 122, 56–69.

Okwuashi, O., Ndehedehe, C.E., 2020. Deep support vector machine for hyperspectral image classification. Pattern Recognition 103, 107298.

OpenCV, 2023. opencv-python — pypi.org. https://pypi.org/project/opencv-python/. Accessed 25-Jan-2023.

Parodi, S., Manneschi, C., Verda, D., Ferrari, E., Muselli, M., 2018. Logic learning machine and standard supervised methods for hodgkin's lymphoma prognosis using gene expression data and clinical variables. Health Informatics Journal 24, 54–65. doi:https://doi.org/10.1177/1460458216655188.

Parteek, 2020. Multi-class Weather Dataset. URL: https://kaggle.com/pratik2901/multiclass-weather-dataset.

Pintelas, E., Liaskos, M., Livieris, I.E., Kotsiantis, S., Pintelas, P., 2020. Explainable machine learning framework for image classification problems: case study on glioma cancer prediction. Journal of imaging 6, 37. doi:https://doi.org/10.3390/jimaging6060037.

Premsankar, G., Di Francesco, M., Taleb, T., 2018. Edge computing for the internet of things: A case study. IEEE Internet of Things Journal 5, 1275–1284. doi:https://doi.org/10.1109/JIOT.2018.2805263.

Prophet, R., Hoffmann, M., Ossowska, A., Malik, W., Sturm, C., Vossiek, M., 2018. Pedestrian classification for 79 ghz automotive radar systems, in: 2018 IEEE Intelligent Vehicles

Symposium (IV), IEEE. pp. 1265–1270. doi:https://doi.org/10.1109/IVS.2018.8500554.

Pycaret, 2023. pycaret-python — pypi.org. https://pypi.org/project/pycaret/. Accessed 25-Jan-2023.

Qureshi, M.B., Afzaal, M., Qureshi, M.S., Fayaz, M., et al., 2021. Machine learning-based eeg signals classification model for epileptic seizure detection. Multimedia Tools and Applications 80, 17849–17877. doi:https://doi.org/10.1007/s11042-021-10597-6.

Ramasubramanian, K., Singh, A., 2019. Deep learning using keras and tensorflow, in: Machine Learning Using R. Springer, pp. 667–688. doi:https://doi.org/10.1007/978-1-4842-4215-5_11.

Rasheed, K., Qayyum, A., Qadir, J., Sivathamboo, S., Kwan, P., Kuhlmann, L., O'Brien, T., Razi, A., 2020. Machine learning for predicting epileptic seizures using eeg signals: A review. IEEE Reviews in Biomedical Engineering 14, 139–155.

Rasp, S., Lerch, S., 2018. Neural networks for postprocessing ensemble weather forecasts. Monthly Weather Review 146, 3885–3900. doi:https://doi.org/10.1175/MWR-D-18-0187.1.

Reda, M., 2021. Satellite Image Classification. URL: https://kaggle.com/mahmoudreda55/satellite-image-classification.

Ren, H., Anicic, D., Runkler, T.A., 2021. Tinyol: Tinyml with online-learning on microcontrollers, in: 2021 international joint conference on neural networks (IJCNN), IEEE. pp. 1–8.

Rizik, A., Randazzo, A., Vio, R., Delucchi, A., Chible, H., Caviglia, D.D., 2019. Feature extraction for human-vehicle classification in fmcw radar, in: 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), IEEE. pp. 131–132. doi:https://doi.org/10.1109/ICECS46596.2019.8965072.

Rizik, A., Tavanti, E., Chible, H., Caviglia, D.D., Randazzo, A., 2021. Cost-efficient fmcw radar for multi-target classification in security gate monitoring. IEEE Sensors Journal 21, 20447–20461. doi:https://doi.org/10.1109/JSEN.2021.3095674.

Roh, D., Shin, H., 2021. Recurrence plot and machine learning for signal quality assessment of photoplethysmogram in mobile environment. Sensors 21, 2188.

Rohling, H., Heuel, S., Ritter, H., 2010. Pedestrian detection procedure integrated into an 24 ghz automotive radar, in: 2010 IEEE Radar Conference, pp. 1229–1232. doi:https://doi.org/10.1109/RADAR.2010.5494432.

Roser, M., Moosmann, F., 2008. Classification of weather situations on single color images, in: 2008 IEEE intelligent vehicles symposium, IEEE. pp. 798–803.

Rymarczyk, T., Kozłowski, E., Kłosowski, G., Niderla, K., 2019. Logistic regression for machine learning in process tomography. Sensors 19, 3400.

Sahoo, S., Dash, M., Behera, S., Sabut, S., 2020. Machine learning approach to detect cardiac arrhythmias in ecg signals: A survey. Irbm 41, 185–194.

Sajjad, M., Khan, S., Muhammad, K., Wu, W., Ullah, A., Baik, S.W., 2019. Multi-grade brain tumor classification using deep cnn with extensive data augmentation. Journal of computational science 30, 174–182. doi:ttps://doi.org/10.1016/j.jocs.2018.12.003.

Salmi, N., Rustam, Z., 2019. Naïve bayes classifier models for predicting the colon cancer, in: IOP conference series: materials science and engineering, IOP Publishing. p. 052068.

Sanchez-Iborra, R., Skarmeta, A.F., 2020. Tinyml-enabled frugal smart objects: Challenges and opportunities. IEEE Circuits and Systems Magazine 20, 4–18.

Schonlau, M., Zou, R.Y., 2020. The random forest algorithm for statistical learning. The Stata Journal 20, 3–29.

Shahraki, A., Abbasi, M., Haugen, Ø., 2020. Boosting algorithms for network intrusion detection: A comparative evaluation of real adaboost, gentle adaboost and modest adaboost. Engineering Applications of Artificial Intelligence 94, 103770.

Singh, J., Banerjee, R., 2019. A study on single and multi-layer perceptron neural network, in: 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), IEEE. pp. 35–40.

Skolnik, M., 2001. Introduction to Radar Systems. McGraw-Hill. doi:https://doi.org/10.1108/sr.1999.08719bae.001.

Speiser, J.L., Miller, M.E., Tooze, J., Ip, E., 2019. A comparison of random forest variable selection methods for classification prediction modeling. Expert systems with applications 134, 93–101.

Sree, V., Mapes, J., Dua, S., Lih, O.S., Koh, J.E., Ciaccio, E.J., Acharya, U.R., et al., 2021. A novel machine learning framework for automated detection of arrhythmias in ecg segments. Journal of Ambient Intelligence and Humanized Computing 12, 10145–10162. doi:https://doi.org/10.1007/s12652-020-02779-1.

Sreedharan, M., Khedr, A.M., El Bannany, M., 2020. A multi-layer perceptron approach to financial distress prediction with genetic algorithm. Automatic Control and Computer Sciences 54, 475–482.

Stieler, F., Rabe, F., Bauer, B., 2021. Towards domain-specific explainable ai: model interpretation of a skin image classifier using a human approach, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1802–1809.

Sudharsan, B., Breslin, J.G., Ali, M.I., 2020a. Edge2train: A framework to train machine learning models (svms) on resource-constrained iot edge devices, in: Proceedings of the 10th International Conference on the Internet of Things, pp. 1–8. doi:https://doi.org/10.1145/3410992.3411014.

Sudharsan, B., Breslin, J.G., Ali, M.I., 2020b. Rce-nn: a five-stage pipeline to execute neural networks (cnns) on resource-constrained iot edge devices, in: Proceedings of the 10th International Conference on the Internet of Things, pp. 1–8.

Sudharsan, B., Salerno, S., Nguyen, D.D., Yahya, M., Wahid, A., Yadav, P., Breslin, J.G., Ali, M.I., 2021. Tinyml benchmark: Executing fully connected neural networks on commodity microcontrollers, in: 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), IEEE. pp. 883–884.

Takahashi, D., 2019. Fast Fourier transform algorithms for parallel computers. Springer. doi:https://doi.org/10.1007/978-981-13-9965-7.

Tang, P., Wang, H., Kwong, S., 2017. G-ms2f: Googlenet based multi-stage feature fusion of deep cnn for scene recognition. Neurocomputing 225, 188–197. doi:https://doi.org/10.1016/j.neucom.2016.11.023.

Tanwani, A.K., Afridi, J., Shafiq, M.Z., Farooq, M., 2009. Guidelines to select machine learning scheme for classification of biomedical datasets, in: European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, Springer. pp. 128–139. doi:https://doi.org/10.1007/978-3-642-01184-9_12.

Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K., 2013. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms, in: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 847–855. doi:https://doi.org/10.1145/2487575.2487629.

Tyralis, H., Papacharalampous, G., Langousis, A., 2019. A brief review of random forests for water scientists and practitioners and their recent history in water resources. Water 11, 910.

Venkatesan, C., Karthigaikumar, P., Paul, A., Satheeskumaran, S., Kumar, R., 2018. Ecg signal preprocessing and svm classifier-based abnormality detection in remote healthcare applications. IEEE Access 6, 9767–9773. doi:https://doi.org/10.1109/ACCESS.2018.2794346.

Vermeire, T., Brughmans, D., Goethals, S., de Oliveira, R.M.B., Martens, D., 2022. Explainable image classification with evidence counterfactual. Pattern Analysis and Applications , 1–21doi:https://doi.org/10.1007/s10044-021-01055-y.

Vujović, V., Maksimović, M., 2014. Raspberry pi as a wireless sensor node: Performances and constraints, in: 2014 37th international convention on information and communication technology, electronics and microelectronics (MIPRO), IEEE. pp. 1013–1018. doi:https://doi.org/10.1109/MIPRO.2014.6859717.

Wan, A., Dunlap, L., Ho, D., Yin, J., Lee, S., Jin, H., Petryk, S., Bargal, S.A., Gonzalez, J.E., 2020. Nbdt: neural-backed decision trees. arXiv preprint arXiv:2004.00221 .

Wang, F., Li, Z., He, F., Wang, R., Yu, W., Nie, F., 2019. Feature learning viewpoint of adaboost and a new algorithm. IEEE Access 7, 149890–149899.

Wang, J., Liu, P., She, M.F., Nahavandi, S., Kouzani, A., 2013. Bag-of-words representation for biomedical time series classification. Biomedical Signal Processing and Control 8, 634–644. doi:https://doi.org/10.1016/j.bspc.2013.06.004.

Wang, J.X., 2021. Meta-learning in natural and artificial intelligence. Current Opinion in Behavioral Sciences 38, 90–95.

Wang, W., Sun, D., 2021. The improved adaboost algorithms for imbalanced data classification. Information Sciences 563, 358–374.

Ward, L., Agrawal, A., Choudhary, A., Wolverton, C., 2016. A general-purpose machine learning framework for predicting properties of inorganic materials. npj Computational Materials 2, 1–7.

Wasimuddin, M., Elleithy, K., Abuzneid, A.S., Faezipour, M., Abuzaghleh, O., 2020. Stages-based ecg signal analysis from traditional signal processing to machine learning approaches: a survey. IEEE Access 8, 177782–177803.

Weiss, K., Khoshgoftaar, T.M., Wang, D., 2016. A survey of transfer learning. Journal of Big data 3, 1–40. doi:https://doi.org/10.1186/s40537-016-0043-6.

Wojtkiewicz, A., Misiurewicz, J., Nalecz, M., Jedrzejewski, K., Kulpa, K., 1997. Two-dimensional signal processing in fmcw radars. Proc. XX KKTOiUE , 475–480.

Xu, D., Li, T., Li, Y., Su, X., Tarkoma, S., Jiang, T., Crowcroft, J., Hui, P., 2020. Edge intelligence: Architectures, challenges, and applications. arXiv preprint arXiv:2003.12172 .

Yang, Z., Li, M., Ai, H., 2006. An experimental study on automatic face gender classification, in: 18th International Conference on Pattern Recognition (ICPR'06), IEEE. pp. 1099–1102. doi:https://doi.org/10.1109/ICPR.2006.247.

Yazici, M.T., Basurra, S., Gaber, M.M., 2018. Edge machine learning: Enabling smart internet of things applications. Big data and cognitive computing 2, 26. doi:https://doi.org/10.3390/bdcc2030026.

Yoo, S.H., Geng, H., Chiu, T.L., Yu, S.K., Cho, D.C., Heo, J., Choi, M.S., Choi, I.H., Cung Van, C., Nhung, N.V., et al., 2020. Deep learning-based decision-tree classifier for covid-19 diagnosis from chest x-ray imaging. Frontiers in medicine 7, 427.

Yu, W., Liang, F., He, X., Hatcher, W.G., Lu, C., Lin, J., Yang, X., 2017. A survey on the edge computing for the internet of things. IEEE access 6, 6900–6919. doi:https://doi.org/10.1109/ACCESS.2017.2778504.

Yu, Y., Si, X., Hu, C., Zhang, J., 2019. A review of recurrent neural networks: Lstm cells and network architectures. Neural computation 31, 1235–1270. doi:https://doi.org/10.1162/neco_a_01199.

Zhang, J.w., Wang, L.p., Liu, X., Zhu, H.h., Dong, J., 2010. Chinese cardiovascular disease database (ccdd) and its management tool, in: 2010 IEEE International Conference on BioInformatics and BioEngineering, IEEE. pp. 66–72.

Zhang, Q., Hu, W., Liu, Z., Tan, J., 2020. Tbm performance prediction with bayesian optimization and automated machine learning. Tunnelling and Underground Space Technology 103, 103493. doi:https://doi.org/10.1016/j.tust.2020.103493.

Zhang, X., Wang, Y., Shi, W., 2018. Pcamp: Performance comparison of machine learning packages on the edges, in: USENIX workshop on hot topics in edge computing (HotEdge 18). URL: https://www.usenix.org/conference/hotedge18/presentation/zhang.

Zhang, Z., Ma, H., Fu, H., Zhang, C., 2016. Scene-free multi-class weather classification on single images. Neurocomputing 207, 365–373. doi:https://doi.org/10.1016/j.neucom.2016.05.015.

Zhou, K., Yang, Y., Qiao, Y., Xiang, T., 2021. Domain adaptive ensemble learning. IEEE Transactions on Image Processing 30, 8008–8018.

Appendix A

Explaining *VAMPIRE FE2* in Detail

A.1 Further Explaining the *VAMPIRE FE2* Algorithm by Example

This appendix aims to provide a step-by-step application of **Algorithm (8.5)** from *VAMPIRE FE2* in order to clarify the pseudo-code by supplying real data taken from the Python interpreter and to clear up any ambiguity. Initially, in the array below, we can find all the zero-crossings of an input signal. To determine the crossings, the minimum local maxima and the maximum local minima are used as reference points to detect crossings rather than zero. The name of the array from **Algorithm (8.5)** to be computed is *Cross*.

Cross:
[***100, 169, 201, 269, 293, 335, 361, 379****, 405, 484, 506, 576, 604, 682, 708, 777, 811, 880, 909, 990, 1022, 1081, 1116, 1182, 1211, 1279, 1309, 1373, 1402, 1475, 1503, 1570, 1600, 1683, 1705, 1779, 1809, 1884, 1914, . . .]*

As shown in Cross for every Cross[i]:

Cross [0] = 100 Cross [2] = 201
Cross [1] = 169 Cross [3] = 269 . . .

In the array *Cross*, *Cross[i]* represents a Negative-to-Positive crossing *(N-to-P)* if the index is zero or is an even number, and *Cross[i]* represents a Positive-to-Negative crossing *(P-to-N)* if the index *i* is odd. For every pair of crossings, the maximum voltage should be computed. The array *Patt* is used for storing the maximum value in the signal between every

N-to-P and *P-to-N* pair. *Patt []* represents the maximum voltage between every crossing pairs:

Patt:
*[**57.878, 64.533, 64.724, 3.538**, 42.306, 45.081, 51.199, 49.983, 42.531, 48.166, 28.105, 39.79, 53.489, 62.38, 61.229, 54.553, 60.818, 56.603, 54.73, 51.671, 57.029, 66.834, 65.955, 32.123, 34.94, 30.187, 55.639, 43.408, 49.639, 62.696, 59.278, 64.183, 48.3, 65.897, 142.026, 72.131, 511.249, 338.739, 367.466,10.085, 101.134, 105.386, 47.537, 28.63, 52.903, 43.38]*

Consequently, after computing the maximum voltages between crosses in the *Patt []* array, the window-size of every beat or repeating series of crossings needs to be determined. In order to do this, we can rely on the mean for every possible window size. The array *Mean []* below contains all the computed means:

Mean:
*[0.0, 0.0, 0.0, **47.668**, 43.775, 38.912, 35.531, 47.142, 47.198, 47.97, 42.196, 39.648, 42.388, 45.941, 54.222, 57.913, 59.745, 58.301, 56.676, 55.956, 55.008, 57.566, 60.372, 55.485, 49.963, 40.801, 38.222, 41.044, 44.718, 52.846, 53.755, 58.949, 58.614, 59.414, 80.102, 82.089, 197.826, 266.036 . . .]*

For example:
For Window = 4
Mean [3] = Mean of: Patt [3], Patt [3-1], Patt [3-2], Patt [3-3]
Mean [3] = Mean of: Patt [3], Patt [2], Patt [1], Patt [0]
Mean [3] = (57.878 + 64.533 + 64.724 + 3.538)/ (4 = Window Size)
Mean [3] = 47.668

Therefore, after computing the mean for some window size, as in **Algorithm 4.5**, the standard deviation is computed and if *STD* is higher than a certain *Threshold*, the *Window* variable is incremented by one. This will go on recursively until the correct *Window* size is determined. It should also be pointed out that multiples of a correct window-size hold and may incorrectly detect *Window*. In case *Window = 3*, Windows of sizes 2, 4, and 5 will give a higher *STD* for *Mean []*, which implies that this is an incorrect *Window*.

For example:

$$\textit{Mean (Window size of 3)} = \textit{Mean (Window size of 3*N)} \tag{A.1}$$

If we choose for *Window* a starting point of 4, instead of starting at 2, and the correct Window is 3, *Window* may be detected as 6. This is true since the mean of array *Mean []* with *Window* size 3 is almost equal to the array *Mean []* of *Window* size *3*N*. A description of this is presented in *Equation (A1)*.

After detecting *Corss* and *Window* for that series, *Crosspx* is generated from the collection of *Cross* pairs of size *2*Window*. **Algorithm 4.5** demonstrated how to collect the first and last crossings from *Cross* into *Crosspx* of size *Window*2*. The *Crosspx []* array is used to transform the signal into the D-Domain which in terms is used to extract the features for ML classification.

Crosspx:
*[**100, 379**, 405, 777, 811, 1182, 1211, 1570, 1600, 1978, 2012, 2329, 2365, 2726, 2761, 3097, 3137, 3508, 3541, 3853, 3886,...]*

Appendix B

Tutorial for Running Low-level C-Code through the *Ctypes* Python Interface for TS Extraction

B.1 C-Code structure used in compilation

This appendix provides a detailed description on how to low-level C-Code through Python using the Ctypes module, where this section relates to the low-level part to be compiled to consequently generate a shared library file. This has been discussed within chapters 5 and 6 where the Rulex software has been ported by means of compiling C/Cpp code using the basic procedure discussed in chapter 7. Therefore, since the code in the former chapter has been compiled/ported from scratch, in this appendix, a detailed recipe will be outlined describing step-by-step the procedure for running code remotely on a Linux system (Raspberry Pi) through a high-level programming language (Python).

Firstly, the decorator DLLIMPORT should be declared. This addition instructs the compiler to when the shared library file will be exported the required functions (which are to be called by Python) and when to import them into the running program:

```
#if BUILDING_DLL
        #define DLLIMPORT __declspec(dllexport)
#else
        #define DLLIMPORT __declspec(dllimport)
#endif
```

Consequently, DLLIMPORT is used before initializing each function required by Ctypes. In this Appendix, we will assume the following function "Image_to_TS_Engine" which converts an image into a TS equivalent using Algorithm 7.1 which has been compiled to run on the Raspberry Pi 4. An example of this function setup is outlined below:

```
DLLIMPORT struct  tsStruct  Image_to_TS_Engine(struct imStruct image);
```

Additionally, every variable which is used as input or output of this function should also be preceded by the same decorator in order to have the function "Image_to_TS_Engine" operational within the Python environment. The input is a 112x112 Sobel-filtered image which is implemented as a structure imStruct, and the output is another structure of six TS representing each statistical function as described in Algorithm 7.1. the C language format of these variables is shown here:

```
DLLIMPORT typedef struct imStruct imStruct;
DLLIMPORT typedef struct tsStruct tsStruct;

typedef struct imStruct {
        int im[112][112];
} imStruct;

typedef struct tsStruct {
        float ts[6][360];
} tsStruct;
```

Moreover, in the case of generating of a windows library ".DLL" file rather than a Linux ".SO" file in case of testing purposes, the following source code should be included in the main file as recommended by some C/Cpp IDE's:

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL,DWORD fdwReason,LPVOID lpvReserved)
{
        switch(fdwReason)
        {
                case DLL_PROCESS_ATTACH:
                {
                        break;
                }
                case DLL_PROCESS_DETACH:
                {
                        break;
                }
                case DLL_THREAD_ATTACH:
                {
                        break;
                }
                case DLL_THREAD_DETACH:
                {
                        break;
                }
        }

        return TRUE;
}
```

B.2 Python C-Class Variables and the *Ctypes* interface

Furthermore, the two C structures which are compiled in C should be re-declared with the same dimensions inside the Python part and need to be properly interfaced for optimal performance. Since inefficiently converting the 2D C structure as a Numpy arrays may render the act of generating the shared library useless or at the very least much slower than properly interfacing a classed C function through the Ctypes module.

Therefore, imStruct and tsStruct have to be initialized inside Python primarily using the same dimensions in the form of Ctypes structures class as shown in the Python code below:

```
class imStruct(ctypes.Structure):
        _fields_ = [("im", (ctypes.c_int32 * 112)*112)]

class tsStruct(ctypes.Structure):
        _fields_ = [("ts", (ctypes.c_float * 360 )*6)]
```

After compiling the C-Code, a shared library file is generated which is in our case named "Clibrary.so". Using the Ctypes pre-installed package, the compiled version of the functions may be accessed using the "CDLL" module which in terms is employed to call the "Image_to_TS_Engine" function through the described Ctypes classes, where the inputs and outputs are to be initialized. The required commands to be used are shown here:

```
Clibrary = ctypes.CDLL("./Clibrary.so")
Clibrary.Image_to_TS_Engine.restype = tsStruct
ts_s = tsStruct()
im_s = imStruct()
```

After properly setting the return type using the restype command, the Python variables used to store the inputs and outputs of the C function are initialized. However, to convert the C structures into Python Numpy arrays (Which is the best choice for performance in Python), the simple approach is to set each value in each 2D array by index using for loops. Although this approach may be valid, however, it drastically degrades the performance since Python for loop or considerably slow and the process of using C functions wouldn't be as beneficial.

To address this issue, the "Numpy ctypeslib" sub-module has been employed for the conversion from "Numpy to C" in case of the input image, and from "C to Numpy" in case of the TS return type. The source which was used to perform the Image-to-TS conversion is show here:

```
im_s.im = np.ctypeslib.as_ctypes(input_sobel_image.astype(np.int32))
ts_s =Clibrary.Image_to_TS_Engine(im_s)
ts_ = np.ctypeslib.as_array(ts_s.ts)
```

The code previously described in this appendix is applied per image in the inference stage on the IoT board (Raspberry Pi 4) as a deployment setup. However, in the training phase in order to generate a dataset, the TSs are generated for all images in each dataset. Consequently, depending on each image, its corresponding TS may vary when compared to other samples, therefore, interpolation should be applied after having generated the dataset such data all the smaller TS are resized using the OpenCV library till the maximum length available after conversion. The efficient source code for this conversion may be observed as follows:

```
max = len(ts[0])
for i in range(len(ts)):
        if max < len(ts[i]):
                max = len(ts[i])
ts__ = np.zeros((len(x), max))
for i in range(len(ts)):
        ts__[i] =np.array(cv2.resize(ts[i], (1,max))).flatten()
ts__ =np.array(cv2.resize(ts__, (1,max))).flatten()
xsamples = np.arange(0,len(ts__)+1,max)
```