MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES

IN PURE AND APPLIED SCIENCES

HIGH LEVEL SYNTHESIS IMPLEMENTATION OF CLASSIFICATION ALGORITHMS FOR BREAST CANCER DETECTION ON XILINX FPGAS

ASLIHAN HACER TÜFEKCİOĞLU

MASTER THESIS

Department of Electrical and Electronics Engineering

Thesis Supervisor

Asst. Prof. Dr. Salih BAYAR

İSTANBUL, 2024

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES

IN PURE AND APPLIED SCIENCES

HIGH LEVEL SYNTHESIS IMPLEMENTATION OF CLASSIFICATION ALGORITHMS FOR BREAST CANCER DETECTION ON XILINX FPGAS

ASLIHAN HACER TÜFEKCİOĞLU

MASTER THESIS

Department of Electrical and Electronics Engineering

Thesis Supervisor

Asst. Prof. Dr. Salih BAYAR

İSTANBUL, 2024

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

Aslihan Hacer TUFEKCIOGLU a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended her thesis entitled "High Level Synthesis implementation of Classification Algorithms for Breast Cancer Detection on Xilinx FPGAs", on September , 2024 and has been found to be satisfactory by the jury members.

Jury Members

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Aslıhan Hacer TUFEKCIOGLU be granted the degree of Master of Science in Department of Electrical and Electronics Engineering on September , 2024. (Resolution no:).

Director of the Institute

ACKNOWLEDGMENT

I would like to thank Asst. Prof. Dr. Salih BAYAR for guiding me to his important publications and for his leadership on artificial intelligence and Xilinx FPGAs. He assisted me in every doubt that I encountered with his acquaintance and experience.

Also, I am very thankful to my family for supporting me during my education life and writing this thesis.

CONTENTS

ÖZET

MAKİNE ÖĞRENMESİ SINIFLANDIRMA ALGORİTMALARININ XİLİNX FPGA ÜZERİNDE GERÇEKLENMESİ

Tezin amacı göğüs kanseri datasındaki değerlere göre K En Yakın Komşu , Karar Ağacı Öğrenimi, Rastgele Orman, Lojistik Regresyon ve Naif Bayes makine öğrenimi algoritmalarını kullanarak hem yüksek seviye programlama dilleri vasıtası ile hem de FPGA de sınıflandırma yapmaktır.

Bu tez, makine öğreniminin donanım ve yazılım uygulamasına odaklanmaktadır. Donanım uygulaması, sistemin performansını artırır ve gerçek zamanlı uygulamalar için güç tüketimini azaltır. FPGA, yazılım algoritmalarının donanımsal olarak uygulanmasında kullanılacaktır. FPGA de makine öğrenmesi sınıflandırma algoritmasını uygulamak için önerilmiştir. FPGA teknolojisi, makine dili algoritması programlama dilleri olan VHDL veya Verilog gibi bir donanım dilleri ile programlanabilmektedir. Ayrıca, FPGA makine öğrenmesi sınıflandırma uygulamaları için yüksek performans sağlar. Sınıflandırma algoritmaları, ortamdan alınan veriler üzerinden karar verebilir. Sınıflandırma yöntemlerinden Lojistik Regresyon, Naif Bayes Sınıflandırması, Karar Ağacı Öğrenimi, Rastgele Orman ve K En Yakın Komşu Algoritmasıdır . Yöntemlerin her biri, verileri sınıflandırmak için farklı yöntemler kullanarak bir sonuç elde etmeyi amaçlar. Önerilen sınıflandırma tasarımları, Xilinx Zynq-7000 SoC ZC702 FPGA platformunda toplam 699 hastalıklı ve sağlıklı bireyin bilgilerini içeren data üzerinde test edilmiştir.Her sınıflandırma algoritması için, başarı oranı (örneğin doğruluk), uygulama süresi, toplam alan (örneğin dilimler veya arama tabloları (LUT) cinsinden), maksimum saat frekansı ve güç tüketimi gibi bilgiler elde edilmesi amaçlanmaktadır.

Bu anlamda, bu tez karar verme sürecinde ilk aşamadan son aşamaya kadar olan tüm adımları içerecektir. İlk aşama, makine öğrenme sınıflandırma algoritmalarının yazılım uygulamasıdır. Bu aşamada hem C/C++ hem de Python dilleri kullanılmıştır. İkinci aşama, bir vaka çalışmasının uygulanması olacak ve son aşama ise bu algoritmaların başarı oranı (örneğin doğruluk), uygulama süresi, toplam alan (örneğin dilimler veya arama tabloları (LUT) cinsinden), maksimum saat frekansı ve güç tüketimi gibi değerlerin yazılım tabanlı sonuçlarla kıyaslanması şeklinde olacaktır.

ABSTRACT

HIGH LEVEL SYNTHESIS IMPLEMENTATION OF CLASSIFICATION ALGORITHMS FOR BREAST CANCER DETECTION ON XILINX FPGAs

The aim of the thesis is to classify using machine learning algorithms such as K Nearest Neighbor, Decision Tree Learning, Random Forest, Logistic Regression and Naive Bayes Classification based on the values in breast cancer data with high level programming languages and FPGA.

This thesis focuses on the hardware and software implementation of machine learning. The hardware implementation can enhance the performance of the system and reduce the power consumption for real-time applications. FPGA will be used for the implementation of software algorithms as hardware. Hardware design is proposed to implement machine learning classifiers on FPGA. FPGA technology provides a way to concentrate on the balance between programming machine language algorithms and programming in a hardware language such as VHDL or Verilog. Also, it provides high performance for those kinds of applications. Classification algorithms can make a decision on the received data from the environment. The most important classification methods are Logistic Regression, Naive Bayes Classification, Decision Tree Learning, Random Forest algorithm and K Nearest Neighbor Algorithm (KNN). Each of the methods aims to achieve a result using different ways in order to classify data. The proposed classification designs were tested on the Xilinx Zynq-7000 SoC ZC702 FPGA platform on data containing the information of a total of 699 cancerous and non-cancerous individuals. For each candidate classification algorithm, it is aimed to obtain information such as the success rate (e.g. accuracy), execution time, total

area (e.g. in terms of Slices or Look Up Tables (LUT)), maximum clock frequency, and power consumption.

In this sense, this thesis will include all the steps from the first stage to the final stage in the decision-making process. The first stage is software implementation of machine learning classification algorithms implemented as a high level language C/ C++ and Python ,the second stage will be the implementation of a case study in image processing domain and the last stage will be the performance evaluation of these algorithms in terms of success rate (e.g. accuracy), execution time, total area (e.g. in terms of Slices or Look Up Tables (LUT)), maximum clock frequency, and power consumption.

SYMBOLS

$(x, x_i) = \sqrt{\Sigma\,(x_i - x_{ii})^2}$: Euclidean Distance

$z = b + w_0 * x_0 + w_1 * x_1 + \ldots\ldots\ldots + w_n * x_n$: Forward Propagation

$\hat{y} = sigmoid(z) = \frac{1}{a+e^{-z}}$: Sigmoid Function

$J(\theta) = \frac{1}{n}\Sigma\, log\hat{y} - (1 - y)log(1 - \hat{y}) + \frac{\lambda}{2n}(\Sigma\,\theta^2)$: Cost Function

$w := w - \alpha\,\nabla_\theta J(\theta)$: Updating weight values

$b := b - \alpha\,\nabla_\theta J(\theta)$: Updating bias value.

ABBREVIATIONS

FPGA: Field-Programmable Gate Array

HLS: High-Level Synthesis

HDL: Hardware Description Language

k-NN: k-Nearest Neighbors

CAD: Computer-Aided Design

MRI: Magnetic Resonance Imaging

LUT: Look-up Table

UCI: University of California Irvine

MUX: Multiplexer

ANN: Artificial Neural Network

MLP: Multi-Layer Perceptron

GRU: Gated Recurrent Unit

ISFET: Ion-Sensitive Field Effect Transistor

DT: Decision Tree

RF: Random Forest

NB: Naive Bayes

HQC: High-quality cryptography

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Field-Programmable Gate Array (FPGA) technology has become a significant tool for high-performance computing, data processing, and custom hardware acceleration applications in today's world. FPGAs are known for their capabilities in providing hardware acceleration optimized for specific workloads. This demand has surged due to the growth of data-centric applications where performance, efficiency, and low latency are crucial (Jemal et al., 2011). FPGAs have emerged as an attractive hardware platform to accelerate computationally intensive tasks with parallel processing capabilities.

Traditionally, developing FPGA-based designs required a deep understanding of Hardware Description Languages (HDLs), which are complex and time-consuming due to their low-level structure and unique syntax (Sun et al., 2011). HDLs can be challenging due to their low-level structure, unique syntax, and time-consuming nature of hardware design. Debugging and testing are also complicated. However, High-Level Synthesis (HLS) provides a solution that makes FPGA development more accessible to software developers by abstracting away low-level complexities.

HLS has been a tool that bridges the gap between software and FPGA development. HLS tools allow users to take advantage of FPGA features without needing a deep understanding of HDLs. This paradigm shift has made FPGA programming more accessible and expanded the range of applications that can benefit from FPGA acceleration. The use of HDLs has traditionally been a significant barrier for developers interested in FPGA programming. HLS effectively eliminates this barrier by allowing developers to define the desired behavior of their FPGA-based systems using high-level programming languages such as C, C++, or Python. This approach not only streamlines the development process but also allows developers to focus on algorithmic optimizations and application-specific customizations rather than dealing with HDL complexities.

Many machine learning and classification algorithms are inherently parallelizable, making them ideal candidates for FPGA implementation. FPGAs excel at executing tasks simultaneously, fitting perfectly with the parallel processing requirements of these algorithms. The combination of HLS and FPGA acceleration can lead to significant performance improvements and reduced execution times, especially when working with large-scale datasets and real-time applications.

Binary classification is a category of algorithms that enables software applications to make accurate predictions without explicit programming. The primary goal of binary classification is to create algorithms that can read input data, use statistical analysis to update output, and estimate output when new data becomes available. This study involves several binary classification algorithms, such as Artificial Neural Networks, Naive Bayes Classification, Decision Tree Learning, Random Forest algorithm, and k-Nearest Neighbor Algorithms (k-NN). We plan to use built-in or general classification libraries, such as NumPy and TensorFlow, for software implementation in different high-level programming languages like Python and C++.

We selected the 'Breast Cancer' dataset as a sample dataset for conducting performance comparisons. The Wisconsin Breast Cancer Database, which contains 699 instances, ten attributes, and 16 missing attribute values, will be used in this study. The class results can be either malignant (cancerous) or benign (non-cancerous), with grades ranging from 1 to 10 for the attributes. The class distribution of the dataset is 65.5% benign and 34.5% malignant. This dataset is widely used for binary classification methods and is part of the UCI (University of California Irvine) machine learning repository.

Table 1.1. Wisconsin Breast Cancer Database

#	Attribute	Domain
1	Clump Thickness	1-10
2	Uniformity of Cell Size	1-10
3	Uniformity of Cell Shape	1-10
4	Marginal Adhesion	1-10
5	Single Epithelial Cell Size	1-10
6	Bare Nuclei	1-10
7	Bland Chromatin	1-10
8	Normal Nucleoli	1-10
9	Mitoses	1-10
10	Class:	2 for benign 4 for malignant

Key challenges in implementing binary classification algorithms on both software and FPGA platforms include algorithm selection, programming language compatibility, hardware-software co-design, performance evaluation, and dataset preprocessing. Addressing these challenges is essential for ensuring the accuracy, efficiency, and reliability of the solutions.

- **Algorithm Selection**: Choosing appropriate binary classification algorithms for FPGA acceleration was challenging due to factors like complexity, parallelizability, and resource utilization. Balancing performance and resource constraints was significant.
- **Programming Language Compatibility**: Integrating high-level programming languages like C/C++ and Python with FPGA development tools proved

non-trivial. Ensuring syntactical correctness and optimizing code for FPGA synthesis required meticulous attention.

- **Hardware-Software Co-Design**: Integrating hardware and software components demanded understanding both paradigms and achieving seamless data transfer, synchronization, and algorithm partitioning across platforms.
- **Performance Evaluation**: Devising metrics for performance evaluation, including accuracy, execution time, FPGA resource utilization, power consumption, and maximum clock frequency, posed unique challenges. Ensuring fair comparisons demanded rigorous experimental design.
- **Dataset Preprocessing**: Preprocessing the dataset involves ensuring data quality, handling missing values, and normalizing features. The dataset required careful handling to avoid biases and ensure integrity.

This study aims to provide a comprehensive performance analysis of FPGA-accelerated binary classification algorithms, highlighting the advantages of integrating hardware and software, utilizing FPGA technology for enhanced performance, and evaluating performance metrics such as accuracy, execution time, FPGA resource utilization, power consumption, and maximum clock frequency. By merging high-level programming with FPGA hardware, this study lays a robust foundation for leveraging FPGA technology effectively in binary classification algorithms. The proposed methodology can be extended to other datasets, showcasing the potential of FPGA technology in accelerating binary classification tasks across diverse applications. This flexibility underscores the versatility and adaptability of FPGA technology in various fields, from medical diagnostics to real-time data processing.

Breast cancer diagnosis benefits significantly from advancements in FPGA technology. The real-time processing capabilities of FPGAs, combined with the accuracy of machine learning algorithms, provide a powerful tool for clinicians. This integration allows for rapid prototyping and application development, enabling the medical

community to quickly adapt to new findings and technologies. The methodology proposed in this study is not limited to breast cancer and can be applied to other types of cancer and diseases, providing a framework for future research and development in medical diagnostics.

- **Hardware and Software Integration**: Merges high-level programming (C/C++ and Python) with FPGA hardware.
- **Enhanced Performance via FPGA**: Employs FPGA technology to boost real-time application performance, facilitating software developers' use of FPGAs without HDL knowledge.
- **Diverse Algorithms**: Includes algorithms like k-NN, Decision Tree, Random Forest, Logistic Regression, and Naive Bayes.
- **Performance Metrics Evaluation**: Assesses designs using accuracy, execution time, area, clock frequency, and power consumption.
- **Data-set Classification**: Applies these algorithms to distinguish between cancerous and non-cancerous data effectively.

In conclusion, this study advances FPGA-accelerated binary classification algorithms, addressing key issues such as algorithm selection, compatibility between programming languages, and hardware-software co-design. It lays a robust foundation for utilizing FPGA technology effectively in binary classification tasks, offering promising prospects for future applications in various domains. The potential for extending the methodology to other datasets and applications showcases the versatility and adaptability of FPGA technology, providing a powerful tool for innovation and efficiency in high-performance computing and data processing.

2. RELATED WORKS

Binary classification teaches computers to make decisions between two options, aiming to replicate certain human cognitive processes without direct programming. This is achieved by providing machines with data and information. The most efficient classification method depends on the specific application requirements.

FPGA development boards have been reviewed in the literature for implementing entire systems. An FPGA is an integrated circuit allowing functionality changes post-production. The main structure of an FPGA is a Logic Cell, typically consisting of a Look-Up Table (LUT), a Flip Flop, and a 2-to-1 Multiplexer (MUX). However, the specific architecture of a Logic Cell, also referred to as a Configurable Logic Block, varies depending on the FPGA family or model.

This paper (El Bouazzaoui et al., 2024), introduces an adaptive k-NN methodology for FPGAs, using Dynamic Classifier Selection and K-Nearest Centroids to improve accuracy and efficiency. It shows up to 10.66% accuracy improvement and 3.918 times latency reduction.

Another study (Xu et al., 2023) leverages artificial neural networks (ANNs) for identifying electrochemical signals and drift estimation in Ion-Sensitive Field Effect Transistor (ISFET) sensors. The model, combining Multi-Layer Perceptrons (MLPs) and Gated Recurrent Units (GRUs), achieves 89.71% reaction classification accuracy with an average delay of 15.73 seconds. The feasibility of FPGA implementation through HLS is also demonstrated.

The paper (Abreu et al., 2022) explores power-accuracy trade-offs in decision trees (DT) and random forests (RF) accelerators for embedded devices. It translates tree-based structures to HDL and conducts power-performance analysis, enabling optimization of power consumption while maintaining prediction accuracy.

This paper (Karn et al., 2024) presents a method for confidential inference on FPGAs using McEliece and HQC cryptography for order-preserving encryption in RF models. It leverages FPGAs' low latency for encrypted data processing. High-level synthesis simplifies implementation, with tests on Xilinx FPGA using Jet and MNIST datasets. Jet results were chosen for comparison with our studies.

Another paper (Memon et al., 2021) introduces a Naive Bayes (NB) classifier implemented on FPGA hardware for epileptic event classification from EEG signals. The study achieves real-time classification with 95% to 100% accuracy, demonstrating the feasibility of hardware implementation despite constraints.

In summary, the results of related works are given in Table 2.1 k-NN, Logistic Regression (LR), Decision Tree, Random Forest, and Naive Bayes are commonly used algorithms for classification problems. These algorithms are suitable for implementation on both software and hardware platforms.

Tablo 2.1. Related Works' Accuracies, Elapsed Times, and Resource Usages

-	[3]	[4]	[5]	[6]	[7]
Accuracy for kNN	87.46%	-	-	-	-
Accuracy for LR	-	89.71%	-	-	-
Accuracy for DT	-	-	72.10%	-	-
Accuracy for RF	-	-	-	75%	-
Accuracy for NB	-	-	-	-	95-100%
BRAM	5.5	160	0	0	0
DSP	12	912	0	0	804
FF	1119	103132	536	4633	96375
LUT	1632	190763	3473	4059	95940

3. METHODOLOGY

Machine learning classification methods will be described in this section detail.

3.1 k- Nearest Neighbor Classification (KNN)

The k-nearest neighbors (k-NN) algorithm is a non-parametric and supervised learning algorithm used to classify data. Supervised machine learning algorithms like k-NN need labeled input data. The k-NN algorithm is widely selected for classification due to its ability to distinguish outputs easily. Instead of learning from the training data, it memorizes the dataset and looks for the closest neighbor in the entire data set to make a prediction. After the well-tagged data is chosen, the k value and the distance function must be decided. Each data point in the test dataset will have its class determined using the distance equation with the specified k values. Following these distances found, the minor k-distance results are chosen as a classification set.

The main distance equations include Euclidean distance, Manhattan distance, Chebyshev distance, standardized Euclidean distance, Mahalanobis distance, Bhattacharyya distance, Kullback-Leibler divergence, Hamming distance, and cosine distance (Zhang & Li, 2021). However, we proceeded with the Euclidean distance.

The Euclidean distance equation is used in this study to classify the results as benign or malignant. The equation calculates differences between a new point (x) and an existing point (xi) across all input attributes j.

$$(x, x_i) = \sqrt{\Sigma\,(x_i - x_{ii})^2}$$

After obtaining the Euclidean distance, all points are compared to find the smallest distance. The most accurate results, obtained from the minimum distance criterion, are

selected for classifying the test data (Asri et al., 2016).

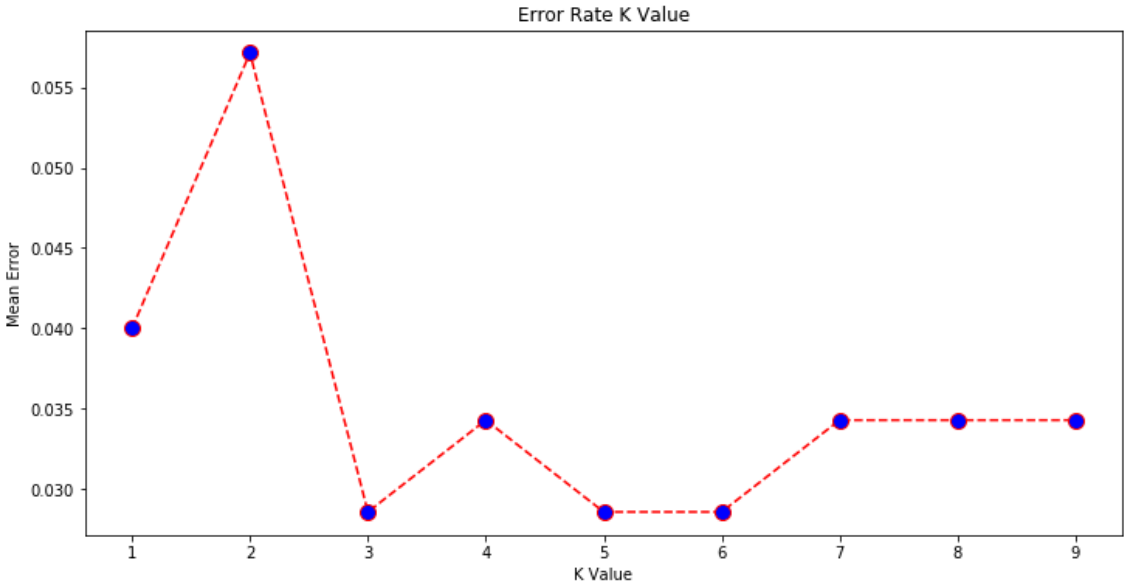

Figure 3.1. K-Nearest Neighbors Algorithm based on Mean Error Analysis

As shown in Figure 3.1, we analyzed the performance of the KNN algorithm by varying the value of K, which represents the number of nearest neighbors considered during classification. The mean error, calculated over a range of K values, was plotted against the K values on the x-axis. The resulting graph illustrates that as the value of K increases, the mean error initially decreases, reaching its lowest point at K=3, K=5, and K=6, and then starts to rise again. Based on this analysis, we selected K=3 as the optimal value for our KNN model due to its associated lowest mean error. This decision ensures improved classification accuracy and robustness in our model, providing better performance in the applications.

3.2 Logistic Regression

Logistic Regression is used as a classification analysis method with models that have a dependent variable with binary possibilities such as yes/no, healthy/patient, and 0/1. In (Zou et al., 2019) mentioned, Logistic Regression is widely utilized in machine learning

due to its higher accuracy compared to other methods (Salazar et al., 2012). Moreover, Logistic regression research on particular data sets' rationality and classification effect will help to understand the excellent utility of artificial neural networks and other complex structures (Kang et al., 2018).

There are mainly two stages for the logistic regression method: forward propagation and backward propagation. With forward propagation, the weights, the bias, and the loss-cost function are constructed.

3.2.1 Forward Propagation

To commence the analysis, the dataset is partitioned into training and test subsets. In adherence to the methodology of this study, the training subset constitutes 80% of the dataset, while the test subset constitutes the remaining 20%. Subsequently, with the creation of the separated data, the imperative task of probability calculation ensues. This calculation necessitates the initiation of forward propagation through parameter initialization. As seen in Figure 3.2, weights multiply the features of the data. After this multiplication, bias will be added to this multiplication. During the application, weight values and bias values were initialized intuitively.

The sigmoid function is referred to in Logistic Regression. A sigmoid function is a bounded, differentiable, non-negative derivative at each point (Han & Moraga, 1995).

As seen from Figure 3.2, the z function gives probabilistic results. Specifically, when $z>0$, the outcome is interpreted as 1, and when $z<0$, the outcome is interpreted as 0, following the application of the sigmoid curve. More specifically, in the context of binary classification, the sigmoid function serves to transform the output of a linear model into a probability score between 0 and 1. When the transformed value, denoted as z, is greater than 0, the resulting probability tends to be closer to 1. Conversely, when z is less than 0, the probability tends to be closer to 0. This behavior is attributed to the sigmoid function's characteristic S-shaped curve, which effectively 'squashes' any

real-valued input into the range [0, 1]. Consequently, decisions regarding class membership can be made based on whether the probability exceeds a certain threshold, typically set at 0.5. Thus, the sigmoid function's output allows for the interpretation of probabilistic outcomes in binary classification tasks. The data feature as input is expressed with x (x0, x1, x2, ..., xn), and the weights are signed with w (w0, w1, w2, ..., wn). These two vectors are multiplied, and the multiplication is summed up with bias (Zou et al., 2019).

$$z = b + w_0 * x_0 + w_1 * x_1 + \ldots\ldots\ldots + w_n * x_n$$
$$\hat{y} = sigmoid(z) = \frac{1}{a+e^{-z}}$$

Cost Function:

$$J(\theta) = \frac{1}{n}\Sigma\, log\hat{y} - (1 - y)log(1 - \hat{y}) + \frac{\lambda}{2n}(\Sigma\, \theta^2)$$

The cost function shows the total error in the forward propagation. According to the Cost function, the algorithm is required to optimize the algorithm according to the best cost function, which means that it should be very close to 0 in the total loss function.

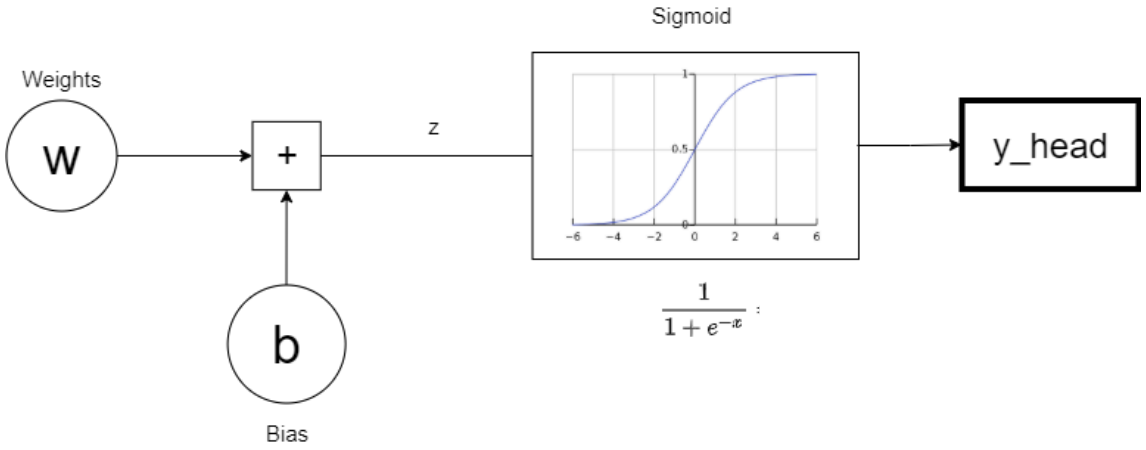

Figure 3.2 Computational graph of Logistic Regression

3.2.2 Backward Propagation

After getting the cost, which is the error, the next step is decreasing the cost because a

high cost means that the prediction arrives with quite a significant error. Thus, the weight and bias values must be updated in the model to avoid significant errors in the prediction. The gradient descent method is helpful in finding the minimum cost with the curve approach to update the parameters.

Gradient Descent aims to reach the global minimum value, starting with random variables. Until reaching the minimum cost, the parameters are constantly updated to get the lower error.

The α (alpha) value here represents the learning rate, a parameter that determines the step size at which the weights (w) and biases (b) are updated during the training process. Essentially, it controls the speed at which the model learns from the training data. A higher learning rate implies larger steps in parameter updates, potentially leading to faster convergence but also increasing the risk of overshooting the optimal solution. Conversely, a lower learning rate results in smaller updates, ensuring more cautious adjustments to the model parameters and potentially better stability during training. Selecting an appropriate learning rate is crucial as it directly influences the model's convergence and performance.

Updating parameters such as weight and bias (Ruder, 2016):

$$w := w - \alpha \nabla_{\theta} J(\theta)$$
$$b := b - \alpha \nabla_{\theta} J(\theta)$$

When the Gradient Descent is 0, we can calculate the parameters. The probability can be defined; in this step, the prediction value and the absolute values have matched. So, the accuracy of the Logistic Regression can be found.

3.4 Decision Tree

A decision tree is a supervised learning technique. It has a tree structure that comprises the nodes, branches, and leaf nodes. The internal nodes represent the features of the dataset. The branches symbolize the rule to separate data and a choice between many

alternatives. After all, the leaf nodes are coming, and it shows the outcome of the Decision Tree. Decision tree learning is a commonly used algorithm for inductive inference (Peng, Chen, & Zhou, 2009).

ID3, CART (Classification and Regression Trees), and C4.5 are the most common decision tree algorithms in data mining that use different splitting criteria to split the node at each level to create a homogeneous node.

Figure 3.3. Generated Decision Tree

A decision tree offers various advantages for classifying datasets. These methods are particularly efficient as their computational complexity scales proportionally with the number of training instances observed. Decision trees embody a conceptual framework that is intuitive for human interpretation, simplifying the classification process. By recursively partitioning the data based on questions with binary responses (Yes/No), decision trees create decision nodes and leaf nodes. This approach facilitates the classification of both categorical and numerical data.

The working principle behind the decision tree is making comparisons in the dataset

starting from the tree's root node; after that, the decision node exists based on the comparison, and the process keeps going until the last leaf nodes recursively.

There are generally two methods called Information Gain and Gini Index to select the best attributes with Attribute Selection Measure (ASM) based on the impurity of a node to find the best attribute to create the nodes for the decision tree algorithm. The main aim is to reduce the impurity of the nodes.

3.4.1 ID3

The ID3 (Iterative Dichotomiser 3) is a simple building algorithm for Decision Tree, which Quinlan developed in 1983. The idea behind the algorithm is to specify the classification testing of the values of the features. It continues from top to bottom for each node in the Decision Tree. We are using this to maximize information gain and minimize entropy, and we are using the results to divide the set of objects. This method proceeds until the sub-tree is homogeneous. It uses the information gain criteria regardless of not looking for alternatives, so this increases the efficiency in terms of the executing times. To sum up, ID3 applies information gain and entropy.

3.4.2 Information Gain

This splitting process generally executes in ID3 and C4.5 Decision Tree algorithms. The information gain is the difference between the entropy before/after splits. To summarize, the information gain is calculating the change in entropy which consists of an impurity criterion.

$$Gain(\Delta) = Entropy(ParentNode) - Entropy(ChildNode)$$

3.4.3 Entropy

The entropy measures the impurity of a node. The formula is below:

$$Entropy(t) = -\Sigma\, p(i/t) log_2 p(i/t)$$

Entropy(t)=−∑p(i/t)log2p(i/t)Entropy(t)=−∑p(i/t)log2p(i/t)

All equations related to the Decision tree are mentioned in (Singh & Gupta, 2014).

The performance metrics of the splitting nodes are as follows:

$$Sensitivity = True\ Positive/Positive$$
$$Specificity = True\ Negative/Negative$$
$$Accuracy = \frac{TruePositive+TrueNegative}{Positive+Negative}$$

Equation 9 defines Sensitivity, which is also known as True Positive Rate or Recall. It measures the proportion of actual positive cases that are correctly identified by the classifier. Mathematically, it is calculated as the ratio of True Positives to the total number of actual positive cases.

Equation 10 represents Specificity, also referred to as True Negative Rate. It measures the proportion of actual negative cases that are correctly identified as negative by the classifier. It is computed as the ratio of True Negatives to the total number of actual negative cases.

Equation 11 defines Accuracy, which is a measure of the overall correctness of the classifier. It quantifies the proportion of all correctly classified cases (both positive and negative) among all cases. Mathematically, it is expressed as the ratio of the sum of True Positives and True Negatives to the total number of cases (both positive and negative).

In the course of this study, we employed the decision tree algorithm to construct a decision tree based on entropy. The 'X' variables depicted in Figure 3.3 denote the features within the dataset. As indicated in Table 1.1, the dataset comprises nine input features and one output feature (benign or malignant).

By employing the decision tree algorithm, we obtained the classification results shown in Figure 3.3, and the accuracy achieved for this classification task using Python is documented as 96%. This notable level of accuracy not only underscores the effectiveness of our approach but also highlights the substantial predictive capacity of

the decision tree model within the context of this specific dataset.

3.5 Random Forest

To understand the concept of the Random Forest, it is important to be familiar with the structure of Decision Trees. The Decision Tree algorithm sequentially asks questions to maximize information gain and aims to make well-directed predictions. However, one of the major issues with decision trees, as a traditional method, is overfitting. To address this, the Random Forest model randomly selects subsets of data and features, ranging from tens to hundreds, and trains multiple decision trees. Each decision tree provides predictions for the data set individually. Subsequently, the algorithm selects the most commonly voted classification among all the predictions.

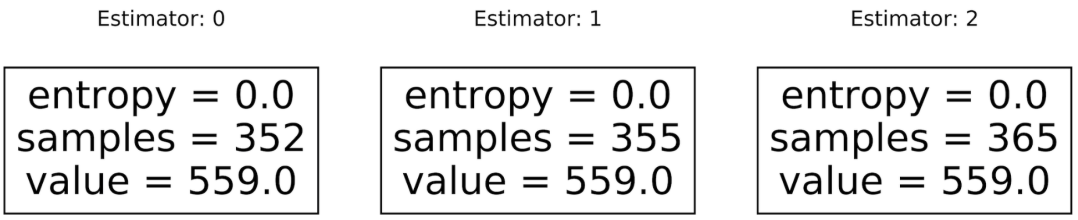

Figure 3.4. Random Forest with three Trees

Random Forest is a supervised machine learning algorithm. It builds a random forest by randomly selecting subsets from the dataset. Unlike the Decision Tree algorithm, Random Forest helps minimize overfitting (Zhou et al., 2020)

In this study, we employed three Decision Trees to construct the random forest algorithm. It is important to note that each tree independently contributes to the final decision-making process. Consequently, when the algorithm makes decisions, votes are aggregated, and the result with the highest number of votes is chosen. Therefore, the number of trees is selected to be odd to prevent tie situations and ensure a decisive outcome.

3.6 Naive Bayes

The Naive Bayes algorithm can be characterized as a probabilistic classifier, operating based on Bayes' Theorem. It assesses the probability of each state for an element and classifies it according to the highest probability value. In cases where a value in the test set is unobserved in the training set, the algorithm returns a probability value of 0, leading to what is termed as zero-frequency. To address this issue, a common solution involves adding one to the frequency of every feature, mitigating the zero-frequency problem (Schütze et al., 2008).

The zero-frequency problem arises when a feature or value present in the test dataset has not been observed during the training phase of the algorithm. Consequently, when the algorithm encounters this unobserved feature during classification, it assigns a probability of zero to it. This renders the algorithm unable to make predictions for that specific feature, potentially leading to inaccuracies in the classification process. To mitigate this issue, techniques such as Laplace smoothing can be employed, where a small constant value is added to the observed frequencies during training, ensuring that even rare features have non-zero probabilities assigned to them, thereby improving the algorithm's robustness and accuracy.

Naive Bayes classifies the data according to maximum likelihood, which is a supervised machine learning algorithm. In this study, we used the Simple Naive Bayes classification. This classification uses Bayes Theorem:

$$P(A \mid B) = P(B \mid A)P(A)P(B)P(A \mid B)=P(B)P(B \mid A)P(A)$$

Bayes' Theorem, represented by the formula "posterior = (likelihood * prior) / (evidence)," stands as a foundational principle in probabilistic reasoning and statistical inference. It offers a framework for updating beliefs or hypotheses based on new evidence or observations. The posterior probability, denoting the probability of a hypothesis or event given the observed data, is calculated by combining the prior

probability (the initial belief about the hypothesis before observing the data) with the likelihood (the probability of observing the data given the hypothesis) and dividing by the evidence (the total probability of observing the data across all possible hypotheses). This theorem allows the integration of new information to refine or revise existing beliefs, making it a cornerstone in various fields such as machine learning, Bayesian statistics, and decision-making under uncertainty. By systematically updating beliefs in light of new evidence, Bayes' Theorem facilitates informed decision-making processes and enhances our understanding of uncertain phenomena.

4. FPGA SIMULATION AND IMPLEMANTATION

4.1 Code Structures

In this section, we delve into the code structures of C++, Python, and HLS. By focusing on a single algorithm, we aim to illustrate and highlight the specific features and considerations involved in using each tool. Examining how the algorithms operate in these programming environments, our objective is to clarify the distinct features and optimizations introduced by each tool. This comparative analysis showcases the adaptability of the chosen tools and provides insights into their specific applications in binary classification algorithms.

For each binary classification task described in Section 3, we normalized the data to a range between 0 and 1. This preprocessing step facilitated ease of manipulation and comparison during subsequent operations, enabling more efficient data processing. Following the normalization process, we partitioned the dataset into 80% for training and 20% for testing purposes. For each binary classification method utilized, we trained the model using the training dataset and then applied the trained model to the test dataset to make predictions, classifying instances as either "benign" or "malignant." These predictions were compared with our ground truth labels from the test dataset to calculate the accuracy rate, expressed as a percentage. Additionally, we measured the elapsed time between the start and end of the testing process to determine the processing time for each language.

In our study, we present code implementations in three different programming languages to showcase the versatility and comparative performance of binary classification methods. The use of C++, Python, and HLS allows us to demonstrate the algorithm's adaptability across different development environments. To ensure a fair and consistent performance evaluation for all these implementations, the same dataset was employed. By presenting code structures in different languages and highlighting the advantages of HLS in FPGA-based implementations, our study aims to provide a

comprehensive view of the versatility of binary classification and its potential to address a wide range of application requirements.

4.1.1 C++ Code Structure on CPU

In our pursuit of implementing diverse classification models, we opted for the C++ programming language, leveraging its versatile features and employing a consistent coding approach across various algorithms.

Our classification models share a modular and dynamic approach in handling datasets. Nested loops for calculating distances and determining nearest neighbors form the backbone of the classification algorithms. Dynamic arrays and conditional statements are employed for efficient binary classification searches.

To facilitate meaningful evaluations, real-world data is dynamically populated into training and testing datasets across all classification models. The accuracy of each model is evaluated by comparing predicted labels with actual labels, providing a comprehensive performance assessment. Moreover, the overall execution time for each classification model is systematically measured and reported, contributing to a thorough understanding of computational efficiency.

```
double sigmoid;
double lossFunction;
double costFunction;
const int feature = 9;
double weight[feature];
int train = 560;
int test = 139;
int totaldatanumber = 699;
double results[test];
double b = 0.0;
double ydiff[train];
for (int i = 0; i < feature; i++) {
    weight[i] = 0.01;
}
for (int i = 0; i < train; i++) {
    ydiff[i] = y_head[i] - y_tr[i];}
for (int i = 0; i < train; i++) {
    der_values[i];}
for (int i = 0; i < train; i++) {
    sumOfdiff = sumOfdiff + ydiff[i];}
b = b - static_cast<double>((lr * sumOfdiff) / train);
for (int i = 0; i < feature; i++) {
    w[i] = w[i] - (lr) * (der_values[i]);}
for (int i = 0; i < test; i++) {
    test_Z[i] = (w[j] * TestL[i].feature[j]) + b;}
for (int i = train; i < totaldatanumber; i++) {
    m_result[i - train] = BorM[i];}
for (int i = 0; i < test; i++) {
 results[i] = 0.0;}
// Accuracys
```

Figure 4.1. C++ Code Illustrating the Logic of Logistic Regression

4.1.2 Python Code Structure on CPU

In our Python implementation, we incorporated various features and optimization techniques to ensure both efficiency and accuracy.

Data preprocessing is a crucial step in any machine learning pipeline. In our implementation, we employed the Pandas library to handle dataset operations. Furthermore, we applied feature scaling using Scikit-learn to normalize the feature values. Normalizing the features is essential for the algorithms as they are sensitive to the scale of input features. This step ensures that each feature contributes equally to the distance computation.

The proposed Python implementation of the binary classification algorithms leverages a combination of time tracking, data preprocessing, feature scaling, and Scikit-learn functionalities to create an efficient and effective solution for binary classification tasks.

```python
def sig(z):
    pass

def init_w_and_b(dim):
    w = 0.01
    b = 0.0
    return w, b

def f_b_prop(w, b, x_tr, y_tr):
    # Forward propagation
    z = np.dot(w.T, x_tr) + b
    y_head = sigmoid(z)
    # loss function
    # cost function
    # Backward propagation
    grad = {"der_w": None, "der_b": None}
    return cost, grad

for i in range(no_iter):
    cost, grad = f_b_prop(w, b, x_tr, y_tr)
    cost_list.append(cost)
    w = w - lr * grad["der_w"]
    b = b - lr * grad["der_b"]

return {w, b}, grad, cost_list

# Prediction function
def predict(w, b, x_test):
    z = sig(np.dot(w.T, x_test) + b)
    Y_pred = np.zeros(z.shape[1])
    for i in range(z.shape[1]):
        if z[0, i] <= 0.5:
            Y_pred[0, i] = 0
        else:
            Y_pred[0, i] = 1
    return Y_pred

# Initialize weights-bias
dim = x_tr.shape[0]
w, b = init_w_and_b(dim)

# Train update function
par, grad, cost_list = update(w, b, x_tr, y_tr, lr=1, no_iter=300)

# Predict
y_pred_test = predict(par["w"], par["b"], x_test)

# Accuracy
```

Figure 4.2. Python Code Illustrating the Logic of Logistic Regression

4.1.3 HLS Code Structure on FPGA

In our implementation of binary classification algorithms, we leveraged the power of

HLS on FPGA to create an efficient and hardware-accelerated solution. HLS allowed us to take advantage of various features and pragmas to optimize our FPGA-based implementation. In HLS, a pragma (short for "pragmatic information") is a directive or annotation provided by the programmer to guide the synthesis tools in generating hardware from high-level language descriptions. Pragmas are used to convey additional information to the synthesis tool regarding optimizations, constraints, or specific hardware requirements. We provide insight into some of the key HLS features and pragmas, exemplifying how we harnessed the capabilities of HLS for enhanced performance and accessibility in the following paragraphs:

Interface Pragmas: We utilized HLS pragmas to define the interface. The pragmas facilitated the management of input and output ports, ensuring seamless communication between the software and hardware components. By specifying interfaces for each parameter, we streamlined data transfer and interaction.

Control Pragmas: To control the synthesis process, we employed control pragmas to designate the return value of our function. This pragma indicated that our function does not require a specific control interface, simplifying the integration into the larger FPGA design.

Resource Optimization: HLS allowed us to optimize the allocation of FPGA resources efficiently. We specified resource pragmas for input parameters, ensuring that they are appropriately mapped to FPGA resources. This fine-grained control over resource allocation contributed to improved performance and resource utilization.

Parallelism: One of the inherent strengths of FPGA technology is parallelism. We took advantage of this feature by designing our algorithms to process multiple input parameters simultaneously. By parallelizing the computations, we achieved significant speedup in the binary classification methods inference process.

Function Decomposition: To further enhance the efficiency of our implementation, we

decomposed our algorithms into modular functions. This allowed us to focus on optimizing specific components of the algorithm independently, resulting in a more streamlined and efficient hardware design.

To summarize this subsection, our FPGA-based binary classification implementation benefited greatly from the use of HLS features and pragmas. These tools enabled us to harness the full potential of the FPGA platform, optimizing resource utilization, improving performance, and enhancing the accessibility of our binary classification algorithms. By providing these examples of HLS utilization, we aim to illustrate the effectiveness of FPGA acceleration in binary classification applications and inspire further exploration of FPGA-based solutions.

4.2 Experimental Results

In this paper, we used sample data from the UCI Repository to make a comprehensive comparison of software and hardware implementations of binary classification algorithms. The sample dataset for this purpose is breast cancer detection. However, our proposed solutions can also work on other datasets. For this data set with 10 attributes, 699 samples were used. Training sets of classification studies were carried out using Python and C++ programming languages. 80% of this data set was chosen as the training set and the remaining 20% as the test set.

Consistency was maintained throughout the study by employing a uniform CPU-based system for software implementations in all phases, ensuring the meaningfulness of inter-comparisons. The utilized computer is equipped with an Intel Core i5-7200U dual-core processor, featuring a base frequency of 2.5 GHz and a Turbo Boost frequency of up to 3.1 GHz, accompanied by a 3MB cache. Moreover, it is outfitted with 8GB DDR4-2133 SDRAM to facilitate efficient memory management and a capacious 1TB 5400rpm SATA hard drive.

The hardware design of classification methods was carried out using the Vivado

simulation platform. In addition, the Xilinx Zynq-7000 family FPGA xczu9eg-1ffvc900e-3-e device was used in the implementations.

4.3 FPGA Implementations

In this section, we present the results of FPGA implementations for the execution of binary classification algorithms using HLS. The objective is to assess the resource utilization of these algorithms implemented on an FPGA, focusing on key metrics such as Block RAM (BRAM18K), DSP48E slices, Flip-Flops (FF), and Look-Up Tables (LUT). The tables outline the specific resource consumption details for different components of the FPGA design. These results provide valuable insights into the efficiency and effectiveness of the k-NN algorithm when implemented on an FPGA using HLS.

Table 4.1. Resource Utilization of k-NN Implementation with HLS

Name	BRAM	DSP	FF	LUT
Instance	2	1	434	1632
Memory	1	-	0	0
Multiplexer	-	-	-	193
Register	-	-	8	-
Total	**3**	**1**	**442**	**1825**

In Table 4.1, the resource utilization of k-NN Implementation with HLS results indicates that the design is relatively small and efficient in terms of hardware resources.

For example, the design only requires three BRAMs, one DSP, 442 FFs, and 1825 LUTs. This suggests that the design can be easily implemented even on low-cost tiny FPGAs. The largest resource utilization in the design comes from LUTs, which is not surprising given that k-NN involves a large number of conditional statements and data comparisons. However, the number of LUTs used in the design is still relatively small, indicating that the design can run at high clock frequencies and achieve high throughput. The design does not require any DSP blocks for computation, which is a positive indication of its low-power and low-cost implementation. The design requires solely two BRAMs for storing the instance data, suggesting that the memory usage is reasonable, and the design can handle moderate-sized datasets. The design only requires 8 registers for storing the control signals, which is a small overhead and does not significantly impact resource utilization.

Table 4.2. Resource Utilization of Logistic Regression Implementation with HLS

Name	BRAM	DSP	FF	LUT
Expression	-	-	0	113
Instance	-	40	3753	7191
Multiplexer	-	-	-	587
Register	-	-	428	-
Total	-	**40**	**4184**	**7891**

Table 4.2 presents the resource utilization results for a design that implements logistic regression using Vivado HLS. It is noteworthy that the resource utilization of this

design surpasses that of the earlier discussed k-NN implementation. Specifically, the design necessitates 40 DSP blocks, 4184 FFs, and 7891 LUTs. This observation suggests that the design exhibits greater complexity and requires a larger number of hardware resources for successful implementation. The design efficiently employs a relatively modest number of registers (428) for storing control signals, indicating a favorable level of efficiency. However, the most substantial resource utilization within the design arises from the LUTs (7891), signifying the presence of numerous conditional statements and data comparisons. In contrast to the k-NN implementation, the logistic regression design incorporates 40 DSP blocks for computation. This implies that the design entails a significant number of multiplications and may benefit from utilizing dedicated DSP blocks to achieve high-performance implementation. Furthermore, it is important to note that the design does not utilize any BRAM blocks for data storage, indicating its ability to accommodate datasets that can fit within on-chip memory. Overall, the results of resource utilization highlight the heightened complexity of the logistic regression design compared to the k-NN implementation. Nevertheless, the design remains feasible for implementation on FPGA devices with reasonable resource availability. It facilitates high-performance computation through the utilization of DSP blocks and effectively handles moderate-sized datasets by leveraging on-chip memory. However, it is advisable to consider higher clock frequencies for the design due to its increased resource utilization.

Table 4.3. Resource Utilization of Decision Tree Implementation with HLS

Name	BRAM	DSP	FF	LUT

Expression	-	-	0	282
Multiplexer	-	-	-	110
Register	-	-	49	-
Total	**-**	**-**	**49**	**392**

Table 4.3 gives the resource utilization results for the design that implements the decision tree using Vivado HLS. Remarkably, this design exhibits significantly lower resource requirements compared to the previously discussed k-NN and logistic regression implementations. Specifically, the decision tree design necessitates only 49 FFs and 392 LUTs, implying a notably simpler structure and a reduced demand for hardware resources. Notably, this design does not utilize any DSP blocks or BRAM blocks, indicating that it does not rely on high-performance computation or extensive on-chip memory. The largest resource utilization within the design stems from the LUTs (392), suggesting a moderate presence of conditional statements and data comparisons. Moreover, the design efficiently employs a small number of registers (49) for storing control signals, underscoring its commendable efficiency. Furthermore, the design does not incorporate any expression blocks or multiplexers, signaling its straightforward nature and lack of complex logic requirements. In summary, the resource utilization results highlight the substantially simpler nature of the decision tree design compared to the k-NN and logistic regression implementations discussed earlier. It can be implemented on FPGA devices with minimal resource allocations and offers moderate-performance computation through the utilization of LUTs. It is worth noting that this method may be compatible with lower clock frequencies due to its modest resource utilization.

Table 4.4. Resource Utilization of Random Forest Implementation with HLS

Name	BRAM	DSP	FF	LUT
Expression	-	-	0	661
Total	-	-	0	**661**

Table 4.5. Software and Hardware Simulation Results

#	k-NN	LR	DT	RF	NB
Accuracy in Python	97%	97%	96%	100%	100%
Accuracy in C/C++	98%	95%	98%	99%	99%
Elapsed time Python (s)	13.50	0.30	2.20	0.92	0.37

Elapsed time C/C++ (s)	0.142	0.001	0.110	0.144	0.004
Elapsed time HLS (µs)	58.291	1.009	5.931	4.133	0.329
BRAM	3	0	0	0	0
DSP48E	1	40	0	0	22
FF	442	4181	49	0	3151
LUT	1825	7891	392	661	1094

Table 4.4 shows the resource utilization results for a design implementing random forest using Vivado HLS. Compared to the k-NN and logistic regression designs, the random forest design requires fewer hardware resources, with only 661 LUTs. This indicates

that the design is less complex and can be implemented on FPGA devices with small resource constraints. The random forest design does not require any BRAM blocks or DSP blocks, which suggests that it does not involve high-performance computation or large on-chip memory. The largest resource utilization in the design comes from LUTs, which indicates that the design may involve a moderate number of decision trees. The random forest design does not use any expression blocks, multiplexers, or registers, which suggests that the design is relatively straightforward and does not require complex logic. Compared to the decision tree design, the random forest design requires more LUTs but does not require any registers. This indicates that the random forest design may involve multiple decision trees, and the LUT utilization increases accordingly. Overall, the resource utilization results suggest that the random forest design is less complex than the k-NN and logistic regression designs, but more complex than the decision tree design. The random forest design requires moderate resources in terms of LUTs and can be implemented on FPGA devices with small resource constraints. The method may require lower clock frequencies due to its small resource utilization.

Table 4.6. Resource Utilization of Naive Bayes Implementation with HLS

Name	BRAM	DSP	FF	LUT
Expression	-	-	0	94
Instance	-	22	680	501

Memory	0	-	1152	198
Multiplexer	-	-	-	301
Register	-	-	1319	-
Total	0	22	3151	1094

Table 4.7. Comparison of Related Works and Proposed Work

#	**Related Works**					**Proposed Work**				
	[3]	**[4]**	**[5]**	**[6]**	**[7]**	**kNN**	**LR**	**DT**	**RF**	**NB**
Accuracy for kNN	87.46 %	-	-	-	-	97 %	-	-	-	-
Accuracy for LR	-	89.71 %	-	-	-	-	97 %	-	-	-
Accuracy for DT	-	-	72.10 %	-	-	-	-	96 %	-	-
Accuracy for RF	-	-	-	75 %	-	-	-	-	100 %	-
Accuracy for NB	-	-	-	-	95-100 %	-	-	-	-	100 %
BRAM	5.5	160	0	0	0	0	0	0	0	0
DSP	12	912	0	0	804	1	40	0	0	22
FF	1119	103132	536	4633	96375	442	4181	49	0	3151
LUT	1632	190763	3473	4059	95940	1825	7891	392	661	1094

Table 4.5 shows the resource utilization of the Naive Bayes algorithm implemented in Vivado HLS. Compared to k-NN and Logistic Regression, Naive Bayes requires more resources, with a total of 22 DSP48E slices and 1,094 LUTs. However, it uses only 22 instances of DSP48E, which is less than that required by the Logistic Regression algorithm. In comparison to Decision Tree and Random Forest, Naive Bayes uses more resources in terms of LUTs and FFs but requires fewer resources in terms of BRAM18K and DSP48E slices. Overall, it can be concluded that Naive Bayes has moderate resource requirements compared to the other algorithms in this study.

Regarding resource utilization, we can see that the logistic regression and Naive Bayes models use significantly more FF, LUT, and DSP48E resources than the other models, especially compared to the decision tree and random forest models.

Overall, Table 4.5 provides a comprehensive comparison of the performance and resource utilization of the different models using different programming languages and hardware implementation. However, it is important to note that the specific results may vary depending on the dataset and the specific implementation details.

In Table 4.5, as illustrated, the hardware and software results have been gathered for each algorithm. Accordingly, the comparison of elapsed times for C/C++, Python, and HLS is as follows. It is evident that HLS significantly outperforms C/C++, while C/C++ surpasses Python by a substantial margin.

In general, the models achieve higher accuracy in C/C++ than in Python. The elapsed time for the models is significantly faster in C/C++ compared to Python. Additionally, the hardware implementation using Vivado HLS achieves an extremely fast elapsed time in microseconds (μs), which is orders of magnitude faster than the software implementations in Python and C/C++.

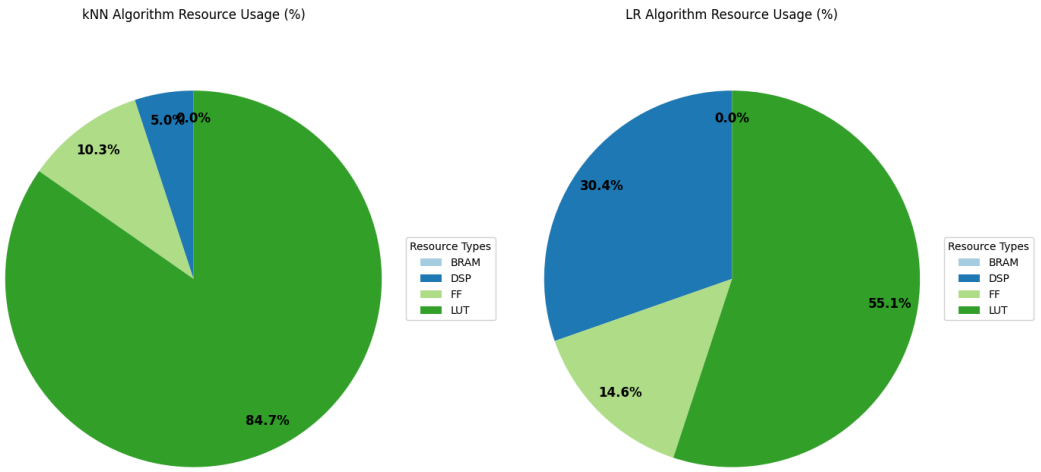

Figure 4.3. Resource Usage for kNN and Logistic Regression in FPGA

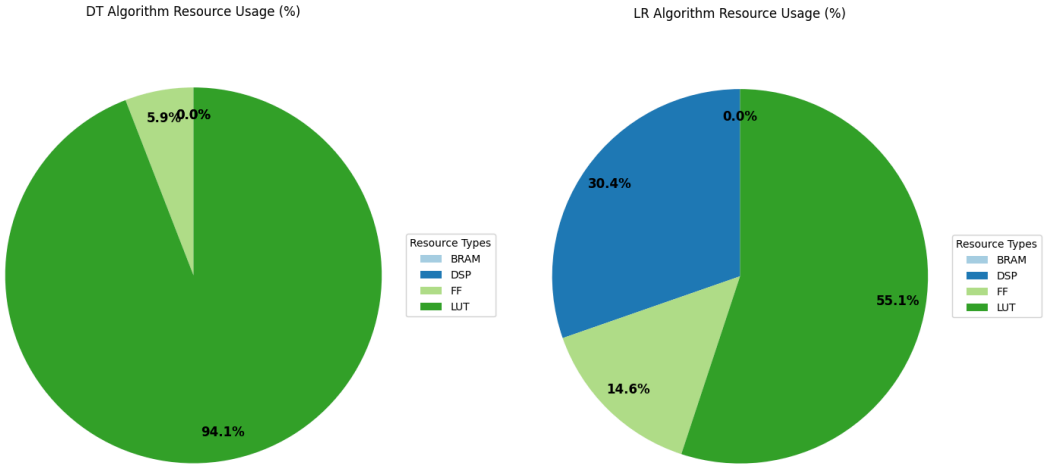

Figure 4.4. Resource Usage for Decision Tree and Random Forest in FPGA

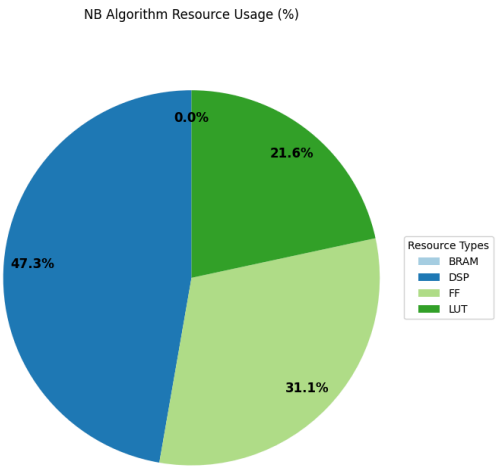

Figure 4.5. Resource Usage for Naive Bayes in FPGA

The outcomes, organized by algorithms and elapsed times, are presented as follows:

- **For the k-NN Algorithm:**
    - C/C++ exhibits a speed advantage over Python by approximately a factor of 95.
    - HLS demonstrates a remarkable speed advantage over Python by approximately 232 thousand times.
    - HLS surpasses C/C++ in speed by roughly 2.5 thousand times.
- **For the Logistic Regression Algorithm:**
    - C/C++ outpaces Python by approximately 317-fold times.
    - HLS showcases an impressive speed advantage over Python by roughly 297 thousand times.
    - HLS surpasses C/C++ in speed by approximately a factor of 1000.
- **For the Decision Tree Algorithm:**
    - C/C++ boasts a speed advantage over Python by approximately 20 times.
    - HLS showcases an extraordinary speed advantage over Python by approximately 370 thousand times.
    - HLS surpasses C/C++ in speed by approximately 18 thousand times.

- **For the Random Forest Algorithm:**
    - C/C++ outperforms Python by a factor of roughly 6.4.
    - HLS demonstrates a substantial speed advantage over Python by approximately 222 thousand times.
    - HLS surpasses C/C++ in speed by approximately 34 thousand times.
- **For the Naive Bayes Algorithm:**
    - C/C++ outpaces Python by approximately a factor of 103.
    - HLS showcases an exceptional speed advantage over Python by approximately 1 million times.
    - HLS surpasses C/C++ in speed by approximately a factor of 10 thousand.

In summary, our analysis reveals significant performance variations across binary classification algorithms when implemented in different programming languages and with hardware acceleration. C/C++ consistently outperforms Python, with speed advantages ranging from 6.4 to 317-fold for various algorithms. Additionally, HLS demonstrates remarkable acceleration capabilities, surpassing Python by billions of times and C/C++ by millions to billions of times. These findings highlight the pivotal role of language and hardware choices in optimizing algorithmic performance, offering valuable insights for both practitioners and researchers in diverse fields.

The comparison of related studies and our proposed implementations is presented in Table 4.7. In this table, while columns from references [3] to [7] show related works, the last five columns, namely, kNN, LR, DT, RF, and NB show our proposed implementations. We can observe notable differences in terms of accuracy and resource utilization metrics, particularly in the fields of binary classification and hardware implementation.

Regarding accuracy, our approach outperforms the related studies significantly. For instance, our Logistic Regression model achieves an accuracy of 97%, while the related

works exhibit lower accuracy values for Logistic Regression. Similarly, our Decision Tree model attains an accuracy of 96%, surpassing the accuracy of 72.10% in related work. The kNN and Random Forest models in our work also demonstrate a marked increase in accuracy compared to the related works.

Furthermore, when considering resource utilization metrics such as BRAM, DSP, FF (Flip-Flops), and LUT (Look-Up Tables), our work exhibits efficient use of resources. Notably, in contrast to the relatively high resource utilization in some related works, we have managed to achieve resource-efficient implementations, as indicated by the lower values in BRAM, DSP, FF, and LUT. This reflects our ability to optimize the hardware resources effectively in our study.

In conclusion, our proposed work, as presented in Table 4.7, demonstrates superior accuracy and resource utilization compared to the related studies, signifying the advancements and optimizations we have made in the fields of binary classification and hardware implementation. These outcomes highlight the strengths and contributions of our research in this domain.

5. CONCLUSIONS

In conclusion, this paper presented a study on the classification of breast cancer data using various binary classification algorithms, including K Nearest Neighbor, Decision Tree Learning, Random Forest, Logistic Regression, and Naive Bayes Classification. The focus of this work was on the hardware and software implementation of these algorithms using FPGA technology. The hardware implementation offered the potential to improve system performance and reduce power consumption for real-time applications.

The proposed HLS-based hardware design aimed to implement binary classification classifiers on the Xilinx Zynq-7000 FPGA platform. By evaluating each classification algorithm, the paper investigated their success rate (accuracy), execution time, total area

(in terms of Slices or Look-Up Tables), maximum clock frequency, and power consumption. The implementation was carried out in two stages: software implementation in high-level languages such as C/C++ and Python, and performance evaluation of the algorithms.

The results demonstrated the effectiveness of the implemented binary classification classifiers in accurately predicting breast cancer based on the provided dataset. Our proposed work also provided insights into the trade-offs between different algorithms and their hardware resource requirements. This research opens possibilities for utilizing FPGA technology to optimize binary classification applications, particularly in the medical field.

6. References

1 Jemal, A., et al. (2011). Global cancer statistics. *CA: A Cancer Journal for Clinicians, 61*(2), 69-90.

2 Sun, Y., et al. (2011). Circulating tumor cells: advances in isolation and analysis, and applications in cancer detection and treatment. *Clinical Chemistry, 57*(10), 1292-1295.

3 El Bouazzaoui, A., Jariri, N., Mouhib, O., & Hadjoudja, A. (2024). Adaptive K values and training subsets selection for optimal K-NN performance on FPGA. *Journal of King Saud University-Computer and Information Sciences*. Elsevier, 102081.

4 Xu, Y., Kuang, L., Zhu, T., Zeng, J., & Georgiou, P. (2023). Drift Prediction and Chemical Reaction Identification for ISFETs using Deep Learning. In *Proceedings of the 2023 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1–5). https://doi.org/10.1109/ISCAS46773.2023.10181442.

5 Abreu, B., Grellert, M., & Bampi, S. (2022). A framework for designing power-efficient inference accelerators in tree-based learning applications. *Engineering Applications of Artificial Intelligence, 109*, 104638.

6 Karn, R. R., Knechtel, J., & Sinanoglu, O. (2024). Code-Based Cryptography for Confidential Inference on FPGAs: An End-to-End Methodology. In *Proceedings of the 2024 25th International Symposium on Quality Electronic Design (ISQED)* (pp. 1-8). IEEE.

7 Memon, N., Zaidi, S. S. H., & Suleman, M. (2021). The Prognosis of Epilepsy with Naive Bayes Classifier On FPGA Using HDL Coder. In *Proceedings of the 2020 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)* (pp. 18–23). https://doi.org/10.1109/IECBES48179.2021.9398794.

8 Zhang, S., & Li, J. (2021). Z Distance Function for KNN Classification. *arXiv preprint arXiv:2103.09704*, 706.

9 Asri, H., Mousannif, H., Al Moatassime, H., & Noel, T. (2016). Using machine learning algorithms for breast cancer risk prediction and diagnosis. *Procedia Computer Science, 83*, 1064–1069.

10 Zou, X., Hu, Y., Tian, Z., & Shen, K. (2019). Logistic Regression Model Optimization and Case Analysis. In *Proceedings of the 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)* (pp. 135–139). IEEE.

11 Salazar, D. A., Vélez, J. I., & Salazar, J. C. (2012). Comparison between SVM and logistic regression: Which one is better to discriminate? *Revista Colombiana de Estadística, 35*, 223–237.

12 Kang, K., Gao, F., & Feng, J. (2018). A new multi-layer classification method based on logistic regression. In *Proceedings of the 2018 13th International Conference on Computer Science & Education (ICCSE)* (pp. 1–4). IEEE.

13 Han, J., & Moraga, C. (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning. In *Proceedings of the International Workshop on Artificial Neural Networks* (pp. 195–201). Springer.

14 Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 717.
15 Peng, W., Chen, J., & Zhou, H. (2009). An implementation of ID3-decision tree learning algorithm. From web.arch.usyd.edu.au/wpeng/DecisionTree2.pdf Retrieved date: May 2009, 13.
16 Singh, S., & Gupta, P. (2014). Comparative study ID3, cart and C4.5 decision tree algorithm: a survey. *International Journal of Advanced Information Science and Technology (IJAIST), 27*, 97–103.
17 Zhou, Q., Lan, W., Zhou, Y., & Mo, G. (2020). Effectiveness Evaluation of Anti-bird Devices based on Random Forest Algorithm. In *Proceedings of the 2020 7th International Conference on Information, Cybernetics, and Computational Social Systems (ICCSS)* (pp. 743–748). IEEE.
18 Schütze, H., Manning, C. D., & Raghavan, P. (2008). Introduction to information retrieval; Vol. 39, Cambridge University Press Cambridge.

RESUME

Aslıhan Hacer TÜFEKCİOĞLU

Bursa

EDUCATION

2022- Present	MSc in Electrical and Electronic Engineering Marmara University, Istanbul (Turkiye) **GPA 3.3 /4.00** Language of Education: English
2012–2017	BSc in Mechatronics Engineering Bursa Technical University, Bursa (Turkiye) **GPA 2.59 /4.00** Language of Education: 30% English

WORK EXPERIENCE

03/2022–Present	**Development Engineer - BOSCH, Bursa** *Product Development - Quality Engineer*
03/2020–03/2022	**Process Development Engineer -VALEO, Bursa** *Process Development*
12/2018–03/2020	**Project Engineer -Beyçelik Gestamp, Bursa** *Project Engineer*