Software and FPGA-Based Hardware to Accelerate Machine Learning Classifiers

by

Parisa Abdolrahim Poorheravi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

Abstract

This thesis improves the accuracy and run-time of two selected machine learning algorithms, the first in software and the second on a field-programmable gate array (FPGA) device. We first implement triplet loss and triplet mining methods on large margin metric learning, inspired by Siamese networks, and we analyze the proposed methods. In addition, we propose a new hierarchical approach to accelerate the optimization, where triplets are selected by stratified sampling in hierarchical hyperspheres. The method results in faster optimization time and in almost all cases, and shows improved accuracy. This method is further studied for high-dimensional feature spaces with the goal of finding a projection subspace to increase and decrease the inter- and intra class variances, respectively.

We also studied hardware acceleration of random forests (RFs) to improve the classification run-time for large datasets. RFs are a widely used classification and regression algorithm, typically implemented in software. Hardware implementations can be used to accelerate RF especially on FPGA platforms due to concurrent memory access and parallel computational abilities. This thesis proposes a method to decrease the training time by expanding on memory usage on an Intel Arria 10 (10AX115N 3F 45I2SG) FPGA, while keeping high accuracy comparable with CPU implementations.

Acknowledgements

I would like to thank all the people who made this thesis possible. First, my supervisor Dr.Vincent Gaudet, who have been patient and supportive throughout my time in University of Waterloo. Having his guidance and encouragements gave me confidantes in exploring a new field. Furthermore, for providing me opportunities to grow academically.

It was all not possible without the care and the emotional support of my family, Dr. Houshmand Abdolrahim Poorheravi, Dr. Mojgan Kashanchi Langroodi, my brother and my engineer in crime, Pedram Abdolrahim Poorheravi whom I need to specially thank, for being there for me in shadows, darkness and lights since his birth and share a close bond with me. Their tolerance, patient and sacrifices has an enormous and dominant impact on my life and who I am today as a person. I am more than I could be because of them and I am thankful, grateful and lucky.

I also have to mention my partner Dr. Javad Rahimipour Anaraki for his emotional support, counseling, mentor-ship and enlightenment both professionally and personally during hard times and disappointments. His support and patience is invaluable to me.

Finally, I have been inspired by many people along my way in University of Waterloo, my colleagues such as Dr. Benyamin Ghojogh, co-workers and friends for their love, guidance and their impact on my research career.

Dedication

This is dedicated to my parents Dr. Houshmand Abdolrahim Poorheravi (M.D), Dr. Mojgan Kashanchi Langroudi (M.D) and my brother Pedram Abdolrahim Poorheravi for their sacrifices, leaving their home, position, career, possessions and their beloved family behind for giving me the opportunity of a new life in a new country.

The hardship of immigration they tolerated, trusting our little family, showed me it is never too late to start again and to perform to the best of my ability. They are not only a symbol of love and care but also are a symbol of strength, dedication, hard-work and dare.

For all the nights my parents would sit and help me with my homework. For all the fights we had over mathematics and for all the questions they told me I did wrong to learn. I am forever thankful and in debt to them for their love an patience with me.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BA Batch All 14

BH Batch Hard 14

BSH Batch Semi-Hard 14

EPEN Easy Positive Easy Negative 14, 24

EPHN Easy Positive Hard Negative 14, 24

FCBF Fast Correlation-Based Filter 6

FLDA Fisher Linear Discriminant Analysis 9

FPGA Field Programming Gate Array 20

GDA Generalized Discriminant Analysis 10, 36

HPEN Hard Positive Easy Negative 14, 24

HPHN Hard Positive Hard Negative 14, 24

ICA Independent Component Analysis 8

K-NN K - Nearest Neighbor 15

K-NS K-Negative Sampling 25

LDA Linear Discriminant Analysis 15, 16, 32, 34–36

PCA Principle Component Analysis 8, 26, 35, 36

PCC Pearson Correlation Coefficient 6

QDA Quadratic Discriminant Analysis 15, 16

RF Random Forest 15, 16

SDP Semidefinite Programming 22

SU Symmetrical Uncertainty 7

Chapter 1

Introduction

1.1 Thesis Objectives

This thesis focuses on the acceleration of machine learning algorithms, presenting two different approaches, one in software and one in hardware. The software implementation focuses on triplet loss and triplet mining methods for large margin metric learning. A new hierarchical approach is presented and analyzed for both large margin metric learning and subspace learning methods. The thesis then transitions to the hardware implementation of an accelerator for random forests using field programmable gate arrays (FPGA).

1.2 Contributions

Distance metric learning is a fundamental and widely used technique in machine and manifold learning [68]. The general idea is to find a metric to discriminate classes by increasing and decreasing the inter- and intra- class variances, respectively [47, 49]. This goal was first applied to Fisher Discriminant Analysis (FDA) [39, 49] and to Principal Component Analysis (PCA) [90] as the fundamental metrics of feature extraction and metric learning.

Most metric learning methods can be defined as anchor-positive-negative triplets, where anchor and positive are defined as same class label instances while negative is defined as an instance with a different class label as anchor. Large Margin Metric Learning for Nearest Neighbor Classification is defined based on the concept of triplets [109, 110]. This method uses Semi-Definite Programming (SDP) optimization, which can be iterative and slow for complex datasets [107]. Later, Siamese Networks introduced triplet cost functions [54, 61, 96] in two forms of Hinge loss [96, 109, 110] and Softmax [50, 83, 113, 115].

Triplet mining methods [96] were used to speed up the SDP optimization by selecting the most important data points, rather than to use them all. However, the triplet mining methods for Siamese networks were not explored for previously developed large margin metric learning for nearest-neighbor classification. In this thesis, inspired by the mining techniques for Siamese networks, different triplet mining methods are proposed for large-margin metric learning in order to speed up SDP optimization. In addition to triplet mining techniques, this thesis discusses a hierarchical approach to further increase the optimization speed by using iterative selection of most important data instances with stratified sampling [44].

The complexity of a dataset is not contained only to its number of instances but also the size of its feature space. High numbers of instances are definitely beneficial for classification or function estimation. However, high dimensional observation spaces are non-intuitive and therefore, visualization and exploration of such data requires dimensionality reduction in order to simplify the data. Having a high number of features compared to samples in a dataset is referred by a phenomenon called the "curse of dimensionality", which raises a need for applying a dimensionality-reduction method to remove redundant and irrelevant features. There are two main approaches for reducing the size of a dataset: feature selection [18] and feature extraction [98]. Feature selection chooses a subset of features with respect to their correlation with labels. It does not change the feature space and it is most effective when intrinsic complexity of data is lower than the complexity of its feature space. Feature extraction, however, can reduce the dimensionality of data by changing the data space and creating new feature set that best represents the data in a lower-dimension data space with the hope of discovering more correlated features with respect to classification outcome. Some of the well-known approaches are principal component analysis (PCA) [63] and linear discriminant analysis [33].

Dimensionality reduction allows a more interpretable representation of data. However, it does not necessarily result in higher accuracy due to potentially losing some informative features during the reduction. But it is a trade-off to gain better explorability and visualization of data. In this thesis, the hierarchical approach for large margin metric learning using stratified sampling [92], is expanded to a hierarchical approach for manifold learning dimensionality-reduction that improves accuracy. This approach includes iterative selection of features by hierarchical sampling, feature selection, and feature extraction.

In addition to the proposed method to reduce run-time on software, the other main topic of this thesis focuses on acceleration of random forest classifiers through hardware implementation.

A random forest (RF) is a well-known ensemble-learning method [28] for classification and regression [14] problems. RFs use a collection of many independently trained decision trees where each tree is created by sampling the entire training set. Test instances pass through a forest (collection) of trees and a final decision is made based on a majority vote [28] [74]. Generally, RFs use both random sub-sampling and feature selection to create trees with low inter-correlation, hence achieving high accuracy and robustness to overfitting.

Software implementations of machine learning algorithms such as RFs are widely used for multiple applications, and can have performance that is comparable to that of some deep neural networks [118]. However, RFs can take a long time to run, especially when using large datasets, which makes them an interesting subject for hardware developers, where certain architectures can lead to significant speed-ups. Many studies have investigated the implementation and acceleration of both the training and testing phase of RFs. The high level of available parallelism in the RF algorithm lends itself to hardware implementation.

Implementing machine learning algorithms such as random forests in software using a high-level language, is a balance between run time and accuracy. However, hardware implementations of these algorithms focus mainly on run-time vs. memory issues and do not address the classification accuracy. The hardware calculated accuracy is low or it is out-resourced to software implementation. In this thesis, we propose a hardware architecture to decrease the training time, while maintaining a comparable accuracy with existing software implementations.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. In chapter 2, we review the foundations of large margin metric learning, triplet loss, Siamese networks, feature selection, feature extraction, the foundation of a decision tree and a general hardware approach to random forest implementation. The triplet mining methods that have already been proposed for Siamese triplet training, batch all [30], batch hard [58], batch semi-hard [96], easiest/hardest positives and easiest/hardest negatives, negative sampling [111] are discussed as well as the classifiers that were used to compare the accuracies of the methods under study followed by related studies on hardware implementations of random forests. In chapter 3, the proposed triplet mining techniques in the SDP optimization of large margin metric learning, the hierarchical approach for large margin metric learning and the hierarchical approach for subspace learning methods for dimensionality reduction are explained in details. Chapter 4 is dedicated to the proposed hardware architecture for random forest acceleration. The experiments and results are also explained and compared to the literature. Finally, chapter 5 concludes the thesis by summarizing the work and discusses possible future directions.

1.4 Publications

Poorheravi, P. A., Ghojogh, B., Gaudet, V., Karray, F., & Crowley, M. (2021). Acceleration of Large Margin Metric Learning for Nearest Neighbor Classification Using Triplet Mining and Stratified Sampling. Journal of Computational Vision and Imaging Systems, 6(1), 1–5.

Poorheravi, P. A., & Gaudet, V. (2021, May). Hierarchical Subspace Learning for Dimensionality Reduction to Improve Classification Accuracy in Large Data Sets. In 2021 IEEE 51st International Symposium on Multiple-Valued Logic (ISMVL) (pp. 136-141). IEEE.

Poorheravi, P. A., & Gaudet, V. (2022, Augest). FPGA-Based Architectures for Random Forest Acceleration. In 2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS) (pp. *to appear*). IEEE.

Chapter 2

Background And Related Works

In this chapter, we first summarize several dimensionality-reduction methods, including feature selection and feature extraction, Siamese Network including triplet loss plus triplet mining methods, and large margin metric learning for nearest-neighbor classification. Afterwards, we focus on Random Forest (RF) classifiers. Moreover, we discuss the literature around hardware implementations of RF classifiers and discuss some state-of-the-art implementations.

2.1 Dimensionality Reduction

In data classification, either with discrete finite labels for classification or continuous labels for regression, a model needs to be trained to identify patterns in data with regards to its associated labels. In real-world complex datasets, and high-dimensional observation spaces are not intuitive, and hence it is hard for the model to identify useful information. In order to help the model, we add a pre-processing stage to better discriminate the data in terms of its classes, or to find a better representation (visualization) of data. The goal of dimensionality reduction is to either find a better discrimination of data [15] or a more intuitive visualization of data [34, 77]. The pre-processing stage should either highlight the useful features or remove redundant and irrelevant features. There are two main approaches to achieve this goal, i.e., feature selection and feature extraction [65].

Both of these methods are common for reducing the dimensionality of data by mapping $\boldsymbol{x} \mapsto \mathbf{y}$, where $\boldsymbol{x} \in \mathbb{R}^d$ and $\boldsymbol{y} \in \mathbb{R}^p$ are column vectors and $p \leq d$. the goal is to achieve a better data visualization in p dimensions. In feature selection, the selected features are a subset of the original feature space, while feature extraction will result in a completely new set of features created from the original feature space from the patterns of data [98].

2.1.1 Feature Selection

Feature selection [16, 18, 80] is a pre-processing feature-reduction method that selects a subset of the original feature space to map a complex raw feature space $\boldsymbol{x} \in \mathbb{R}^d$ to $\boldsymbol{y} \in \mathbb{R}^p$ where $p \leq d$. The feature space's subset is chosen based on some criterion. The hope is to better represent the dataset and simplifying it while maintaining its classification accuracy. One criterion to choose a subset of features is Filter Methods.

In filter methods, the features are ranked based on their redundancy and their Relevance. Redundancy factors shows how much of redundant information a feature carries and relevancy represents the correlation of feature with the classification label. After ranking, the features are removed and filtered based on a threshold before the model is trained [46, 53].

Correlation Based Feature Selection

As the name suggests, this criterion is based on the correlation of each feature, $\boldsymbol{x}^j$, with the target label vector $\boldsymbol{t}$ where $\boldsymbol{x}^j$ represents the j-th feature. The higher the correlation value, the more relevant the feature is for class determination, hence the more important the feature is for classifying the data and therefore is ranked higher in the hierarchy of selected features [46, 55].

The most common correlation criterion is Pearson Correlation Coefficient (PCC) defined as [7, 53]:

$$\rho_{\boldsymbol{x}^j,\boldsymbol{t}} := \frac{Cov(\boldsymbol{x}^j, \boldsymbol{t})}{\sqrt{Var(\boldsymbol{x}^j)Var(\boldsymbol{t})}}, \tag{2.1}$$

where $Cov(.,.)$ represents co-variance and $Var(.)$ represents variance.

Fast Correlation Based Filter Feature Selection

Fast Correlation-Based Filter (FCBF) [116] is another correlation-based feature-selection criterion adopted for high-dimensional data to find predominant correlation in order to find the subset of features that reduces dimensionality while increasing classification accuracy. FCBF uses an entropy-based measurement known as Symmetrical Uncertainty (SU) to find the relevancy and redundancy of features. The SU is defined as:

$$SU(X, Y) := 2\left(\frac{IG(X, Y)}{H(X) + H(Y)}\right), \tag{2.2}$$

where X is a random feature and Y is the label. The $IG(X, Y)$ is the information gain defined as:

$$IG(X, Y) := H(X) - H(X|Y). \tag{2.3}$$

$H(X)$ is the Shannon's definition of entropy defined as [46]:

$$H(X) := -\sum_{x} p(x) log(p(x)), \tag{2.4}$$

which calculates the uncertainty of the random variable X.

The symmetrical uncertainty between a feature x^i and the target label vector $\boldsymbol{t}$ is measured for each feature of $i = 1, 2, ..., d$. The feature with the largest value of $SU(x^i, \boldsymbol{t})$ is considered the predominant feature. A threshhold is later applied to filter the features and predominant features are used to eliminate other features. The redundancy among features can also be calculated using $SU(x^i, x^j)$ between the *i-th* and *j-th* feature.

2.1.2 Feature Extraction

Feature extraction is based on the manifold hypothesis stating that a high-dimensional data point lies on a low-dimensional sub-manifold or sub-space that is embedded within the original high-dimensional data space [36]. Feature extraction is a dimensionality-reduction method involving around creating an entirely new set of features $\boldsymbol{y} \in \mathbb{R}^p$ different from the original dimensions of $\boldsymbol{x} \in \mathbb{R}^d$ where $p \leq d$. The change of the feature space allows a better representation and visualization of dataset [17, 98]. The new smaller feature space $\boldsymbol{y}$ is called the embedded feature space and p is the intrinsic dimension of data.

There are two main categories of supervised and unsupervised feature extraction methods. The former takes the label vector, $\boldsymbol{t}$, into use during the training process while the latter discovers the patterns in the data.

Unsupervised Feature Extraction

Unsupervised feature-extraction methods do not use labels for feature extraction. Instead they look for patterns in data [56, 93].

- Principle Component Analysis (PCA): PCA was originally proposed in [90] as a linear unsupervised method meaning that the data lies on a linear sub-manifold and the classes can be distinguished linearly. The goal of PCA is to find an orthogonal direction in the space of data that best captures its variations [41]. The maximum variation of data is called the first principal component and is denoted by $\boldsymbol{u}$. Then the dataset is projected onto that direction, considered as rotating the coordinate system such that the main coordinate is in the direction of $\boldsymbol{u}$ [42].

  The data projected onto the direction of $\boldsymbol{u}$ is presented as $\boldsymbol{u}^\intercal \boldsymbol{X}$ where $\boldsymbol{X}$ is the data matrix. The variance of this projection is then $\boldsymbol{u}^\intercal S \boldsymbol{u}$ where S is the $d \times d$ co-variance matrix of the dataset. PCA tries to maximize this variance, which leads to eigenvalue problem of $S\boldsymbol{u} = \lambda \boldsymbol{u}$ meaning that the principal directions are the eigenvectors of the co-variance matrix of data. The first p terms of $\boldsymbol{U}$ are chosen for dimensionality reduction, where $\boldsymbol{U}$ is the principal component matrix [41, 56].

  The data can be reconstructed back using $\hat{\boldsymbol{X}} = \boldsymbol{U}\,\boldsymbol{Y}$, but with distortion error due to reducing the feature space to p sub-manifolds.

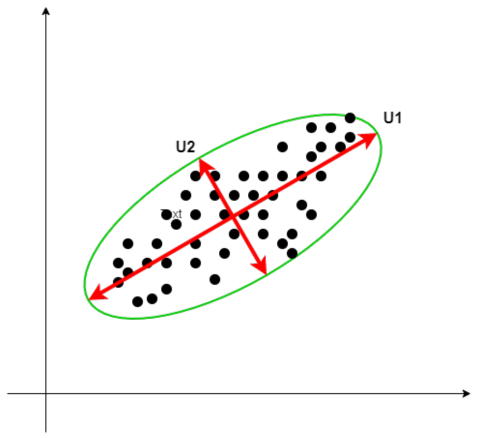


Figure 2.1: PCA with two main projection directions

- Independent Component Analysis (ICA): Originally defined by Jutten and Hérault in 1991 [64] and Comon in 1994 [23], is an extension of PCA that helps with non-Gaussian data. This method is best for a mixed set of independent sources to linearly transform the data with the hope of separating those sources [10]. Despite PCA with orthogonal projection axes, ICA finds a linear non-orthogonal coordinate system by finding the independent component vector $\boldsymbol{u}$ and the weight vector $\boldsymbol{w}$. By linear, we mean that ICA assumes the observation x_i is a linear mixture of the independent components of $\boldsymbol{u}$. Hence we can define $\boldsymbol{u} = \boldsymbol{w}.\boldsymbol{x}$ and it tries to minimize the mutual information among the axial projection of the input data [71].

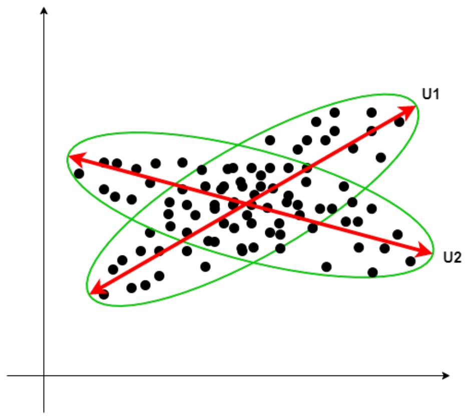


Figure 2.2: ICA with two main projection directions

Supervised Feature Extraction

Supervised feature extraction uses the labeled dataset to train its model. Using the label vector, the model can measure its accuracy during training, reduce the redundancy of the data and improved its classification accuracy [82, 94].

- Fisher Linear Discriminant Analysis (FLDA): Also known as FDA or LDA in literature, is a supervised feature-extraction method that projects data onto a new space with lower dimension based on an eigenvalue resolution. It was originally proposed by Fisher in [38]. Similar to PCA and IDA, FDA finds the projection direction of data as well but unlike the previous cases, it does not try to maximize the variation of data. FDA uses the Fisher criterion to find the optimum variation of data that increases intra-class variance while decreasing the inter-class variance [46].

  If we denote the intra-class scatter as S_b, inter-class scatter as S_w and the projection direction by u, we can formulate the FDA as [45, 112]:

$$J(\boldsymbol{u}) := \frac{\boldsymbol{u}^\top \boldsymbol{S}_B \boldsymbol{u}}{\boldsymbol{u}^\top \boldsymbol{S}_W \boldsymbol{u}}, \tag{2.5}$$

The goal of FDA is to maximize this ratio, which is $J(u)$ which leads to an eigenvalue problem of $\boldsymbol{S}_B\boldsymbol{u} = \lambda \boldsymbol{S}_W \boldsymbol{u}$ [40, 117]. Solving this optimization problem results in a projection matrix $\boldsymbol{U}$, where the projection directions are the column vectors of matrix $\boldsymbol{U}$ and the eigenvectors of $\boldsymbol{S}_W^{-1}\boldsymbol{S}_B$ sorted in descending order [84].

- Generalized Discriminant Analysis (GDA): This is a non-linear version of FDA in which the input space is mapped into a convenient higher-dimensional feature space, where the variables within this new feature space are non-linearly related to the original input space. Similar to FDA, the goal of GDA is to maximize the ratio of intra- and inter- class scatter in the new feature space using kernel functions. GDA solves the same eigenvalue problem [8, 114].

2.2 Large Margin Metric Learning

Measuring distance is a main part of pattern recognition in classification and regression. Normally, this is Euclidean distance defined as:

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 := (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top (\boldsymbol{x}_i - \boldsymbol{x}_j), \tag{2.6}$$

Euclidean distance, however, is not the best metric for high-dimensional datasets due to the geometry of the high dimensional spaces in those datasets. Hence, in high dimensional data, euclidean distance is not very effective. Metric learning suggests finding a new distance metric for the data based on the properties of the data itself, which means finding a new definition of similarity and dissimilarity between points in a dataset in a different way [88, 110]. Learning a metric for a model is equivalent to transforming the data into a new space and then computing the Euclidean distance in that new space. In this way, we might be able to represent some intrinsic structure of the data [3].

Fig. 2.3 shows that the Euclidean distance, which is the same in different directions for different data points, does not represent the true structure of the data. By choosing another metric and transforming the dataset into a discriminative subspace, we can see class 2 and 3 are above and below the separation line for class 1.

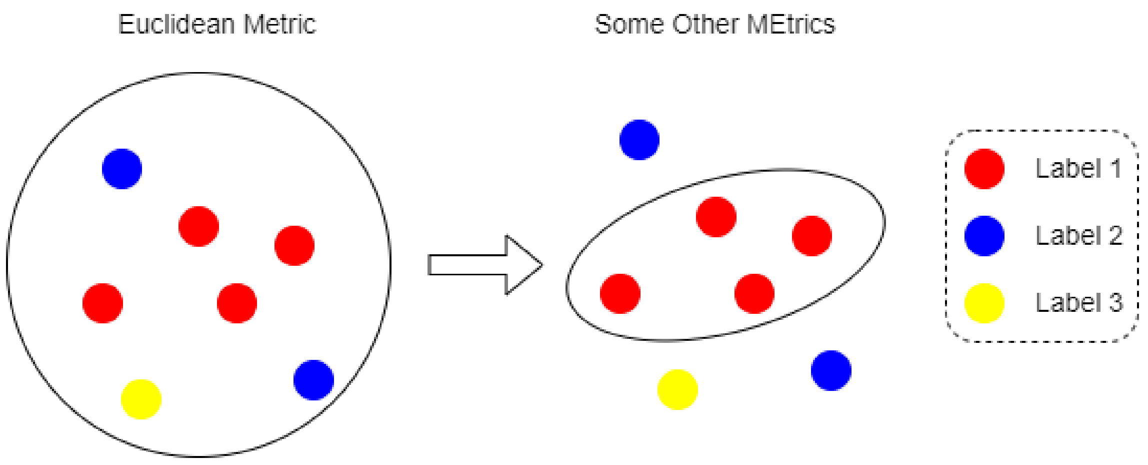


Figure 2.3: Metric learning transformation from euclidean metric

2.2.1 Large Margin Metric Learning for Nearest-Neighbor Classification

K-Nearest Neighbor (K-NN) is a supervised machine learning classifier based on feature similarity of a point's neighbors. It includes the known neighbors from the training dataset, in the majority vote to determine the class of an unknown instance. As was mentioned before, Euclidean distance does not weight the points and values all of them equally. However, K-NN highly depends on the metric by which the distance between data points is calculated.

In K-NN, Mahalanobis distance [79] is used to replace Euclidean distance. Mahalanobis metric can be viewed as a linear transformation with a projection matrix denoted by L [108]. Mahalanobis distance can be formulated as a general version of Euclidean distance with the formula below [48, 67, 110]:

$$\mathcal{D} := \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2 := \|\boldsymbol{L}^\top(\boldsymbol{x}_i - \boldsymbol{x}_j)\|_2^2 = (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{M}(\boldsymbol{x}_i - \boldsymbol{x}_j), \tag{2.7}$$

The matrix $\boldsymbol{M}$ should be positive semi-definite, i.e., $\boldsymbol{M} \succeq 0$, to meet the convexity and the triangle inequality criterion and to be considered as a metric. Due to its positive semi-definite nature, it can be decomposed to $\boldsymbol{M} := \boldsymbol{L}\boldsymbol{L}^\top$. It can be seen that the Euclidean distance is in fact a special case of Mahalanobis distance, where $\boldsymbol{M}$ is the identity matrix $\boldsymbol{i}$ [11].

Since distances are a crucial part of the metric-learning methods, we can improve these methods by trying to increase the intra-class variances while decreasing the inter-class variances by pulling the points from the same class towards each other and pushing points from other classes further [47]. Fig. 2.4 represents the push and pull method to improve K-NN.

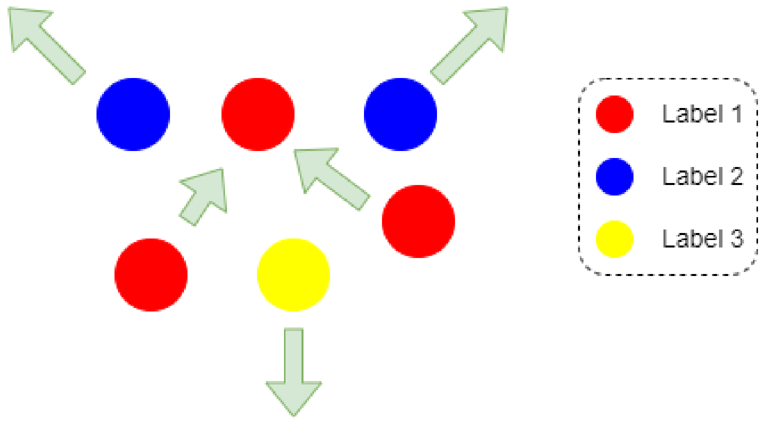


Figure 2.4: Metric learning for decreasing and increasing the intra- and inter-class variances, respectively, by pulling the positives (same class instances) toward the anchor but pushing the negatives (other class instances) away.

Let y_{il} denote whether two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_l$ are from the same class or not by having value of one or zero respectively. Also, let η_{ij} denote if a point $\boldsymbol{x}_j$ is among the K-nearest neighbors of the point $\boldsymbol{x}_i$ or not by having the value of one and zero respectively. In the push and pull method, the optimization problem of increasing and decreasing the intra- and inter-class scatter, can be defined as below and it should be minimized [109]:

$$\sum_{i,j} \eta_{ij} \|\boldsymbol{L}^\top(\boldsymbol{x}_i - \boldsymbol{x}_j)\|_2^2 + c \sum_{i,j,l} \eta_{ij}(1 - y_{il}) \Big[1 + \|\boldsymbol{L}^\top(\boldsymbol{x}_i - \boldsymbol{x}_j)\|_2^2 - \|\boldsymbol{L}^\top(\boldsymbol{x}_i - \boldsymbol{x}_l)\|_2^2\Big]_+, \tag{2.8}$$

where $[.]_+ := \max(., 0)$ is the standard Hinge loss, $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, and $\boldsymbol{x}_l$ are anchor, positive, and negative instances, respectively. The first half of the cost function in Equation 2.8 is for pushing the same-labeled classes towards each other and the second half of this equation represent a triplet loss [96]. Triplet loss considers a triumvirate of anchor, neighbor and distant, where anchor and neighbor belong to the same class and distant is a different-class point, with the purpose to reduce the distance between anchor and neighbors while increasing the distance between anchor and distant.

The cost function in Equation 2.8 can be re-stated as an SDP optimization problem below:

$$\begin{aligned}
\underset{\boldsymbol{M},\,\xi_{ijl}}{\text{minimize}} \quad & \mathcal{L} := \sum_{i,j} \eta_{ij} \, \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2 + c \sum_{i,j} \eta_{ij} \, (1 - y_{il}) \, \xi_{ijl}, \quad \forall l \\
\text{subject to} \quad & \|\boldsymbol{x}_i - \boldsymbol{x}_l\|_{\boldsymbol{M}}^2 - \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2 \geq 1 - \xi_{ijl}, \\
& \xi_{ijl} \geq 0, \\
& \boldsymbol{M} \succeq \boldsymbol{0},
\end{aligned} \tag{2.9}$$

The first half of the equation is not changed because of Equation 2.7. The hinge loss is represented in the second half of the equation above and is denoted by slack variable ξ_{ijl}.

2.2.2 Triplet loss and Siamese Network

As was mentioned before, triplet loss is a combination of anchor, neighbor and distant. It tries to pull the anchor and neighbor (positive) with the same class towards each other and push the distant (negative) away from them by minimizing the general loss function below [96]:

$$\ell_t = \sum_{i=1}^{b} [\|\boldsymbol{f}(\boldsymbol{x}_{\boldsymbol{a}}^{\boldsymbol{i}}) - \boldsymbol{f}(\boldsymbol{x}_{\boldsymbol{n}}^{\boldsymbol{i}})\|_2^2 - \|\boldsymbol{f}(\boldsymbol{x}_{\boldsymbol{a}}^{\boldsymbol{i}}) - \boldsymbol{f}(\boldsymbol{x}_{\boldsymbol{d}}^{\boldsymbol{i}})\|_2^2 + \alpha]_+, \tag{2.10}$$

The goal is for the anchor-distant distance, to be larger than the anchor-neighbor distance by a margin of $\alpha \geq 0$. In this formulation, $f(x)$ represents the embedding (output) of input x, x^i is the ith instant, b is the batch size or the number of triplets, $[.]_+ := \max(., 0)$ is the standard Hinge loss, and $\|.\|$ is the ℓ_2 norm. The anchor, neighbor and distant are denoted by x_a, x_n and x_d respectively. Fig. 2.4 also represents triplet loss's concept in which the anchor and positive, in this case, would be class 1 and the negative can be any of the class 2 or 3.

A Siamese network is a neural network method to train multiple sub-networks that share weights with each other [96]. The sub-networks can be trained using a triplet loss function, meaning that the anchor, neighbor and distant will be fed into these sub-networks and their weights are tuned using formula 2.10.

2.2.3 Triplet Mining Methods

the triplets can be sampled from the dataset using triplet mining methods such as Batch All (BA) [30], Batch Hard (BH) [58], Batch Semi-Hard (BSH) [96], Extreme Distances [97] and Negative Sampling [111]. These sampling methods work very well for datasets with large numbers of instances; however, their long training time compared to other triplet mining methods, is a disadvantage.

- Batch All (BA): This is a triplet mining method that considers all the positives and negative combination from the dataset, for each anchor [30].
- Batch Hard (BH): This is another hard triplet mining method meaning that it takes the farthest positive and the nearest negative with respect to the anchor into account for a Siamese neural network. The farthest positive and the nearest negative are the hardest to push and pull [58].
- Batch Semi-Hard (BSH): A triplet mining method that takes into account the nearest neighbor to the anchor that are farther from the positive [96].
- Extreme Distances: Distance plays an important role on choosing triplets in large margin metric learning. Neighbors, positive or negative, can be chosen as positive and negative based on their distances to the anchor (being near or far). This results in four different cases of Easy Positive Easy Negative (EPEN), Easy Positive Hard Negative (EPHN), Hard Positive Easy Negative (HPEN) and Hard Positive Hard Negative (HPHN). Easy and hard positives are the nearest and farthest positives to the anchor respectively while easy and hard negatives are the farthest and nearest negatives to the anchor respectively [97].
- Negative Sampling: In this mining method, for every positive to the anchor, negatives are chosen based on an occurrence probability that is calculated using stochastic probability of occurrence. The distribution of pairwise distances, denoted by $q(\mathcal{D})$, of two points can be estimated as [111]:

  $$q(\mathcal{D}) \propto \mathcal{D}^{d-2}(1 - 0.25\mathcal{D}^2)^{\frac{d-3}{2}}, \tag{2.11}$$

  Here d is the dimensionality of data and $\mathcal{D}$ is defined by Eq. (2.7). If anchor is represented as $\boldsymbol{x}_i$, and the negative is represented as $\boldsymbol{x}_l$, the probability of negative sample with distance $\mathcal{D}$ from the anchor can be presented using [102, 111]:

$$\mathbb{P}(\boldsymbol{x}_l \,|\, \boldsymbol{x}_i) \propto \min(\lambda, q^{-1}(\mathcal{D})), \tag{2.12}$$

Here, λ (e.g., 1.4) is for giving all the negatives a minimum chance of selection.

2.3 Classifiers

Classification is the task of assigning a label y_j to an unknown input test instant x_i, where $i \in \mathbb{R}^n$ based on the test instance attributes and the previously obtained machine learning model from training procedure. The algorithm mainly calculates a probability function of a test instance belonging to a class with respect to its features and attributes. Many machine learning classifiers exist and can be named such as K - Nearest Neighbor (K-NN), Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), and Random Forest (RF).

2.3.1 K - Nearest Neighbor (K-NN)

$K - NN$ is a well-known Euclidean-based classifier for discriminant analysis that don't have a prior knowledge of data distribution. In $K - NN$, a test instance's probability of belonging to a class is determined by taking a majority vote of K number of its nearest neighbors in the model training set [25].

If the training set is denoted by $T = (x_i, y_i)$ for $i = 0, ..., N$ where x_i is the training vector and y_i is the corresponding label vector. To classify a test instance x', N nearest neighbor is identified based on the euclidean distance to the instance, as a vector set of $T' = (x_k^{NN}, y_k^{NN})$ where $k = 0, ..., K$, using the formula in equation 2.6 [91]. The majority vote of these neighbours, determine the predicted class of test instance x' using the formula below [52]:

$$y' = argmax_y \sum_{(x_i^{NN}, y_i^{NN}) \in T'} \delta(y = y_i^{Nn}) \tag{2.13}$$

where y_i^{NN} is the label of the ith instance from the nearest neighbor vector T' and δ takes the value of one if $y = y_i^{NN}$ and zero otherwise.

2.3.2 Quadratic Discriminant Analysis (QDA)

QDA is a supervised classification method [69] that assumes a Gaussian distribution of classes. The co-variance matrices of classes are not assumed to be necessary equal. Hence the decision boundaries between classes are quadratic. The probability of a point belonging to a class is based on Bayes posterior using the formula below [57, 69]:

$$P(x \in \mathcal{C}_1 | X = x) = \frac{P(X = x | x \in \mathcal{C}_1) P(x \in \mathcal{C}_1)}{P(X = x)} = \frac{f_1(x)\pi_1}{\sum_{k=1}^{|c|} P(X = x | x \in \mathcal{C}_k)\pi_k}, \tag{2.14}$$

where x represents an instance, $|\mathcal{C}|$ is the number of classes, $f_1(x)$ is the likelihood and π_1 is the prior probability. k is chosen so the posterior probability is maximized.

2.3.3 Linear Discriminant Analysis (LDA)

As was mentioned in section 2.1.2, LDA is a supervised feature extraction and classification method [69] that is a special case of QDA. In LDA, the classes have equal co-variance matrices and data distribution is considered to be Gaussian. Hence, the decision boundary between classes is a linear line in this method [43, 101, 103]. The probability of a point belonging to a class is based on Bayes classification in equation 2.14.

2.3.4 Random Forest (RF)

RF is a classifier that uses an ensemble of decision trees to train a model to predict the labels of unknown instances. The decision trees are created by sampling instances and features with replacement from the training set. The test instance traverses the created trees and a majority vote among the results determines the final class of data point [26, 74].

2.4 Random Forest (RF)

Random Decision Forests were first introduced by Ho in [59, 60] and were further developed to their current format by Breiman in [13]. An RF is an ensemble of uncorrelated decision trees that are created using bagging from a training dataset, for classification and regression purposes. This classifier predicts a label for the unknown input instance by taking a majority vote among traversed trees. The training model is created using a subset of data called the training dataset $X_{train} = \{(x_m^d, y_m) | m, d \in \mathbb{R}\}$, where d is the number of attributed or features of dataset, m is the number of instances in X_{train} and y is the associated label. An RF classifier creates a forest of trees by sampling m and d randomly from X_{train} to create a forest of uncorrelated trees [74].

Fig. 2.5 shows the general structure of random forest in which the training subset of dataset is partitioned recursively starting at a root node, until it reaches the leaf node, based on an objective function. The leaf node can not be split further due to reaching either purity or a certain criteria. A label is assigned to every leaf node by majority vote as well [14].

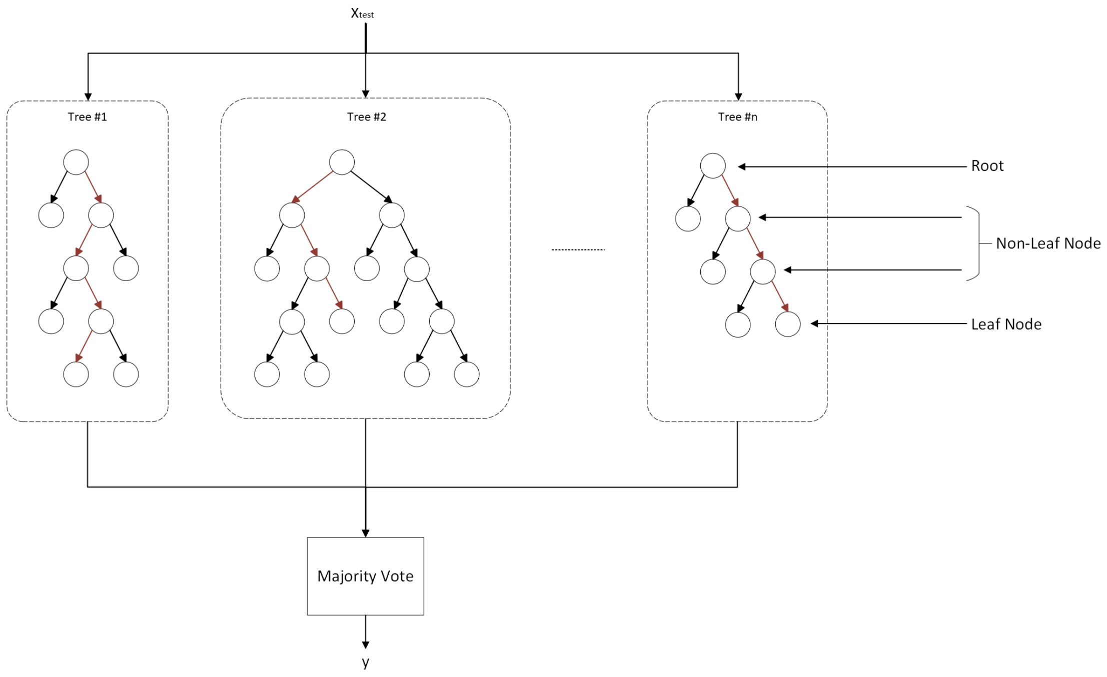


Figure 2.5: Random Forest

A test instance should traverse all trees, starting from the root node, passed through non-leaf node to reach a leaf node. The next destination is determined via a sensitivity function at every non-leaf node. The label of the final leaf node, in which the x_{test} lands, is the predicted label for that particular tree. An RF classifier takes a majority vote over all trees in the ensemble, to determine the actual class of the unknown instance.

2.4.1 Decision Tree Algorithm

A subset of the training dataset, X_{train}, is chosen randomly in the sense of both feature and instance using sampling with replacement technique [13]. This subset is then partitioned recursively from root to leaf nodes, where no further split is possible due to purity or meeting a stopping criterion, as shown in Fig. 2.6. A non-leaf node divides based on a split criterion also referred to as a weak learner. Fig. 2.6 is a 2-dimensional feature space example with two labels. Finally, a label will be associated to leaf nodes by taking majority vote among its instances.

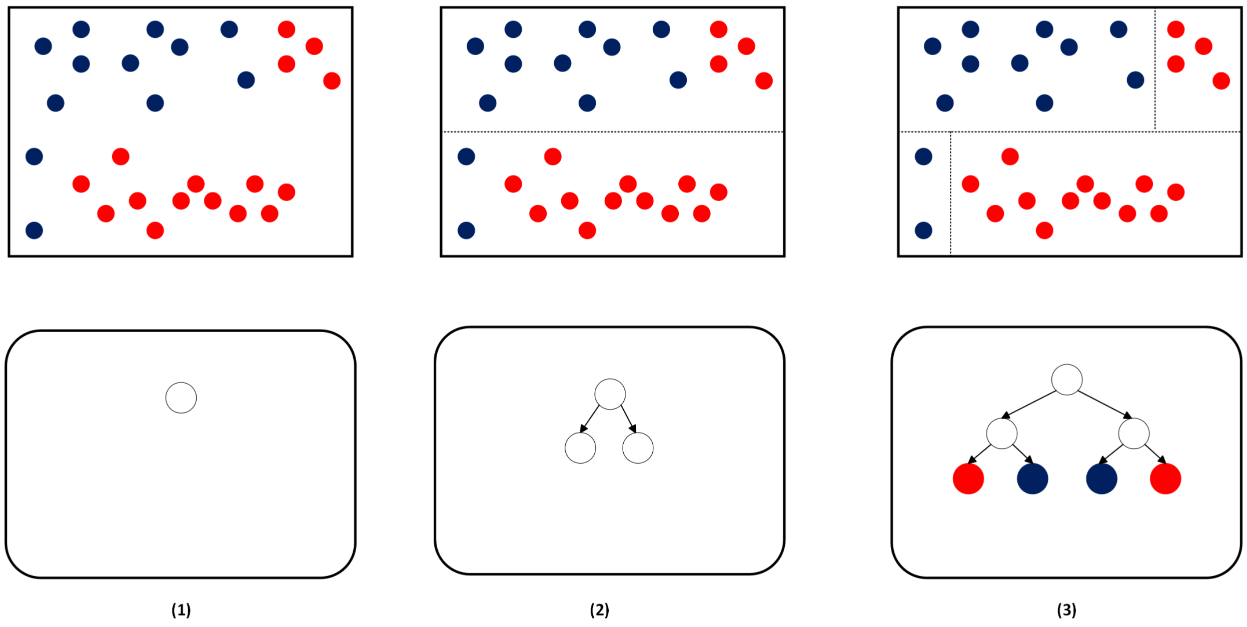


Figure 2.6: Decision Tree Algorithm

2.4.2 Split Criteria

A non-leaf node splits based on a split criterion, also known as a weak learner, which is defined as an optimization problem in which we optimize function I [21]:

$$\theta_i = \arg\max I(S_i, \theta), \tag{2.15}$$

where θ_i represents the weak learner that is chosen among a list of potential criteria θ for splitting a node. S_i the subset of instances under split belonging to the node i. The list of potential split criteria θ is defined as [21]:

$$\theta = (\phi(v), \tau) \;\; v \in S_i, \tag{2.16}$$

where v is the feature vector of instances belonging to S_i, $\phi(v)$ is simply the value of chosen attribute v, τ is the threshold by taking median of two non-identical adjacent nodes, and the relation of $\phi(v) \geq \tau$ in order to split the S_i to two branches at node i.

In summary, θ is created by selecting an attribute and extracting all the values of that attribute from instances in S_i. A list of thresholds τ is then defined by taking median of two non-identical adjacent values.

As was mentioned, I is the objective function that determines the optimal weak learner θ_i in equation 2.15 from the list of θ and it is defined as [21]:

$$I = i(N) - P_L i(N_L) - P_R i(N_R), \tag{2.17}$$

where i is the impurity measurement function, N is the node under study, L and R substances refer to the left and right partitions to be constructed from the possible split respectively, and P_L and P_R are the proportions of the node instances to be in left and right branches, respectively.

The impurity measurement function is method-dependant and can be either Gini impurity based defined in equation 2.18, or Entropy impurity based defined in equation 2.19 [1].

$$\text{Gini i} = \sum_{(i \neq j)} P_i P_j = \frac{1 - \Sigma_j P_j^2}{2}, \tag{2.18}$$

$$\text{Entropy i} = \sum_{i=1} P_i.(-\log_2 P_i), \tag{2.19}$$

where P_i and P_j are the proportion of instances in classes i and j, respectively.

2.4.3 Related Work on Random Forest Implementation in Hardware

This section is dedicated to the previous works on the implementation of random forests or on decision trees as the building unit of an RF classifier, on various platforms, with a focus mostly on the training phase of the algorithm. Hardware implementation of random forests, especially on Field Programming Gate Array (FPGA) FPGA, is an interesting subject for scientists due to FPGAs' on-chip memory bandwidth, flexible logic and memory access, and possibility to exploit parallelism, which can result in faster execution time.

Narayanan, *et al.* [86] implemented a gini calculation unit for FPGA implementation that accelerates the search for optimal split point by using only dividers and adders. In this implementation, the run time of a single decision tree was accelerated by 5.58×. The sorting module, however, was done in the host computer. Their method is based on vertical data parallelism in which the data of different attributes are distributed to different nodes to be processed and a global comparison among all, will find the optimal split point.

With the importance of decision trees in the random forest algorithm, Saqib, *et al.*, implemented a pipelined decision tree to achieve a faster execution time of a single tree due to concurrent execution, with a reasonable resource efficiency. Their method was capable of independently processing the data from a streaming source for parallel processing nodes by incorporating concurrent engine for parallel operations on a dataset. Their design is highly scalable but is limited to binary trees [95].

Essen, *et al.* proposed a novel hardware architecture to handle the memory utilization problem in FPGA in previous works by using a compact random forest (CRF) with shallow-depth trees (with a maximum depth of six) [106]. Using compact decision trees made the FPGA implementation more VLSI-friendly. They also compared a high-utilization FPGA implementation against a single-instruction multi-thread (SIMT) algorithm on GP-GPU and multi-core implementation. In this comparison, the FPGA showed higher classification performance with lower power consumption.

Rastislav, *et al.* [100] proposed an FPGA architecture with an oblique/non-linear weak learner, parallel calculations for weak learners, and assimilate a new algorithm called HereBoy [73] to explore large search spaces.

An architecture was proposed in [20] to accelerate not just a single decision tree, but a complete training phase for a random forest classifier. This method incorporates parallel computation of trees to improve the execution speed but it also used batch-learning method in which the dataset is presented to the algorithm batch-by-batch from off-chip memory. The number of rounds of transmission from off-chip memory to FPGA (for input data) and from FPGA back to double-data-rate (DDR) SDRAM memory (to write the predictions back) is not very efficient for large datasets. They also had memory issues for having parallel computations. A FIFO-based merge sorter was also developed for its fast output throughput and low resource utilization. The memory utilization problem was later improved by suggesting data compression and decompression scheme to facilitate sorting in [21]. Different methods and solutions were later proposed in [19] to address small problems or improve upon previous works.

Oberg, *et al.* [87] use the decision tree classifier for body part and gesture recognition in a Microsoft Kinect image sensor. They propose an optimized FPGA implementation of a Forest Fire pixel classification algorithm. Their design communicates with external DDR memory and the paper discussed the opportunity of expanding bandwidth to external DDR memory.

Lin and Zhang [76] explore a quantile-based online approach to train a random forest using Hoefding trees and compares their result with batch learning method in [19]. This work optimizes the quantile parameters and incorporates pipelining and incorporates dynamic memory, which results in higher accuracy comparing to batch learning method.

Nakahara, *et al.*, uses fixed-point representation of data instead of a floating point in [85] for a pipelined multi-valued decision diagram random forest. Parallel pipelined decision trees are used in other works as well such as [66]. However, they have a common issue of how many trees can be fitted in the design and how deep each tree can be. Other methods, such as sub-sampling, are presented in works such as [72] to accelerate the training process, in comparison to the online training and batch learning method. This approach eases the training process on large datasets but the result is not comparable with previous methods.

2.5 Conclusion

This chapter was dedicated to defining concepts in details that are used in the proposed methods in the next chapters. The topics discussed were dimensionality reduction and methods to achieve it, large margin metric learning approaches to classification and regression, followed by classifiers, and finally random forest classifiers. Decision trees, as the building blocks of RF ensembles, were also discussed and the previous works in the literature regarding hardware implementation of RF classifiers was introduced as well. The purpose of this section was to achieve an understanding of topics discussed in order to realize the importance of an optimized hardware architecture for machine learning algorithms.

Chapter 3

Hierarchical Approach for Large Margin Metric Learning and Subspace Learning in Software

In this chapter, we evaluate the run-time and accuracy improvements of triplet mining methods, originally introduced for Siamese networks, for Large Margin Metric Learning. We study a new hierarchical approach as well that shows improvement in run-time compared to other explored methods in literature.

As was mentioned in section 2.2.1, the general optimization problem to be solved is bringing the anchor and positive closer together by decreasing the inter-class variances and distancing the negative by increasing the intra-class variances. The optimization problem in equation 2.9 is the mathematical representation of this goal. However, this equation is taking all the negative instances into consideration. This will make the Semidefinite Programming (SDP) problem time-consuming for large datasets [107]. Inspired by metric learning definitions in [109, 110], we proposed triplet mining methods based on the objective functions of equation 2.9, to speed up the SDP for solving the optimization problem. The methods described in this chapter are $K-$Batch All, $K-$Batch Hard, $K-$Batch Semi-Hard, Extreme Distances, and Negative Sampling, for large margin metric learning.

3.1 K-Batch All

Batch All triplet mining takes all the possible combinations of positive-negatives into account, which is very time-consuming for SDP due to the huge number of permutations. The method in [109, 110] is $K-$ Batch All version, which only considers K nearest positives and all negatives. The objective function in equation 2.9 can be re-phrased as:

$$\begin{aligned}
\underset{\boldsymbol{M}, \xi_{ijl}}{\text{minimize}} \quad & \mathcal{L} = \sum_{i,j} \eta_{ij} \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_{\boldsymbol{M}}^2 + c \sum_{i,j} \eta_{ij} \left(1 - y_{il}\right) \xi_{ijl}, \quad \forall l \\
\text{subject to} \quad & \left\|\boldsymbol{x}_i - \boldsymbol{x}_l\right\|_{\boldsymbol{M}}^2 - \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_{\boldsymbol{M}}^2 \geq 1 - \xi_{ijl}, \\
& \xi_{ijl} \geq 0, \\
& \boldsymbol{M} \succeq \boldsymbol{0},
\end{aligned} \tag{3.1}$$

where η_{ij} is one if x_j is among the K nearest neighbors of x_i and zero otherwise, $\|.\|_M$ denotes the Mahalanobis distance, c is a constant, y_{il} is one if x_i and x_j belong to the same class and zero otherwise, ξ_{ijl} is the non-negative slack variable.

3.2 K-Batch Hard

Inspired by the Batch Hard method, which considers the nearest negatives and farthest positives to the anchor [58], $K-$ Batch Hard considers K farthest positives and K nearest negatives. The objective function of equation 2.9 now becomes:

$$\begin{aligned}
\underset{\boldsymbol{M}, \xi_{ijl}}{\text{minimize}} \quad & \mathcal{L} = \sum_{i,j} \gamma_{ij}^{+} \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_{\boldsymbol{M}}^2 + c \sum_{i,j,l} \gamma_{ij}^{+} \, \eta_{il}^{-} \left(1 - y_{il}\right) \xi_{ijl}, \\
\text{subject to} \quad & \left\|\boldsymbol{x}_i - \boldsymbol{x}_l\right\|_{\boldsymbol{M}}^2 - \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_{\boldsymbol{M}}^2 \geq 1 - \xi_{ijl}, \\
& \xi_{ijl} \geq 0, \\
& \boldsymbol{M} \succeq \boldsymbol{0},
\end{aligned} \tag{3.2}$$

where γ_{ij}^{+} is one if x_j is among the K farthest positive neighbors of x_i and zero otherwise. The positive sign is for same class neighbors. Here, η_{il}^{-} is one if x_l is among the K nearest negative neighbors of x_i and the negative sign is for distant classes.

3.3 K-Batch Semi-Hard

Batch Semi-Hard is a Siamese network mining method that considers the closest negatives to the anchor that are farther from the positives [96]. Inspired by this method, we consider K positive instances and for each positive, we have K nearest negatives that are farther from the chosen positive. The cost in function 2.9 accordingly will be:

$$\begin{aligned} \underset{\boldsymbol{M}, \xi_{ijl}}{\text{minimize}} \quad & \mathcal{L} = \sum_{i,j} \eta_{ij} \, \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2 + c \sum_{i,j} \eta_{ij} \sum_{l} \eta_{il}^{\mp} \, (1 - y_{il}) \, \xi_{ijl}, \\ \text{subject to} \quad & \|\boldsymbol{x}_i - \boldsymbol{x}_l\|_{\boldsymbol{M}}^2 - \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2 \geq 1 - \xi_{ijl}, \\ & \xi_{ijl} \geq 0, \\ & \boldsymbol{M} \succeq \boldsymbol{0}, \end{aligned} \tag{3.3}$$

where η_{ij} is one if x_j is among the K nearest neighbors of x_i that are positive and zero otherwise, $\eta_{il}^{\mp}$ is one if x_l is among the K negative neighbors of x_i that are farther from x_j to x_i and zero otherwise.

3.4 K-Extreme Distances

Based on the four cases introduced in section 2.2.3, four new cases of EPEN, EPHN, HPEN, HPHN can be introduced in which only K positive and negative instances are chosen. $K-$HPHN is equivalent to the Batch Hard method previously explained in section 2.2.3. The hard cases are interesting due to the opposition learning [104] and also because they are the more difficult cases and solving them increases the value of the solution. However, it does not decrease the importance of easy cases in literature [113]. The objective of the cost function in equation 2.9 to be implemented will change accordingly:

$$k\text{-EPEN:} \quad \mathcal{L} = \sum_{i,j} \eta_{ij} \, \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2 + c \sum_{i,j,l} \eta_{ij} \, \gamma_{il}^{-} \, (1 - y_{il}) \, \xi_{ijl}, \tag{3.4}$$

$$k\text{-EPHN:} \quad \mathcal{L} = \sum_{i,j} \eta_{ij} \, \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2 + c \sum_{i,j,l} \eta_{ij} \, \eta_{il}^{-} \, (1 - y_{il}) \, \xi_{ijl}, \tag{3.5}$$

$$k\text{-HPEN:} \quad \mathcal{L} = \sum_{i,j} \gamma_{ij}^{+} \, \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2 + c \sum_{i,j,l} \gamma_{ij}^{+} \, \gamma_{il}^{-} \, (1 - y_{il}) \, \xi_{ijl}, \tag{3.6}$$

where η indicates the nearest neighbors, γ is for farthest neighbors, the positive sign is for same class points and the negative sign is for distant classes. η_{ij} is one if x_j is among the K nearest neighbors of x_i and is zero otherwise. η_{il}^{-} is one if x_l is among the K nearest negatives of x_i. γ_{ij}^{+} is one if x_j is among the K farthest positive neighbors of x_i and is zero otherwise. γil^{-} is one if x_l is among the K farthest negative neighbors of x_i. $y_{il}$is one if x_l is a positive to anchor x_i. Finally ξ_{ijl} is the non-negative slack variable.

3.5 K-Negative Sampling

Based on the negative sampling method explained in section 2.2.3, K-Negative Sampling (K-NS) is introduced in which K positives and K negatives for every anchor-positive pair are chosen. The cost function in equation 2.9 becomes:

$$\begin{aligned}
\underset{\boldsymbol{M},\,\xi_{ijl}}{\text{minimize}} \quad & \mathcal{L} = \sum_{i,j} \eta_{ij}\, \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2 + c \sum_{i,j,l} \eta_{ij}\, \rho_{il}^{-}\, (1 - y_{il})\, \xi_{ijl}, \\
\text{subject to} \quad & \|\boldsymbol{x}_i - \boldsymbol{x}_l\|_{\boldsymbol{M}}^2 - \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\boldsymbol{M}}^2 \geq 1 - \xi_{ijl}, \\
& \xi_{ijl} \geq 0, \\
& \boldsymbol{M} \succeq \boldsymbol{0},
\end{aligned} \tag{3.7}$$

where η_{ij} is one if x_j is among the K nearest neighbors of x_i and is zero otherwise. ρ_{il}^{-} is one if x_l is among the negatives for the anchor-positive pair of (x_i, x_j).

3.6 Proposed Hierarchical Large Margin Metric Learning with Stratified Sampling

Even though the previous methods increase the speed of the SDP for solving the optimization problem, we propose a new hierarchical method to speed up SDP. The main idea relies on the previous concept of training only on portions of data. However, SDP needs the whole dataset in order to solve the optimization problem in equation 2.9. Hence the data is introduced in hierarchical portions. It is similar to a divide-and-conquer method to accelerate SDP [24].

This method is a hierarchical approach with stratified sampling [6, 44] and model averaging [12, 22]. It is an iterative method that uses several hyper-spheres to solve the

SDP at every iteration. The triplets are sampled from the points within each hyper-sphere in the data space by stratified sampling [6] where every class of data is homogeneous subgroups or strata. The sampled data points will be used to solve the optimization problem in equation 2.9 rather than using the whole dataset. In this way, the pace of the SDP improves.

Algorithm 1 can be used to follow the text explaining the algorithm in details in this chapter. This approach, in general, trains using portions of data iteratively in order to use the whole dataset while improving the speed of training.

At every iteration, several hyper-spheres, denoted by n_s, are applied on the data space and then triplets are sampled from the data points within each hyper-sphere using stratified sampling (see Line 10 in Algorithm 1). At every iteration, the optimization equation 2.9 is solved using the sampled triplets rather than the whole dataset (see Line 11 in Algorithm 1). Matrix $\boldsymbol{M}$ in equation 2.7 is positive semi-definite $\boldsymbol{M} \succeq 0$ to meet the convexity criteria and hence can be factorized to $\boldsymbol{L}\boldsymbol{L}^\top$ using eigenvalue decomposition:

$$\boldsymbol{M} = \boldsymbol{\Psi}\boldsymbol{\Sigma}\boldsymbol{\Psi}^\top = \boldsymbol{\Psi}\boldsymbol{\Sigma}^{(1/2)}\boldsymbol{\Sigma}^{(1/2)}\boldsymbol{\Psi}^\top = \boldsymbol{L}\boldsymbol{L}^\top, \tag{3.8}$$

The column space of $\boldsymbol{L}$ is used in algorithm 1 to project the whole dataset, similar to the principles of PCA (see Line 13 in Algorithm 1) [62]. In order to not collapse into low-rank subspaces, the diagonal of matrix $\boldsymbol{M}$ can be strengthened. This results in larger eigenvalues but it does not affect the projection direction [81].

Fig. 3.1 indicates the procedure in algorithm 1. At every iteration, the radius of hyper-spheres denoted by r, is increased with respect to the iteration index, denoted by τ, because as we progress through the algorithm in data space, we want to see more data and cover more of the data space, hence we need to have larger hyper-spheres. Contradictory to the radius, the number of hyper-spheres, n_s, is decreasing, in order for the hyper-spheres not to overlap the sampling area. As the iteration index enlarges, the more data has already been seen, so there is no need to consider all data points to sample from in each hyper-sphere. In addition to this, the more data is in each hyper-sphere to sample from, and this can slow the SDP optimization. For these reasons, at every iteration, the size of the stratified sampling, denoted by n_τ is decreasing. The sampling portion, denoted by p_τ, is a function of iteration index.

The initial radius, number of hyper-spheres, and the portion of sampling are $r := 0.1\sigma$, $n_s := \lfloor 0.01 \times n \rfloor$ (clipped to $10 \leq n_s \leq 20$), and $p_\tau := 1$. At every iteration, the function is updated as $r := r + \Delta r$, $n_s := \max(n_s - \lceil 0.2 \times n_s \rceil, 1)$, and $p_\tau := \max(p_\tau - 0.05, 0.2)$, where $\Delta r := 0.3\sigma$ where σ is the average standard deviation along features.

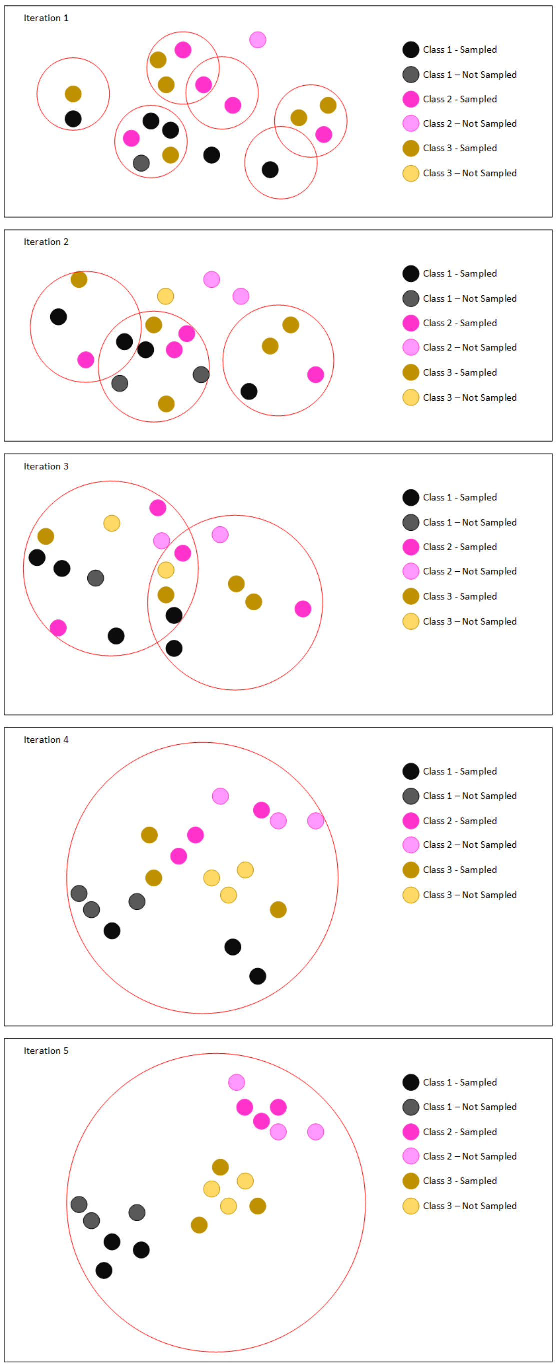


Figure 3.1: Hierarchical large margin metric learning with stratified sampling

```
Procedure: Hierarchical Metric Learning($\boldsymbol{X}$, $k$)
Input: $\boldsymbol{X}$: dataset, $k$: number of neighbors
Initialize $r$, $n_s$, and $p_\tau$
for $\tau$ from 1 to $T$ do
    $r$ := increasing function of $\tau$
    $n_s$ := decreasing function of $\tau$
    $p_\tau$ := decreasing function of $\tau$
    for $s$ from 1 to $n_s$ do
        $\boldsymbol{c}_s \sim \text{range}(\boldsymbol{X})$
        $\{\boldsymbol{x}_{i,a}, \boldsymbol{x}_{i,p}, \boldsymbol{x}_{i,n}\}_{i=1}^{n_\tau} \leftarrow$ draw a stratified triplet sample with sampling
          portion $p_\tau$ within the $s$-th hypersphere
        Solve optimization (2.9)
        Decompose $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^\top$ using Eq. (3.8)
        Project $\boldsymbol{X}$ onto $\mathbb{C}\text{ol}(\boldsymbol{L})$: $\boldsymbol{X} \leftarrow \boldsymbol{L}^\top \boldsymbol{X}$
```

Algorithm 1: Hierarchical Large Margin Metric Learning

3.7 Experimental Results and Analysis

3.7.1 Datasets and Setup

The proposed metric algorithms were tested on three publicly available datasets of Fisher Iris [31, 38] with 150 data points in three classes and feature space of four. The second dataset is ORL faces [4] with 40 classes. Each class has 10 subjects, each with the size of 112×92 pixels. Lastly, the MNIST digit dataset [70] with 28×28 pixel images.

In the pre-processing step, Fisher Iris and MNIST datasets were split to train-validation-test subsets with portions of $70\%-15\%-15\%$ respectively. The MNIST dataset was further projected on principal component analysis subspace with dimensionality of 30 [62]. For the ORL faces, the first 6 images were chosen for training and the rest were equally divided for validation and test. The ORL faces were also projected on their 15 leading eigenfaces [105]. The validation determines the optimal value of k and c.

3.7.2 Comparing Large Margin Metric Learning Methods in Non-Hierarchical and Hierarchical Approaches

Table 3.1 represents the accuracy and the run time of tested datasets for all the non-hierarchical approaches and proposed hierarchical one using Mahalanobis distance. High and compatible accuracies can be observed in non-hierarchical methods in the Iris dataset. However, the hierarchical approach shows perfect accuracies in this dataset. This is because of the small size and the simplicity of Fisher Iris.

Among non-hierarchical approaches, $k-$BH reports the highest accuracy in all datasets. In the hierarchical method, $k-$BSH reports top accuracies especially in ORL faces and MNIST. This is due to both methods using hard negatives (negatives closer to anchor) for training. This helps with avoiding over-fitting the training data. The same logic is applied on ORL faces with $k-$BH and $k-$EPHN reporting top accuracies. This logic can also be validated by observing $k-$BSH having the second top accuracy in ORL dataset. The $k-$NS has acceptable performance due to the effectiveness of the probability distribution used for sampling from the negative instances [111].

It can be seen from table 3.1 that the hierarchical approach is reporting either better accuracies or comparable results with lower run-times than non-hierarchical approaches due to model averaging. There are some cases in which the non-hierarchical approach is reporting better accuracies such as $k-$BH and $k-$EPHN in ORL faces and MNIST. This can be explained with random sampling from hyper-spheres rather than using the whole data. The sampling makes the hierarchical approach faster and more scalable.

The time consumed by hierarchical approach is less than non-hierarchical approaches in most cases. This is highlighted, especially in $k-$BA for ORL faces and MNIST datasets. While these datasets are too large for the non-hierarchical $k-$BA and they can take forever, they can easily be run on the hierarchical approach. $k-$BA obtains the longest run-time due to considering all the negative instances. $k-$BSH and $k-$HPEN also show long run-times due to handling the hard cases.

3.7.3 Comparing Triplet Mining Methods by Ghost Faces

Motivated by using eigenfaces for face recognition [105] and fisherfaces [9], we can illustrate the large margin metric learning methods, using the eignevalue decomposition in equation 3.8. The top ten columns of L (projection matrix), sorted in descending order, are chosen to display the ghost faces. Fig. 3.2 represents ghost faces for different triplet mining methods described in this chapter. As can be seen in Fig. 3.2, the projection directions

Table 3.1: Comparing accuracies and run-time of the proposed triplet mining methods in both non-hierarchical and hierarchical metric learning for nearest-neighbor classification.

Dataset			k-BA	k-BH	k-BSH	k-HPEN	k-EPEN	k-EPHN	k-NS
Iris	Non-Hierarchical	Accuracy (%)	72.73	100	86.36	95.45	81.82	95.45	72.73
		Time (sec)	832.85	5.51	6.62	4.77	5.34	5.11	5.06
	Hierarchical	Accuracy (%)	100	100	100	100	100	100	100
		Time (sec)	23.73	9.72	4.54	7.25	4.73	5.05	4.64
ORL Faces	Non-Hierarchical	Accuracy (%)	–	85.00	78.75	72.50	75.00	85.00	77.50
		Time (sec)	–	16.13	18.61	19.59	19.19	16.31	19.05
	Hierarchical	Accuracy	76.25	76.25	81.25	78.75	78.75	81.25	63.75
		Time (sec)	0.39	0.93	0.79	4.36	1.07	0.95	0.39
MNIST	Non-Hierarchical	Accuracy (%)	–	82.00	79.00	82.00	78.00	82.00	78.00
		Time (sec)	–	122.21	182.13	152.89	173.18	135.64	170.33
	Hierarchical	Accuracy (%)	71.00	77.00	79.00	81.00	75.00	78.00	79.00
		Time (sec)	27.17	1.56	1.55	0.49	1.02	1.70	1.55

of ORL facial dataset are some facial features which are like ghost faces. Each face vary within a range of gray where some facial feature differ in color among the set of training sets. This figure captures different facial features that are discriminative to the dataset under study.

All triplet mining methods extracted discrimination features by finding facial features such as eye, eyebrow, cheeks for glasses, chin, hair, and nose. Since there are cases of faces with eye glasses in the dataset, many features related to eyes and cheeks have been extracted. $k-$NS shows more distinguished features. After $k-$NS, $k-$BSH, $k-$HPEN, and $k-$EPEN also show diverse features of eyes, cheeks, nose and hair. Eyes and eyebrows are mostly chosen for discriminating classes especially in $k-$BH and $k-$EPHN, due to faces with eye glasses in the dataset.

3.7.4 Comparison of Triplet Mining Methods by Subspaces

The projection of training and test data, in the wine dataset, onto the column space of $\boldsymbol{L}$ is shown in Fig. 3.3 for the different triplet mining methods. This figure shows acceptable discrimination of classes after projection onto the metric subspaces for all different triplet mining techniques. As can be seen, separation is better for k-BH, k-EPHN, and k-HPEN where in k-HPEN, the view is from the top and separation in the third dimension is not visible. The reason for better discrimination of classes in these cases is considering the hard instances which helps the metric subspace to learn the worst-case scenarios. Interestingly, these results correlate with the timing measured in Table 3.1. As we can see, the cases considering the hard instances also take longer to run.

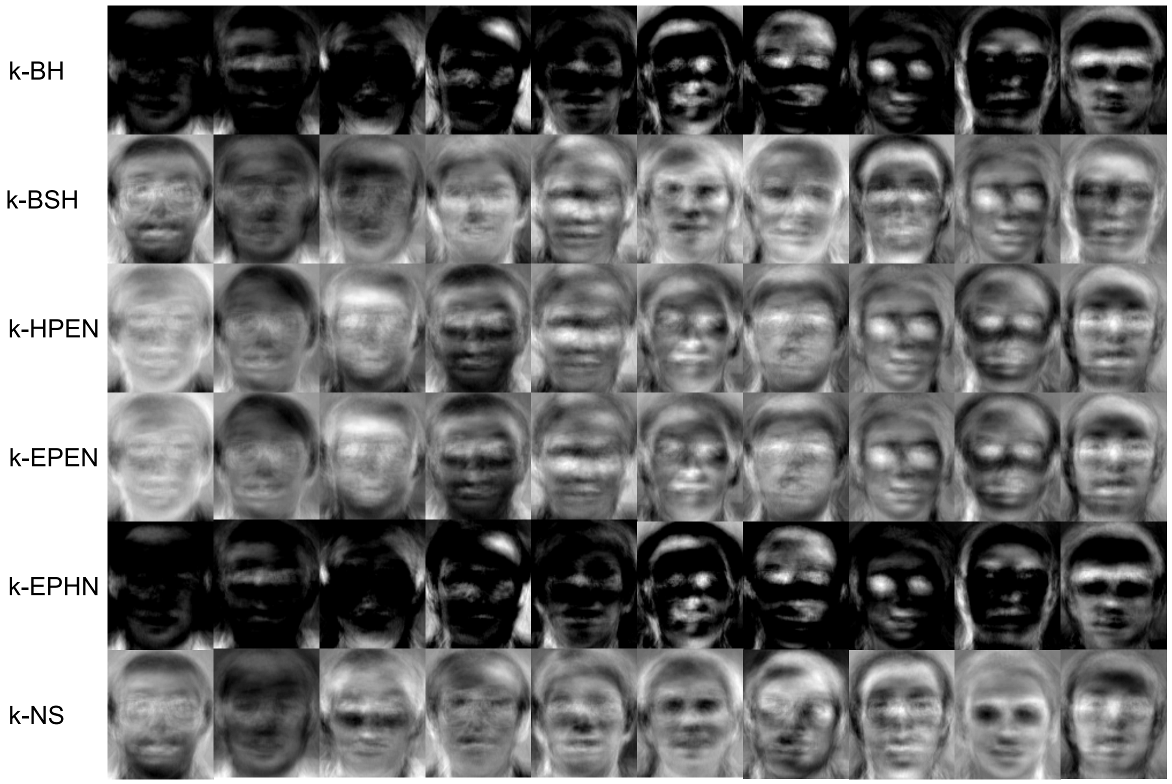


Figure 3.2: The top ten ghost faces in different triplet mining methods.

3.8 Proposed Hierarchical Approach for Subspace Learning Methods

As was seen in section 3.7, the hierarchical approach improves the accuracy and run-time for datasets with large number of instances. However, most large datasets not only have a high number of instances, but also a large feature space. Visualization, representation, and exploration of complex datasets with high number of features is hard and non-intuitive. The hierarchical approach in section 3.6 can further be improved to reduce the dimensionality of data as well to better discriminate the classes.

The procedure is shown in algorithm 2 and it is very similiar to the original algorithm introduced in section 3.6. At every iteration, several hyper-spheres are applied on the feature space. The feature space of every hyper-sphere is either untouched, which is the same as the original feature space of $1 \times d$, reduced to a random subset of $1 \times p$ where

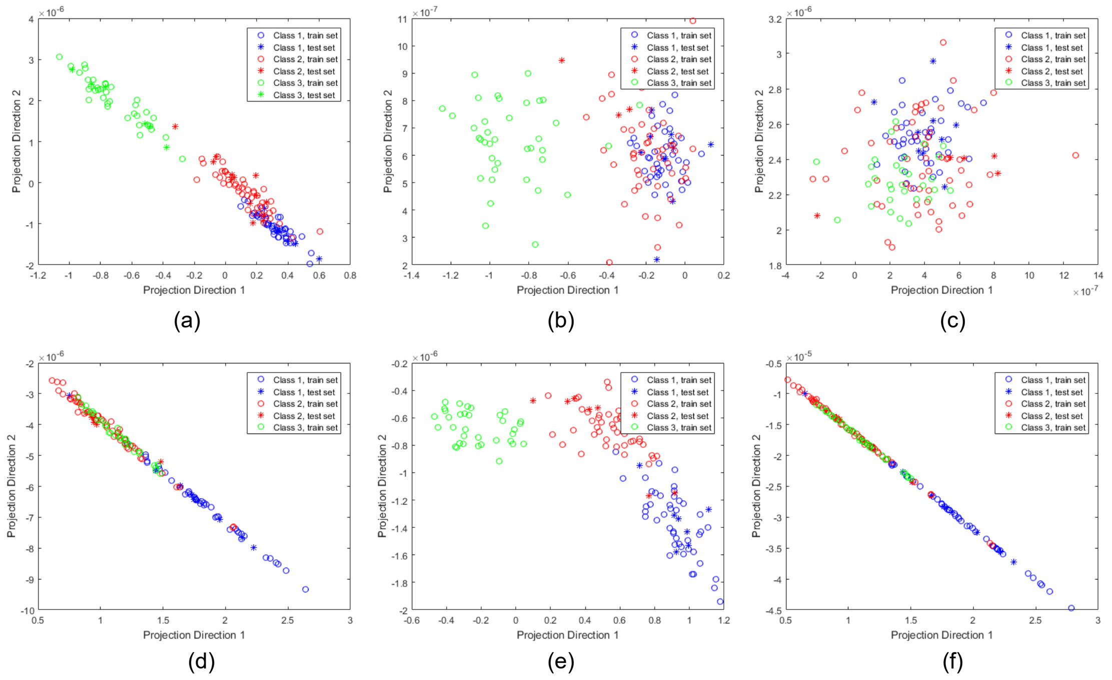


Figure 3.3: The top two leading dimensions of the metric subspace for wine dataset: (a) k-BH, (b) k-BSH, (c) k-HPEN, (d) k-EPEN, (e) k-EPHN, (f) k-NS.

$p < d$, or reduced to a subset of $1 \times p$ where $p < d$ using correlated feature selection where features are correlated in descending order in respect to their correlation with labels (see line 11 in Algorithm 2).

The instances are then sampled from within hyper-spheres using stratified sampling similar to the previous hierarchical method (see line 12 in Algorithm 2). Feature extraction methods described in section 2.1.2, are then applied to the resulting features and instances at that iteration. The resulting weights from the feature extraction method are used to transform the whole data on to the new subspace at the end of the iteration. The same weights are used to transform the test set later as well due to the fact that feature extraction creates a new feature space (see line 14 and 15 in Algorithm 2).

Note that the diagonal of the coefficient matrix, denoted by W, in LDA method, is slightly strengthened to avoid collapsing into subspace due to low ranks. However, it does not affect the projection directions. Similar to the hierarchical large margin metric learning approach, the number of hyper-spheres, denoted by n_s, decreases with respect to iteration index while the radius of hyper-spheres, denoted by r, increases. This is because, as we

```
Procedure: Hierarchical Subspace Learning($\boldsymbol{X}$)
Input: $\boldsymbol{X}$: dataset
Initialize $r$, $n_s$, and $p_\tau$
for $\tau$ from 1 to $T$ do
    $r$ := increasing function of $\tau$
    $n_s$ := decreasing function of $\tau$
    $p_\tau$ := decreasing function of $\tau$
    $n_f$ := decreasing function of $\tau$
    for $s$ from 1 to $n_s$ do
        $\boldsymbol{c}_s \sim$ range($\boldsymbol{X}$)
        Feature selection : Correlation based feature selection with portion size
          $n_f$ within the $s$-th hypersphere
        Instance Sampling: Draw a stratified sample with sampling portion $p_\tau$
          within the $s$-th hypersphere
    Apply desired subspace learning method
    Store the weight matrix
    Project $\boldsymbol{X}$ onto $\mathbb{C}$ol($\boldsymbol{W}$): $\boldsymbol{X} \leftarrow \boldsymbol{XW}$
```

Algorithm 2: Hierarchical Subspace Learning

proceed through the algorithm, we want to explore more of the data and feature space while avoiding the overlap area among sampling spaces. In cases of feature selection application, the portions of features, denoted by n_f, is altered with respect to iteration index as well since new features are being created by feature extraction at every iteration.

The initial radius, number of hyper-spheres, instance sampling portion, feature selection subset portion, and feature extraction subset portion are $r = 0.1\sigma$, $n_s = \lfloor 0.01 \times n \rfloor$ (clipped to $10 \le n_s \le 20$), and $p_\tau = 1$, $n_f = 90\%$, and $FE = 90\%$. At every iteration, the functions are updated as $r = r + \Delta r$, $n_s = \max(n_s - \lceil 0.2 \times n_s \rceil, 1)$, $p_\tau = \max(p_\tau - 0.05, 0.2)$, where $\Delta r = 0.3\sigma$ and σ is the average standard deviation along features, $n_s = \max(n_f - \lceil 0.95 \times n_f \rceil, 50)$, and $FE = \max(FE - \lceil 0.95 \times FE \rceil, 50)$.

Table 3.2: Dataset specifications

Datasets	Size	No Classes	Type	Task
Iris	150×4	3	Digit	Pattern Recognition
Breast Cancer	569×32	2	Digitized Image	Diognostic
Isolet	7797×617	26	Digitized Voice	Voice Recognition
MNIST	3000×784	10	Image, Pixel	Digit recognition
ORL	400×1178	10	Image, Pixel	Face Recognition

3.9 Experimental Results and Analysis

3.9.1 Dataset and Setup

Five publicly available datasets of Fisher Iris [31, 38], MNIST [70], ORL faces [4] (all described in section 3.7), Breast Cancer Wisconsin [78, 99] with 569 instances plus 32 features and Isolet [29, 35] with 617 attributes were tested. Datasets cover a range of Diagnostic, voice, pattern, digit and face recognition with different range of features. The specification of datasets are provided in table 3.2. Datasets are split to $70\% - 15\% - 15\%$ training, validation, and test subsets. The ORL faces were further projected to their 38 leading eigenfaces.

3.9.2 Comparing Manifold Learning Methods in Hierarchical and Non-Hierarchical Approaches

As was described before, there are three cases of no feature selection, random feature selection and correlation based feature selection. For each cases, the algorithm performs the previously described manifold learning methods. To see and compare the performance of the methods, different classifiers were applied as well. The algorithm is also compared to the raw original data going through classifiers as well as commonly applying feature extraction method once. Table 3.3 illustrates the result for comparison.

As table 3.3 presents, the hierarchical approach shows higher accuracy results in overall. Iris and Breast Cancer Wisconsin show similar accuracies in different methods due to the small feature space and the simplicity of datasets. However, the larger the number of features get, the gap between the hierarchical and other methods is highlighted more. It is more obvious in cases such as Breast Cancer Wisconsin with LDA on $K-$NN classifier and

Table 3.3: Comparing accuracies of the proposed manifold learning methods in raw, non-hierarchical and hierarchical subspace learning for classification.

Datasets		Feature Extraction	Classifiers											
			No Feature Selection				Random Feature Selection				Feature Selection			
			LDA	KNN	RF	QDA	LDA	KNN	RF	QDA	LDA	KNN	RF	QDA
Iris	Raw		81.80	72.70	81.80	76.30	81.80	72.70	81.80	76.30	81.80	72.70	81.80	76.30
	Original	LDA	81.81	72.70	81.80	77.70	81.81	72.70	81.80	77.70	81.81	72.70	81.80	77.70
		PCA	72.27	71.00	81.81	72.72	77.27	72.71	81.81	77.80	77.27	73.63	82.30	75.20
		GDA	81.80	72.00	80.76	77.65	81.80	80.80	86.30	86.30	81.80	83.50	91.90	86.30
		RICA	81.81	69.09	65.45	61.81	81.81	69.09	65.45	61.81	81.81	69.09	65.45	61.81
	Hierarchical	LDA	86.36	77.27	86.36	77.00	81.81	77.27	86.36	86.36	81.81	77.27	86.36	86.36
		PCA	81.81	72.45	82.34	77.00	81.81	72.45	82.34	77.00	81.81	77.60	82.34	77.27
		GDA	81.80	72.00	80.76	77.65	81.80	80.80	86.30	86.30	81.80	83.50	91.90	86.30
		RICA	81.81	72.72	65.45	61.81	81.81	72.72	65.45	61.81	81.81	72.72	65.45	61.81
Breast Cancer	Raw		97.14	97.14	98.09	99.04	97.14	97.14	98.09	99.04	97.14	97.14	98.09	99.04
	Original	LDA	96.19	96.19	96.19	96.19	94.28	93.33	94.28	98.95	96.19	96.19	94.28	94.28
		PCA	95.23	97.14	94.28	95.23	96.19	93.33	92.38	93.33	95.23	96.19	96.19	98.09
		GDA	96.19	96.19	96.19	97.14	97.14	96.19	96.19	100	93.33	94.28	92.38	94.28
		RICA	96.19	95.23	94.28	95.23	98.09	98.09	99.04	99.04	94.28	95.23	94.28	96.19
	Hierarchical	LDA	96.19	98.09	97.14	97.14	94.28	95.23	96.19	93.33	96.19	96.19	97.14	97.14
		PCA	95.23	98.09	96.19	95.28	95.23	96.19	93.33	94.28	98.09	97.14	99.04	96.19
		GDA	96.19	96.19	96.19	97.14	99.04	98.09	97.14	98.09	93.33	94.28	92.38	94.28
		RICA	96.19	96.19	96.19	95.23	98.09	98.09	99.04	99.04	94.28	96.19	96.19	96.19
Isolet	Raw		99.87	92.05	93.58	95.71	99.87	92.05	93.58	95.71	99.87	92.05	93.58	95.71
	Original	LDA	84.35	82.30	82.05	81.53	81.7	81.53	78.97	79.23	83.58	82.82	78.46	81.02
		PCA	80.51	71.53	75.20	81.02	81.02	74.10	81.02	81.02	81.53	74.00	79.74	80.00
		GDA	79.23	77.14	77.14	78.40	76.41	73.84	71.79	73.58	76.12	72.84	71.02	73.30
		RICA	77.94	71.28	73.07	80.00	71.53	73.07	77.69	81.53	73.07	74.35	72.85	72.30
	Hierarchical	LDA	91.28	88.46	88.46	81.53	80.00	74.35	76.92	75.89	89.48	88.17	89.48	88.17
		PCA	91.02	80.51	86.15	94.87	78.00	70.60	75.21	75.00	86.64	88.36	83.33	82.80
		GDA	85.41	90.00	90.00	92.10	63.07	67.17	57.69	63.39	88.71	88.46	88.46	84.10
		RICA	77.43	77.69	79.74	80.51	80.00	77.69	79.48	80.00	73.07	79.23	70.00	73.58
MNIST	Raw		75.00	80.00	88.00	61.00	75.00	80.00	88.00	61.00	75.00	80.00	88.00	61.00
	Original	LDA	74.00	72.00	62.00	69.00	74.00	72.00	63.00	69.00	73.00	73.00	65.00	67.00
		PCA	75.00	81.00	84.00	68.00	75.00	80.00	82.00	68.00	75.00	81.00	84.00	68.00
		GDA	70.00	79.90	65.00	58.90	70.00	79.90	65.00	58.90	70.00	79.90	65.00	58.90
		RICA	71.00	70.00	73.00	64.00	71.00	67.00	68.00	57.00	71.00	62.00	66.00	64.00
	Hierarchical	LDA	74.00	74.00	70.00	69.00	74.00	72.50	65.00	68.00	73.00	74.00	69.00	68.00
		PCA	80.00	87.00	82.00	60.00	79.00	80.00	84.00	69.00	80.00	84.00	86.00	70.00
		GDA	74.00	79.00	68.00	60.00	58.00	45.00	58.00	47.00	80.00	79.00	70.00	76.00
		RICA	71.00	74.00	75.00	75.00	68.00	60.00	62.00	72.00	71.00	74.00	764.00	64.00
ORL Faces	Raw		92.50	87.50	92.50	89.90	92.50	87.50	92.50	89.90	92.50	87.50	92.50	89.90
	Original	LDA	85.00	88.75	67.5	82.20	85.00	88.75	70.00	75.00	85.00	88.7	71.2	80.2
		PCA	92.50	83.75	83.75	87.5	92.50	83.40	83.75	86.00	92.50	84.00	81.25	88.67
		GDA	74.00	72.13	63.00	52.22	73.50	66.32	63.00	52.22	73.70	68.23	63.00	52.22
		RICA	73.75	75.00	83.75	81.25	73.75	75.00	82.50	81.25	73.00	75.00	85.00	81.25
	Hierarchical	LDA	85.60	88.75	70.86	84.00	83.50	85.00	65.30	69.00	88.00	89.7	81.45	82.43
		PCA	92.50	84.50	84.50	90.12	58.50	76.45	73.50	67.90	82.50	85.30	85.42	89.00
		GDA	78.54	72.13	66.60	60.00	67.50	66.25	56.73	57.00	80.00	76.25	67.50	60.50
		RICA	73.75	75.00	83.75	81.25	70.86	70.86	80.00	79.40	76.07	79.92	85.00	83.12

MNIST with PCA on LDA classifier where the hierarchical accuracy exceeds the raw data classification accuracy. This presents the curse of lower dimensionality and how having higher dimensions does not guarantee better exploration of data and higher accuracy. This is more common on datasets with a larger feature space than the data space.

Random selection of features does not provide higher accuracies consistently due to its random nature in which it does not consider any correlation among features or between features and label. Hence, it is expected to perform better in cases when features with more correlation with label were chosen and perform worse otherwise. However, the iteration nature of the algorithm makes up the random selection of features to a considerable degree where the results are comparable with raw data classification.

As can be seen in the table, feature extraction alone, without any feature selection, outperforms other cases due to keeping more of the feature space by eliminations feature selection while staying true to the goal of dimensionality reduction in a slower pace by feature extraction. As expected, this method is more time consuming due to the projection direction matrix which is highly dependable on the feature space.

Table 3.3 shows comparable results among LDA, PCA and GDA, outperforming other approaches with LDA leading. This is due to both methods being linear manifold learning methods and are expected to be more compatible with linear datasets. However, LDA is a supervised method while PCA is unsupervised and hence it is expected for LDA to have higher accuracy results. Moreover, GDA as an extension of LDA should provide comparable results.

Moreover, among classification methods used, Random forest performs better due to minimizing the probability of error by creating an ensemble of trees and taking a majority vote among all. LDA classifier, when paired with LDA feature extraction, is presenting high accuracies as well due to linear dataset.

Even though the run-time was an advantage in hierarchical large margin metric learning with coplec data space, the time consumption increases to 297.12, 154.56 and 145.31 seconds for Isolet, MNIST, and ORL faces respectively in hierarchical approach for subspace learning methods. This is due to the large feature space of the datasets. As the number of features increases, due to hierarchically exploring the feature space, it will take longer. For smaller datasets such as Iris and Breast Cancer, the time consumption is comparable due to the size and simplicity of datasets. The run-time is specially highlighted among hierarchical approach and raw dataset classification, again due to the feature space complexity of input matrix. Even though the sub-manifold learning methods are decreasing the dimensionality of data in overall, the hierarchical nature of the method is time consuming specially during early iterations.

3.10 Conclusion

In this chapter, inspired by triplet mining methods for Siamese network, we proposed several triplet mining methods for large margin metric learning. The methods were based on making the data space smaller by sampling the most important instances. This method increased the accuracy while speeding up the SDP optimization. In addition to the proposed methods, a new hierarchical approach was proposed to reduce the SDP timing further in combination with triplet mining methods. With considerable results of hierarchical approach on data space, it was further improved to perform on feature space as well. The methods were tested on five publicly available datasets combined. A possible future direction would be implementing the hierarchical approach for non-linear kernel sub-manifold learning methods.

Chapter 4

FPGA-Based Architectures for Random Forest Acceleration

Random forest is a very popular algorithm for both classification and regression due to it's flexibility among real-life datasets to be used on different variety of datasets regardless of their dimensions. It is an intriguing topic for the cross-section of machine learning on hardware. Upon approaching with different solutions to the problem of complex datasets with huge dimensions in chapter 3, the familiarity and popularity of random forest classifier, in addition to the previous works on the topic in literature, the random forest algorithm is chosen for hardware implementation.

In this Chapter, we explore the efficient hardware implementation of random forest classifiers. In section 2.4.3, we showed that there is a trade-off between accuracy, memory, and run-time with a greater focus on the latter two. In this Chapter, by bringing accuracy into consideration alongside run-time, we propose a method to achieve a comparable accuracy to the CPU models of random forests while reducing the run-time and optimizing memory use. We describe our approach for training on an Intel FPGA - Arria 10 device that also results in higher accuracy than previous models. The Arria FPGA is programmed using Quartus Prime Pro.

The chapter is structured as follows. It starts with explaining the logical framework of the implemented architect followed by the hardware design in detail. Finally, an evaluation of the design is performed to assess the architecture.

4.1 Algorithm Overview

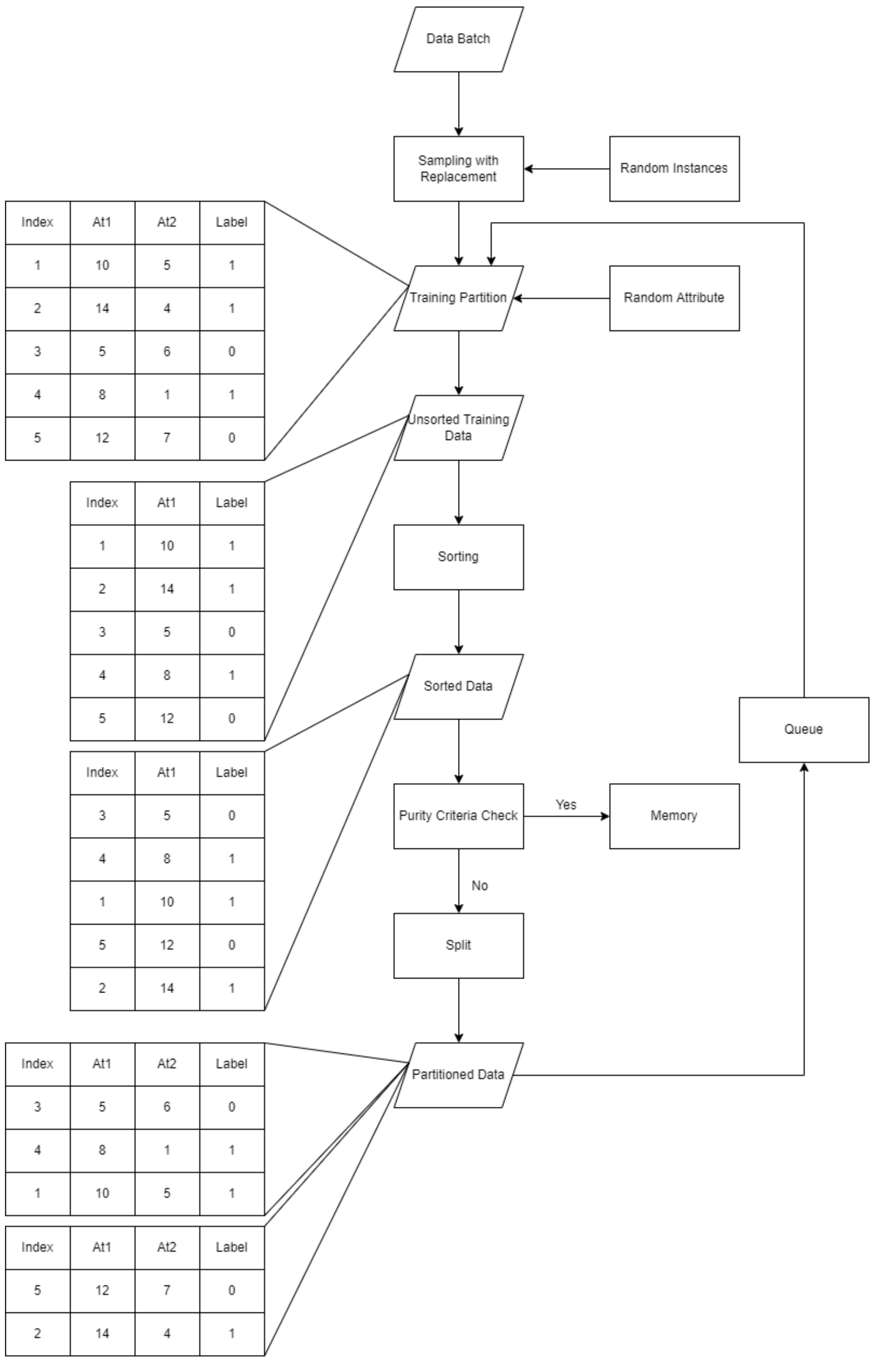
The dataset is originally stored in an external memory and will be transferred to the FPGA embedded memory in batches when the training process starts. The instances are randomly selected for creating a decision tree using sampling with replacement method. The training process is presented in Fig. 4.1. The training process starts with training partition, sampled with replacement in instances and attribute, entering the sorting module. In this example, instances 1 to 5 and attribute 1 are randomly selected from the data partition in the memory. The data will be sorted with respect to the selected attribute in the sorting module and the sorted data will then be checked for purity criteria.

If the purity check is met, the data will be stored in memory as a branch; otherwise, it will enter the split module where the data will be searched for the optimal weak learner and will be split to two branches. Here, the optimal weak learner is chosen to be gini impurity. The branches will be stored in a queue and the split information will be stored in memory as well. The branches in the queue will be checked for purity criteria again and follow the cycle. In the example of Fig. 4.1, the first branch includes instances $3, 4, 1$, and the second branch includes instance $5, 2$ with respect to attribute 1 and the split point of 11. A flag based on whether the queue branches are empty, indicates the termination of cycles and the tree information will be saved on FPGA.

4.2 Top Level Architecture

Fig. 4.2 below represent the hardware presentation of Fig. 4.1. The top level architecture includes Memory, Forest Control System, PEs and Tree Storage. The data will be transferred from external memory to FPGA internal memory. From memory, the data will travel to a FIFO controlled by the Forest Control Module, to enter to a proper processing element(PE). The PE includes the sort module, the purity criteria check, split module and the queue memory to store branches. The data will iterate through PE and the complete tree will exit the PE entering the FPGA Tree Memory module.

The Memory module stores the batch of data under study. Each PE includes the Sorting module, a PE Control module, purity check and Split modules which will be explained in details further in this chapter. The workflow of the top level will be managed by the Forest Control System and the final results are collected in Tree Memory.



Index	At1	At2	Label
1	10	5	1
2	14	4	1
3	5	6	0
4	8	1	1
5	12	7	0

Index	At1	Label
1	10	1
2	14	1
3	5	0
4	8	1
5	12	0

Index	At1	Label
3	5	0
4	8	1
1	10	1
5	12	0
2	14	1

Index	At1	At2	Label
3	5	6	0
4	8	1	1
1	10	5	1

Index	At1	At2	Label
5	12	7	0
2	14	4	1

Figure 4.1: Training Process Top-Level Architecture

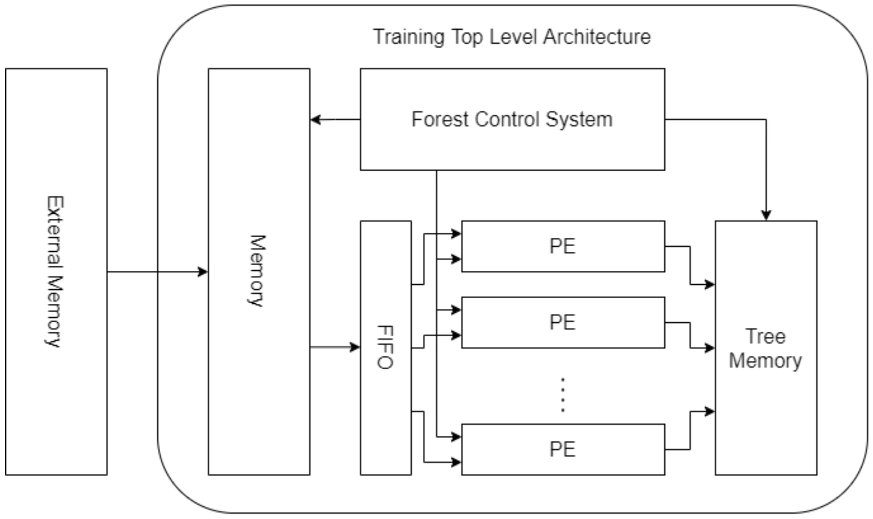


Figure 4.2: Training Process Top-Level Architecture

4.3 Memory

To map the external memory to the internal memory, Quartus internal IPs are used with soft memory controller to ensure maximum bandwidth for control and data communication. The latency varies with the IP setting such as frequency, data width, the type and etc. However, Quartus uses Ping Pong FIFO buffers to ensure its fullness. Hence, the latency of data transfer is hidden under the process latency.

The DDR Controller is responsible for read and write requests from external memory and for moving the appropriate batch to internal memory. In this way, while the trees are being processed, we can replace the contents of the Internal memory with new batches of data. The internal memory communicates with the DDR Controller through read requests and a counter to enable the controller to load the appropriate batch of data and navigate it through the Write Address Generator to appropriate blocks. The write address consists of two parts. The MSB part is the block number based on the features and the LSB part is the memory address in which the value is stored in. The read address is created from two parts. The MSB part is the selected attribute and the LSB part is the random index. Only the LSB part is used for loading the appropriate label.

The final step is to load the read instances and the labels in two temporary storage areas for each tree. The instances are stored in a BRAM FIFO while the labels are stored in an appropriately sized register as one element. Bear in mind that a combination of parallel and series tree processing is implemented. To have parallel trees, we need a FIFO for storage at the beginning of the training of every tree. In order to have parallel tree computation without data loss, FIFOs are implemented at the beginning of each tree as well in order to have data available for processing at all times, so when the first tree is being processed, the second tree can be loaded in the queue. The top level architecture of the training process is presented in Fig. 4.2.

4.4 Processing Element

At the beginning of the training process, the unsorted tree data enters the Processing element. The processing element (PE) consist of four major sub-modules, Sort, Split Zero, Right Split and Left Split modules. Fig. 4.3 represents the top-level architecture of one PE.

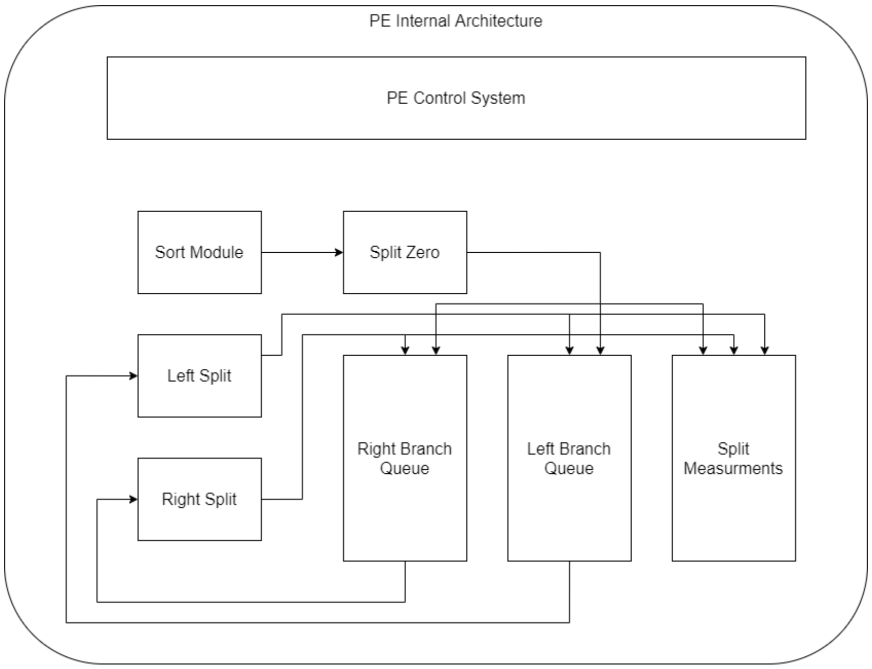


Figure 4.3: PE internal architecture

The process starts with the unsorted data entering the sorting module where the tree instances and labels are sorted in ascending order with respect to the attribute value. The sorted label array is stored in one register in a bit-serial manner for internal PE process. Working with sorted labels require less hardware utilization for processing. A pool of potential splits are created by taking the median of two non-identical adjacent data instances. The possible split indices are stored to be later used by PE control module. It is recommended for the subset of training dataset for a single tree to be multiplication of 16 for optimized memory utilization.

Sorted labels originally enter the Split Zero module, in which we divide a root into two branches of right and left. The Split Zero module is responsible for the first split of a tree for which no parent node information is provided. Both the right and left partitions are fixed sized zero arrays shifted with the actual branches and hence emulate zero-padding for a fixed size of the original partition. The new branches from split zero are stored in first element of right-branch queue and left-branch queue while split point index will be stored in Split Indices in memory.

In the following iterations, the input to the Right Split and Left Split are provided by the Right and Left queues. The Right(Left) Split module is responsible for finding the best split for all the right(left) branches only. At the beginning of every split module, the data is checked for purity and leaf node criteria. In case of meeting either criteria, a leaf node flag is raised preventing further process. Every split module creates two new right and left branches. The parent node split information also enters the right/left-split modules to indicate the size of the actual instances contained within that partition. The Right Split module stores its new branches in $2i$ addresses of both queues, and the Left Split module stores its new branches in $2i+1$ addresses of the queues. Hence, during the PE process, at every iteration, two branches are read and four new ones are created.

The PE contains an internal PE Control System, responsible for determining if the process is at split zero, synchronizing the internal PE processes, tracking the iterations, tracking read/write addresses for queues as well as split modules entries and departures. It is also responsible for split indices to be loaded in the split modules and to be stored from the split modules.

The quality of a split point is assessed with a gini impurity assessment. The gini impurity formula for two classes is represented below with two main part of gini left and gini right:

$$\mathcal{S}(.) = \frac{n(L,i) \times n(L,j)}{n(L,i) + n(L,j)} + \frac{n(R,i) \times n(R,j)}{n(R,i) + n(R,j)}, \tag{4.1}$$

where L/R stands for left/right and i/j stands for class one and two respectively. $n(.,.)$ is the number of entries of that particular branch belonging to a specific class. The formula, however, can be reorganized and expanded for categorical data with more than two classes as:

$$\mathcal{S}(.) = \frac{1}{n_L}[n^2_{(L,1)} + n^2_{(L,2)} + ... + n^2_{(L,m)}] + \frac{1}{n_R}[n^2_{(R,1)} + n^2_{(R,2)} + ... + n^2_{(R,m)}] \qquad (4.2)$$

As mentioned before, the assessment of the best split is done in a parallel manner since the labels enter the split module all at once. For every possible split point, gini left, gini right, and gini total are calculated and possible branches are stored in a label histogram register-based memory. The histogram contains all the possible future split points. The depth of the histogram is 2 × (tree instances − 1) and the width should be tree instances × label width. After assessment, the best threshold index and split point are stored in a split point information module, and the new branches are chosen from the label histogram in an index-based manner and are sent to appropriate Right branch RAM and Left branch RAM.

The tree information that is saved to be traveresed later by test dataset, is minimal. Every tree has two static fixed size array dedicated to it where the information is stored. One is for the split information and one is for the label. Both arrays are sorted in ascending order. The label array contains one extra element because whit k split points, $k+1$ possible labels are expected. The only information that is stored is the split point happening at every node. The label array takes majority vote among left and right branches and stores them accordingly. For instance, if we have a tree with 16 instances and the depth of 4, we will have 15 split nodes and hence we will have an array of 15 elements for the split information sorted in ascending order and an array of 16 elements for the labels. Eventually, outside of every PE, we will have two matrices of $n \times m$ dimension where n is the number of trees and m is either an array of 16 or 15.

The parallel processing is achieved by not only having parallel tree computations but also by having parallel processes for left and right branches to decrease the tree processing time approximately in half number of cycles than what originally anticipated.

4.5 Serialization and Parallelization Within PE

Every entry to the split module has a package of information to inform the module about the specification of that data partition that is to be processed. The sort or split module should know what data it should collect from the memory arrays and other components

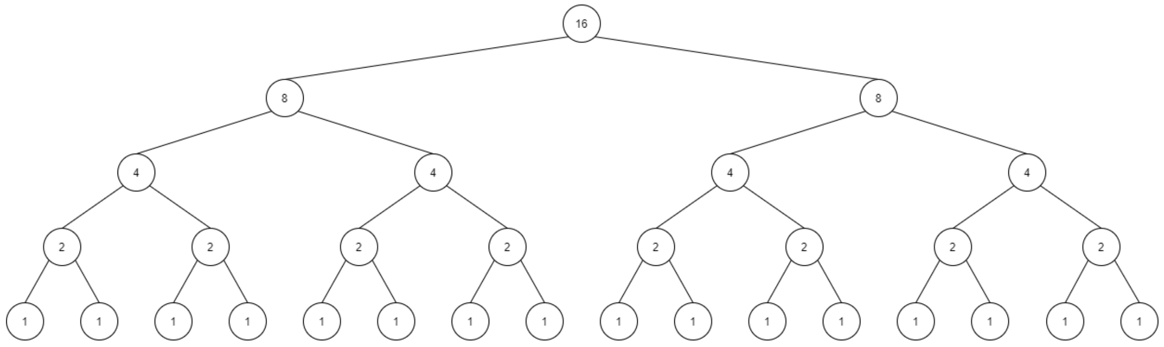


Figure 4.4: Tree with 16 instances and depth of 4

in order to perform the split. Each entry hence, carries information such as the length indicating the size or the number of actual instances contained within each partition. It also carries information about the parent split module such as it being a left branch or right originally. It also includes addresses to the appropriate data.

The general idea for this thesis is to add parallel processing within each PE. One way in which this is accomplished is by having two separate internal split process within PEs for right and left branches. In this way, the process for one tree is approximately half of what is originally anticipated.

A tree with n instances can have a maximum depth of $\lceil \log_2^n \rceil$ and a maximum of $n-1$ split points with one being the split zero. Hence, in total, we have $n-2$ splits to be processed by left and right branches of split modules. Accordingly, we have $(n-2)/2$ iterations in total since the right split module and left split module work in parallel. For example, for 16 instances with depth of 4, we have a tree represented in Fig. 4.4 with maximum of 15 split points. This example results in 7 iterations for left and right branches to process. Every iteration includes many clock cycles for internal processes but the unit of measure in this section is the number of iterations.

With a correct and accurately designed control system, in addition to using pipelines, we can replace the first half of the Right and Left branch memory after the half-point of the iterations. This is because whenever an address is read from the branch memory and produces the resulting split point information, that information is no longer useful and can be replaced with new information and values queued to be processed later. The same strategy can be applied to the split information unit, albeit with slight changes since the resulting tree information should be stored for one tree as a unit. The split information module needs to be pipelined and buffered in order to avoid re-writing the split information of one tree half-way through iterating the branches. Also, the processing element unit needs to process the remaining half of the right and left branch memories. In the example of 16 instances with 7 parallel iterations, we can start importing a new tree in the PE, sort and start the split zero for the next tree after passing the 4^{th} iteration. Upon starting the new tree, the remaining half of the current tree is processed. This is indicated by a flag. during this time, the new processes are directed to the split zero module. The speed of the process is halved due to one module processing both right and left branch in RAM, but in this way, resources are used more efficiently.

In our example, we can save three iterations out of eight. The pipeline process of a tree within a PE is shown in Fig. 4.5.The process in in each PE is a combination of serial, data passing through split zero and move forward until the end of the tree serially) and parallel (having right and left branches processing in parallel fashion). Note that the bigger the desired tree, the greater the number of instances, and the higher the number of iterations. The greater the number of instances, the the fewer the number of iterations that can be optimized. The complication of this process is due to re-using the split zero module besides its original purpose. A control system needs to be implemented for this purpose so the order of instances entering and result collection is preserved. The bigger the tree size, the bigger the FIFO, and then, the longer the training time.

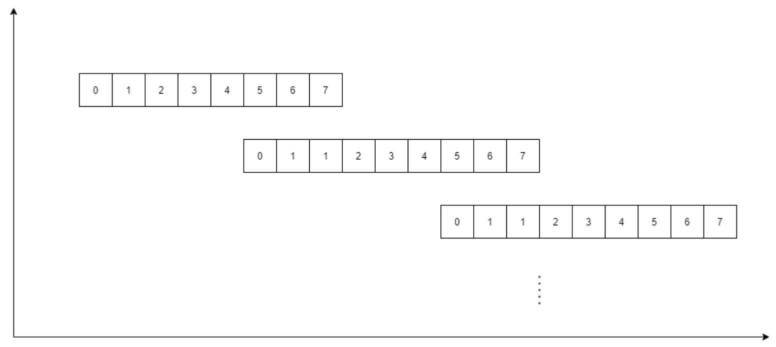


Figure 4.5: Example tree serialization of PE process for 16 instances

4.6 Testing

The resulting trees from the training phase are stored on FPGA memory, making it less complicated for testing. It also makes the testing dependant on the training phase being concluded. The latency and the throughput of testing depends on the number of parallel PEs that are processed, due to accessing the trees in memory, the number of trees being traversed, and the number of instances in the testing partition of the dataset. There are also accuracy calculations, but the bottleneck of the testing latency is the former two reasons related to memory read access.

4.7 Experimental Results and Analysis

4.7.1 Datasets and Setup

To validate our approach, we used six publicly available datasets to cover a variety of applications with a range of features and instances. A summary of the datasets is given in Table 4.1; datasets are available in [32]. The first dataset is the Fisher Iris data first introduced by Ronald Fisher in [37] with 150 records under four features. The second dataset is Sonar, which is a multivariate dataset with binary classes by Terry Sejnowski. It was first used in [51] with 60 attributes and 208 instances in total. This dataset uses floating-point numbers, and is useful to demonstrate the effectiveness of the proposed hardware approach. The third dataset is Breast Cancer Coimbra with 116 instances, 10 attributes. It is a binary class dataset indicating the presence or absence of breast cancer [27, 89]. The next dataset is Heart Failure, first introduced in [2], is a multivariate dataset with binary classification including 299 patients with heart failure. This dataset includes 13 features of both continuous and categorical kinds. The sixth dataset is Contraceptive Method Choice data [75] with 1473 instances, nine features and three classes. The final dataset is Cardiography data with 2126 instances, 23 features and three classes [5].

Table 4.1: Datasets specifications

Datasets	Size	No Classes	Type	Task
Fisher Iris	150×4	3	Real	Pattern Recognition
Sonar	208×60	2	Real	Pattern Recognition
Breast Cancer Coimbra	116×10	2	Integer	Diognastic
Heart Failure	299×13	2	Integer	Diognastic
Contraceptive	1473×9	3	Integer	Pattern Recognition
Cardiography	2126×23	3	Real	Diognastic

All datasets are divided to approximately 70% training and 30% testing subsets. The best result is achieved by having the test subset a multiple of 16 since the RAMs are designed in this manner. The architecture is implemented on an Intel Arria 10 family

($10AX115N3F45I2SG$) FPGA, using Quartus Prime Pro 20.0 and simulated using Modelsim. The FPGA is clocked at 100 MHz.

4.8 Data Pre-Processing

Each dataset is pre-processed in Python, and is divided into training and testing subsets. For best hardware utilization, in the testing subsets, the number of instances is a multiple of 16. In smaller datasets such as Iris [32], we can store the dataset completely on FPGA memory as only one partition. For such datasets, the algorithm can process as a conventional random forest training algorithm with access to the whole training dataset. However, the FPGA on-chip memory space is an important constraint when it comes to large datasets. Due to memory issues, we utilize the batch-learning model similar to [19] by transferring datasets from external memory to FPGA batch by batch.

In addition to the usual bootstrap bagging algorithm, an $N \times d$ array of uniformly distributed random numbers is generated from 1 to d, where N is the number of trees and d is the number of features in order to create a tree based on random attribute selection.

4.8.1 Comparison of Training Accuracy and Run-Time

The training accuracy of the proposed method is compared with a CPU implementation of RF on Python, as well as the results presented in [19, 20]. It is known that the accuracy is, to some extent, dependent on the random train-test division of dataset. Hence, to avoid the effect of the random selection of points, we reported an a $5\times$ cross validation for both the hardware and the software implementations. Table 4.2 summarises the comparison results between our proposed method on hardware and the established software method. The software implementation is on Intel Core$-i7$ CPU @3.00GHz and the propose method results are based on 250 trees with depth of 4.

As can be seen in table 3.1, the accuracies in proposed hardware implementation cases are lower but compatible with software version of a random forest. However, both the proposed method and software accuracies, are far from the average accuracy reported for the Sonar dataset in [19] of 48% with 200 trees and up to 300 batches. This is due to the fact that, the tree results in [19, 20] are not stored on the FPGA and are transferred to the external memory while in this proposed method, the tree results are stored on FPGA

Table 4.2: Comparing accuracies and run time of the proposed hardware implementation and Scikit-learn random forest

Dataset		Proposed Method	Scikit-learn
Fisher Iris	Accuracy (%)	86	96.25
	Time (msec)	29.64	1710.24
Sonar	Accuracy (%)	53	71.25
	Time (msec)	37.23	2099.39
Breast Cancer Coimbra	Accuracy (%)	59	76.25
	Time (msec)	37.7	1732.03
Heart Failure	Accuracy (%)	75	86.25
	Time (msec)	38.56	1846.68
Contraceptive	Accuracy (%)	49	51.25
	Time (msec)	37.91	2192.42
Cardiography	Accuracy (%)	74	98.75
	Time (msec)	38.08	3336.08

resulting in more accurate classification. However, the number of trees that can be stored on FPGA is limited and depends on both the tree instances and depth of each tree. The accuracy of iris dataset is closest to the software implementation due to the simplicity and the size of dataset while more complex datasets such as Sonar need more number of trees for more accurate results.

Another reason for the difference in accuracies between the proposed implementation and the software implementation, is due to the transformation of double value format to fixed-point format. The fixed-point format affects finding the best split point since it is found by taking the median of two consecutive numbers. This gap is more obvious in the case of sonar since this dataset has the precision of four decimal points in real values so the effect of data format is more clear in this case. Also standard deviation of a feature, though small, has a strong affect on transferring the feature's trace on label on the software implementation in contradictory to hardware. If the standard deviation is too small, the effect of that feature will not fully be represented in tree forest which can cause the small difference in accuracies.

Cheng *et al.* provides the run-time of 43.7 millisecond for only the sonar dataset in [19]. As can be seen in table 4.2, the proposed architecture is proven to be faster due to parallel processing of right and left branches. The variation between the run time of the proposed method on different datasets is due to the number of instances, the width of the data, number of attributes, etc. The run time of the proposed method is on average 50× faster than the software implementation of random forest on python.

4.8.2 Comparison of Hardware Utilization and Latency

The hardware resource utilization of our proposed method, depends on the dataset under study to some extent. The width, and depth of most registers, memory blocks, the number of attributes plus entries are user specified. In addition to these, the depth of the tree is also an important factor for how many trees and PEs can be fitted in the FPGA internal memory. Other important factors can be name such as the length of the entries and how many labels the dataset inquires. Table 4.3 summarizes a comparison of hardware utilization for one PE among the proposed method and [20].

Table 4.3: Single PE hardware utilization

Datasets	No Registers	Block Mem Bits	No DSP Blocks
Proposed Method	3420	9152k	497
Task Parallelism	4596	4102k	9

As can be seen, the number of memory bits is increased drastically due to having a memory centric approach in which most mid-stages result are stored on FPGA embedded memory.

The total time of the training process depends on many factors such as the processing time of the task defined in previous section and the time it takes to move from external memory to internal embedded memory. For loading the data from the off-chip DDR4 memory to on-chip embedded memory, we are using Intel Quartus Prime IPs for control and communication. The latency varies with the setting of IP such as data width, the frequency used. The data accessing patterns such as how many agents attempt to read or write data, the frequency, etc., also affect the performance. Intel built in IP is optimized to use a type of ping-pong buffer FIFO to consistently buffer data from DDR until it is full. In this case, the latency of data reading stage overlaps with the data processing stage, making the latency not influential.

Hence, the main part of latency will be the training process itself which is heavily pipelined. From the training process also, the first batch will be the most influential one as the starting point. The working frequency after resolving the timing violations are 120 MHz. With dissolving the first batch of training through the whole training process, having

2 PEs, the resulting throughput will be $(2 \times 16 \times 120)/30$ equal to 128 bps. One framework of data is 16 bits and hence it takes about 30 cycles to process 32 frameworks using two PEs.

4.9 Conclusion and Future Direction

Software implementations of RFs can be very slow on large datasets. To accelerate the process, different hardware implementations are proposed in the literature on batch learning, sub-sampling and online training for FPGA or GP-GPU. However, FPGA implementation,seems to be more promising due to their parallel computation power and its concurrent memory access.

In this thesis, inspired by prior work by Cheng et al. [20], we proposed an architecture to fasten the training part of the classification process by an average of 50 times compared to software implementation. The proposed method also shows better run-time and accuracy result compared to the reference papers in literature.

The proposed method incorporates separate processing for creating right and left branches within every PE, while having parallel PE processes compared to previous works in literature [20, 21]. By sacrificing memory utilization in addition to storing tree information regarding on FPGA memory for faster investigation on test data, we achieve more accurate classification results. The method also incorporates a control system and final state machine to fully utilize its resources.

A possible future direction is to optimize the split zero module by improving the control unit for better hardware utilization. We can also implement this logic for online training of RFs for future work. The testing phase is also an unexplored area for this thesis, which can be a possible future contribution.

Chapter 5

Conclusions and Future Directions

Large margin metric learning for nearest-neighbor classification is a metric learning method that finds a metric to better discriminate classes by using SDP optimization and the anchor-positive-negative triplet concept. The SDP optimization is very slow and computationally expensive, due to the use of interior point optimization method. In this thesis, inspired by triplet mining techniques for Siamese network training, by considering the most important triplets rather than all, we proposed several triplet mining methods for large margin metric learning for nearest-neighbor classification. The analysis of the proposed methods show a speed-up in the optimization and presents a more computationally efficient method. The proposed triplet mining techniques were k-BA, k-BH, k-BSH, k-HPEN, k-EPEN, k-EPHN, and k-NS. Moreover, we proposed a new hierarchical approach, in combination with the triplet mining methods, to further increase the pace of optimization and reduce the training time. Our experiments on three public available datasets verified the effectiveness of the proposed approaches.

However, the bottleneck of the dataset is not always the number of instances but also the number of features in a dataset. Dimensionality reduction techniques are used for map the dataset onto lower sub-manifolds in order to aid in visualization and explorability. It is commonly used as a pre-processing method in machine learning and statistics. Inspired by the hierarchical approach for large margin metric learning using stratified sampling, we introduced a hierarchically iterative approach as a pre-processing method. Analysis of the proposed method shows an increase in classification accuracy. A possible future direction can be testing non-linear manifold learning methods such as kernel PCA or kernel LDA using the hierarchical approach or to try the proposed hierarchical approach using stratified sampling on other subspace learning methods.

Software implementations of some machine learning algorithms such as random forest classifiers on CPU can be very slow, especially on huge datasets. To accelerate the process, different hardware implementations are proposed in literature on FPGA due to its parallel computation power and its concurrent memory access. In this thesis, we proposed an architecture to speed up the training part of the classification process by an average of 50 times compared to software implementation. The proposed method also shows better run-time and accuracy compared to the state of the art. The proposed method incorporates separate processing for creating right and left branches by sacrificing memory utilization and also stores some information regarding the trees in FPGA memory for more accurate classification results. The method also incorporates a control system and finite state machine to fully utilize its resources. A possible future direction is to optimize the split-zero module by improving the control unit for better hardware utilization. We can also implement this logic for online, rather than offline training. The testing phase is also an unexplored area for this thesis, which can be a possible future contribution.

References

[1] RA Pearson Ah Chung Tsoi. *Comparison of three classification techniques: CART, C4. 5 and multi-layer perceptrons.* In fAdvances in neural information processing systems, 1991.

[2] Tanvir Ahmad, Assia Munir, Sajjad Haider Bhatti, Muhammad Aftab, and Muhammad Ali Raza. Survival analysis of heart failure patients: A case study. *PloS one*, 12(7):e0181001, 2017.

[3] Babak Alipanahi, Michael Biggs, Ali Ghodsi, et al. Distance metric learning vs. fisher discriminant analysis. In *Proceedings of the 23rd national conference on Artificial intelligence*, volume 2, pages 598–603, 2008.

[4] AT&T Laboratories Cambridge. Orl face dataset, 2001.

[5] Diogo Ayres-de Campos, Joao Bernardes, Antonio Garrido, Joaquim Marques-de Sa, and Luis Pereira-Leite. Sisporto 2.0: a program for automated analysis of cardiotocograms. *Journal of Maternal-Fetal Medicine*, 9(5):311–318, 2000.

[6] Vick Barnett. *Elements of sampling theory.* English Universities Press, London, 1974.

[7] Roberto Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks*, 5(4):537–550, 1994.

[8] Gaston Baudat and Fatiha Anouar. Generalized discriminant analysis using a kernel approach. *Neural computation*, 12(10):2385–2404, 2000.

[9] Peter N. Belhumeur, João P Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):711–720, 1997.

[10] Anthony J Bell and Terrence J Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural computation*, 7(6):1129–1159, 1995.

[11] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[12] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[13] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[14] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[15] Christopher JC Burges. *Dimension reduction: A guided tour*. Now Publishers Inc, 2010.

[16] Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. Feature selection in machine learning: A new perspective. *Neurocomputing*, 300:70–79, 2018.

[17] Miguel A Carreira-Perpinán. A review of dimension reduction techniques. *Department of Computer Science. University of Sheffield. Tech. Rep. CS-96-09*, 9:1–69, 1997.

[18] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[19] Chuan Cheng. *Random forest training on reconfigurable hardware*. PhD thesis, Imperial College London, 2015.

[20] Chuan Cheng and Christos-Savvas Bouganis. Accelerating random forest training process using fpga. In *2013 23rd International Conference on Field programmable Logic and Applications*, pages 1–7. IEEE, 2013.

[21] Chuan Cheng and Christos-Savvas Bouganis. Memory optimisation for hardware induction of axis-parallel decision tree. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, pages 1–5. IEEE, 2014.

[22] Gerda Claeskens, Nils Lid Hjort, et al. Model selection and model averaging. *Cambridge Books*, 2008.

[23] Pierre Comon. Independent component analysis, a new concept? *Signal processing*, 36(3):287–314, 1994.

[24] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 2009.

[25] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[26] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision*, 7(2–3):81–227, 2012.

[27] Joana Crisostomo, Paulo Matafome, Daniela Santos-Silva, Ana L Gomes, Manuel Gomes, Miguel Patrício, Liliana Letra, Ana B Sarmento-Ribeiro, Lelita Santos, and Raquel Seiça. Hyperresistinemia and metabolic dysregulation: a risky crosstalk in obese breast cancer. *Endocrine*, 53(2):433–442, 2016.

[28] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[29] Thomas G Dietterich and Ghulum Bakiri. Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *The Mathematics of Generalization*, pages 395–407. CRC Press, 2018.

[30] Shengyong Ding, Liang Lin, Guangrun Wang, and Hongyang Chao. Deep feature learning with relative distance comparison for person re-identification. *Pattern Recognition*, 48(10):2993–3003, 2015.

[31] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[32] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[33] Richard O Duda, Peter E Hart, et al. *Pattern classification*. John Wiley & Sons, 2006.

[34] Daniel Engel, Lars Hüttenberger, and Bernd Hamann. A survey of dimension reduction methods for high-dimensional data analysis and visualization. In *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering-Proceedings of IRTG 1131 Workshop 2011*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[35] Mark Fanty and Ronald Cole. Spoken letter recognition. *Advances in neural information processing systems*, 3, 1990.

[36] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

[37] RA Fisher. The use of multiple measurements in taxonomic problems, annual eugenics, 7, part ii, 179-188 (1936); also in contributions to mathematical statistics, 1950.

[38] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[39] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer Series in Statistics New York, 2001.

[40] Keinosuke Fukunaga. *Introduction to statistical pattern recognition.* Elsevier, 2013.

[41] Ali Ghodsi. Dimensionality reduction a short tutorial. *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada*, 37(38):2006, 2006.

[42] Ali Ghodsi. Data visualization course, winter, 2017.

[43] Benyamin Ghojogh and Mark Crowley. Linear and quadratic discriminant analysis: Tutorial. *arXiv preprint arXiv:1906.02590*, 2019.

[44] Benyamin Ghojogh and Mark Crowley. The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial. *arXiv preprint arXiv:1905.12787*, 2019.

[45] Benyamin Ghojogh, Fakhri Karray, and Mark Crowley. Fisher and kernel fisher discriminant analysis: Tutorial. *arXiv preprint arXiv:1906.09436*, 2019.

[46] Benyamin Ghojogh, Maria N Samad, Sayema Asif Mashhadi, Tania Kapoor, Wahab Ali, Fakhri Karray, and Mark Crowley. Feature selection and feature extraction in pattern analysis: A literature review. *arXiv preprint arXiv:1905.02845*, 2019.

[47] Benyamin Ghojogh, Milad Sikaroudi, Sobhan Shafiei, Hamid R Tizhoosh, Fakhri Karray, and Mark Crowley. Fisher discriminant triplet and contrastive losses for training siamese networks. In *2020 international joint conference on neural networks (IJCNN)*, pages 1–7. IEEE, 2020.

[48] Amir Globerson and Sam Roweis. Metric learning by collapsing classes. *Advances in neural information processing systems*, 18:451–458, 2005.

[49] Amir Globerson and Sam T Roweis. Metric learning by collapsing classes. In *Advances in Neural Information Processing Systems*, pages 451–458, 2006.

[50] Jacob Goldberger, Geoffrey E Hinton, Sam T Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems*, pages 513–520, 2005.

[51] R Paul Gorman and Terrence J Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural networks*, 1(1):75–89, 1988.

[52] Jianping Gou, Lan Du, Yuhong Zhang, Taisong Xiong, et al. A new distance-weighted k-nearest neighbor classifier. *J. Inf. Comput. Sci*, 9(6):1429–1436, 2012.

[53] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[54] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1735–1742, 2006.

[55] Mark Andrew Hall. *Correlation-based feature selection for machine learning.* PhD thesis, University of Waikato Hamilton, 1999.

[56] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning. springer series in statistics. In *:*. Springer, 2001.

[57] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Science & Business Media, 2009.

[58] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[59] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[60] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[61] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

[62] Ian Jolliffe. *Principal component analysis.* Springer, 2011.

[63] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[64] Christian Jutten and Jeanny Herault. Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal processing*, 24(1):1–10, 1991.

[65] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. A survey of feature selection and feature extraction techniques in machine learning. In *2014 science and information conference*, pages 372–378. IEEE, 2014.

[66] Rafal Kulaga and Marek Gorgon. Fpga implementation of decision trees and tree ensembles for character recognition in vivado hls. *Image Processing & Communications*, 19(2-3):71, 2014.

[67] Brian Kulis et al. Metric learning: A survey. *Foundations and trends in machine learning*, 5(4):287–364, 2012.

[68] Brian Kulis et al. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364, 2013.

[69] Peter A Lachenbruch and M Goldstein. Discriminant analysis. *Biometrics*, pages 69–85, 1979.

[70] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[71] T. W Lee. *Independent component analysis – Theory and applications.* Kluwer Academic Publishers, Boston, 1988.

[72] Ping Lei and Shinji Kimura. Rf-sm: Random forest training process acceleration with subsampling method on fpga. *Training*, 171:48, 2018.

[73] Delon Levi. Hereboy: A fast evolutionary algorithm. In *Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware*, pages 17–24. IEEE, 2000.

[74] Andy Liaw, Matthew Wiener, et al. Classification and regression by random forest. *R News*, 2(3):18–22, 2002.

[75] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine learning*, 40(3):203–228, 2000.

[76] Zhe Lin, Sharad Sinha, and Wei Zhang. Towards efficient and scalable acceleration of online decision tree learning on fpga. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 172–180. IEEE, 2019.

[77] Shusen Liu, Dan Maljovec, Bei Wang, Peer-Timo Bremer, and Valerio Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE transactions on visualization and computer graphics*, 23(3):1249–1268, 2016.

[78] Olvi L Mangasarian, W Nick Street, and William H Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, 1995.

[79] Goeffrey J McLachlan. Mahalanobis distance. *Resonance*, 4(6):20–26, 1999.

[80] Jianyu Miao and Lingfeng Niu. A survey on feature selection. *Procedia Computer Science*, 91:919–926, 2016.

[81] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop*, pages 41–48. IEEE, 1999.

[82] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of machine learning.[sl], 2012.

[83] Yair Movshovitz-Attias, Alexander Toshev, Thomas K Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 360–368, 2017.

[84] Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.

[85] Hiroki Nakahara, Akira Jinguji, Simpei Sato, and Tsutomu Sasao. A random forest using a multi-valued decision diagram on an fpga. In *2017 IEEE 47th international symposium on multiple-valued logic (ISMVL)*, pages 266–271. IEEE, 2017.

[86] Ramanathan Narayanan, Daniel Honbo, Gokhan Memik, Alok Choudhary, and Joseph Zambreno. An fpga implementation of decision tree classification. In *2007*

Design, Automation & Test in Europe Conference & Exhibition, pages 1–6. IEEE, 2007.

[87] Jason Oberg, Ken Eguro, Ray Bittner, and Alessandro Forin. Random decision tree body part recognition using fpgas. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 330–337. IEEE, 2012.

[88] Mícheál O'Searcoid. *Metric spaces*. Springer Science & Business Media, 2006.

[89] Miguel Patrício, José Pereira, Joana Crisóstomo, Paulo Matafome, Manuel Gomes, Raquel Seiça, and Francisco Caramelo. Using resistin, glucose, age and bmi to predict the presence of breast cancer. *BMC cancer*, 18(1):1–8, 2018.

[90] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[91] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[92] Parisa Abdolrahim Poorheravi, Benyamin Ghojogh, Vincent Gaudet, Fakhri Karray, and Mark Crowley. Acceleration of large margin metric learning for nearest neighbor classification using triplet mining and stratified sampling. *Journal of Computational Vision and Imaging Systems*, 6(1):1–5, 2020.

[93] Christian Robert. Machine learning, a probabilistic perspective, 2014.

[94] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2002.

[95] Fareena Saqib, Aindrik Dutta, Jim Plusquellic, Philip Ortiz, and Marios S Pattichis. Pipelined decision tree classification accelerator implementation in fpga (dt-caif). *IEEE Transactions on Computers*, 64(1):280–285, 2013.

[96] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[97] Milad Sikaroudi, Benyamin Ghojogh, Amir Safarpoor, Fakhri Karray, Mark Crowley, and Hamid R Tizhoosh. Offline versus online triplet mining based on extreme distances of histopathology patches. In *International Symposium on Visual Computing*, pages 333–345. Springer, 2020.

[98] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014.

[99] W Nick Street, William H Wolberg, and Olvi L Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *Biomedical image processing and biomedical visualization*, volume 1905, pages 861–870. SPIE, 1993.

[100] Rastislav JR Struharik and Ladislav A Novak. Evolving decision trees in hardware. *Journal of Circuits, Systems, and Computers*, 18(06):1033–1060, 2009.

[101] Masashi Sugiyama. Dimensionality reduction of multimodal labeled data by local fisher discriminant analysis. *Journal of machine learning research*, 8(5), 2007.

[102] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[103] Alaa Tharwat, Tarek Gaber, Abdelhameed Ibrahim, and Aboul Ella Hassanien. Linear discriminant analysis: A detailed tutorial. *AI communications*, 30(2):169–190, 2017.

[104] Hamid R Tizhoosh. Opposition-based learning: a new scheme for machine intelligence. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 1, pages 695–701. IEEE, 2005.

[105] Matthew Turk and Alex Pentland. Face recognition using eigenfaces. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–587, 1991.

[106] Brian Van Essen, Chris Macaraeg, Maya Gokhale, and Ryan Prenger. Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga? In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 232–239. IEEE, 2012.

[107] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[108] René Vidal, Yi Ma, and S Shankar Sastry. Principal component analysis. In *Generalized principal component analysis*, pages 25–62. Springer, 2016.

[109] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480, 2006.

[110] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of machine learning research*, 10(2), 2009.

[111] Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. Sampling matters in deep embedding learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2840–2848, 2017.

[112] Yong Xu and Guangming Lu. Analysis on fisher discriminant criterion and linear separability of feature space. In *2006 International Conference on Computational Intelligence and Security*, volume 2, pages 1671–1676. IEEE, 2006.

[113] Hong Xuan, Abby Stylianou, and Robert Pless. Improved embeddings with easy positive triplet mining. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 2474–2482, 2020.

[114] Jieping Ye, Ravi Janardan, Cheong Hee Park, and Haesun Park. An optimization criterion for generalized discriminant analysis on undersampled problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8):982–994, 2004.

[115] Mang Ye, Xu Zhang, Pong C Yuen, and Shih-Fu Chang. Unsupervised embedding learning via invariant and spreading instance feature. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6210–6219, 2019.

[116] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 856–863, 2003.

[117] Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, et al. *Subspace linear discriminant analysis for face recognition*. Citeseer, 1999.

[118] Zhi-Hua Zhou and Ji Feng. Deep forest. *arXiv preprint arXiv:1702.08835*, 2017.