POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Ternary Neural Networks for Efficient Biometric Data Analysis

Supervisors

Prof. Enrico MAGLI

Prof. Tiziano BIANCHI

Prof. Andrea MIGLIORATI

Candidate

Giacomo AGNETTI

December 2024

Summary

Advancements in artificial intelligence, especially deep neural networks (DNNs), have driven significant progress in fields like computer vision and natural language processing. However, the high computational demands of these models present challenges for deployment on power-constrained and memory-limited devices, such as wearable ones. This thesis addresses these challenges in the context of gait analysis for biometric applications on resource-constrained wearable devices. While smartphones and smartwatches offer convenient platforms for capturing gait data via IMU sensors, their limited computational resources hinder real-time DNN deployment.

To overcome this, we introduce a Ternary Neural Network (TNN) framework that combines quantization and pruning to achieve high sparsity, setting most weights to zero to reduce memory and energy usage while maintaining model accuracy. Our approach dynamically adjusts quantization during training, achieving sparsity rates above 90% with entropy levels below 1 bit per symbol, making the model both highly compressible and effective. We evaluate the model on two biometric tasks: identification, where the model differentiates individuals based on gait patterns, and authentication, which verifies identity by comparing gait against a reference. Results indicate that TNNs retain strong discriminative power, enabling efficient and accurate gait-based biometric recognition.

Table of Contents

List of Tables

List of Figures

Acronyms

Adam

adaptive moment estimation

AI

artificial intelligence

BNN

binary neural network

CNN

convolutional neural network

COS

cosine similarity

CS

closed set

DCNN

deep convolutional neural network

EUC

Euclidean distance

FC

fully connected

FP

full precision

FNN

feed-forward neural network

GPU

graphics processing unit

HAMM

Hamming distance

ML

machine learning

NN

neural network

OS

open set

RNN

recurrent neural network

ReLU

rectified linear unit

SGD

stochastic gradient descent

TNN

ternary neural network

Chapter 1

Introduction

Recent advancements in artificial intelligence, particularly deep neural networks (DNNs), have driven progress across diverse fields, from computer vision to natural language processing. However, the high computational demands of these models pose challenges for deployment on resource-constrained devices such as wearables, which are limited by power and memory. Wearable devices, including smartwatches and smart glasses, have surged in popularity and are equipped with sensors that generate substantial data streams useful for various applications. Yet, the limited computational resources of these devices make running complex DNNs in real time a significant challenge. This has led to an increased focus on model compression techniques that reduce model size and energy consumption, enabling efficient deployment without sacrificing performance.

Biometric systems have become crucial for secure authentication and identification, often outperforming traditional methods like passwords. These systems leverage unique physiological and behavioral characteristics, providing a reliable and user-friendly alternative for verifying identity. Among these modalities, gait analysis has gained attention as a non-invasive biometric that can be captured unobtrusively, making it particularly suitable for wearable devices. This study investigates how neural network-based models can improve the accuracy and efficiency of gait-based biometric systems by exploring quantization techniques that optimize model performance within the constraints of wearable devices.

While neural networks have shown significant promise in biometric applications, deploying these networks on edge devices requires carefully designed, resource-efficient architectures. This thesis investigates specific neural network architectures and training methods optimized for biometric recognition within memory and power constraints. By optimizing model size and efficiency, we aim to make biometric models viable for real-world, low-power scenarios.

To achieve this, we employ Ternary Neural Networks (TNNs), which combine quantization with parameter pruning to reduce computational requirements. TNNs set a large proportion of weights to zero, resulting in a sparse model that maintains performance while using significantly less memory and storage space. We propose a ternarization framework that aims at achieving high sparsity rates while minimizing the performance loss due to quantization. It operates by dynamically changing the quantization function during training in order to push the model into higher sparsity gradually during training. Our approach achieves high sparsity rates—often exceeding 90%, with less than 10% of parameters active. This results in a ternary parameter distribution with entropy significantly below 1 bit per symbol, making it more compressible than traditional binary networks while also delivering superior performance.

The specific biometric signal used in this study is gait, represented by Inertial Measurement Unit (IMU) signals from a smartphone. Smartphones offer the advantage of widespread availability and ease of use, as they can unobtrusively capture gait data during daily activities without requiring dedicated equipment. This allows for a more practical and scalable biometric system that leverages widely available hardware. The models are tested in two scenarios: identification, where the model must recognize individuals based on their unique gait patterns, and authentication, where the goal is to verify a person's identity by comparing their gait against a stored reference.

This thesis is structured as follows: Chapter 2 provides an overview of neural networks, biometric data, and network compression techniques, establishing the foundation for our methods. Chapter 3 details the experimental setup, including dataset preparation, network architecture, and training methods. Chapter 4 presents the results, comparing ternary quantization with baseline models and evaluating model performance on key metrics. Finally, Chapter 5 summarizes the findings and discusses future directions for enhancing efficient biometric systems in resource-constrained environments.

Chapter 2

Background

In this chapter, we provide the foundational concepts and techniques essential for understanding the methods and experiments conducted in this study. This background knowledge sets the stage for the work that follows, covering the core topics necessary to contextualize our approach and results.

We begin by introducing neural networks, which serve as the primary models used in our work. This section covers the general architecture and training process of neural networks, as well as key network types relevant to this study. Next, we delve into biometric data, focusing on its role in identification and authentication tasks. This section explores various biometric modalities, the unique challenges they present, and their suitability for different applications. Lastly, we examine compression techniques for neural networks, an important consideration for deploying models in resource-constrained environments. Here, we review popular methods for reducing model size and improving computational efficiency, including quantization and pruning, which are relevant to the efficient deployment of models trained on biometric data.

2.1 Neural Networks

Neural networks are computational models inspired by the structure and functioning of the human brain. Comprising interconnected layers of nodes or "neurons," these networks are designed to process information by emulating how biological neurons transmit and process signals. Neural networks form the backbone of many AI applications, particularly in areas that require pattern recognition, classification, and decision-making. Unlike traditional algorithms based on rule-based logic, neural networks learn directly from data, adjusting weights and biases to reduce error and improve accuracy over time.

Applications of neural networks span a wide range of fields, including computer vision, natural language processing, healthcare, and finance. They are especially valuable for tasks involving high-dimensional and complex data. Neural networks are adept at identifying non-linear relationships and intricate patterns that traditional methods may overlook, providing enhanced predictive power and automation capabilities.

2.1.1 Origins and Evolution

The development of neural networks has been marked by significant advancements over the decades. In the following, we report an exploration of key models that contributed to the evolution of neural networks, along with essential formulas that define their mechanisms.

McCulloch and Pitts Model (1943)

The McCulloch and Pitts model is often regarded as the first theoretical model of a neuron, laying the groundwork for artificial neural networks [1]. Proposed by Warren McCulloch and Walter Pitts, this model introduced a binary neuron that activates based on a threshold rule, analogous to a step function. Each neuron receives multiple binary inputs (0 or 1) from other neurons, with each connection assigned an equal weight of one. If the sum of these inputs exceeds a certain threshold, the neuron "fires" or activates.

The McCulloch and Pitts model can be represented mathematically as follows:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

where y represents the output, x_i denotes each binary input, and θ is the threshold. Despite its simplicity, this model introduced key concepts, such as thresholds and binary activation, that remain foundational in modern neural networks.

Rosenblatt's Perceptron (1957)

Frank Rosenblatt expanded on the concept of artificial neurons with the development of the Perceptron, an algorithm capable of learning from data [2]. The perceptron model is a single-layer neural network that can classify linearly separable data. During the training process, the perceptron algorithm adjusts weights based on errors in its output, allowing it to "learn" simple patterns.

The mathematical representation of the perceptron's output is as follows:

$$y = \text{sign}\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

where w_i represents the weights associated with each input x_i, b is a bias term, and sign is the activation function that determines the output (usually 1 or -1 for binary classification). Rosenblatt's Perceptron introduced the concept of trainable weights, which are adjusted to minimize classification errors during training.

Hopfield Networks (1982)

John Hopfield introduced a different type of neural network, known as the Hopfield Network, designed for tasks involving associative memory [3]. Unlike the feed-forward structure of perceptrons, Hopfield networks are recurrent, with neurons interconnected in a way that each neuron influences every other neuron. Hopfield networks aim to minimize an "energy" function, guiding the network to stable states that represent stored patterns or memories.

The energy function E in a Hopfield network is given by:

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}s_i s_j$$

where s_i and s_j are the states of neurons i and j, and w_{ij} is the weight connecting them. The minimization of this energy function enables the network to reach stable patterns, which can be used to retrieve stored memories, making Hopfield networks useful for certain optimization and pattern recognition tasks.

Backpropagation and Multilayer Perceptrons (1986)

The introduction of the backpropagation algorithm in 1986 by David Rumelhart, Geoffrey Hinton, and Ronald Williams marked a pivotal advancement in neural networks, allowing for the training of multilayer perceptrons (MLPs) [4]. Unlike single-layer perceptrons, MLPs consist of multiple layers of neurons, enabling them to learn more complex patterns. The backpropagation algorithm calculates the gradient of the error with respect to each weight, adjusting weights to minimize error iteratively.

The weight update rule in backpropagation is given by:

$$w_i \leftarrow w_i - \eta\frac{\partial L}{\partial w_i}$$

where η is the learning rate, L is the loss function (typically Mean Squared Error or Cross-Entropy for classification tasks), and $\frac{\partial L}{\partial w_i}$ represents the partial derivative of the loss function with respect to weight w_i. This algorithm allowed MLPs to improve their performance through learning, driving significant advancements in neural networks and enabling the development of deeper architectures.

Convolutional Neural Networks (CNNs) (1989)

Yann LeCun introduced Convolutional Neural Networks (CNNs) in 1989, designing them for image recognition tasks [5]. CNNs utilize convolutional layers with spatial filters to capture local patterns and spatial hierarchies, enabling efficient processing of high-dimensional data by sharing weights across regions. This architecture is particularly effective for images, as it reduces the number of parameters required, allowing CNNs to handle complex visual data with improved efficiency.

Core components of CNNs include convolutional layers for feature extraction, pooling layers to reduce spatial dimensions, and fully connected layers for high-level classification. Together, these components allow CNNs to excel in computer vision applications such as image classification, object detection, and facial recognition. CNNs' capability to learn hierarchical spatial features has solidified their role as a fundamental model in deep learning and AI.

2.1.2 Common Network Architectures

Neural network architectures consist of interconnected layers, each designed to transform the input data through various learned parameters and activation functions. Typically, these networks contain input, hidden, and output layers, where each hidden layer processes input data through a series of transformations and non-linear activation functions to capture complex patterns.

Activations play a critical role by introducing non-linearity, enabling networks to approximate more complex functions beyond linear mappings. Some of the most used activation functions are:

- **ReLU:** Rectified Linear Unit (ReLU) is one of the most widely used activation functions in deep learning, particularly in convolutional networks, due to its computational efficiency and ability to mitigate the vanishing gradient problem. ReLU is defined as:
$$f(x) = \max(0, x),$$
where values below zero are set to zero, introducing sparsity and reducing computation in subsequent layers.

- **HardTanh:** HardTanh is a bounded, non-linear activation function that limits the output within a fixed range, making it useful in situations where controlled output ranges are needed, such as signal processing. The function is defined as:
$$f(x) = \begin{cases} 1 & \text{if } x > 1 \\ x & \text{if } -1 \le x \le 1 \\ -1 & \text{if } x < -1 \end{cases}$$

This bounded range helps prevent extreme output values, stabilizing learning in certain network structures.

- **Softmax:** The Softmax activation function is commonly used in the output layer of classification networks, where it converts raw output values into probabilities that sum to one across classes. It is defined as:

  $$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}},$$

  where x_i represents the output of each class, and the function scales these into a probability distribution over classes.

Neural networks come in various architectures, each tailored to specific data types and learning tasks. Different network structures, such as Feed-forward Neural Networks (FNN), Recursive Neural Networks (RNN), and Convolutional Neural Networks (CNN), are designed to handle unique data characteristics—whether they be tabular, sequential, or spatial data. These architectures differ in how they connect neurons, process information, and learn from data, allowing them to address a wide range of applications. Below, we explore the structures, functionalities, and use cases for each of these common neural network types.

Feed-forward Neural Network (FNN)

A Feed-forward Neural Network (FNN) is a simple yet foundational neural network structure where information flows in a single direction from the input layer through one or more hidden layers to the output layer. There are no loops or cycles, making the network structure straightforward.

In FNNs, each layer is typically a fully connected layer, where each neuron in a layer is connected to every neuron in the following layer. The output of each neuron in a fully connected layer is computed as:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right),$$

where w_i represents the weights, x_i the inputs, b the bias term, and f the activation function applied to the weighted sum. This structure enables FNNs to learn intricate representations, especially in tabular data contexts.

FNNs are commonly used in supervised learning tasks with structured data, such as classification and regression problems, where spatial or sequential patterns in data are less critical.

Recursive Neural Network (RNN)

Recursive Neural Networks (RNNs) are designed to process sequential data, such as time series or natural language, by maintaining an internal memory to capture dependencies between sequence elements. Each layer in an RNN is connected in a loop, allowing information to persist across time steps.

A simple RNN layer computes its output based on the current input and the previous hidden state:

$$h_t = f(Wx_t + Uh_{t-1} + b),$$

where h_t is the hidden state at time t, x_t the input at time t, W and U are weight matrices, b is the bias, and f is the activation function, often a non-linear function like tanh or ReLU. This formulation allows RNNs to capture dependencies across time steps.

RNNs are widely used for applications where data has temporal dependencies, such as language modeling, machine translation, and speech recognition, where the order and context of data points play a critical role.

Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are specialized for spatial data, especially image and video analysis, by exploiting spatial hierarchies through convolutional layers.

The convolutional layer is the fundamental building block of CNNs, where a small filter (or kernel) slides over the input data to produce feature maps that capture local patterns in the data. Each filter learns a unique set of weights to detect specific features, such as edges or textures, within its receptive field. Mathematically, the convolution operation for an input x and a filter w is defined as:

$$y_{i,j} = \sum_m \sum_n x_{i+m,j+n} \cdot w_{m,n},$$

where $y_{i,j}$ is the output at position (i, j) in the resulting feature map, and $w_{m,n}$ represents the filter weights. By convolving multiple filters across the input, the convolutional layer creates multiple feature maps, each highlighting different patterns or aspects of the input data.

Pooling layers reduce the spatial dimensions of feature maps $\mathcal{X}$, preserving essential information while reducing computational load. The two common types of pooling are max pooling and average pooling, given a region $\mathbf{x} \in \mathcal{X}$ with N elements in it, the output corresponding to that region y can be defined as:

- **Max Pooling:** selects the maximum value within each pooling window:

$$y = \max_0^N(x_i).$$

This operation helps highlight the most prominent features in each region, making the model more robust to minor variations in the position of features.

- **Average Pooling:** computes the average of values within each pooling window:
$$y = \frac{1}{N}\sum_{i=1}^{N} x_i.$$
This operation is useful for retaining more spatial information than max pooling, which can be helpful for tasks requiring fine details.

Convolutional networks often conclude with fully connected layers that transform the learned spatial features into a final classification or regression output. These layers aggregate information from the convolutional layers, acting as a final decision-making component.

CNNs are extensively used in computer vision tasks, including image classification, object detection, time series processing, and segmentation, where they excel in learning and extracting hierarchical spatial features from visual data.

2.1.3 Training Neural Networks

Training a neural network is the process of optimizing its parameters to minimize the difference between predicted and actual outcomes. This involves multiple steps, including forward propagation to produce predictions, evaluating prediction accuracy using a loss function, and applying backpropagation to adjust weights based on error gradients. To efficiently reach optimal parameter values, various optimization techniques are used, along with regularization methods to prevent overfitting.

Forward Propagation

Forward propagation is the process by which input data flows through a neural network to produce an output prediction. In each layer, data is transformed by layer-specific operations, such as convolutions in convolutional layers or weighted summations in fully connected layers, followed by an activation function.

For a general layer l, the output $a^{[l]}$ can be represented as:

$$a^{[l]} = f^{[l]}(\mathcal{L}(a^{[l-1]}; W^{[l]})),$$

where $f^{[l]}$ is the activation function, $\mathcal{L}$ denotes the layer-specific transformation (e.g., convolution, matrix multiplication), and $W^{[l]}$ represents the parameters (such as weights or filters) of layer l.

The final output layer produces the network's prediction, which is then evaluated with a loss function to measure the prediction error.

Loss Functions

The loss function quantifies the difference between the network's predictions and the actual target values. Minimizing the loss function is essential, as it directly correlates with improving the network's prediction accuracy. During training, the network's parameters (weights and biases) are adjusted to minimize this loss, leading to better performance on unseen data.

- **Mean Squared Error (MSE):** Mean Squared Error (MSE) is commonly used in regression tasks, measuring the average squared differences between predicted and true values. It is given by:

  $$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

  where y_i is the true value, $\hat{y}_i$ the predicted value, and N the number of samples. MSE penalizes larger errors more heavily, which can be beneficial for achieving high accuracy in regression tasks.

- **Cross Entropy:** Cross Entropy is widely used in classification tasks, as it measures the difference between two probability distributions: the true labels and the predicted probabilities. For a target class y and predicted probability $\hat{y}$, cross-entropy loss is defined as:

  $$\text{Cross Entropy} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i),$$

  where C is the number of classes. Cross Entropy encourages the model to increase the likelihood of the correct class, leading to more confident and accurate predictions.

- **Triplet Loss:** Triplet Loss is used in tasks requiring similarity learning, such as facial recognition, to ensure that similar inputs are closer in the feature space than dissimilar ones. It operates on triplets of anchor A, positive P, and negative N samples, with the loss defined as:

  $$\text{Triplet Loss} = \max(0,\ d(f(A), f(P)) - d(f(A), f(N)) + m),$$

  where $d(\cdot,\cdot)$ represents a generic distance function (e.g., Euclidean or cosine distance), and m is a margin parameter. This loss encourages the distance between A and P to be smaller than the distance between A and N by a margin of α, promoting tighter clustering of similar inputs in the feature space.

Backpropagation

Backpropagation is the algorithm used to minimize the loss function by iteratively adjusting the network's parameters. It calculates the gradient of the loss function with respect to each parameter in the network, enabling efficient parameter updates in the direction that minimizes the loss. This process involves two passes: the forward pass computes predictions, while the backward pass propagates the loss gradients back through the network.

Mathematically, backpropagation uses the chain rule to compute gradients for each layer. For a weight $W^{[l]}$ in layer l, the gradient $\frac{\partial L}{\partial W^{[l]}}$ is computed as:

$$\frac{\partial L}{\partial W^{[l]}} = \frac{\partial L}{\partial a^{[l]}} \cdot \frac{\partial a^{[l]}}{\partial W^{[l]}},$$

where $\frac{\partial L}{\partial a^{[l]}}$ represents the gradient of the loss with respect to the activations at layer l. This process continues for each layer until gradients for all parameters are computed, enabling to update them according to the optimization method of choice.

Optimization

Optimization in neural networks refers to the process of updating model parameters to minimize the loss function. By iteratively adjusting weights and biases, optimizers guide the network toward a configuration that reduces prediction error. Optimization algorithms use gradient information to determine the step size and direction for each parameter update, balancing convergence speed and accuracy. Two of the most used optimization methods are:

- **Stochastic Gradient Descent (SGD):** Stochastic Gradient Descent (SGD) is a fundamental optimization algorithm that updates parameters based on individual samples, rather than the entire dataset. The update rule for a weight W in SGD is:

  $$W_t = W_{t-1} - \eta \frac{\partial L}{\partial W},$$

  where η is the learning rate, and $\frac{\partial L}{\partial W}$ is the gradient of the loss with respect to W. By processing one sample at a time, SGD introduces noise that can help the network escape local minima, though it may converge slowly.

- **Adaptive Moment Estimation (Adam):** Adam is an adaptive optimization algorithm that combines the benefits of SGD with momentum and adaptive learning rates, making it effective for a wide range of tasks. Adam maintains two moving averages, m_t and v_t, for the gradients and squared gradients, respectively, and updates each parameter W as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\frac{\partial L}{\partial W},$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\left(\frac{\partial L}{\partial W}\right)^2,$$

$$W_t = W_{t-1} - \eta\frac{m_t}{\sqrt{v_t} + \epsilon},$$

where β_1 and β_2 are decay rates for the moving averages, η is the learning rate, and ϵ is a small constant for numerical stability. Adam's adaptive design allows it to converge faster and more reliably in practice.

Regularization Techniques

Overfitting is a common problem in training neural networks, where the model learns to perform exceptionally well on the training data but fails to generalize to new, unseen data. This occurs when the model captures noise or random fluctuations in the training data, leading to poor performance on the test set.

Regularization is a set of techniques used to prevent overfitting by adding constraints or modifications during training. These techniques help the network maintain simplicity and robustness, reducing its capacity to fit noise in the data. It is essential for achieving models that generalize well and perform accurately on new data, leading to more stable and reliable networks. Some of the most regularization techniques are:

- **L1 and L2:** L1 and L2 regularization are techniques that add penalty terms to the network's loss function based on the magnitude of the weights, effectively discouraging large weights. L1 regularization adds an absolute value penalty, promoting sparsity by setting some weights to zero. The L1 penalty term is:

  $$L_{L1} = \lambda \sum_i |w_i|,$$

  where λ is the regularization parameter controlling the penalty's strength, and w_i represents individual weights.

  L2 regularization, also known as Ridge regularization, penalizes the square of the weight values, which discourages large weights but tends to keep all weights small without driving them to zero. The L2 penalty term is:

  $$L_{L2} = \lambda \sum_i w_i^2.$$

Both L1 and L2 regularization terms are added to the loss function to achieve more generalized weight distributions, improving the model's generalization ability.

- **Dropout:** Dropout is a regularization technique where randomly selected neurons are "dropped" or set to zero during each training iteration. This prevents neurons from becoming overly reliant on specific patterns and forces the network to learn more robust features. During training, a neuron is retained with probability p and dropped with probability $1-p$, defined as:

$$\text{Output of neuron} = \begin{cases} 0 & \text{with probability } (1-p) \\ \text{original output}/p & \text{with probability } p \end{cases}$$

  where dividing by p ensures that the expected output during training matches the output during inference, when dropout is not applied. Dropout is especially effective in large networks where overfitting is prevalent.

- **Batch Normalization:** Batch Normalization is a technique that normalizes the input of each layer to have a mean of zero and a variance of one, reducing internal covariate shift. This stabilizes the training process and allows the network to use higher learning rates, which can also serve as an indirect form of regularization. For a batch of inputs x, batch normalization transforms each input as follows:

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}},$$

  where μ_B and σ_B^2 are the mean and variance of the batch, and ϵ is a small constant to prevent division by zero. Additionally, two trainable parameters, γ and β, scale and shift $\hat{x}$ to allow the layer to learn an optimal representation:

$$y = \gamma\hat{x} + \beta.$$

  Batch Normalization helps reduce dependency on initialization and acts as a regularizer by introducing noise from batch statistics, mitigating overfitting.

2.2 Biometric Data

Biometric data refers to the measurement and statistical analysis of people's unique physical and behavioral characteristics. Unlike traditional identification methods such as passwords or tokens, biometric systems leverage intrinsic human features for identification, providing a robust and often more secure means of recognizing individuals. These characteristics are typically challenging to replicate or steal, thereby offering a higher level of assurance in authentication and security processes.

The significance of biometric data in modern security and authentication systems has grown substantially, as these methods promise improved safety, reduced fraud, and streamlined access control. By relying on the distinct attributes of individuals, biometric authentication offers greater protection against unauthorized access. Consequently, biometric systems are increasingly deployed across sectors where security is critical, including financial services, healthcare, government facilities, and personal devices.

Over recent decades, biometric technologies have evolved from experimental applications to mainstream tools for identity verification. This progression has been driven by advances in data capture, machine learning, and data storage, as well as a growing acceptance of biometric methods. As a result, biometrics are now embedded in a variety of industries, offering solutions for various security challenges and enhancing user convenience.

2.2.1 Biometric Modalities

Biometric modalities are generally categorized into two main types: physiological and behavioral. Physiological modalities rely on biological traits like fingerprints, facial features, and iris patterns, which remain relatively constant over time. Conversely, behavioral modalities focus on how individuals perform certain actions, such as their voice characteristics or walking style, which can vary but are distinct to each person. Each type of modality offers unique applications and is associated with particular strengths and challenges.

Physiological

Physiological biometric modalities leverage biological features that are inherently unique to individuals. These features are typically stable, providing reliable identifiers that can be used across various applications.

- **Fingerprint** Fingerprint recognition is one of the oldest and most widely adopted biometric methods. It analyzes the unique ridge patterns on an individual's fingers, which remain largely unchanged over a person's lifetime.

Fingerprint systems are commonly used in smartphones, secure entry systems, and law enforcement due to their high accuracy and cost-effectiveness.

- **Face** Facial recognition technology analyzes facial features such as the distance between eyes, nose shape, and jawline. Facial biometrics are advantageous for non-intrusive, contactless authentication, making them suitable for surveillance, personal devices, and border control. However, face recognition systems can be vulnerable to variations in lighting, pose, and facial expressions, which may affect accuracy.
- **Iris, Retina** Iris and retinal scanning are highly accurate biometric modalities that analyze unique patterns in the eye. Iris recognition captures the intricate structures of the iris, which remain stable throughout life, whereas retinal scanning maps the unique patterns of blood vessels at the back of the eye. These methods are frequently used in high-security environments, as they offer a high level of precision but may require more complex equipment and user cooperation.

Behavioral

Behavioral biometrics focus on the unique ways individuals perform certain actions, which are often less stable over time compared to physiological traits. These biometrics add an extra layer of security by analyzing habits that are difficult to replicate.

- **Voice** Voice recognition analyzes an individual's speech patterns, including pitch, tone, and rhythm. It has become increasingly common in customer service and mobile applications, as it allows hands-free interaction. However, voice biometrics face challenges with accuracy due to background noise, health conditions affecting vocal quality, and the potential for imitation.
- **Gait** Gait analysis identifies individuals based on their unique walking patterns. This modality can be captured from a distance and does not require active cooperation from the individual, making it useful for surveillance. However, gait can be affected by temporary factors like injury or mood, posing challenges for consistent identification.

Others

Emerging biometric modalities, such as vein pattern recognition and electrocardiogram (ECG) signals, are gaining attention for their potential in secure authentication. Vein pattern recognition uses infrared imaging to map the vein patterns beneath the skin, which are unique to each individual and difficult to replicate.

ECG biometrics capture the heart's electrical activity, providing a unique and continuous form of authentication that is both physiological and behavioral. These technologies are still developing but hold promise for future applications in medical and high-security environments.

2.2.2 Applications

Biometric technologies have found numerous applications across various sectors, where their unique identification capabilities provide advantages over traditional methods. By utilizing inherent physiological and behavioral traits, biometrics offer enhanced security, user convenience, and efficiency, which can be particularly advantageous in sensitive or high-demand environments. Below are several key areas where biometric applications are gaining traction.

Health In healthcare, biometric systems are used to securely manage patient records, streamline access to medical information, and authenticate staff. For example, hospitals implement fingerprint or facial recognition systems to ensure that only authorized medical personnel access restricted areas or sensitive data. Biometrics also aids in patient identification, reducing errors in treatment by confirming the correct patient identity, particularly in emergency or high-traffic settings. Additionally, emerging biometrics like vein and ECG recognition are being researched for their potential to continuously monitor patients, adding a new layer to remote healthcare solutions.

Security Security is one of the primary fields where biometrics have become essential, offering reliable solutions for identity verification and access control. For instance, fingerprint and iris scanning are widely used in secure entry systems for government buildings, airports, and financial institutions, ensuring that only authorized individuals gain entry. Facial recognition is also extensively deployed in surveillance, enabling quick identification of individuals in public spaces or during large events. Furthermore, biometric authentication in personal devices, such as smartphones and laptops, has added an extra layer of security to protect users' personal data.

Transportation In the transportation sector, biometrics enhance both security and passenger convenience. Airports, for example, utilize biometric boarding systems where passengers use facial recognition to verify their identity, expediting the boarding process. Biometric systems are also applied in immigration control, where iris and fingerprint recognition streamline the identification process, reducing wait times and increasing security. Public transportation systems in several cities now experiment with biometric fare systems, where commuters can use fingerprint or facial recognition for seamless ticketing, bypassing the need for physical passes or tickets.

IoT The integration of biometrics with the Internet of Things (IoT) opens up innovative possibilities for smart home and connected device security. Biometric authentication in IoT applications can control access to smart homes, cars, and personal devices through fingerprint, facial, or voice recognition, making these devices more secure and user-friendly. For instance, a smart lock system might use facial recognition to allow household members or trusted individuals to enter, while denying access to unknown persons. Biometrics also enhance security in wearable IoT devices, like fitness trackers or smartwatches, which can use ECG or fingerprint data for user authentication, providing access only to authorized individuals.

2.2.3 Biometric Systems

A biometric system is a specialized pattern recognition system designed to automatically recognize individuals based on their unique physiological and/or behavioral characteristics. As Jain et al. (2004) discuss [6], these systems operate by capturing biometric data from individuals, extracting features, and comparing them against stored templates in a database. This approach ensures secure, efficient identification or verification by analyzing distinct personal attributes, enabling authentication processes based on "who a person is" rather than "what they possess" (e.g., an ID card) or "what they know" (e.g., a password).

Biometric systems operate in two primary modes: authentication and identification. In authentication (or verification) mode, the system performs a one-to-one comparison, confirming a user's claimed identity by matching their biometric data against a stored template associated with that identity. In contrast, the identification mode involves a one-to-many search across a database of biometric templates to identify an individual without requiring them to claim an identity.

Identification Identification mode in biometric systems involves determining an individual's identity by comparing their biometric data against multiple stored templates, without any prior identity claim. Formally, given an input feature vector $\mathbf{x}$, the system must determine the identity i from a set $\{1, 2, \ldots, N\}$, where N is the number of enrolled users, by maximizing the similarity function $S(\mathbf{x}, \mathbf{t}_i)$ over all templates $\mathbf{t}_i$. Mathematically, this can be expressed as:

$$\text{identify } i \text{ such that } \max_{j \in \{1,2,\ldots,N\}} S(\mathbf{x}, \mathbf{t}_j) \geq \tau,$$

where τ is a predefined threshold that determines the acceptance level for a match. Identification systems are critical in contexts requiring high security, as they establish identity through a one-to-many search, making them suitable for both positive (e.g., airport security) and negative recognition applications (e.g., identifying a person against criminal databases).

Authentication Authentication, also referred to as verification, is a process in which the biometric system validates a user's claimed identity by performing a one-to-one comparison with the stored template associated with that individual. If a user claims to be individual i, the system matches the input feature vector $\mathbf{x}$ with the template $\mathbf{t}_i$ and accepts the claim if the similarity score $S(\mathbf{x}, \mathbf{t}_i)$ meets or exceeds a threshold τ:

$$\text{verify identity if } S(\mathbf{x}, \mathbf{t}_i) \geq \tau.$$

Authentication is typically employed in scenarios where positive recognition is required, such as secure access control, where the aim is to prevent multiple people from using the same identity. This mode allows systems to offer reliable identity verification while ensuring user convenience and security, crucial in both commercial and personal applications.

Approach

The approaches used in biometric systems have evolved significantly over recent years, moving from traditional pattern-matching methods to more advanced machine learning techniques. This shift has been driven by a need for higher accuracy, robustness, and adaptability in diverse application environments. While traditional methods provided foundational solutions, modern deep learning approaches have improved the precision and scalability of biometric systems, despite introducing new challenges related to computational demands.

Traditional Approach Traditional biometric systems primarily relied on statistical and pattern recognition techniques, using fixed feature extraction methods and simple classifiers. Examples include fingerprint recognition systems based on minutiae matching, where ridge endings and bifurcations in fingerprint images are compared, or facial recognition systems using geometric features like distances between facial landmarks (e.g., eyes, nose, and mouth). These methods were often implemented with algorithms such as principal component analysis (PCA) for dimensionality reduction or linear discriminant analysis (LDA) for enhancing class separability.

However, these traditional approaches come with notable limitations. They typically lack flexibility in handling variations in data, such as changes in lighting, pose, or noise, which are common in real-world environments. Furthermore, they may fail to generalize effectively across diverse datasets or adapt to new data, requiring manual adjustment and optimization of feature extraction methods for different biometric traits, which limits scalability.

Deep Learning Approach The deep learning approach represents a modern shift in biometric recognition, leveraging neural networks to automatically learn relevant features from raw biometric data. Deep learning models, especially convolutional neural networks (CNNs), have proven highly effective in facial recognition, iris detection, and fingerprint analysis due to their capacity to learn complex patterns directly from data, minimizing the need for manual feature engineering. These networks enable the creation of high-dimensional embeddings that can distinguish individuals with exceptional precision and resilience to variations, such as lighting and pose.

Deep learning-based biometric systems operate by feeding raw or pre-processed biometric data into a network, which processes it through multiple layers, each extracting increasingly complex features. For instance, in face recognition, a CNN model learns low-level features such as edges in the initial layers and progressively builds more abstract representations, capturing unique aspects of an individual's face in the deeper layers. This approach results in robust feature vectors that enable accurate and efficient matching against stored templates.

This method outperforms traditional approaches in terms of accuracy, adaptability, and scalability. Deep learning models are particularly adept at handling complex variations in data, as they learn hierarchical representations that make them resilient to environmental changes. Additionally, deep learning systems often achieve higher accuracy with larger datasets, making them ideal for large-scale biometric applications where consistent accuracy is critical. Their ability to generalize across varied datasets has made deep learning a standard in modern biometric solutions.

Despite their advantages, deep learning approaches present challenges, primarily in terms of computational complexity. Training and running deep networks require significant processing power and memory, making them less suitable for resource-constrained devices, such as embedded systems or mobile platforms. Moreover, their high storage and power requirements complicate deployment on devices where low latency and efficiency are essential. Addressing these limitations has led to recent research in model compression and optimization techniques, aiming to make deep learning approaches more compatible with embedded applications, which will be discussed in the next section.

2.3 Model Compression

Model compression is essential for deploying deep neural networks in real-world applications, particularly when resource constraints are a concern. Large neural networks often require significant memory, computational power, and energy, which can limit their deployment on devices with limited resources, such as mobile and embedded systems. Compression techniques aim to reduce the size and computational complexity of these networks without significantly degrading their performance. By compressing a network, we can make it more efficient, enabling faster inference and reducing memory usage.

Compression is particularly beneficial for biometric data processing. Biometric models, which often involve high-dimensional data such as fingerprints, iris scans, and gait patterns, require efficient inference to enable real-time authentication and identification on resource-constrained devices. Compressing these models reduces latency and power consumption, making biometric data processing feasible on mobile and embedded platforms.

In this section, we review several compression techniques that can reduce the size of neural networks, including pruning, parameter quantization, and other advanced methods [7].

2.3.1 Pruning

Pruning is a model compression technique that reduces the number of parameters in a neural network by removing less significant weights or neurons. By eliminating weights that have minimal impact on the model's performance, pruning can significantly reduce model size and improve computational efficiency. The goal is to maintain accuracy while reducing the network's footprint, making it more efficient for deployment in environments with resource constraints.

Unstructured Pruning

Unstructured pruning removes individual weights based on their importance, leaving the network's structure largely unchanged. In their seminal work, LeCun et al. [8] introduced this concept, demonstrating that removing certain weights has little impact on the model's accuracy. Unstructured pruning is typically followed by a process of retraining, as outlined by Han et al. in [9], to recover any accuracy lost during pruning.

Regularization techniques, such as L1 and L2 regularization, are often employed during training to encourage sparsity, further enhancing the effects of pruning. Additionally, energy metrics can be used to guide pruning, as proposed by Yang et al. [10], which helps in designing energy-efficient models. However, unstructured pruning introduces irregular memory access patterns, which can be inefficient on certain hardware architectures.

Structured Pruning

Structured pruning is a model compression technique that removes entire structures within a neural network, such as neurons, filters, channels, or layers. Unlike unstructured pruning, which removes individual weights in an irregular pattern, structured pruning targets groups of parameters, resulting in a more organized and hardware-friendly network architecture. By reducing larger structures, structured pruning decreases both parameter count and computational complexity, making neural networks more feasible for resource-limited devices.

This technique is often preferred over unstructured pruning because it preserves the overall structure of the network, which enables more efficient processing on hardware like GPUs or mobile processors. The regular pattern of parameter removal facilitates better memory access patterns, minimizing irregular memory access that could hinder inference speed. Additionally, structured pruning aligns well with hardware accelerators, which benefit from dense computations and regular network patterns.

Structured pruning can be applied at different levels of network architecture, each with distinct advantages. Neuron pruning removes entire neurons based on their impact, often applied to fully connected layers to simplify computation. Filter or kernel pruning focuses on removing redundant filters in convolutional layers, reducing the computational load in CNNs [11] [12]. Channel-based pruning eliminates feature map channels, saving memory while preserving layer connectivity [13]. Finally, layer-based pruning removes entire layers or blocks, which can dramatically reduce model size and computation. These structured pruning methods offer adaptable strategies to create compressed models suited for various hardware and efficiency requirements.

2.3.2 Parameter Quantization

Parameter quantization is a model compression technique that reduces the precision of weights and activations by representing them with fewer bits. By limiting the number of possible values, quantization reduces memory requirements and computational load, making neural networks more suitable for deployment on resource-constrained devices, such as mobile and embedded systems. Quantization is particularly useful in applications requiring low latency and power efficiency, as it enables faster inference by simplifying arithmetic operations.

Quantization-aware training (QAT) is a technique where the model is trained to accommodate quantized parameters from the beginning. During QAT, quantization effects are simulated in each training step, allowing the model to learn to maintain accuracy despite lower precision. This approach minimizes accuracy loss, as the network adapts to quantization during the learning process, making it ideal for high-stakes applications where model performance cannot be compromised.

Post-training quantization (PTQ), on the other hand, involves training a model in full precision and then applying quantization to the trained weights and activations. PTQ is less computationally intensive than QAT and is often faster to implement, but it may result in a slight accuracy drop due to the model's lack of adaptation to quantized representations during training. PTQ is useful for models that can tolerate minor accuracy losses, offering a quick method to reduce model size and complexity after training.

In this thesis, we adopt a standard QAT method as a baseline to compare with our proposed ternarization approach, allowing us to assess whether ternarization provides meaningful improvements in sparsity and efficiency while maintaining accuracy. The following sections offer a general summary of binarization and ternarization methods, while Chapter 3 provides a detailed explanation of the specific methods considered in this study.

Binary Quantization

Binary quantization is a form of quantization-aware training where weights and activations are restricted to two possible values, typically $\{-1, +1\}$. This method minimizes memory and computational requirements by converting standard multiplications into simpler binary operations, which makes binary quantization highly efficient for embedded and low-power devices.

In binary quantization, the model is trained with binarized weights and activations, while the straight-through estimator (STE) is used during back-propagation to approximate gradients [14]. The STE allows gradients to flow through the binary layers despite the non-differentiable binarization step. This approach ensures that the network can learn meaningful representations even under extreme quantization, making binary quantization a powerful tool for resource-limited environments.

Ternary Quantization

Ternary quantization is similar to binary quantization but allows weights and activations to take on three possible values, typically $\{-1, 0, +1\}$. The inclusion of zero enables increased sparsity, which can lead to additional memory savings and power efficiency, as zero-valued weights can be pruned from the network structure, further reducing computational load.

Compared to binarization, ternary quantization provides more flexibility and often retains higher accuracy due to the added zero value. This additional degree of freedom allows the model to approximate complex functions more closely than a purely binary model.

2.3.3 Other Methods

Beyond pruning and quantization, several other model compression techniques exist that aim to reduce model complexity while maintaining performance. These methods provide alternative approaches to creating lightweight models, particularly useful when specific hardware requirements or model architectures do not align well with pruning or quantization. Techniques like low-rank decomposition and knowledge distillation offer different strategies for compressing models by focusing on the internal representations of parameters and outputs.

Low-rank Decomposition

Low-rank decomposition compresses a neural network by approximating large weight matrices with lower-rank representations. By factoring weight matrices into products of smaller matrices, low-rank decomposition reduces the number of parameters in dense layers, which is particularly effective for compressing fully connected layers and certain convolutional layers. This decomposition leverages the idea that the full rank of a matrix may not be necessary to capture the most informative features within the data.

One of the main advantages of low-rank decomposition is its ability to significantly reduce the parameter count and computational load, especially for large-scale networks. However, it may lead to a slight performance loss if the rank is reduced too aggressively, as lower-rank approximations may fail to capture some details in the data. Low-rank decomposition is most effective in cases where network redundancy is high, making it ideal for tasks with large amounts of data and dense model structures but less suitable for tasks where high-rank representations are crucial.

Knowledge Distillation

Knowledge distillation is a compression technique where a smaller "student" model is trained to mimic the behavior of a larger "teacher" model. Introduced by Hinton et al., this method involves using the teacher model's output, often softened by temperature scaling, as labels for the student model. By learning from the teacher's output distribution, the student model can achieve performance levels close to the teacher's, despite having significantly fewer parameters.

The primary advantage of knowledge distillation is that it enables the student model to inherit the performance of a larger model without needing its full complexity, making it highly suitable for resource-constrained environments. However, knowledge distillation may require extensive computational resources for training, as the student model must learn the teacher's nuanced output behavior. Additionally, this method depends heavily on the quality of the teacher model; if the teacher model has limitations, these may be transferred to the student model.

Chapter 3

Methodology

In this chapter, we outline the methodologies and experimental setups used to investigate the effectiveness of neural network models for biometric tasks. This chapter provides a detailed overview of the data, network architectures, training approaches, and quantization techniques central to our approach.

We begin by describing the dataset and data types considered in this study, including the specific biometric features used for identification and authentication. This section establishes the basis for subsequent experiments and contextualizes the data requirements for effective model training. Next, we introduce the basic network topology employed across our experiments, detailing the foundational architecture on which our models are built. This consistent structure serves as a comparative baseline for evaluating different quantization methods. Then, we present the binary training approach, which serves as the initial baseline for quantized models. Binary quantization reduces the model weights to two levels, providing a benchmark for efficiency and performance. Further on, we introduce our proposed ternary quantization method, which extends the binary approach by allowing weights to take on three discrete values. This section details the implementation of ternarization within the network and highlights its potential advantages in terms of model efficiency and performance. Finally, we formalize the biometric tasks of identification and authentication, explaining each task's requirements, evaluation metrics, and relevance to the study.

3.1 Dataset

While the proposed ternarization method could be applied to various biometric signals, this thesis focuses on smartphone IMU data collected from multiple subjects in an unconstrained environment for gait analysis. The dataset utilized for this study is `Dataset #1` from the WHUGAIT dataset collection [15]. This section outlines the data format, the collection process, and the steps involved in data preprocessing.

3.1.1 Data Format

The dataset contains input samples $\mathbf{x} \in \mathcal{X}$ in the form of $6 \times T$ matrices, representing the evolution of three-axis accelerometer and three-axis gyroscope readings over time. Specifically, we have:

$$\begin{aligned} \mathbf{x} &= \begin{pmatrix} x_1 & x_2 & \dots & x_{T-1} & x_T \end{pmatrix} \\ x_t &= \begin{pmatrix} a_{xt} & a_{yt} & a_{zt} & g_{xt} & g_{yt} & g_{zt} \end{pmatrix}^T \end{aligned}$$

where T is the number of readings in each sample, a_{xt}, a_{yt}, and a_{zt} represent the accelerometer readings along the X, Y, and Z axes respectively for the t-th reading, and similarly, g_{xt}, g_{yt}, and g_{zt} represent the gyroscope readings. Each sample $\mathbf{x}$ is associated with a label $y \in \{1, 2, \dots, S\}$, indicating which of the S subjects the sample belongs to.

3.1.2 Data Collection

The WHUGAIT dataset was collected by having each subject install an Android app on their smartphone, which continuously recorded IMU signals in the background at a sampling rate of 50 Hz while they performed everyday activities, such as commuting, working, or cooking, with no specific restrictions on their behavior.

As a result, the recorded data streams include both walking and non-walking sessions without any predetermined structure. Several preprocessing steps are required to achieve the data format described in Section 3.1.1.

3.1.3 Preprocessing

Semantic Partition Since only walking data is relevant for gait analysis, the first preprocessing step involves partitioning each inertial time series into walking and non-walking sections. This can be accomplished using a one-dimensional semantic segmentation DCNN inspired by U-NET [15]. The network can be trained on a small subset of the collected data that can be manually labeled with minimal time investment, as only a few labeled samples are needed. Additionally, the segments of the raw data stream where a subject was walking can be easily identified by visual inspection, making the labeling process straightforward.

Figure 3.1 illustrates the network architecture in detail. The raw inertial time series are divided into samples made of 1024 readings, which, at a sampling rate of 50 Hz, correspond to about 20 seconds of data. Then each input sample goes through a compression network which consists of blocks composed of two 1×16

convolution layers with ReLU activations, followed by 2 × 2 max-pool layers with stride 2 that act as down-samplers. The expansion network is composed of similar blocks where the max-pooling layer is replaced by an up-convolution layer. The number of channels in the feature maps is doubled with every down-sample and halved with every up-sample. The resulting output is a 1 × 1024 array used to identify parts of the sample likely representing walking.

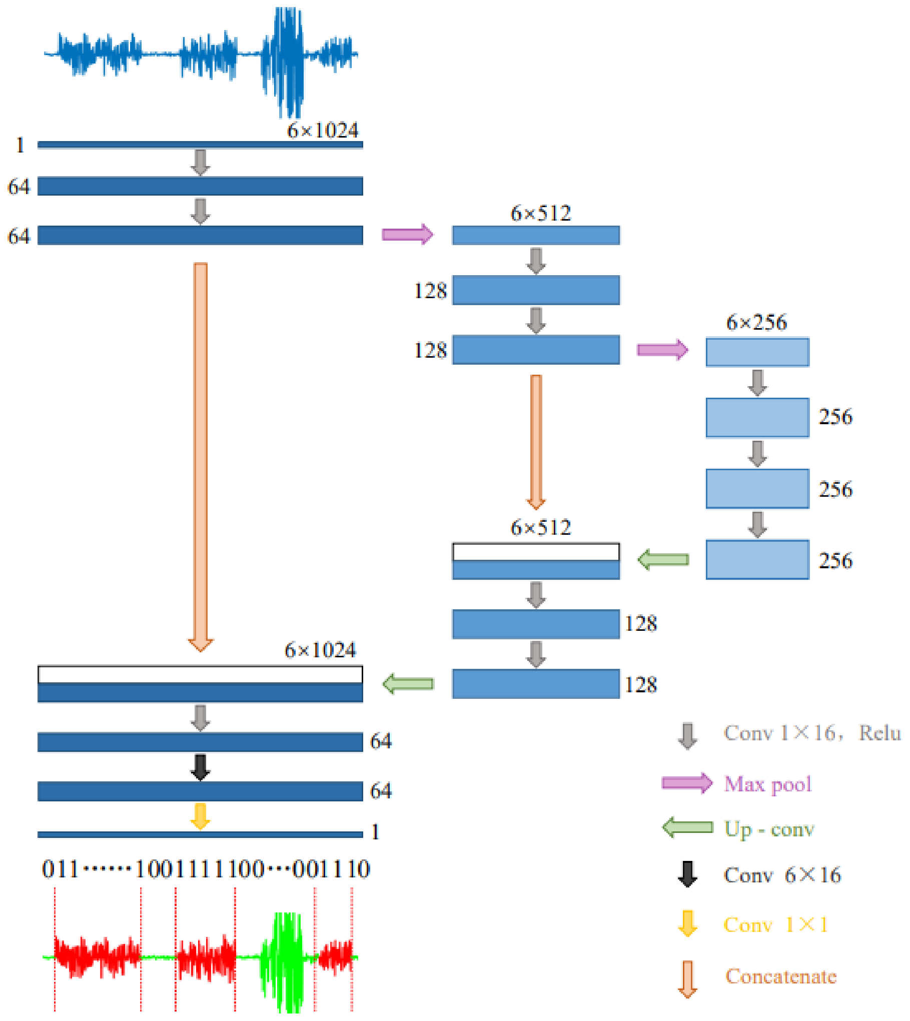

Figure 3.1: DCNN used to partition walking and non-walking sections; originally from [15].

Gait Cycle Partition Once the walking sections are extracted by the segmentation network, they are subsequently divided into separate steps, where a step is defined as the interval between two consecutive ground contacts of the same foot. Without loss of generality, each ground contact can be detected by identifying local maxima in the magnitude of the acceleration vector, $\|a_t\| = \sqrt{a_{x,t}^2 + a_{y,t}^2 + a_{z,t}^2}$, as shown in Figure 3.2. The magnitude is used instead of individual axis values because the smartphone orientation is unknown. The peak detection algorithm is calibrated by analyzing a small subset of samples to determine appropriate amplitude and cycle time thresholds for each subject.

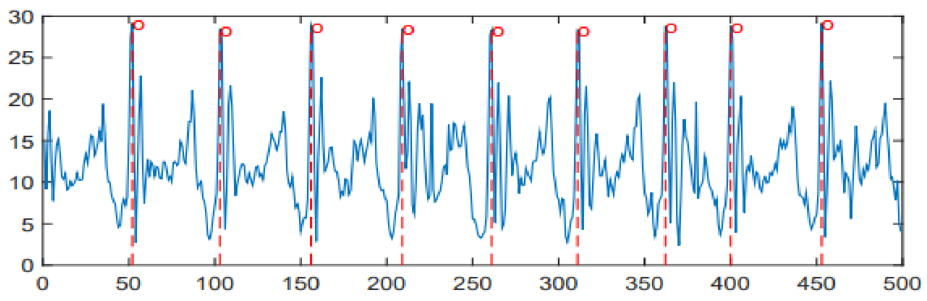

Figure 3.2: Acceleration vector module evolution over time. Used to detect steps cycles [15].

Linear Time Interpolation Once the beginning of each step is annotated, each input sample **x** is generated by extracting slices that contain two consecutive steps and fitting them into an array of size 128 using linear interpolation. It is important to note that this two-step slicing method is specific to `Dataset #1`, which is the dataset used for the experiments. The samples in the other datasets are generated using different methods. For example, `Dataset #2` creates fixed-length slices starting from the beginning of each step annotation, while `Dataset #3` employs fixed-length slices without considering step annotations.

3.2 Network Topology

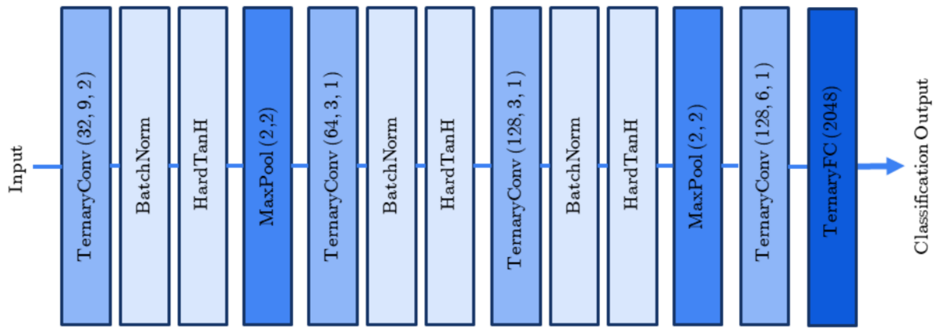

Figure 3.3: Base network architecture used for identification and authentication tasks. Convolutional layers are swapped for quantized ones when testing compressed models. The last FC layer is discarded when extracting embeddings for the authentication tasks.

Figure 3.3 illustrates the network architecture that serves as the foundation for the models discussed in the following sections. This architecture consists of two main components: a convolutional section followed by a fully connected layer.

The convolutional section contains four blocks, each composed of a convolutional layer, batch normalization, and an activation function. Max-pooling is applied after the first and third blocks, using a 1×2 max-pooling layer with a stride of 2, which reduces the size of the feature maps by half along the time dimension. The number of channels increases from 32 to 128 as data progresses through the network. The first three convolutional layers use one-dimensional kernels of varying sizes $(1 \times K)$ to operate along the time dimension. The fourth layer, however, combines both the 3-axis accelerometer and 3-axis gyroscope data using a 6×1 kernel, which operates along the spatial dimension (size 6). As a result, an input of size 6×128 is reduced to a feature map of size 1×16, with a depth of 128. These feature maps, now of size $128 \times 1 \times 16$, are flattened into arrays of size $128 \times 1 \times 16 = 2048$. These arrays are then passed into the final fully connected layer, whose output size is customizable based on the task, such as matching the number of classes in a classification problem.

When evaluating quantized models, the network architecture remains largely unchanged. The main difference lies in the activation function: the full-precision model uses `ReLU`, whereas the quantized models employ `HardTanh`, as recommended in previous studies [16]. Additional considerations specific to quantized training are necessary; these will be discussed in detail in the following section (3.3).

3.3 Quantized Training

The network described in Section 3.2 contains approximately 337K parameters, which are typically represented using 32-bit floating-point precision. As a result, running inference for a single sample requires millions of floating-point multiplications. While this is manageable for devices equipped with specialized hardware, such as GPUs, which support heavy parallel processing, deploying such a network on power- and computation-constrained devices—like wearables or IoT devices—poses significant challenges [17]. As explained previously, several compression methods have been developed to address the challenges posed by computational and storage constraints by reducing the computational burden on the hardware. This thesis investigates the effectiveness of quantization as a compression method when applied to IMU signals for gait authentication and identification. Specifically, we propose a novel method for ternarization and compare its performance with both a full-precision baseline and more conventional quantization techniques, such as binarization.

In binary networks, weights and activations are limited to $\{-1, +1\}$. In contrast, ternary networks allow weights to take the additional value of 0, which enables further memory reduction by eliminating zero-weighted neurons. This makes ternary architectures particularly appealing for applications where maximizing storage compression is essential. A straightforward approach to obtaining a quantized network involves taking a fully-trained, full-precision model and applying a typical quantization function. However, this would result in quantized weights that are too different from the optimal ones learned during training, leading to a significant loss in performance. To address this issue, it is necessary to incorporate quantization during the training process itself. By doing so, the network can adapt to the quantization constraints and retain better performance. The following sections describe the binarization method used to train a quantized baseline model. This is followed by the introduction of a custom ternarization method, specifically designed to produce a highly compressible model by encouraging a large portion of the weights to be set to zero.

3.3.1 Binary Training

While it is possible to implement Binarized Neural Networks (BNNs) using full precision activations, in this study we quantize both the weights and activations to maximize efficiency, which is crucial for resource-constrained devices. The sign quantization function used is defined as follows:

$$x^b = \text{Binarize}(x) = \text{Sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise.} \end{cases} \qquad x \in \mathbf{R} \tag{3.1}$$

Here, x is a floating-point value (either a weight W or an activation a), and x^b represents its binarized version (denoted as W^b or a^b). Since the derivative of the quantization function 3.1 is zero for all input values, standard back-propagation cannot be used to accumulate gradients in the weights, as these gradients would also be zero everywhere. To understand this issue, let us consider a generic layer k with weights W_k. During the forward pass, this layer receives the quantized activations from the previous layer, a^b_{k-1}, as input and produces the activations for the current layer, $a_k = W^b_k \cdot a^b_{k-1}$. These activations will then be quantized before being forwarded to the next layer, $a^b_k = \text{Sign}(a_k)$.

During back-propagation, the gradient is computed as:

$$\frac{\partial C}{\partial a_k} = \frac{\partial C}{\partial a^b_k} \cdot \frac{\partial a^b_k}{\partial a_k} \quad \text{or equivalently} \quad g_{a_k} = g_{a^b_k} \cdot \frac{\partial a^b_k}{\partial a_k}$$

where $g_{a^b_k} = \frac{\partial C}{\partial a^b_k}$ is the gradient of the cost function C with respect to the quantized activation a^b_k. However, since $\frac{\partial a^b_k}{\partial a_k}$ is the derivative of the sign function, which is always zero, the gradient $g_{a^b_k}$ cannot propagate any further, effectively blocking the update of the weights. To overcome this issue, we need a method to calculate the gradient g_{a_k} that bypasses standard back-propagation for the quantization function.

In [14], Bengio studied several methods for estimating gradients in stochastic discrete neurons and found that the "straight-through estimator" (**STE**), originally introduced in Hinton's lectures [18], resulted in faster training. The method used in this study was first presented by *Courbariaux et al.* in [19], where they adapted the **STE** for use in Binarized Neural Networks (BNNs) that use a deterministic quantization function, such as the sign function. This approach approximates the gradient g_{a_k} as:

$$g_{a_k} = g_{a^b_k} \mathbf{1}_{|a_k| \leq 1}$$

where $\mathbf{1}_{|a_k| \leq 1}$ is the indicator function, which equals 1 when $|a_k| \leq 1$ and 0 otherwise.

Instead of using the derivative of the sign function, $\frac{\partial a^b_k}{\partial a_k}$, we simply set $g_{a_k} = g_{a^b_k}$ when the absolute value of the neuron's activation is relatively small (i.e., $|a_k| \leq 1$). Conversely, we set $g_{a_k} = 0$ when the neuron's activation is large (i.e., $|a_k| > 1$). This introduces a saturation effect, where neurons that are strongly activated, either positively or negatively, do not propagate their gradient backward. This behavior is equivalent to back-propagating the gradient through a $\text{HardTanh}(a_k)$ function, whose derivative is $\mathbf{1}_{|a_k| \leq 1}$, as illustrated in Figure 3.4.

Algorithm 1 Binary training. C is the cost function for mini-batch, the functions Binarize($\cdots$) and Clip($\cdots$) specify how to binarize and clip weights, L is the number of layers, Update($\cdots$) is the optimization method of choice (such as `SGD` or `Adam`. `BatchNorm` layers and layer biases b_t are omitted for clarity.

```
Require: a mini-batch of $(a_0,\ a^*)$ (inputs, targets), previous parameters $W_t$
    (weights) and learning rate $\eta$.
Ensure: updated weights $W_{t+1}$.
                                                        ▷ 1. Forward propagation
for $k = 1$ to $L$ do
    $W_t^b \leftarrow$ Binarize($W_t$)
    if $k = 1$ then
        $a_k \leftarrow a_{k-1} \cdot W_b^t$                    ▷ Do not binarize inputs
    else
        $a_k \leftarrow a_{k-1}^b \cdot W_b^t$
    end if
    if $k \neq L$ then
        $a_k^b \leftarrow$ Binarize($a_k$)
    end if
end for
                                                        ▷ 2. Backward propagation
Compute $g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing $a_L$ and $a^*$
for $k = L$ to 1 do
    if $k \neq L$ then
        $g_{a_k} \leftarrow g_{a_k^b} \circ \mathbf{1}_{|a_k| \leq 1}$
    end if
    $g_{a_{k-1}^b} \leftarrow g_{a_k}^T \cdot W_k^b$
    $g_{W_k^b} \leftarrow {a_{k-1}^b}^T \cdot g_{a_k}$
end for
                                                        ▷ 3. Parameter update
for $k = 1$ to $L$ do
    $W_{t+1} \leftarrow$ Clip(Update($W_t,\ \eta,\ g_{W_t^b}$), $-1,\ 1$)
end for
```

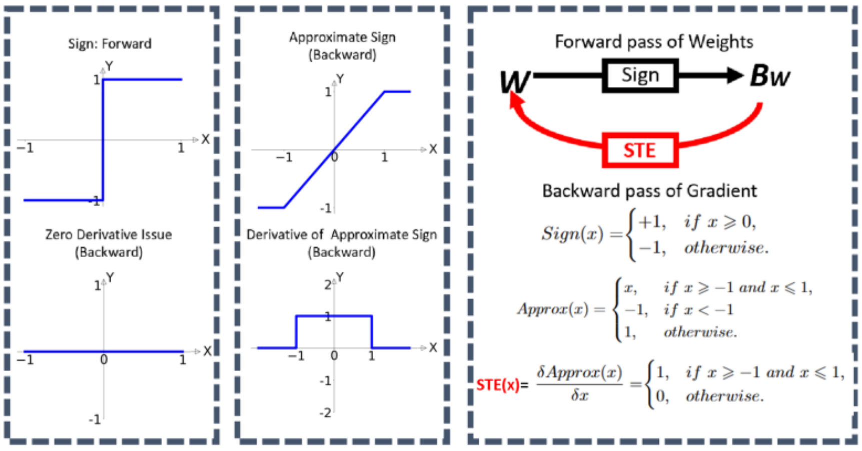
Sign: Forward
Approximate Sign (Backward)
Zero Derivative Issue (Backward)
Derivative of Approximate Sign (Backward)
Forward pass of Weights
W
Sign
Bw
STE
Backward pass of Gradient
$Sign(x) = \begin{cases} +1, & if\ x \geqslant 0, \\ -1, & otherwise. \end{cases}$
$Approx(x) = \begin{cases} x, & if\ x \geqslant -1\ and\ x \leqslant 1, \\ -1, & if\ x < -1 \\ 1, & otherwise. \end{cases}$
STE(x)= $\frac{\delta Approx(x)}{\delta x} = \begin{cases} 1, & if\ x \geqslant -1\ and\ x \leqslant 1, \\ 0, & otherwise. \end{cases}$

Figure 3.4: STE forward versus backward pass difference [16].

Forward Pass During the forward pass, for each of the L layers, we take the binarized activations from the previous layer and combine them with the binarized weights, according to the layer type (i.e., convolutional or linear). However, we do not binarize the input layer a_0 or the output layer a_L, as doing so would result in excessive loss of information.

Backward Pass During the backward pass, for each layer, we take the gradient flowing from the next layer, g_a^b, and bypass the quantization function's chain rule by directly setting $g_a = \mathbf{1}_{|a_k| \leq 1}$ as explained earlier. This modified gradient is then propagated back to the previous layer using the chain rule: $g_{a_{k-1}^b} \leftarrow g_{a_k}^T \cdot W_k^b$, and similarly into the weights: $g_{W_k^b} \leftarrow {a_{k-1}^b}^T \cdot g_{a_k}$. It is important to note that while both the weights W_k^b and activations a_k^b are quantized, their corresponding gradients, $g_{W_k^b}$ and $g_{a_k^b}$, are not. Therefore, these gradients must be accumulated in full precision. This does not pose any issues since these full-precision variables are only needed during training and are discarded afterward.

Parameter Update During optimization, we update the full-precision weights W_{t+1} using the gradients accumulated from the binarized weights, $g_{W_t^b}$. This is necessary because making small adjustments to W_t^b directly would not lead to meaningful changes in the binarized weights. After each update, the full-precision weights W_{t+1} are clamped to the interval $[-1, 1]$ to ensure they do not drift too far from their binarized counterparts. This procedure is compatible with various optimization strategies, such as ADAM or SGD, as they only require the parameter values and their corresponding gradients.

3.3.2 Ternary Training

Ternary networks offer an advantage over binary networks because weights can also take the value 0, allowing for further model compression and enabling faster, more power-efficient inference. If a large portion of the weights are zero, these weights can be removed from the network topology when deploying the model on resource-constrained devices, reducing both storage and computational costs. Therefore, during training, it is crucial to control the percentage of zero weights to strike the right balance between compression and performance.

To train the ternary model, we use the same approach as for the binary network, with the primary difference being the quantization function, which is adapted to allow weights and activations to be set to zero. Specifically, we use the following ternary quantization function:

$$x_q = \text{Ternarize}(x;\ \Delta) = \begin{cases} +1 & \text{if } x > \Delta, \\ 0 & \text{if } |x| \leq \Delta, \\ -1 & \text{if } x < -\Delta. \end{cases}$$

where x is a floating-point value (which could be a weight W or an activation a), and x^q represents its ternarized version (denoted as W^q or a^q). The threshold value Δ is a hyperparameter that controls the number of weights set to zero, directly affecting the sparsity rate. A higher Δ increases the number of weights quantized to zero, allowing for greater compression but potentially limiting the model's learning capacity. Since the derivative of the quantization function is still zero for all input values, we use the same straight-through estimator (STE) as described for binary training in 3.3.1. This approach can be easily adapted to ternary networks, as the gradient estimation is not dependent on the specific quantization function used. As outlined in Algorithm 2, the rest of the training procedure remains mostly unchanged, with two major exceptions.

Delta Regimes

Initial experimental results suggested that increasing the quantization threshold Δ during training, instead of keeping it constant, could lead the model to perform better while achieving higher sparsity rates. Hence, we define the growth schedule for Δ_t as:

$$\Delta_t = \min\left(\Delta_0 + (\Delta_{\max} - \Delta_0) f(t/t_{max}),\ \Delta_{\max}\right),$$

where Δ_0 is the delta value at epoch 0, $\Delta_{\max}$ is the threshold value at epoch t_{max}, and f is a profile function that is defined in the intervals $[0, 1] \rightarrow [0, 1]$. The function f can be used to control the growth of Δ from epoch 0 to t_{max}. After reaching t_{max}, the threshold remains constant at $\Delta_{\max}$.

The growth profiles taken into consideration in this study are:

- **Linear:** constant growth

$$f_{\text{linear}}(x) = x \quad \forall x \in [0,1]$$

- **Square:** slow ramp up

$$f_{\text{square}}(x) = x^2 \quad \forall x \in [0,1]$$

- **Square Root:** fast ramp up with $f'(0) = +\infty$

$$f_{\text{sqrt}}(x) = \sqrt{x} \quad \forall x \in [0,1]$$

- **Logarithmic:** fast ramp up with $f'(0) = 1$

$$f_{\log}(x) = \log(x+1)/\log(2) \quad \forall x \in [0,1]$$

Delta Adjustment

When training with high initial values of Δ_0, it is possible that training fails to start. This happens because most weight initialization methods produce small initial values, and a high threshold Δ_0 would cause all the weights to be set to zero, effectively halting the training process. To address this issue, during the forward pass of each epoch t, we scale down Δ_t as follows:

$$\Delta_t^{\text{adj}} = \Delta_t \cdot \max(|W_t|)$$

where W_t are the full-precision weights for a given layer. If the initial weights are small, this adjustment reduces the threshold Δ_t^{adj} to be more compatible with these small values. Since the weights W_t are clamped within the range $[-1, 1]$, after a reasonable amount of epochs, $\max(|W_t|)$ will equal 1, as at least one of the weights will reach the clipping bound. This ensures that the threshold adjustment only affects the early phases of training, leaving later stages unaffected.

Algorithm 2 Ternary training. C is the cost function for mini-batch, the functions Ternarize$(\cdots)$ and Clip$(\cdots)$ specify how to ternarize and clip weights, L is the number of layers, Update$(\cdots)$ is the optimization method of choice (such as `SGD` or `Adam`). `BatchNorm` layers and layer biases b_t are omitted for clarity.

```
Require: a mini-batch of ($a_0$, $a^*$) (inputs, targets), previous parameters $W_t$ (weights) and learning rate $\eta$, delta regime hyperparameters ($f()$, $\Delta_0$, $\Delta_{max}$, $t_{max}$).
Ensure: updated weights $W_{t+1}$ and delta threshold $\Delta_{t+1}$.
 1:                                                  ▷ 1. Forward propagation
 2: for $k = 1$ to $L$ do
 3:     $\Delta_t^{\text{adj}} \leftarrow \Delta_t \cdot \max(|W_t|)$
 4:     $W_t^q \leftarrow$ Ternarize($W_t$; $\Delta_t^{\text{adj}}$)
 5:     if $k = 1$ then
 6:         $a_k \leftarrow a_{k-1} \cdot W_t^q$                 ▷ Do not binarize inputs
 7:     else
 8:         $a_k \leftarrow a_{k-1}^b \cdot W_t^q$
 9:     end if
10:     if $k \neq L$ then
11:         $a_k^q \leftarrow$ Ternarize($a_k$; $\Delta_{\text{adj}}$)
12:     end if
13: end for
14:                                                  ▷ 2. Backward propagation
15: Compute $g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing $a_L$ and $a^*$
16: for $k = L$ to 1 do
17:     if $k \neq L$ then
18:         $g_{a_k} \leftarrow g_{a_k^q} \circ \mathbf{1}_{|a_k| \leq 1}$
19:     end if
20:     $g_{a_{k-1}^q} \leftarrow g_{a_k}^T \cdot W_k^q$
21:     $g_{W_k^q} \leftarrow {a_{k-1}^q}^T \cdot g_{a_k}$
22: end for
23:                                                  ▷ 3. Parameter update
24: for $k = 1$ to $L$ do
25:     $W_{t+1} \leftarrow$ Clip(Update($W_t$, $\eta$, $g_{W_t^b}$), $-1$, $1$)
26:     $\Delta_{t+1} \leftarrow \min\left(\Delta_0 + (\Delta_{\max} - \Delta_0) f(\frac{t+1}{t_{max}}),\ \Delta_{\max}\right)$
27: end for
```

3.4 Identification Task

Identification in gait analysis refers to the process of recognizing an individual based on their unique gait patterns from a database of multiple potential known subjects. Unlike authentication, which can be seen as a one-to-one verification process, identification operates under a one-to-many comparison model. The goal is to determine the identity of an individual by matching their gait data to the corresponding records within a database of known subjects.

3.4.1 Problem Formulation

Identification can be modeled as a n-class classification task, where, given an input gait curve $\mathbf{x}$ (described in 3.1.1), the objective is to find which of n knows subjects $S = \{s_1, \ \dots, \ s_n\}$ the sample belongs to.

Recalling the network described in 3.3 we consider: $\mathcal{N}$ the convolutional layer, $\mathcal{L}$ the final fully connected layer, than we have:

$$\mathbf{o} = \{o_1, \ \dots, \ o_n\} = \mathrm{softmax}(\mathcal{L}(\mathcal{N}(\mathbf{x}))),$$

where o_k is probability of sample $\mathbf{x}$ belonging to subject s_i, i.e. $o_k = p(s_k|\mathbf{x})$.

3.4.2 Criterion and Training

We define the training dataset as $\mathbf{D} = \{(\mathbf{x}_1, \mathbf{o}_1^*), \dots, (\mathbf{x}_N, \mathbf{o}_N^*)\}$, where N is the total number of training samples, and each pair $(\mathbf{x}_i, \mathbf{o}_i^*)$ consists of an input sample $\mathbf{x}_i$ and its corresponding target label $\mathbf{o}_i^*$. The target labels $\mathbf{o}^*$ are defined such that they represent the class memberships of the input samples. Specifically, $\mathbf{o}^*$ is a one-hot encoded vector, where each element corresponds to a particular class (in this case, a subject) and is defined as follows:

$$\mathbf{o}^* = \{o_1^*, \dots, o_n^*\}, \tag{3.2}$$

$$o_k^* = \begin{cases} 1 & \text{if } \mathbf{x} \text{ belongs to subject } s_k, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

This means that for a given input $\mathbf{x}$, the value $o_k^* = 1$ if the input sample belongs to the k-th subject, and all other entries in the vector $\mathbf{o}^*$ are 0. To train the network, we minimize the cross-entropy loss, a standard loss function for classification tasks, which measures the difference between the true labels $\mathbf{o}^*$ and the predicted outputs $\mathbf{o}$ of the network. The cross-entropy loss function for a single sample is defined as:

$$l_{\text{cross}}(\mathbf{o}^*, \mathbf{o}) = -\sum_{k=1}^{n} o_k^* \log(o_k) + (1 - o_k^*) \log(1 - o_k)$$

This loss function penalizes the model if the predicted probability o_k for the true class $o_k^* = 1$ is low, encouraging the model to assign higher probabilities to the correct class.

Next, in each training iteration, we typically work with mini-batches, i.e., random subsets of the dataset. Let $\tilde{\mathbf{D}} \subset \mathbf{D}$ represent a mini-batch of the dataset. The total training cost $C(W)$, which we seek to minimize, is computed as the average cross-entropy loss over the mini-batch. It is given by:

$$C(W) = \frac{1}{|\tilde{\mathbf{D}}|} \sum_{i=1}^{|\tilde{\mathbf{D}}|} l_{\text{cross}}(\mathbf{o}_i^*, \mathcal{L}_W(\mathcal{N}_W(\mathbf{x}_i)))$$

Here:

- $\tilde{\mathbf{D}}$ is the mini-batch of samples,
- $|\tilde{\mathbf{D}}|$ is the size of the mini-batch,
- $\mathbf{x}_i$ is the i-th input sample in the mini-batch,
- $\mathcal{N}_W(\mathbf{x}_i)$ is the network's output for sample $\mathbf{x}_i$, parameterized by W, and
- $\mathcal{L}_W$ applies the softmax function to the network's output to produce a probability distribution over classes.

In summary, the goal during training is to minimize the total training cost $C(W)$ over all mini-batches. By iteratively updating the model weights W to reduce this cost, the network learns to classify input samples accurately.

3.4.3 Evaluation Metrics

In this section, we outline the evaluation metrics used to assess model performance on the identification task. The primary metric is top-1 accuracy, which measures the each model's ability to correctly identify the target in its highest-ranked prediction. For the ternary model, we also consider the sparsity rate, reflecting the model's compactness and memory efficiency.

Top-1 Accuracy

Top-1 accuracy is a widely used metric in classification tasks and is defined as the percentage of times the model's top prediction matches the true label of the input. It is a robust metric for tasks where identifying the exact class is critical, such as in subject identification.

Formally, let the dataset $\mathbf{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i$ is the input sample and y_i is the true label corresponding to subject s_i. The model produces a set of predicted probabilities $\mathbf{o} = \{o_1, o_2, \ldots, o_n\}$ for n possible classes. The predicted class $\hat{y}_i$ is the one with the highest probability:

$$\hat{y}_i = \arg\max_c o_c$$

Top-1 accuracy is then defined as:

$$\text{Top-1 Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\hat{y}_i = y_i)$$

where:

- N is the total number of samples in the dataset,
- $\mathbb{I}(\hat{y}_i = y_i)$ is an indicator function that equals 1 if the predicted class $\hat{y}_i$ matches the true class y_i, and 0 otherwise.

Sparsity Rate

In the ternary model, the sparsity rate (SR) refers to the percentage of weights that are set to zero, effectively reducing the number of active parameters. A higher sparsity rate indicates that the model is more compact, using fewer weights, which means lower storage requirements and lower power consumption.

Formally, the sparsity rate SR can be defined as:

$$SR = \frac{\text{Number of zero weights}}{\text{Total number of weights}} \times 100\%.$$

For a comprehensive evaluation, the ternary model's performance cannot be assessed based solely on accuracy. Instead, accuracy should be considered along with the sparsity rate, as SR directly influences the model's deployability on resource-constrained devices by defining its compressibility and efficiency.

Shannon Entropy can also serve as a metric to evaluate the compressibility of a quantized model. For a model where weights assume discrete values (e.g., -1, 0, 1 in a ternary model), the entropy H is defined as:

$$H = -\sum_i p_i \log_2(p_i),$$

where p_i is the probability of a weight taking each discrete value i. A lower entropy indicates higher compressibility, as fewer unique weight values lead to greater redundancy and thus a more potentially compact model.

For binary models, where weights are typically equally split between $+1$ and -1, the Shannon entropy is close to 1. In contrast, for ternary models, if weights are evenly distributed among -1, 0, and 1, the entropy would reach approximately

1.6, making the model heavier than its binary counterpart. However, by increasing the sparsity rate to around 75%, a ternary model can achieve the same size as a binary model. With sparsity rates beyond 75%, the ternary model becomes more compressible than the binary model, despite leveraging three possible weight values instead of two.

3.5 Authentication Task

Authentication in gait analysis refers to the process of verifying an individual's identity by comparing their current gait pattern to a previously stored profile. Unlike identification, which seeks to determine who a person is from a group of potential candidates, authentication operates under a one-to-one comparison model, confirming whether the individual is who they claim to be.

3.5.1 Problem Formulation

Authentication can be modeled as a binary classification task where the objective is to determine if two gait curves $(\mathbf{x}_a,\ \mathbf{x}_b)$ belong to the same subject. Recalling the network described in 3.3 we consider: $\mathcal{N}$ the convolutional layer, than we have:

$$\mathbf{e} = \{e_1,\ \dots,\ e_m\} = \mathcal{N}(\mathbf{x}),$$

where $\mathbf{e}$ is an m-dimensional embedding that represents the input sample $\mathbf{x}$, in a lower dimensional space. We can say a pair of input samples $(\mathbf{x}_a,\ \mathbf{x}_b)$ belong to same subject if:

$$d(\mathbf{e}_a,\ \mathbf{e}_b) = d\left(\mathcal{N}(\mathbf{x}_a),\ \mathcal{N}(\mathbf{x}_b)\right) < d^*, \quad d^* \in \mathbb{R},$$

where d is an arbitrary distance function, while d^* is a distance threshold that can be determined during training.

3.5.2 Criterion and Training

To train the network $\mathcal{N}$, we need to ensure that input samples from the same subject produce embeddings that are close to each other, while those from different subjects produce embeddings that are far apart. For this reason, we use the triplet loss as a loss function, which can be defined as:

$$l_{\text{tri}}(\mathbf{e}_0,\ \mathbf{e}_+,\ \mathbf{e}_-) = \max(d(\mathbf{e}_0,\ \mathbf{e}_+) - d(\mathbf{e}_0,\ \mathbf{e}_-) + m,\ 0).$$

where $\mathbf{e}_0$ is the embedding of an arbitrary input sample used as the reference (anchor), $\mathbf{e}_+$ and $\mathbf{e}_-$ are the embeddings of input samples that belong to the same subject (positive) and a different subject (negative), respectively. The triplet loss becomes zero when the negative sample embedding is further away from the anchor than the positive sample embedding by more than a margin m. Conversely, it assumes a high value when the positive embedding is farther from the anchor than the negative sample.

We consider the dataset $\mathcal{D}_3 = \{(x_0, x_+, x_-)_i\}_1^N$ made of N training triplets where x_+ belongs to the same subject as x_0 while x_- does not. Then we define the cost function C on a mini-batch $\tilde{\mathcal{D}}_3 \subset \mathcal{D}_3$ as:

$$C(W) = \sum_{(\mathbf{x}_0, \mathbf{x}_+, \mathbf{x}_-) \in \tilde{\mathcal{D}}_3} l_{\text{tri}}(\mathcal{N}_W(\mathbf{x}_0),\ \mathcal{N}_W(\mathbf{x}_+),\ \mathcal{N}_W(\mathbf{x}_-)).$$

Distance Functions

To evaluate the effectiveness of different embedding representations in distinguishing between subjects, we experiment with various distance functions $d(\mathbf{e}_a, \mathbf{e}_b)$ in the triplet loss. Each distance metric offers unique properties that may impact the model's ability to accurately cluster similar gait patterns while separating dissimilar ones. The specific distance functions tested are:

- **Euclidean Distance:** The Euclidean distance $d_{\text{euc}}(\mathbf{e}_a, \mathbf{e}_b)$ measures the straight-line distance between two embedding vectors $\mathbf{e}_a$ and $\mathbf{e}_b$ in a multidimensional space. This metric is particularly suitable for embeddings where absolute differences in feature space are meaningful, as it emphasizes the overall magnitude of dissimilarity. The Euclidean distance formula is:

  $$d_{\text{euc}}(\mathbf{e}_a, \mathbf{e}_b) = ||\mathbf{e}_a - \mathbf{e}_b||_2$$

  where $||\cdot||_2$ represents the L_2 norm, calculating the square root of the sum of squared differences between corresponding elements in $\mathbf{e}_a$ and $\mathbf{e}_b$.

- **Cosine Distance:** Cosine distance $d_{\cos}(\mathbf{e}_a, \mathbf{e}_b)$ assesses the angular difference between two embedding vectors, making it particularly effective when the direction of embeddings (rather than their magnitude) is crucial for comparison. This is defined as:

  $$d_{\cos}(\mathbf{e}_a, \mathbf{e}_b) = 1 - \frac{\mathbf{e}_a \cdot \mathbf{e}_b}{||\mathbf{e}_a||_2 ||\mathbf{e}_b||_2}$$

  Here, $\mathbf{e}_a \cdot \mathbf{e}_b$ is the dot product, while the denominator normalizes by the Euclidean norms of $\mathbf{e}_a$ and $\mathbf{e}_b$. Cosine distance ranges from 0 (indicating identical embeddings) to 1 (orthogonal embeddings), making it well-suited for high-dimensional embeddings where vector orientation carries more information than magnitude.

- **Pseudo-Hamming Distance:** For binary or ternary embeddings, we apply a custom distance metric, the Pseudo-Hamming distance, optimized for quantized vector spaces. This metric, $d_{\text{hamm}}(\mathbf{e}_a^q, \mathbf{e}_b^q)$, is defined as:

  $$d_{\text{hamm}}(\mathbf{e}_a^q, \mathbf{e}_b^q) = \frac{||\mathbf{e}_a^q - \mathbf{e}_b^q||_1}{m}$$

  where $\mathbf{e}_a^q$ and $\mathbf{e}_b^q$ are the quantized embeddings, and m denotes the dimensionality of the vectors. The L_1 norm $|| \cdot ||_1$ sums the absolute differences between elements of $\mathbf{e}_a^q$ and $\mathbf{e}_b^q$, capturing a distance that approximates Hamming distance but remains differentiable for training. This metric is particularly useful in optimizing quantized models, allowing us to measure similarity efficiently while maintaining model differentiability.

3.5.3 Evaluation Metrics

In this section, we describe the evaluation metrics used to assess the performance of the models on the authentication task. The chosen metrics provide a comprehensive view of how well the learned embeddings can be used to authenticate a subject, as well as how they allow to solve different down-stream tasks such as classification and clustering.

ROC AUC

The ROC AUC (Receiver Operating Characteristic - Area Under the Curve) is a metric used to evaluate the performance of a binary classification model, providing insight into the model's ability to distinguish between two classes. In our case, ROC AUC measures the model's ability to determine if a subject's gait matches a reference gait or not, making it a valuable metric for evaluating identification and authentication tasks. After training, we calculate the embeddings for each sample in the test set using each model. From these embeddings, we compute a pairwise distance matrix where each element represents the distance between a pair of embeddings. This distance is calculated using a specific distance function, which may differ from the one used during training with the triplet loss. The matrix is then divided into two groups: "matching" pairs, where the embeddings correspond to the same subject (positive class, labeled as 1), and "non-matching" pairs, where the embeddings correspond to different subjects (negative class, labeled as 0).

To evaluate model performance, we use the ROC curve, which plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various distance thresholds. The TPR (sensitivity) is defined as:

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}},$$

and the FPR is defined as:

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}.$$

By varying the distance threshold, we observe changes in TPR and FPR, allowing us to assess how well the model differentiates between matching and non-matching pairs.

The Area Under the ROC Curve (AUC) provides a single scalar value summarizing the model's performance across all thresholds. A higher AUC value, closer to 1, indicates better discriminative ability, meaning the model is more effective at separating matching and non-matching pairs. An AUC of 0.5, by contrast, would indicate performance no better than random guessing. Thus, ROC AUC serves as a comprehensive metric for assessing the model's ability to identify matching gaits accurately.

Downstream Classification: Top-1 Accuracy

A downstream task evaluates how effectively the model's learned embeddings perform in a practical application. Here, we select a classification task to assess the model's ability to generate discriminative embeddings for gait curves, enabling differentiation between classes (e.g., individuals). In practice, after training we use the model to generate embeddings for gait curves in both the training and test sets. These embeddings provide compact, high-level representations of the input data. To evaluate their separability, we train a Support Vector Machine (SVM) on the training set embeddings and classify the test set embeddings. The top-1 accuracy metric measures performance, representing the proportion of samples correctly classified when the top predicted class matches the true label.

Downstream Clustering: Rand Index and Silhouette Score

We also consider a clustering downstream task to evaluate the model's effectiveness in an unsupervised scenario, where the goal is to assess how well the learned embeddings represent distinct classes without prior labels. To perform this evaluation, we first use the trained model to generate embeddings for gait curves in the test set. We then apply K-Means clustering on these embeddings to determine how well they naturally group similar samples. For this assessment, we consider the following two metrics:

- **Adjusted Rand Index (ARI):** ARI measures the similarity between the predicted clusters and the true class labels, adjusting for chance. The ARI score ranges from -1 to 1, where a score of 1 indicates perfect agreement between the clustering and the ground truth, 0 indicates random clustering, and negative values suggest worse-than-random clustering. ARI is particularly useful when the number of clusters differs from the number of classes.

- **Silhouette Score:** The silhouette score measures how similar an object is to its own cluster compared to other clusters. The score ranges from -1 to 1, where a score close to 1 indicates that the sample is well clustered, with a large distance between clusters. A score of 0 means the sample is on or very close to the decision boundary between two clusters, while negative values indicate that the sample may have been assigned to the wrong cluster. This metric provides insight into the compactness and separability of clusters formed by the embeddings.

Chapter 4

Experimental Results

In this chapter, we present the experimental results to evaluate the performance of our models across key biometric tasks, focusing on how ternary quantization compares to both full-precision and binary baselines.

We start by detailing the experimental setup, including the configurations, parameters, and evaluation metrics used to ensure consistency and reliability in our results. Next, we present the results for the identification task, comparing the accuracy and efficiency of the ternary model against full-precision and binary models. This analysis highlights the potential benefits of ternary quantization in maintaining or improving accuracy while enhancing model efficiency. Finally, we report the results for the authentication task, using the same comparative framework to assess the effectiveness of ternary quantization in delivering resource-efficient models without compromising authentication accuracy.

4.1 Setup

As outlined in Section 3.1, the dataset used for the experiments is `Dataset #1` from the WHUGAIT dataset collection. It comprises 36,844 labeled gait samples from 118 subjects, split into 33,104 training samples and 3,740 test samples, with no overlap between the splits. The training and test sets are both reasonably balanced across labels, ensuring robustness during model evaluation.

The network topology is detailed in Section 3.2. All models, whether utilizing full, binary, or ternary quantization schemes, follow this same network architecture. The training process is consistent across models, with the primary difference being that, for the binary and ternary models, a quantization function is applied during the forward pass. During backpropagation, gradients are estimated using the straight-through estimator, as discussed in Section 3.3. The models are implemented in PyTorch [20] and trained on a `Linux` system equipped with an `RTX 2070S` GPU.

The number of training epochs is chosen to ensure that both identification and authentication tasks can be completed within one hour while allowing the model to converge. Network parameters are initialized following the method of *He et al.* [21], where weights are sampled from a normal distribution $W \sim \mathcal{N}(0, \sqrt{\frac{2}{n_k}})$, with n_k representing the number of parameters in the convolutional kernel or linear layer.

4.2 Identification Task

For the identification task, models are trained using stochastic gradient descent (SGD) with a momentum of 0.9, regardless of the quantization scheme. Each model is trained for a maximum of 500 epochs. We also use an early stopping criterion to prevent overfitting. Specifically, every 10 epochs, we evaluate the model's accuracy on the validation set, and in the results, we report the best-performing model, not necessarily the final one.

4.2.1 Hyper-Parameter Optimization

To achieve optimal performance, we conduct a grid search to identify the best hyperparameters for each of the three quantization methods considered: full precision, binary, and ternary. For the full-precision model, we search for the optimal values of learning rate η and batch size B. The binary model also relies on these two hyperparameters, as the binarization algorithm does not introduce any additional configurations. For the ternary model, we additionally explore the impact of different growth regimes for the threshold Δ to identify configurations that yield the best performance with high sparsity rates. Figure 4.1 illustrates example growth regimes for each profile described in Section 3.3.2, with a maximum threshold value $\Delta_{\max} = 0.3$ achieved at epoch 100.

Hyper-Parameter	**Values**	**Best Value**		
		Full Precision	**Binary**	**Ternary**
Learning rate η	0.001, 0.01, 0.02, 0.03	**0.001**	**0.02**	**0.01**
Batch size B	128, 256, 512	**256**	**128**	**256**

Table 4.1: Hyper-parameter search values found for full precision, binary, and ternary quantization methods. The best values for the ternary model are referred to a constant Δ value of 0.2.

Table 4.1 shows the range of values explored during the grid search for learning rate and batch size, alongside the best-performing configurations identified for each of the three quantization methods. For the ternary model, we first search for the

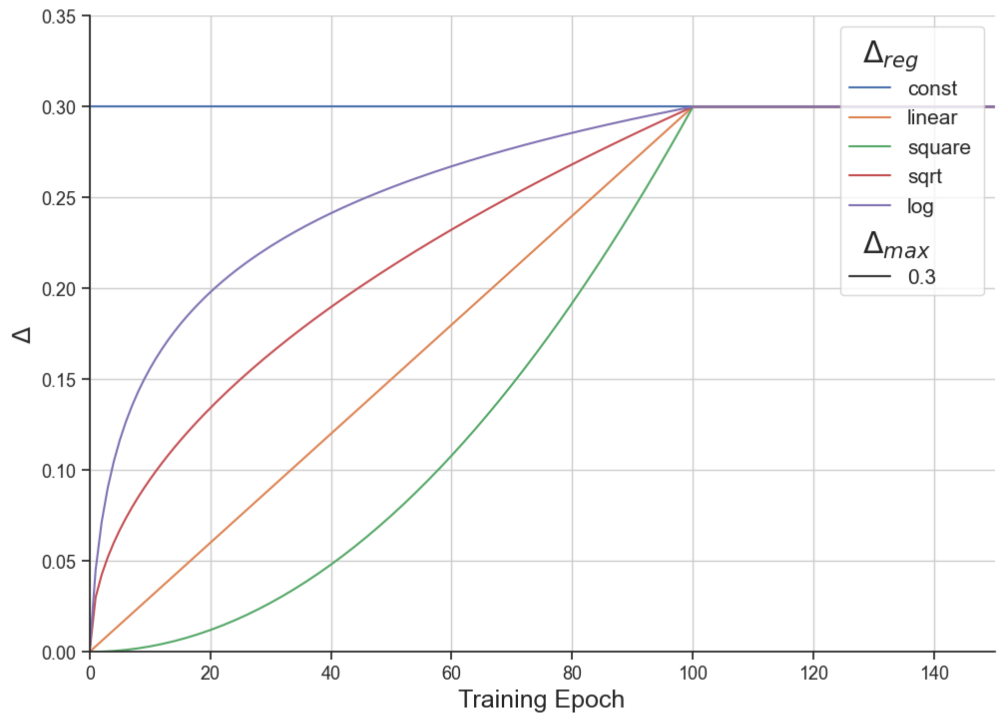

Figure 4.1: Examples of Δ growth regimes for different f functions. Only the growth regimes with $\Delta_{\max} = 0.3$ and $t_{\max} = 100$ are shown in the plot.

optimal η and B with Δ held constant at 0.2. Afterward, we use the best-found values to study how different growth profiles affect the model's performance and sparsity rate. This approach helps reduce the size of the grid search and allows for a more meaningful comparison between growth regimes. The considered growth regimes and their respective performances are presented in the next section, as determining the absolute best model is non-trivial due to the trade-off with sparsity, which impacts the model's compressibility.

4.2.2 Results

Table 4.2 reports the best test accuracy, sparsity rates, and entropy values obtained for each quantization method. In the top half, we show results for state-of-the-art techniques that use full precision weights. In the bottom half, we show results for our full precision and binary baselines, alongside several ternary models trained with different Δ regimes. Our full-precision (FP) baseline performance is comparable to other state-of-the-art models, despite its lightweight design (372k parameters), achieving an accuracy of 94.27%. The binary model loses about 1.8 percentage points due to quantization. Most ternary models outperform the binary models while maintaining high sparsity rates, consistently above 80%. The best-performing growth regime is the *log* profile, which results in a model that loses only 1.2 percentage points relative to the FP baseline while achieving a sparsity rate of 91.1%. This translates to significant model compression, as with a sparsity rate of 90%, only around 37k parameters remain in the network, with the rest being pruned.

We also note that, while maintaining a constant Δ threshold during training can achieve good sparsity rates, it leads to sub-optimal classification accuracy when compared to the increasing Δ growth regimes. This underscores the rationale for gradually increasing Δ with each epoch as training progresses.

	Acc. (%)	**Sparsity (%)**	**Entropy (bits/sym)**	**Acc. Diff. (%)**	**Params. (#)**
CNN+LSTM [22]	93.52	-	-	-	-
IdNet [23]	92.91	-	-	-	-
DeepConvLSTM [24]	92.25	-	-	-	> 996K
MFEBP [25]	95.38	-	-	-	-
FCN-BiLSTM [26]	95.27	-	-	-	2.89M
Ours (FP)	94.27	-	-	-	372K
Ours (BNN)	92.41	1	1	-1.86	372K
Ours (TNN, Δ const)	92.39	91.2	0.54	-1.88	33K
Ours (TNN, Δ linear)	92.02	87.2	0.65	-2.25	48K
Ours (TNN, Δ square)	92.78	**91.3**	**0.53**	-1.49	32K
Ours (TNN, Δ sqrt)	92.68	85.2	0.75	-1.59	55K
Ours (TNN, Δ log)	**93.01**	**91.1**	0.55	**-1.26**	34K

Table 4.2: Best test accuracy, sparsity rate, and entropy comparison for different Δ growth regimes. In the top half, we show results for competing state-of-the-art techniques, obtained from different FP baselines. In the bottom half, we report the full precision and binary baselines for our method. Note that the number of parameters for ternary models is obtained by multiplying the total number of parameters by the sparsity rate.

Figure 4.2 shows the progression of sparsity rates throughout training for two growth regimes, *square* and *log*, across three different values of Δ_{max} ($\Delta_{\text{max}} = 0.1, 0.2, 0.3$), with each maximum value reached at the 250th training epoch ($t_{\text{max}} = 250$). After epoch t_{max}, we observe that the sparsity rate stabilizes, largely determined by Δ_{max} and with limited effect from the growth profile. As expected, higher Δ_{max} values lead to increased sparsity, with $\Delta_{\text{max}} = 0.1$ yielding around 70% sparsity, while $\Delta_{\text{max}} = 0.2$ and $\Delta_{\text{max}} = 0.3$ result in approximately 85% and 90%, respectively. However, setting Δ_{max} above 0.3 destabilizes training, likely due to the high proportion of zero weights, leading to significantly degraded performance. For clarity, results for $\Delta_{\text{max}} > 0.3$ are omitted in Figure 4.2 but can be found in Figure 4.3, which includes multiple experimental runs to provide a comprehensive evaluation of the proposed ternary quantization method.

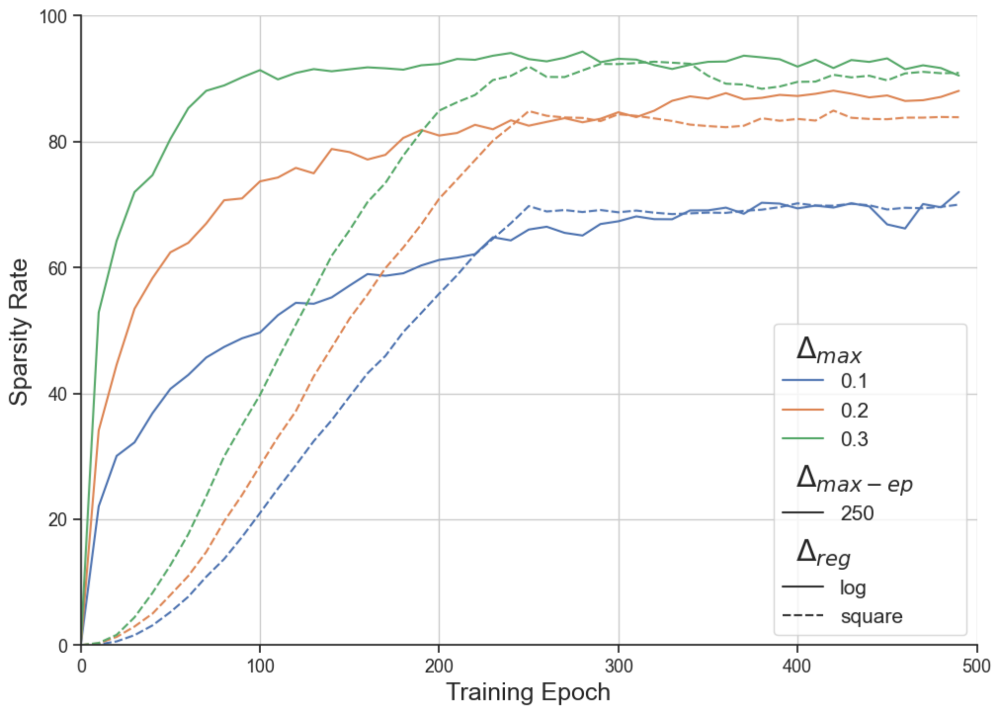

Figure 4.2: Examples of how the sparsity rate changes with the training epoch for two different growth regimes (*square* and *log*) and three different Δ_{max} values ($\Delta_{\text{max}} = 0.1, 0.2, 0.3$), reached at the 250th training epoch. For $\Delta_{\text{max}} > 0.3$, the training becomes unstable and leads to sub-optimal results, as can be seen in Figure 4.3.

Figure 4.3 illustrates classification accuracy and sparsity rate achieved by every growth regime tested. The dashed lines denoted as `W32A32` and `W1A1` represent the accuracy of the full precision and binary model, respectively.

With classification accuracy reaching 93% (a reduction of less than 1.5% compared to the full precision baseline) and sparsity exceeding 91%, the logarithmic growth regime with $\Delta_{\max} = 0.3$ yields the best results among the considered growth regimes. Furthermore, we note that the majority of ternary models significantly outperform the binary baseline in terms of classification accuracy, while also exhibiting entropy rates below 1. This demonstrates the advantage of utilizing the ternary framework over binary networks. As anticipated, growth regimes with $\Delta_{\max} > 0.3$ show heavily degraded performances.

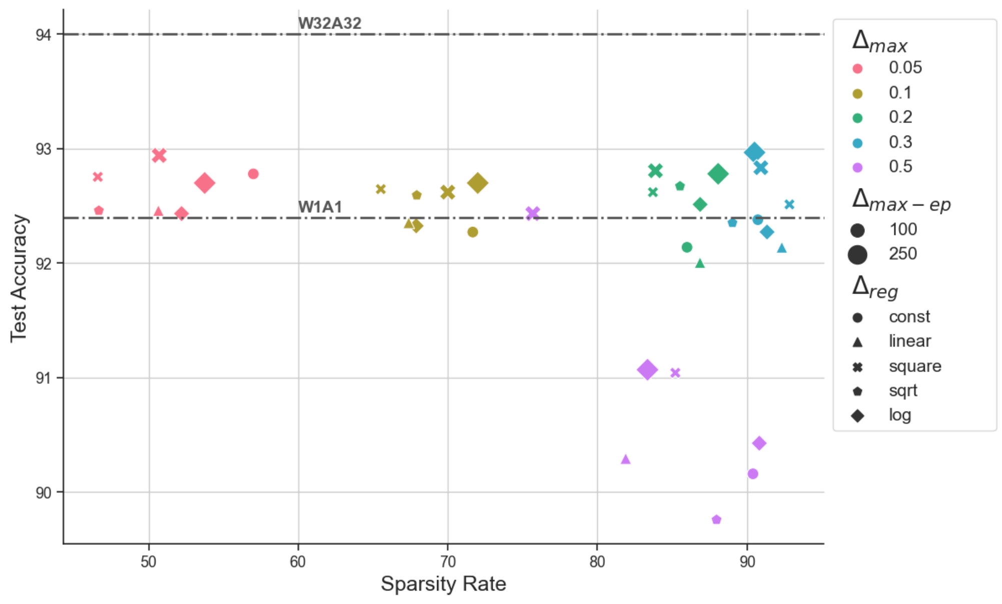

Figure 4.3: Test accuracy against sparsity rates. The charts collect performance on the WHUGAIT dataset [15] for different $\Delta_{\max}$ (reached either at the 100th or 250th training epoch, $t_{\max}$) and Δ growth regimes. The graphs also indicate the full-precision (W32A32) and binary (W1A1) baselines (dotted lines). The best performance is located in the top-right corner, corresponding to high classification accuracy and high sparsity rates.

Figure 4.4 illustrates classification accuracy and entropy achieved by every growth regime tested. It is essentially equivalent to Figure 4.3 with the distinction of a mirrored x-axis scale, due to the inverse monotonic relationship between entropy and sparsity rate.

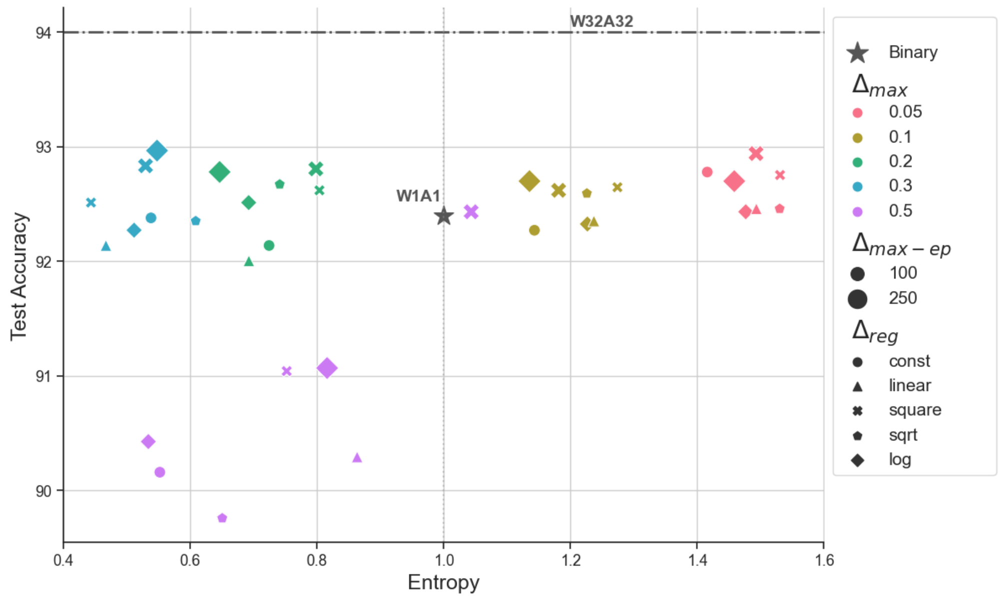

Figure 4.4: Test accuracy against entropy rates. The charts collect performance on the whuGAIT dataset [15] for different $\Delta_{\max}$ (reached either at the 100th or 250th training epoch, $t_{\max}$) and Δ growth regimes. The plot also indicate the full-precision (W32A32) and binary (W1A1) baselines (dotted lines and star, respectively). The best performance is located in the top-left corner, corresponding to high classification accuracy and low entropy. Figure 4.3 is essentially an x-axis mirrored version this plot.

4.3 Authentication Task

For the authentication task, the dataset triplets are constructed at each epoch by associating each input gait with a randomly sampled positive gait, belonging to the same subject, and a negative gait, belonging to a different subject. This process is repeated for both the training and test splits. All models are trained using the Adam optimizer, regardless of the quantization scheme. Each model is trained for a maximum of 250 epochs, compared to 500 epochs for the identification task, as early experiments showed that triplet loss training leads to much faster convergence. We also apply an early stopping criterion to prevent overfitting. Specifically, at every epoch, we evaluate the model's performance by calculating the percentage of triplets in the validation set where the anchor-positive distance is smaller than the anchor-negative distance. For each run, we select the model snapshot from the epoch that produced the highest value for this metric.

4.3.1 Hyper-Parameter Optimization

Similar to the identification task, we perform a grid search on the learning rate η and batch size B. However, for this task, we also include the distance function d and the margin value used in the triplet loss as additional hyperparameters. The ROC AUC score is used as the primary metric for selecting the best values, as it is the most representative of the model's performance on the authentication task, as described in Section 3.5.3. For the ternary model, we use the best-performing delta growth Δ we found for the identification task (*log* profile, $\Delta_{\max} = 0.3$), adjusting $t_{\max}$ to 125 to ensure that the regime still reaches its peak at the halfway point of training, despite the reduced number of epochs (from 500 to 250). Early experiments show that this specific growth regime performs comparably well on the authentication task; consequently, an extensive search for alternative growth regimes was not conducted. This suggests that once an effective regime is identified, it may generalize well across tasks, reducing the need for task-specific fine-tuning.

Table 4.3 presents the values used in the grid search for learning rate, batch size, and margin. Since the distance between two embeddings varies significantly depending on the specific distance function used, we allow for a separate best margin for each function (EUC, COS, HAMM).

4.3.2 Results

Table 4.4 presents the results for the authentication task. Each column contains the evaluation metrics for the models, shown separately for each combination of quantization scheme and distance function used in the triplet loss. This separation allows for a detailed analysis of how each distance function affects the model's ability

Hyper-Parameter	**Values**	**Full Precision**	**Binary**	**Ternary**
Learning rate	0.001, 0.01	**0.001**	**0.01**	**0.01**
Batch size	128, 256	**256**	**128**	**256**
Margin (EUC)	1.5, 5.0, 10.0	**10.0**	**1.5**	**1.5**
Margin (COS)	0.001, 0.004, 0.008, 0.01, 0.04, 0.08, 0.1	**0.2**	**0.01**	**0.01**
Margin (HAMM)	0.001, 0.004, 0.008, 0.01, 0.04, 0.08, 0.1	**-**	**0.08**	**0.04**

Table 4.3: Hyper-Parameter search grid for the authentication task. Best values are shown separately for the full precision, binary, and ternary quantization methods. The margin hyper-parameter has different best values depending on the distance function used in the triplet loss.

to learn meaningful embeddings under different quantization schemes. The models in the first row, denoted as Cross-Entropy, are obtained by selecting the best-performing models from the identification task and discarding the final linear layer used for classification. Then, the flattened output of the last convolutional block is used as the embedding for a given input sample. This setup allows us to evaluate the quality of embeddings produced by models trained with authentication-specific loss function (i.e. triplet loss) against a baseline model that uses a general-purpose loss (i.e. cross-entropy loss).

The metrics adopted are divided into three column groups:

- **ROC Curve, Area Under the Curve:** This metric represents the ease of finding an optimal threshold value d^* to determine whether two embeddings, derived from different samples, belong to the same subject. It is calculated using the pairwise distances between embeddings in the test set. Notably, all three distance functions are applied (not just the one used during triplet loss training) to evaluate the model's robustness to different distance functions.

- **Classification:** A downstream task is used to evaluate the separability of the learned embeddings. After training, each model is used to generate embeddings for both the training and test sets. These embeddings are then treated as a new dataset, on which a Support Vector Machine (SVM) classifier is trained to assess the quality and separability of the embeddings.

- **Clustering:** A downstream task is used to evaluate the robustness of the learned embeddings. After training, each model is used to generate embeddings for both the training and test sets, without using the labels. These embeddings are then treated as a new dataset, on which clustering algorithms such as K-Means are applied. The resulting clusters are analyzed to assess how well the embeddings naturally group samples from the same subject, indicating the quality and cohesion of the embeddings.

Training Method	**Quantization**	**ROC AUC**			**CLASSIFICATION**	**CLUSTERING**	
		EUC	**COS**	**HAMM**	**Accuracy**	**Silhouette**	**Adj. Rand**
Cross-Entropy	Full	94.4	96.5	-	94.2	0.176	0.580
	Binary	97.7	97.7	97.5	93.2	0.144	0.791
	Ternary	96.7	96.5	96.0	92.6	0.176	0.782
Triplet Loss (EUC)	Full	99.5	98.5	-	94.3	0.274	0.890
	Binary	99.1	98.8	98.9	93.5	0.146	0.819
	Ternary	99.3	99.2	98.9	93.6	0.167	0.825
Triplet Loss (COS)	Full	93.9	99.2	-	94.1	0.268	0.788
	Binary	99.1	99.1	99.0	93.3	0.140	0.814
	Ternary	99.3	99.4	98.3	93.3	0.172	0.812
Triplet Loss (HAMM)	Full	-	-	-	-	-	-
	Binary	98.8	98.8	98.7	93.1	0.141	0.830
	Ternary	98.9	98.9	98.8	93.0	0.172	0.833

Table 4.4: Performance metrics for the authentication task. ROC AUC column reports values for ROC curves where pairwise distances are computed with different distance functions. Classification and Clustering columns report metrics used to evaluate the embeddings on the two downstream tasks. Each row corresponds to a model trained with a specific training method and quantization scheme.

ROC Curve, Area Under the Curve When comparing models trained using triplet loss to those trained with cross-entropy, it is evident that triplet loss consistently leads to higher ROC AUC scores across all quantization schemes. For example, the full precision model trained with triplet loss using the Euclidean distance achieves an ROC AUC of 99.5 (compared to 94.4 for cross-entropy), demonstrating the superiority of triplet loss in learning discriminative embeddings for authentication tasks. Similarly, both the binary and ternary models exhibit better performance with triplet loss across all distance functions.

Notably, for models trained with triplet loss, there is a clear hierarchy in performance: the full precision models outperform the ternary models, which in turn outperform the binary models. This pattern is consistent across virtually all distance functions. For instance, in the case of Euclidean distance, the full precision model achieves a ROC AUC of 99.5, followed by the ternary model at 99.3, and the binary model at 99.1. In contrast, this hierarchy is not observed in models trained with cross-entropy, where the full precision model achieves lower results than the quantized ones. This can be explained by the fact that the model is not trained for this specific task, but for multi-class classification problems. The models achieve optimal performance when the distance function used during training matches the one applied in the ROC AUC calculation. For example, the full-precision model trained with Euclidean distance achieves its highest ROC AUC score (99.5) when evaluated with Euclidean distance. This pattern holds across various distance functions, indicating that alignment between training and evaluation distances is critical. Interestingly, as long as this alignment is maintained, the choice of distance function—whether Euclidean, cosine, or Hamming—has minimal impact on performance. This suggests that the embeddings learned by the models are robust and generalize well across different distance metrics when trained with a consistent configuration.

Lastly, the ternary models consistently outperform the binary baselines in terms of ROC AUC, regardless of the distance function or quantization method. Whether using cross-entropy or triplet loss, ternary models show better results across the board. This highlights the effectiveness of ternary quantization in preserving discriminative power while offering higher compression rates compared to binary models.

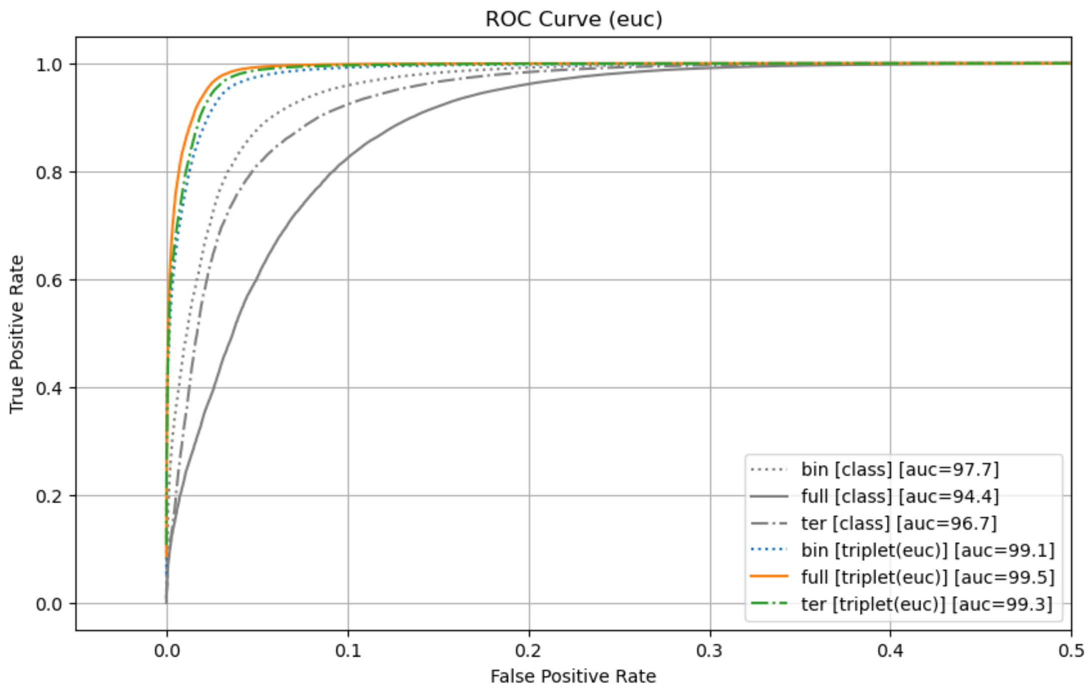

Figure 4.5: ROC curves calculated using Euclidean distance for models trained with triplet loss and cross-entropy loss, shown separately for each quantization scheme.

Figure 4.5 shows how the ROC curves of the models trained with cross-entropy are significantly worse than those trained with the triplet loss. Additionally, the previously mentioned hierarchy among quantization schemes is also evident.

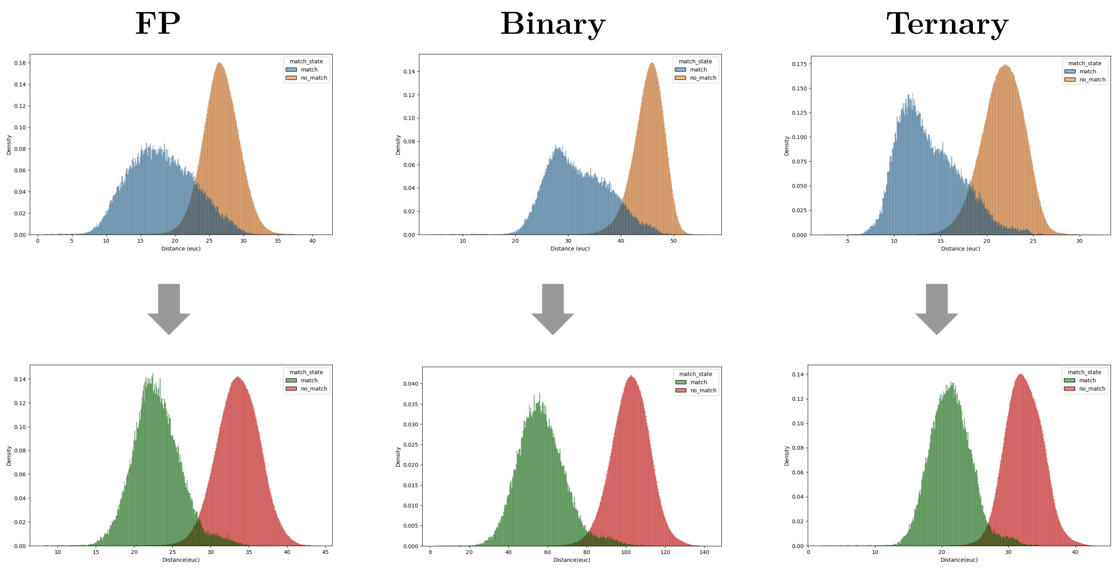

Figure 4.6: Distributions of distances shown separately for matching and non-matching test samples. The top row represents cross-entropy loss training, while the bottom row represents triplet loss training. Matching pairs are shown in blue (top) and green (bottom), while non-matching pairs are shown in orange (top) and red (bottom).

Figure 4.6 shows how the pairwise distance distributions between same-subject and different-subject samples change when using triplet loss. The top row illustrates the distributions for full precision (FP), binary, and ternary models trained with cross-entropy. There is significant overlap between same-subject (blue) and different-subject (orange) distances, indicating poor separability of embeddings. In contrast, the bottom row shows the results after training with triplet loss. For all models, the distributions for same-subject (green) and different-subject (red) samples are much more distinct, with minimal overlap. This demonstrates that triplet loss greatly improves the separability of embeddings across all quantization schemes.

Notably, even in the binary and ternary models, the embeddings retain strong separability after training with triplet loss, showing that it helps maintain robust embedding quality despite quantization.

Classification and Clustering In the classification downstream task, as expected, the full precision model achieves the best performance, similar to when the final linear layer is used directly for classification. Notably, the binary and ternary models show comparable performance to each other, with both models achieving a significant improvement in classification accuracy when an SVM, with full precision parameters, is applied to the learned embeddings instead of using the final quantized layer for end-to-end classification. This suggests that the majority of performance degradation could occur in the final fully connected layer, rather than in the convolutional section of the network. This hypothesis is further explored in a later paragraph.

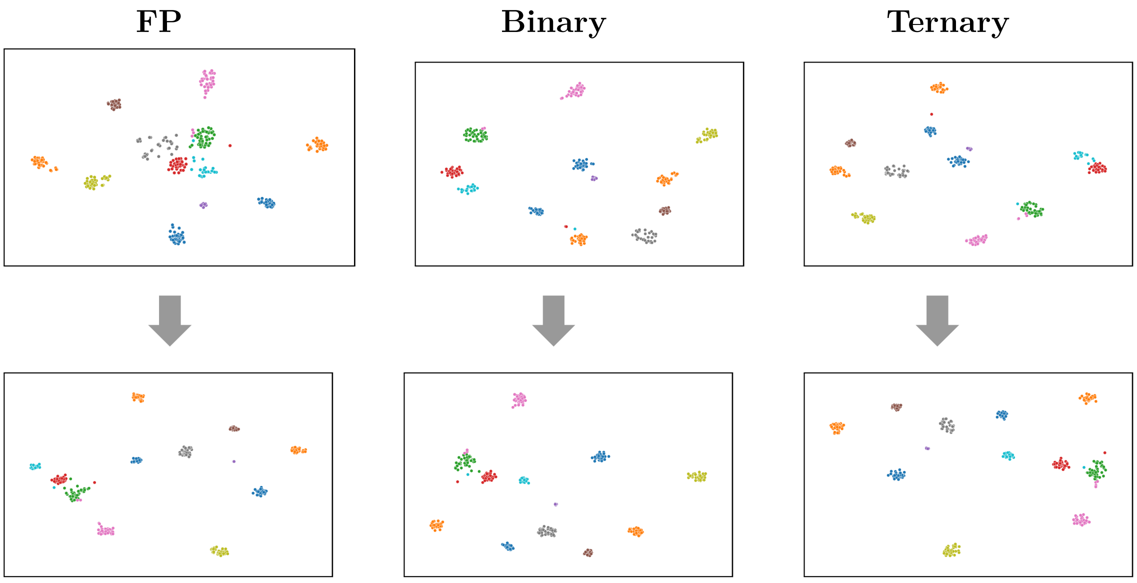

Figure 4.7: T-SNE visualization of test set embeddings for the first 12 subjects. Cross-entropy trained models are shown in the top row, while triplet loss models are shown in the bottom one.

Figure 4.7 presents t-SNE visualizations of the embeddings for a subset of the first 10 subjects from the dataset, before and after training with triplet loss. In the top row, the models are trained with cross-entropy loss, while in the bottom row, the models are trained with triplet loss. A modest improvement in separation is noticeable, particularly for the full precision model, which shows the most significant increase in separation as it performed the worst when trained with cross-entropy.

In the clustering downstream task, models trained with cross-entropy show relatively lower performance, with fewer distinct clusters forming across all quantization schemes. On the other hand, triplet loss significantly enhances clustering performance, leading to better-defined clusters that more accurately correspond to individual subjects. Notably, the Euclidean distance appears to perform better for the full precision model, while the quantized models (binary and ternary) seem less sensitive to the choice of distance function.

Effect of Quantization on the Final Linear Layer We observed that, for the quantized models, training an SVM on the embeddings yields significantly better classification accuracy compared to using a quantized fully connected layer. To examine the extent to which quantizing the final layer degrades performance, we repeat the training for the identification task (i.e., classification), this time using a full-precision linear layer instead of a quantized one.

In Table 4.5, we observe that both the binary and ternary models improved by 0.6 percentage points in classification accuracy, with the ternary model achieving 93.6%, only 0.6 percentage points below the full precision (FP) baseline. Therefore, most of the performance degradation in quantized models seems to stem from the quantization of the final fully connected layer, rather than from the convolutional layers, and using a full precision linear layer when applicable can significantly mitigate this issue.

Training Method	**Quantization**	**Accuracy**
Cross Entropy	Full	94.2
	Binary	92.4
	Binary [full-fc]	93.0 (+0.6)
	Ternary	93.0
	Ternary [full-fc]	93.6 (+0.6)

Table 4.5: Classification results for models trained with cross-entropy when the last fully connected layer is in full precision. Binary[full-fc] and Ternary[full-fc] rows report the delta accuracy compared to their counterparts with a quantized fully connected layer.

4.3.3 Open Set Results

The results in the previous section 4 assume that the problem is a closed-set one, meaning that all classes or subjects present during testing are already known and included in the training set. This makes the task relatively easy, as the model only needs to differentiate between subjects it has already encountered. In contrast, the open set problem is significantly more challenging, as it requires the model to generalize to previously unseen subjects during testing. In this scenario, the model must learn embeddings that not only effectively separate known subjects but can also generalize to new, unseen individuals. Given the relevance of this scenario for real-world authentication tasks, we provide the same results as in the previous section, but for the open set case.

To construct the open set dataset, we merge the original training and test sets of the WhuGait dataset and then re-split the samples so that the training set contains only a subset of the subjects, while the test set consists entirely of subjects not seen during training. Specifically, we hold out 12 of the 118 subjects for the test set, maintaining the original 90/10 split of the dataset. This allows for the creation of 10 possible splits (keeping the subjects in order), which can be averaged to obtain more stable results by smoothing out potential variations in performance due to easier or harder splits.

Training Method	**Quantization**	**ROC AUC**						**CLASS**	**CLUSTERING**			
		EUC	$\Delta_{\text{os-cs}}$	**COS**	$\Delta_{\text{os-cs}}$	**HAMM**	$\Delta_{\text{os-cs}}$	**Accuracy**	**Silh.**	$\Delta_{\text{os-cs}}$	**Adj. Rand**	$\Delta_{\text{os-cs}}$
Classification	Full	94.4	-	96.5	-	-	-	94.2	0.176	-	0.580	-
	Binary	97.7	-	97.7	-	97.5	-	93.2	0.144	-	0.791	-
	Ternary	96.7	-	96.5	-	97.5	-	92.6	0.176	-	0.782	-
Triplet Loss (EUC)	Full	98.3	-1.2	97.4	-1.1	-	-	-	0.264	-3.6%	0.919	+3.3%
	Binary	97.9	-1.2	97.8	-1.0	97.6	-1.3	-	0.218	+49.3%	0.833	+1.7%
	Ternary	98.1	-1.2	98.0	-1.2	97.5	-1.4	-	0.268	+60.5%	0.887	+7.5%
Triplet Loss (COS)	Full	91.0	-2.9	98.2	-1.0	-	-	-	0.219	-18.3%	0.810	+2.8%
	Binary	98.1	-1.0	98.1	-1.0	97.9	-1.1	-	0.168	+20.0%	0.891	+4.3%
	Ternary	98.2	-1.1	98.3	-1.1	97.9	-0.4	-	0.197	+14.5%	0.893	+10.0%
Triplet Loss (HAMM)	Full	-	-	-	-	-	-	-	-	-	-	-
	Binary	98.1	-0.7	98.0	-0.8	98.0	-0.7	-	0.183	+29.8%	0.872	+5.1%
	Ternary	98.3	-0.6	98.3	-0.5	98.3	-0.5	-	0.209	+21.5%	0.864	+3.7%

Table 4.6: Performance metrics for the open set authentication task. Same metrics as in Table 4.4 with deltas $\Delta_{\text{os-cs}}$ representing the difference between open-set (os) and closed-set (cs) results.

Table 4.6 presents the results for the open set authentication, along with the performance delta compared to the closed set case for each metric. The downstream classification with SVM is omitted, as it is nonsensical due to the different classes/subjects in the training and test sets. We can observe that ROC AUC scores consistently decrease by around 1.0 percentage points across all quantization strategies and distance functions, except for Hamming, where the loss is limited to approximately 0.6 percentage points. This demonstrates the models' ability to generalize to the open set case, with quantization not significantly impacting their transferability to open set scenarios. In the downstream clustering task, performance is significantly improved because the clustering only involves 12 classes, rather than 118, making it a less challenging problem. Figure 4.8 extends Figure 4.5 by adding the ROC curves for the models trained with the open set dataset. Notably, despite the slight performance drop, the open-set models trained with triplet loss still significantly outperform those trained with cross-entropy on the closed-set. This highlights the effectiveness of triplet loss in producing embeddings that generalize well, even for unseen subjects.

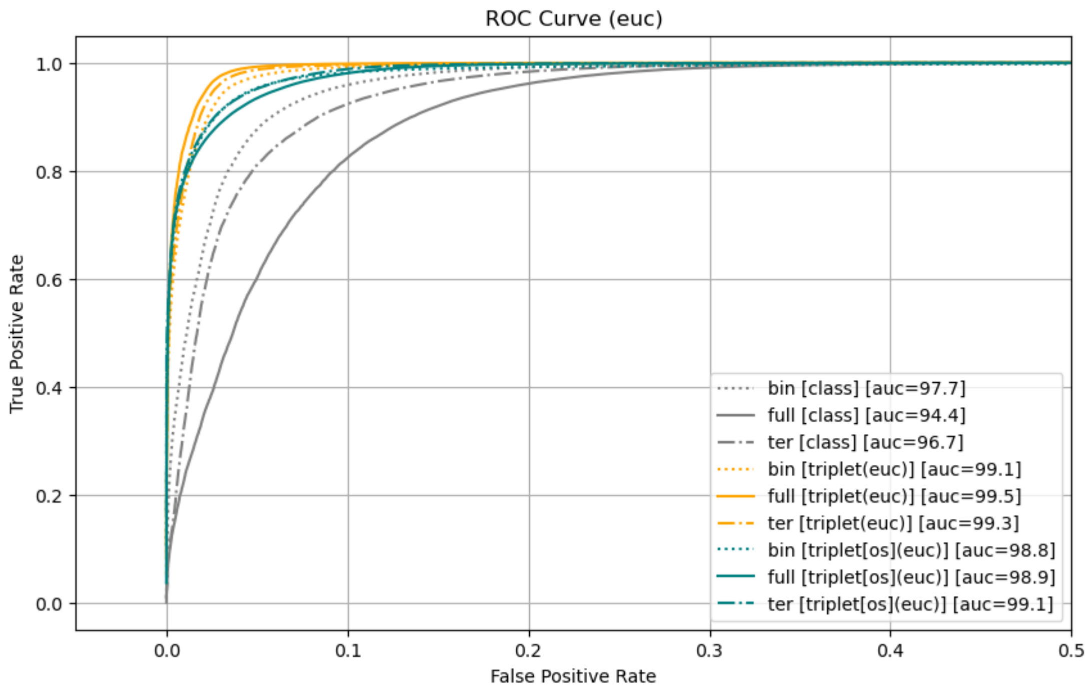

Figure 4.8: ROC curves calculated using Euclidean distance for models trained with triplet loss and cross-entropy loss, shown separately for each quantization scheme. Extension of Figure 4.5 with the addition of the ROC curves for the models evaluated on the open set scenario (green curve).

Chapter 5

Conclusions

This thesis has explored the challenges and solutions for deploying deep neural networks (DNNs) on resource-constrained wearable devices, particularly for gait-based biometric recognition. Despite the advancements in artificial intelligence and the promise of deep learning, the high computational demands of DNNs make real-time deployment on devices like smartwatches and smartphones challenging due to their limited power and memory. This work has addressed these challenges by focusing on model compression techniques, specifically quantization, to enable efficient DNN deployment without compromising performance.

Our study has demonstrated that biometric systems leveraging physiological and behavioral data, such as gait patterns, can effectively provide secure authentication and identification. Gait analysis was chosen for its non-intrusive nature, making it suitable for wearables, which offer practical platforms for capturing gait data through widely available IMU sensors. By utilizing neural network-based models and quantization techniques, this thesis has shown the potential to enhance both the accuracy and efficiency of gait-based biometric systems in resource-limited settings.

Central to our approach is the development of Ternary Neural Networks (TNNs), which combine quantization with parameter pruning, achieving high sparsity and reducing the computational load. Our ternarization framework dynamically adjusts quantization during training, reaching sparsity rates often exceeding 90% with an entropy level below 1 bit per symbol. This approach has proven effective in creating models that are both highly compressible and high-performing, surpassing traditional binary networks in memory efficiency and maintaining robust discriminative power. In evaluating TNNs on identification and authentication tasks, the results confirm that ternary quantization can significantly reduce memory and energy requirements while preserving the accuracy needed for gait-based biometric recognition. By leveraging smartphone-based IMU data, this thesis has shown the practicality of implementing gait recognition on widely available hardware, making the approach viable for real-world applications in secure authentication and identification.

In conclusion, this work provides a framework for achieving efficient and accurate biometric systems deployable on resource-constrained devices. Future research can extend these findings by exploring additional quantization methods, such as adaptive or mixed precision quantization, which could further balance the trade-off between efficiency and accuracy. Additionally, optimizing the training process by incorporating techniques such as knowledge distillation or reinforcement learning could improve the robustness and adaptability of quantized models. Expanding the framework to include other biometric modalities—such as voice, face, or keystroke dynamics—could broaden its applicability to multi-modal biometric systems, enhancing security and reliability. Finally, testing this approach in several real-world conditions and across different hardware platforms would offer valuable insights for further refining resource-efficient biometric recognition in dynamic, low-power environments.

Bibliography

[1] Warren S. McCulloch and Walter Pitts. «A Logical Calculus of the Ideas Immanent in Nervous Activity». In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133 (cit. on p. 5).

[2] Frank Rosenblatt. *The Perceptron: A perceiving and recognizing automaton.* Tech. rep. 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory, Jan. 1957 (cit. on p. 5).

[3] John J. Hopfield. «Neural networks and physical systems with emergent collective computational abilities». In: *Proceedings of the National Academy of Sciences.* Vol. 79. 8. National Academy of Sciences, 1982, pp. 2554–2558. DOI: 10.1073/pnas.79.8.2554 (cit. on p. 6).

[4] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. «Learning representations by back-propagating errors». In: *Nature* 323 (1986), pp. 533–536. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0 (cit. on p. 6).

[5] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Robert E. Howard, Wayne Hubbard, and Lawrence D. Jackel. «Handwritten Digit Recognition with a Back-Propagation Network». In: *Advances in Neural Information Processing Systems.* Vol. 1. Neural Information Processing Systems Foundation, 1989, pp. 396–404 (cit. on p. 7).

[6] Anil K. Jain, Arun A. Ross, and Karthik Nandakumar. *Introduction to Biometrics.* Springer New York, 2011. ISBN: 978-0-387-77325-4. DOI: 10.1007/978-0-387-77326-1 (cit. on p. 18).

[7] Chao Zhong, Xue Mu, Xin He, Jin Wang, and Ming Zhu. «Model Compression for Deep Neural Networks: A Survey». In: *Computers* 12.3 (2023), p. 60. DOI: 10.3390/computers12030060. URL: https://www.mdpi.com/2073-431X/12/3/60 (cit. on p. 21).

[8] Yann LeCun, John S. Denker, and Sara A. Solla. «Optimal Brain Damage». In: *Neural Information Processing Systems.* 1989. URL: https://api.semanticscholar.org/CorpusID:7785881 (cit. on p. 21).

[9] S. Han, H. Mao, and W. J. Dally. «Learning both Weights and Connections for Efficient Neural Networks». In: *Advances in Neural Information Processing Systems*. 2015 (cit. on p. 21).

[10] Y. Yang, H. Qin, R. Gong, and X. Liu. «Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on p. 21).

[11] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. «Pruning Filters for Efficient ConvNets». In: *International Conference on Learning Representations (ICLR)*. 2017 (cit. on p. 22).

[12] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. «ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression». In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 5058–5066 (cit. on p. 22).

[13] Yihui He, Xiangyu Zhang, and Jian Sun. «Channel pruning for accelerating very deep neural networks». In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 1389–1397 (cit. on p. 22).

[14] Yoshua Bengio, Nicolas Léonard, and Aaron C. Courville. «Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation». In: *arXiv preprint arXiv:1308.3432* (2013). URL: `https://arxiv.org/abs/1308.3432` (cit. on pp. 23, 33).

[15] Q. Zou, Y. Wang, Q. Wang, Y. Zhao, and Q. Li. «Deep Learning-Based Gait Recognition Using Smartphones in the Wild». In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3197–3212 (cit. on pp. 27–30, 53, 54).

[16] Chunyu Yuan and Sos S. Agaian. «A comprehensive review of Binary Neural Network». In: *Artificial Intelligence Review* (2021), pp. 1–65. URL: `https://api.semanticscholar.org/CorpusID:238743860` (cit. on pp. 31, 35).

[17] F. J. Ordoñez and D. Roggen. «Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition». In: *Sensors* 16.1 (2016), p. 115 (cit. on p. 32).

[18] Geoffrey Hinton. *Neural Networks for Machine Learning*. Coursera, video lectures. 2012. URL: `https://www.coursera.org/learn/neural-networks-ml` (cit. on p. 33).

[19] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. «Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1». In: *arXiv preprint arXiv:1602.02830* (2016) (cit. on p. 33).

[20] Adam Paszke et al. «PyTorch: An Imperative Style, High-Performance Deep Learning Library». In: *arXiv preprint arXiv:1912.01703* (2019) (cit. on p. 48).

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification». In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV).* 2015, pp. 1026–1034 (cit. on p. 49).

[22] Q. Zou, Y. Wang, Q. Wang, Y. Zhao, and Q. Li. «Deep learning-based gait recognition using smartphones in the wild». In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3197–3212. DOI: `10.1109/TIFS.2020.2964705` (cit. on p. 51).

[23] M. Gadaleta and M. Rossi. «Idnet: Smartphone-based gait recognition with convolutional neural networks». In: *Pattern Recognition* 74 (2018), pp. 25–37. DOI: `10.1016/j.patcog.2017.10.019` (cit. on p. 51).

[24] F. J. Ordoñez and D. Roggen. «Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition». In: *Sensors* 16.1 (2016), p. 115. DOI: `10.3390/s16010115`. URL: `https://doi.org/10.3390/s16010115` (cit. on p. 51).

[25] S. Shen, S.-S. Sun, W.-J. Li, R.-C. Wang, P. Sun, S. Wang, and X.-Y. Geng. «A classifier based on multiple feature extraction blocks for gait authentication using smartphone sensors». In: *Computers and Electrical Engineering* 108 (2023), p. 108663. DOI: `10.1016/j.compeleceng.2023.108663` (cit. on p. 51).

[26] N. Rifaat, U. K. Ghosh, and A. Sayeed. «Accurate gait recognition with inertial sensors using a new FCN-BiLSTM architecture». In: *Computers and Electrical Engineering* 104 (2022), p. 108428. DOI: `10.1016/j.compeleceng.2022.108428` (cit. on p. 51).