TÉCNICO LISBOA

Privacy in Paralinguistic Tasks

Francisco Saraiva Sepúlveda Teixeira

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Isabel Maria Martins Trancoso
Prof. Alberto Abad Gareta

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Isabel Maria Martins Trancoso
Members of the Committee: Prof. Ricardo Jorge Fernando Chaves

September 2018

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to start by expressing my deepest gratitude to my supervisor, Professor Isabel Trancoso, for all her continuous guidance, encouragement, and all the opportunities given to me, without which this work would not have been possible. I would also like to express my deepest thanks to my co-supervisor, Professor Alberto Abad, for all the scientific advice, constant support and availability.

I would like to extend a special thanks to Professor Bhiksha Raj, for our insightful discussions that developed into the 5^{th} Chapter of this Thesis, and to Joana Correia, for all the help and advice throughout this work.

To my friend Miguel Dias, for all the discussions that pushed me towards the topic of this Thesis, and for all the help and support in the beginning of this work.

To all my friends, for being with me throughout this journey.

To my parents, for all their unconditional support, for always pushing me towards my goals and for showing me what science is. To my sister, for always being there, and for showing me that science is not the only way to view the world.

Last, but not least, to Mariana, for all her love, patience and encouragement.

Abstract

The widespread use of devices with internet access, together with the emerging market for data mining applications has raised concerns over the level of privacy currently given to users. Taking advantage of increasingly accurate Machine Learning algorithms, many services use sensitive data to extract information and make predictions about the characteristics of their users. Among other data types, speech stands out for the amount of information it holds. Aside from the linguistic content, from speech one can obtain paralinguistic information, such as the speaker's age, gender, health and personality traits. However, the reasons that make speech useful also make it a target for malicious third parties intending to obtain sensitive information about unsuspecting users. This is especially true for health-related applications where a system may try to uncover whether someone presents symptoms of a medical condition, as this information is deeply sensitive.

In this thesis we show how Homomorphic Encryption can be used to build speech-based privacy-preserving Neural Networks, with focus on three speech affecting conditions: the common Cold, Depression and Parkinson's Disease. To this end, in a first experiment we apply an Encrypted Neural Network, whose operations have been replaced with their encrypted counterparts, to the three aforementioned conditions. On a second approach, we discuss and experiment on the feasibility of building an end-to-end network for the same purpose. Finally, as a last experiment we show how a Neural Network, and its input features, can be discretized in order to allow the use of a Homomorphic Encryption batching technique.

Keywords

Pathological Speech; Paralinguistics; Privacy; Homomorphic Encryption; Machine Learning;

Resumo

O uso generalizado de dispositivos com acesso à internet, em conjunto com o mercado emergente de aplicações de exploração de dados, tem dado origem a preocupações relativas à privacidade dos seus utilizadores. Aproveitando a qualidade dos algoritmos de aprendizagem automática, muitos serviços utilizam dados sensíveis para extrair informações sobre os seus utilizadores. Quando comparada com outros tipos de dados, a fala destaca-se pela quantidade de informação que contém. Para além do conteúdo linguístico, a partir da fala é possível obter conteúdo paralinguístico, como a idade, género, estado de saúde e traços de personalidade do orador. No entanto, estas características também transformam a fala num alvo para entidades mal intencionadas, que pretendem obter informações sensíveis de utilizadores desprotegidos. Isto é especialmente verdade em aplicações relacionadas com saúde, nas quais um sistema tenta descobrir se um utilizador apresenta ou não sintomas de uma doença, uma vez que esta informação é extremamente sensível.

Nesta tese mostramos como a Encriptação Homomórfica pode ser utilizada para construir Redes Neuronais (RN) baseadas em fala, mantendo a privacidade dos dados, dando especial ênfase a três doenças que afectam a fala: Constipação, Depressão e Doença de Parkinson. Para isto, numa primeira experiência, aplicamos às doenças referidas anteriormente uma RN Encriptada, cujas operações foram substituídas pelos seus equivalentes de Encriptação Homomórfica. De seguida, experimentamos a viabilidade de construir uma rede *end-to-end* para o mesmo fim. Por fim, mostramos como é possível discretizar uma RN e as suas entradas, de forma a utilizar uma técnica de *batching* com Encriptação Homomórfica.

Palavras Chave

Fala Patológica; Encriptação Homomórfica; Aprendizagem Automática; Privacidade;

Contents

List of Figures

List of Tables

Acronyms

ANN	Artificial Neural Network
AP	Average Pooling Layer
ASR	Automatic Speech Recognition
BCE	Binary Cross-Entropy
BN	Batch Normalization Layer
BP	Backpropagation
BTT	Backpropagation Through Time
CNN	Convolutional Neural Network
ECNN	Encrypted Convolutional Neural Network
CLDNN	Convolutional Long Short-Term Memory Deep Neural Network
DiNN	Discretized Neural Network
DNN	Deep Neural Network
ENN	Encrypted Neural Network
FC	Fully Connected Layer
FBEs	Log-Mel-Filterbank Energies
FHE	Fully Homomorphic Encryption
FV	Fan-Vercauteren
GC	Garbled Circuit
GMM	Gaussian Mixture Model

HE Homomorphic Encryption

LHE Leveled Homomorphic Encryption

LLD Low-level Descriptor

LV Language Verification

LSH Locality-Sensitive Hashing

LSTM Long Short-Term Memory Layer

LWE Learning with Errors

MAE Mean Average Error

MFCCs Mel Frequency Cepstral Coefficients

MSE Mean Squared Error

ML Machine Learning

MLaaS Machine Learning as a Service

NN Neural Network

OT Oblivious Transfer

PHE Partially Homomorphic Encryption

PD Parkinson's Disease

PLPs Perceptual Linear Predictions

PPML Privacy-preserving Machine Learning

QNN Quantized Neural Network

ReLU Rectified Linear Unit

RLWE Ring Learning With Errors

RMSE Root-Mean-Square Error

RNN Recurrent Neural Network

SaaS Software as a Service

SBE Secure Binary Embedding

SDC Shifted-Delta-Cepstral Coefficients

SEAL Simple Encrypted Arithmetic Library

SFE Secure Function Evaluation

SMLaaS Secure Machine Learning as a Service

SMH Secure Modular Hashing

SMPC Secure Multi-Party Computation

STPC Secure Two-Party Computation

SVM Support Vector Machine

SWHE Somewhat Homomorphic Encryption

UAR Unweighted Average Recall

UBM Universal Background Model

1

Introduction

Contents

In this chapter we present a brief introduction on the topics covered by this thesis (Section 1.1), the motivation behind them and the main goals we expect to achieve (Section 1.2). In addition, in Section 1.3 we provide an outline of the structure of this thesis.

1.1 Motivation

In this work we intent to explore how Privacy-preserving Machine Learning (PPML) frameworks can be applied to health-related speech mining tasks. In this section we will give a brief introduction on the topics of Speech Pattern Recognition and Secure Machine Learning as a Service (SMLaaS), discuss the importance of data privacy in Machine Learning (ML), and in particular the importance of privacy in speech-based ML applications.

1.1.1 Speech Pattern Recognition

Speech is one of the primary means of communication for humans. It can be viewed as a carrier for information on several levels as it conveys not only the meaning and intention predetermined by the speaker, but also information about the age, gender, personality, emotional state, affect and health of the speaker. This information can be divided into two separate categories, *Linguistic* and *Paralinguistic* [80]:

- *Linguistic* information is related to communication and what the speaker intends to relay onto the listener through language.
- *Paralinguistic* information is transmitted in the form of non-linguistic and non-verbal communication, through which the speaker can convey to the listener an attitude, feeling or emotional state. In addition, speech also includes non-communicative information such as the speaker's age, gender, personality and health status. In some works this information is classified as *Extralinguistic*, however, in this thesis we follow the definition used in [80], that categorizes this information as *Paralinguistic*.

The amount of information conveyed in speech makes it the scope of a large number of computational applications that aim to automate tasks such as speech perception (Automatic Speech Recognition), analysis (Speech Pattern Recognition) and synthesis (Text-to-Speech). These tasks have numerous applications, from dialogue-based automated systems, to user authentication, data retrieval as well as educational and medical applications such as the detection, assessment, monitoring and treatment of speech-affecting conditions [82].

However, the reasons that make speech useful for technological applications also make it an uniquely sensitive biometric that should be kept private. It is important to secure not only the user's voice, but also

the information obtained from applying pattern recognition systems. This is specially true for applications in which speech is utilized to determine whether or not a person presents symptoms of a medical condition, as this information is deeply sensitive and personal.

A lot of effort has been put into the development of applications to extract *paralinguistic* information from speech [82]. However, the topic of privacy in speech-based, *paralinguistic* ML frameworks still remains largely unexplored, notwithstanding the important role these applications may play in detecting and monitoring certain health conditions.

1.1.2 Secure Machine Learning as a Service

The growing popularity of devices with internet access has lead to a widespread development of Software as a Service (SaaS) applications. In turn, this has granted researchers and developers access to very large collections of data, which together with recent advances in Machine Learning (ML), has allowed the development of highly accurate ML models. This has induced the creation of a large number of internet based Machine Learning as a Service (MLaaS) applications that employ pattern recognition algorithms as standalone services, and as fundamental parts of larger systems [83].

As the number and reach of internet based application grows, so does the concern on user privacy, as the recently approved European Union General Data Protection Regulation has come to show. MLaaS applications have a particular and privileged access to private information, as most require data regarding the user's preferences, personality traits and even biometric data, such as speech, fingerprints, as well as images and video-recordings of the user's face and physical appearance. However, in most cases, no privacy guarantee is given to the users, for either their data or the information obtained by processing it. This absence of privacy-preserving solutions for MLaaS applications has resulted in an increasingly active field of research that aims to use Secure Multi-Party Computation (SMPC) techniques to develop Privacy-preserving Machine Learning (PPML) solutions for Secure Machine Learning as a Service (SMLaaS) [7].

1.2 Objectives

The topics discussed in the previous section, make clear the importance of developing Privacy-preserving Machine Learning (PPML) frameworks for speech-based applications, with particular emphasis for those that aim to extract health-related *paralinguistic* information from speech. As such, the aim of this thesis is to build on existing PPML frameworks, and to apply them to health-related *paralinguistic* speech tasks.

More specifically we intent to build on the work of [22] and [69], and provide a proof-of-concept on how a Neural Network (NN) combined with Homomorphic Encryption (HE) can provide an accurate

and secure framework for speech-based medical applications. We also intent to explore end-to-end approaches that are compatible with HE, in order to minimize the user's role in the computation, and avoid a feature extraction stage. Finally, we will try to find ways to improve the framework's computational overhead, caused by HE, through the use of SIMD (Single Instruction Multiple Data), HE techniques.

Since our objective is to show how this framework can be applied to health-related tasks, we will focus our experiments on three types of speech-affecting conditions, the common Cold, Depression and Parkinson's Disease (PD):

- *Cold*: Characterized by symptoms such as cough, hoarseness, sore throat, nasal obstruction, sneezing and nasal leakage and stuffiness. Cold and Influenza (or flu) annual epidemics can cause about 3 to 5 million cases of severe illness and from 290,000 to 650,000 deaths, worldwide [42] [87] [62].
- *Depression*: Depression is one of the most prevalent mental disorders, characterized by a persistent low-mood. In 2017 more than 300 million people were living with depression worldwide, representing an increase of more than 18% between 2005 and 2015. It includes several observable symptoms and indicators, that affect facial expressions, demeanor, slowed speech, decreased vocal intensity, reduced interpersonal responsiveness, among others [24] [63].
- *Parkinson's Disease (PD)*: PD is a neurodegenerative disorder that affects 1% of people over the age of 65, 89% of which develop speech disorders. Common symptoms of Parkinsonism include bradykinesia (slowness or difficulty to perform movements), muscular rigidity, rest tremor, as well as postural and gait impairment. The symptoms presented in speech can be grouped as *hypokinetic dysarthria*, and include reduced loudness, monopitch, monoloudness, hypotonicity, breathy and hoarse voice quality, as well as imprecise articulation [92] [46].

Considering their prevalence and serious nature, technology that tests for the existence or severity of the tree diseases described above, using speech, has the potential to become a key tool in the early detection, monitoring and prevention of these conditions [36] [21] [65].

1.3 Outline

This thesis is organized as follows: Chapter 1 contains a small introduction on Privacy-preserving Machine Learning (PPML) and Speech Paralinguistics, as well as the motivation and objectives behind this thesis. Chapter 2 details the state-of-the-art methods and techniques used in Privacy-preserving Machine Learning (PPML) and in Speech Pattern Recognition. In Chapter 3 we provide the results obtained for experiments performed on an NN implemented using HE, for datasets relative to Cold, Depression and PD. Chapter 4 provides a discussion on end-to-end techniques and results on experiments

performed with a CNN implemented using HE. In Chapter 5 we detail the required transformations that need to be applied to an NN in order to allow the use of *batching* with HE. Finally in Chapter 6 we present the main conclusions and contributions that can be taken from this work, along with some topics for future work.

2

Background

Contents

In this chapter we provide an overview of the methods and techniques used in Speech Pattern Recognition as well as in Privacy-preserving Machine Learning. In Section 2.1 we discuss features and machine learning algorithms used in speech processing, while in Section 2.2 we detail frameworks used in Privacy-preserving Machine Learning. Finally, in Section 2.3 we discuss the works related to this topic.

2.1 Speech Pattern Recognition

Speech carries a substantial amount of information, both communicative and non-communicative, as stated in Section 1.1.1. It contains information on the speaker's identity, personality and health state, among many other characteristics. As such, a great deal of effort and research has been placed on the development of methods to detect these patterns. Speech is also highly variable in nature, making it a specially difficult and unique problem for pattern recognition algorithms.

Figure 2.1 shows the common chain of processing used in speech pattern recognition tasks. Usually the input signal is subject to some form of pre-processing, such as normalization and de-noising. It then enters the feature extraction block, where a series of algorithms extract a set characteristics, or *features*, that are considered to be relevant to model the pattern in question. Finally, these features are fed into a Machine Learning (ML) algorithm, typically for classification or regression, which outputs a prediction. Machine Learning (ML) algorithms undergo a training phase, using a set of training data and its corresponding *True* predictions, or labels, with which the model's parameters are adjusted. This phase can be followed by a development phase, where the model is tested with a separate data set, to verify its capability of generalizing what it *learned* for other datasets. This phase also includes the fine tuning of the models' hyper-parameters and the model's structure, which are hand picked by the user and are not normally included in the set of trainable parameters. Finally, to assess the "real world" performance of the model, a final evaluation phase can also be done, where the model is tested with an additional set of unseen data.

In the next subsections we will detail common features and state-of-the-art ML algorithms used in speech pattern recognition, with particular focus on those applied to speech paralinguistic tasks.

2.1.1 Speech Features

The first step in most speech ML algorithms is feature extraction. This process is characterized by several stages, through which the original signal is processed, to create representations that facilitate the identification of the characteristics being modelled. In a first stage, the raw signals are subject to a pre-processing stage, with the goal of enhancing speech, removing noise and separating different speakers. In the following stage, windowing functions (such as the Hamming and Hann windows) are

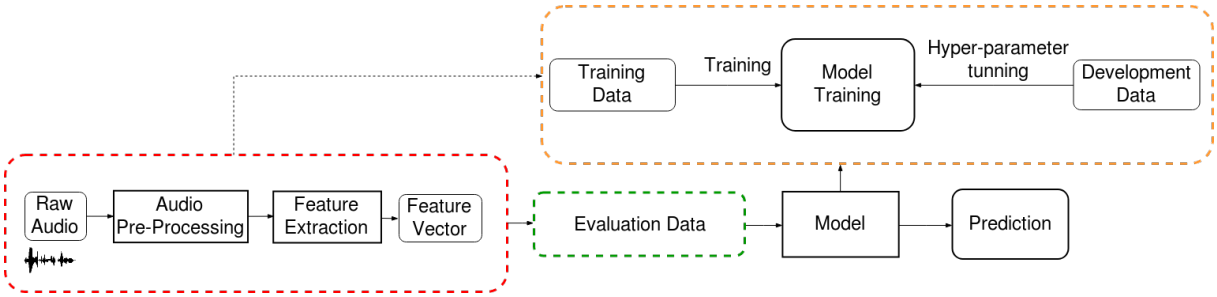


Figure 2.1: Feature Extraction and Machine Learning Model Training

periodically applied along the signal, in order to extract time, frequency or time-frequency domain vectors of Low-level Descriptors (LLDs). To reduce dimensionality, a third stage can be applied, which uses a set of functions that "summarize" the characteristics of each feature present in the vectors. For this purpose a set of functionals are computed from each feature, including the mean, standard deviation, maximum, minimum, skweness and kurtosis, to name a few. Before being fed into a ML algorithm, feature vectors are also commonly normalized.

Spectral features such as Mel Frequency Cepstral Coefficients (MFCCs) and Perceptual Linear Predictions (PLPs), are the most commonly used features in speech processing [40]. While both are perceptually motivated representations of the human auditory system, and go through similar processing chains, there are some key differences between them.

MFCCs aim to represent the sound being produced by the vocal tract by modelling its shape, using the signal's short-term power spectrum, warped with a Mel-scale filter-bank. On the other hand, PLPs model the human auditory system with auto regressive coefficients, and cepstral analysis. PLPs make use of the psycho-acoustic Bark-scale to filter the power spectrum of the input signal, in order to simulate the spectral response of the human ear.

To obtain further information about the dynamic characteristics of the signal, it is common to compute differences, or *deltas*, between cepstral coefficients (such as the MFCCs and PLPs). The first order *delta* coefficients are defined as:

$$\Delta c(t) = c(t+2) - c(t-2) \tag{2.1}$$

The second order *delta* coefficients, the *delta-delta* coefficients are computed as:

$$\Delta\Delta c(t) = \Delta c(t+1) - \Delta c(t-1) \tag{2.2}$$

Additionally, one might need further contextual information, in which case the Shifted-Delta-Cepstral Coefficients (SDC) coefficients can be applied. These coefficients are able to capture information over

several frames, and can be defined as:

$$\Delta c(t,i) = c(t+iP+d) - c(t+iP-d), \tag{2.3}$$

where, $i = 0, ..., N-1$, is the number of coefficients computed in each frame, d is the spread over which deltas are calculated, and P determines the interval between consecutive blocks of frames. The final SDC coefficient vector, for a time frame t is thus:

$$SDC(t) = [\Delta c(t,0), \Delta c(t,1), ..., \Delta c(t,k-1)], \tag{2.4}$$

where k is the number of frames used to compute the final feature vector [13].

For Speech Paralinguistic tasks, most works focus on the acoustic characteristics of speech, having a special emphasis on Low-level Descriptors (LLDs) such as pitch, energy, duration and voice quality, with the latter being measured by features such as jitter, shimmer and Harmonics-to-noise-ratio (HNR), in addition to MFCCs and PLPs [82].

2.1.2 Artificial Neural Networks

Artificial Neural Networks (ANNs), or simply Neural Networks (NNs), are currently one of the most widely used state-of-the-art ML techniques. An NN is a set of interconnected nodes, usually following a feed-forward structure. Based on the human brain's cells, the nodes in an NN try to emulate the functioning principle of neurons. For this reason nodes are also referred to as neurons.

The original construction of a neuron, the *perceptron*, receives a set of weighted inputs, and outputs 0 or 1, depending on whether or not the sum of the inputs is greater than a threshold. Mathematically, this is done by applying the function:

$$\mathbf{y}(\mathbf{x}) = \begin{cases} 1, & \text{if}(\mathbf{w}^T\mathbf{x} + b) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

where $\mathbf{w}$ and $\mathbf{x}$ are N-dimensional weight and input vectors, respectively, and b is a bias term. To expand the number of functions a network can model, neurons are usually organized into layers. For NNs, the most common layer is the Fully Connected Layer (FC) layer, defined as:

$$\mathbf{y}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}, \tag{2.6}$$

where $\mathbf{A}$ and $\mathbf{b}$ are the weight matrix and bias vector, respectively, and $\mathbf{x}$ is the input vector. Additionally, besides the perceptron's binary step function, several other nonlinear Activation functions are usually included after each FC layer, such as:

- Rectified Linear Unit (ReLU): $y(x) = max(0, x)$
- Sigmoid: $y(x) = \frac{1}{1+e^{-x}}$
- Tanh: $y(x) = \frac{1-e^{2x}}{1+e^{2x}}$
- Softmax: $y(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$

An NN can be trained to perform classification or regression tasks, using *backpropagation* algorithms, which take the error of each prediction done over a training set and re-adjust the networks's weights in order to minimize the error in accordance with a *loss* function.

2.1.2.A Convolutional Neural Networks

A specific type of NNs are the Convolutional Neural Networks (CNNs). These networks are usually composed of a Convolutional stage, where the network alternates between Convolutional, Activation and Pooling layers, and a second stage, which follows the same structure of a regular NN, alternating between FC and Activation layers.

In a Convolutional Layer, a set of learnable filters (or filter maps), are convolved with the layer's inputs. Filter maps work similarly to neurons, in the sense that both are weighted sums that have adjustable parameters. Each filter map is applied to small regions of the input, and strided along the full signal, highlighting and identifying the most relevant areas for the task at hand.

Pooling layers work similarly to Convolutional layers, as a filter is strided along the inputs of the layer. However, in this case, the objective is not so much to filter particular features, but to reduce the dimensionality of the input. For this reason, pooling layers work simply by selecting an area of the input signal and applying a certain operation to reduce its dimensionality, having no trainable parameters. One example of this is the Max pooling layer, which takes a region of the input signal and applies the $max(x)$ function, selecting as output the maximum value in that area. Alternatively, the Average Pooling layer takes an area of the input signal and outputs the average of the values encompassed by the pooling filter.

By applying these operations, Convolutional Networks are able to learn and classify high-level features from low-level input representations, making them ideal for applications in Image Processing as well as in Speech Processing.

2.1.2.B Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of NN developed specifically to deal with time-varying inputs. The main motivation behind them is the fact that regular NNs are not able to obtain relevant time-domain information and context from the input without incurring in computational constraints, due

to the high dimensionality of time-sequential inputs such as audio, video and text. Using Recurrent layers, RNNs can receive sequential inputs, retain some of their characteristics, and combine them with the information contained within the subsequent samples.

A Recurrent layer can be defined by the function:

$$\mathbf{h}_t = \phi(W\mathbf{x}_t + U\mathbf{h}_{t-1}), \tag{2.7}$$

where x_t is the input at time t, h is the output of the layer, called the hidden state, ϕ is an activation function, and W and U are matrices applied to the input and to the previous hidden state, respectively. Training recurrent layers is not straightforward, and requires a specific type of backpropagation algorithm, called Backpropagation Through Time (BTT), where the network is "unrolled" (i.e. expanded with regard to each time step), in order to allow the propagation of the output error from the end to the beginning of the time-sequence. However, in Recurrent Layers, for very long time-sequences, BTT suffers from gradient vanishing, or in other words, the backpropagated value diminishes along each time step, quickly reaching zero, and thus, disabling the network's capability of adjusting weights in accordance to the output error. To avoid this, more complex layers as the Long Short-Term Memory Layer (LSTM) and the GRU (Gated Recurrent Unit) are used [19].

2.1.2.C Network Regularization Layers

Over-training and training a network with small datasets may result in it being too adapted, or *over-fitted*, to the training data, and hence, not being able to generalize what it learned with the training set to unseen data. To avoid overfitting, there are several *regularization* techniques, such as *early stopping*, where training is halted after the network's performance on the development set has been stable for a set number of iterations. Additionally, some layers have the specific purpose of preventing the network from overfitting, such as the Batch Normalization Layer (BN) and the Dropout Layer.

The Dropout layer works by setting the weights that connect two neurons of two different layers to zero, or in other words *dropping* the connection, given a certain probability, thus forcing the network to adapt and perform with a reduced number of connections [85].

Batch Normalization Layer (BN) layers, zero-center and normalize to unit variance their inputs, with regard to mean and variance values computed over batches during training, and subsequently scale and shift each value, adding some noise to the value running through the node, which helps to prevent the network from overfitting. Additionally, BN layers also make sure intermediate representations of the input never grow too large, or too small, in between layers which can also be a helpful characteristic

during training [43]. This layer is defined as:

$$y = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{2.8}$$

where μ and σ are the mean and standard deviation, and γ and β are the scale and shift parameters.

2.1.3 Support Vector Machines

Support Vector Machines (SVMs) are a type of ML algorithm, that separates data according to hyperplanes that are constructed using training data. For this purpose, the algorithm first transforms the data from the input space to a higher dimensional space, also called the *feature space*, where the data is linearly separable, using a *kernel*. Some of the most commonly used kernels are the Radial Basis Function Kernel, or RBF Kernel, the Polynomial Kernel and the Linear Kernel. Using data in the feature space, the training algorithm tries to find the hyperplane separating the two classes that maximizes the distance, or *margin*, between itself and the closest vectors of each class, which are referred to as the *support vectors*. After the model is trained, one can classify an unseen feature vector considering where it is placed in relation to the hyperplane [71].

With this algorithm, one is able to perform only binary classification tasks. However, using slightly different constructions and combinations of several SVMs, it is also possible to perform regression, and $n - ary$ classification tasks.

SVMs have the advantage of providing good results when the training data is scarce, which is frequently the case in speech paralinguistic tasks, making them a common choice for these applications [82].

2.1.4 Gaussian Mixture Models

Gaussian Mixture Models (GMMs) are ML models based on linear combination of Gaussian distributions, used in speech pattern recognition, in tasks such as speaker and language verification and identification. In addition, GMMs have also been applied in paralinguistic tasks such as age, gender and emotion classification [82].

GMMs are able to model any distribution, as long as the number of components, in this case *Gaussians*, that make up the linear combination is large enough. They are trained using the Expectation-maximization (EM) algorithm, that iteratively updates the parameters of each Gaussian component, until it is considered that the linear combination of the Gaussian distributions models the training data well enough, or in other words, a convergence criterion is reached [72].

With a trained model one can compute Maximum Likelihood estimates for a given feature vector, to determine whether or not it belongs to a certain class.

For speaker and language verification tasks, Universal Background Models (UBMs) can be trained using a large set of speakers, or languages, and subsequently adjusted to the speaker or language in question. Using both the original and adapted UBM, it is possible to compute the log-likelihood ratio of the scores for each model, and determine whether or not the sample belongs to the class being tested [72].

2.2 Privacy-preserving Machine Learning Frameworks

Privacy-preserving Machine Learning (PPML) is a field of research that aims to adapt state-of-the-art ML algorithms to different privacy conditions, with the intent of making them secure. As explained in Chapter 1, Section 1.1.2, MLaaS frameworks are mostly insecure and are unable to provide privacy for the user's data, as well as for the ML model itself.

PPML applications use Secure Multi-Party Computation (SMPC) (also referred to as Secure Function Evaluation (SFE)) frameworks, a type of secure computational framework that allows several parties $p_i = p_0, ..., p_n$, each holding private inputs x_i, to jointly compute a function $f(x_0, ..., x_n)$, while keeping their data private [33]. A particular case of SMPC is the Secure Two-Party Computation (STPC), that involves only two parties to compute a function f.

In this section we will detail some of the building blocks, or *primitives*, of SMPC, such as Homomorphic Encryption (HE) and Oblivious Transfer (OT), as well as some secure frameworks, with particular emphasis on those used for PPML applied to Speech tasks, as Garbled Circuits (GCs), Secure Binary Embeddings (SBEs) and Secure Modular Hashing (SMH).

2.2.1 Homomorphic Encryption

2.2.1.A Basic Concepts

Homomorphic Encryption (HE) is a type of cryptosystem in which certain operations performed on *ciphertexts* (i.e. encrypted values) are *homomorphic* with regard to the *plaintexts* (i.e. unencrypted values). In other words, considering the encryption of a value x, $E(x)$ and of a value y, $E(y)$, if a homomorphic operation is performed on the two ciphertexts, the result of this operation will correspond to the equivalent unencrypted operation of the two values, as follows:

$$\begin{aligned} E(x) \otimes E(y) &= E(x \times y), \\ E(x) \oplus E(y) &= E(x + y), \end{aligned} \tag{2.9}$$

with $\otimes$ and $\oplus$ corresponding to homomorphic multiplication and addition, respectively.

HE serves as a building block for many secure protocols, having applications in many areas including data mining, forensics, financial privacy and medical applications [3].

A HE scheme can be defined by four algorithms [2]:

- The *Key Generator*, or *KeyGen*, which generates the public and private keys, for asymmetric HE, and a single private key, for symmetric schemes.
- The *Encryptor*, or *Enc*, which takes the encryption key e_{key} (be it public or not) to produce a ciphertext, $c = Enc(m, e_{key})$.
- The *Decryptor*, or *Dec*, which performs the inverse operation of the encryptor, and decrypts a ciphertext using a decryption key d_{key}, $m = Dec(Enc(m, e_{key}), d_{key})$.
- The *Evaluator*, or *Eval*. This algorithm is specific to HE schemes, and performs an operation f over two ciphertexts c_1 and c_2, outputting a ciphertext $c_o = Eval(c_1, c_2, f)$.

Some HE schemes can perform all types of homomorphic operations, while others cannot. Additionally, some HE schemes can perform these operations an unlimited number of times, while others can only perform them a specific number of times. Taking these characteristics into account, HE cryptosystems can be divided into three main categories [2]:

- Partially Homomorphic Encryption (PHE), where only one type of operation is allowed, but it can be performed an unlimited number of times.
- Somewhat Homomorphic Encryption (SWHE), where one or more types of operations can be computed a limited number of times.
- Fully Homomorphic Encryption (FHE), where an unlimited number of operations can be performed an unlimited number of times.

2.2.1.B Partially and Somewhat Homomorphic Encryption

A prominent example of a PHE scheme is RSA. Introduced by Rivest et al. [74], this scheme is homomorphic over multiplication, which means the relation,

$$Eval(Enc(x, e_{key}), Enc(y, e_{key}), \otimes) = Enc(x \times y, e_{key}), \tag{2.10}$$

holds, and this operation can be performed an unlimited number of times. With its security based on the hardness of the *factoring problem* of the product of two large prime numbers, RSA was one of the first PHE schemes, as well as one of the first public-key cryptosystems. In addition to RSA, another important PHE multiplicatively homomorphic cryptosystem is El Gamal [25].

Additively homomorphic cryptosystems also exist, that hold the relation:

$$Eval(Enc(x, e_{key}), Enc(y, e_{key}), \oplus) = Enc(x + y, e_{key}), \tag{2.11}$$

such as Paillier and Goldwasser-Micali, among others [2]. Paillier has the additional advantage of allowing multiplications to be performed between a ciphertext and a plaintext. Using this property it is possible to compute inner products using this cryptosystem, as long as one of the vectors is made of plaintexts.

While some SWHE schemes allow for both additions and multiplications, a restriction is always present, as is the case with the Boneh-Goh-Nissim (BNG) [2], which allows unlimited additions, but it is limited to one multiplication. However, since one multiplication can be performed, it is possible to compute a single inner product of two encrypted vectors. Another SWHE scheme, the Polly-Cracker (PC) [2] scheme, allows unlimited operations to be performed on the ciphertext, although the noise grows exponentially with each operation, making the decryption of the ciphertext meaningless after a certain number of operations [2].

2.2.1.C Fully Homomorphic Encryption

A Fully Homomorphic Encryption (FHE) system was only achieved in 2009 by Craig Gentry [30]. Gentry's scheme was constructed using a lattice-based SWHE cryptosystem. In this scheme, ciphertexts have an associated noise term, which grows with each homomorphic operation, and after a threshold number of operations, the ciphertext can no longer be decrypted correctly. To make this scheme fully homomorphic, Gentry introduced two key concepts, *squashing* and *bootstrapping*. The first concept considers the construction of shallow decryption circuits that can be evaluated homomorphically. On the other hand, the latter proposes homomorphically re-encrypting a ciphertext with the *squashed* decryption circuit, using an encryption of the original encryption key. This re-encryption would reset the noise level in the ciphertext, allowing for an unlimited amount of operations to be performed over it.

Introduced by Dijk, Gentry, Halevi and Vaikutanathan in 2010 [91], DGHV is a conceptually simple example of how a SWHE scheme can be transformed into a FHE scheme. In order to better expose the functioning principles behind homomorphic noise growth, as well as to make clear why *squashing* is necessary and how *bootstrapping* can make a SWHE scheme fully homomorphic, we will now provide a summarized explanation of the SWHE public-key version of this scheme.

In its simple (symmetric) encryption form, considering a message space of $m \in \{0, 1\}$, a ciphertext in DGHV is defined as:

$$c = Enc(m, p) = pq + 2r + m, \tag{2.12}$$

where the secret key p, is an odd positive integer selected from an interval $[2^{\eta-1}, 2^{\eta})$, q and r are random integers sampled from specific distributions depending on p. Additionally r must obey $|2r| < p/2$.

The decryption circuit of this scheme can be defined as:

$$m' = Dec(m, p) = (c \bmod p) \bmod 2 \tag{2.13}$$

From Equation 2.12, it is easy to see that having two ciphertexts $c_1 = q_1 \cdot p + 2r_1 + m_1$ and $c_1 = q_2 \cdot p + 2r_2 + m_2$, their homomorphic addition will be:

$$\begin{aligned} c_1 + c_2 &= p(q_1 + q_2) + 2(r_1 + r_2) + (m_1 + m_2) \\ &= pq' + 2r' + (m_1 + m_2) \\ &= Enc(m_1 + m_2, p) \end{aligned} \tag{2.14}$$

Additionally, the homomorphic multiplication of c_1 and c_2 will be:

$$\begin{aligned} c_1 \times c_2 &= pq'' + 2r'' + (m_1 \times m_2) \\ &= Enc(m_1 \times m_2, p) \end{aligned} \tag{2.15}$$

where $r'' = 2r_1r_2 + r_1m_2 + r_2m_1$. For simplicity we omit the full proof, however it is important to note that, since r_1 and r_2 are integers, r'' is at least twice as large as each of these values.

Considering that r needs to be smaller than p, for the decryption to work correctly, the number of multiplications in this scheme is limited. For this reason, in order to transform the scheme into a FHE scheme, Gentry's techniques of *squashing* and *bootstrapping* need to be applied.

Equation 2.13 cannot be directly expressed as a low depth circuit, and it cannot be correctly performed homomorphically without the noise term growing too large. Consequently, it is first necessary to *squash* it, or in other words, to reduce the number of operations required to perform it. For brevity, we will not include the *squashed* circuit, nor its construction, and instead refer the reader to the original paper [91].

Since the *squashed* circuit can be homomorphically evaluated, it is possible to perform *bootstrapping*. To understand this process we can consider a hypothetical situation where a client encrypts a message m, using a public key pk and a secret key sk, obtaining $c = Enc(m, pk)$, and sends it to the server. The server performs several homomorphic operations on it, obtaining a ciphertext c', that cannot be subject to further operations due to the size of its noise term. As such, it needs to be reset using *bootstrapping*. To this end, the client sends an encryption of the original secret key $Enc(sk, pk)$ to the server. Since the decryption algorithm can be performed using homomorphic operations, the server can homomorphically decrypt c', obtaining an encryption of $c'' = Dec(c, Enc(sk))$, which has a much smaller noise term when compared to c'.

From what was stated above, it is clear how ingenious and powerful *bootstrapping* is. However, in most implementations it is highly inefficient, creating a computational bottleneck. Nevertheless, since

Gentry first introduced the technique, it has been the subject of numerous improvements, that have reduced its computational complexity significantly [15] [2],

2.2.1.D Leveled Homomorphic Encryption

Taking advantage of the fact that, in most applications, the user knows beforehand the amount of operations that he/she intends to perform, and to avoid the computational cost of *bootstrapping*, Leveled Homomorphic Encryption (LHE) schemes, a family of SWHE, rose as an alternative to FHE. These schemes allow the user to select the encryption parameters in such a way that it is possible to determine the maximum noise the ciphertext can have, while still being possible to decrypt it correctly. Conversely, the maximum noise threshold can be defined as a *noise budget*, or the amount of noise that one can "spend", which determines the maximum amount of operations that can be performed on the ciphertext. However, in order to obtain a larger *noise budget*, one has to increase the size of the encryption parameters, which means increasing the computational complexity of each operation in the scheme. For this reason, it is necessary to trade-off between the number of operations to be performed, and the computational cost of the scheme.

To compensate for the computational limitations of LHE, [10] proposed an important optimization, called *batching*. This technique allows several messages to be encrypted in the same ciphertext and thus, to be operated on at the same time. Although the cost of each computation remains the same, the cost of each operation can be divided by the number of messages that can be encoded into a single ciphertext, effectively reducing the computational overhead of the scheme.

Throughout this thesis we will use the Simple Encrypted Arithmetic Library (SEAL)'s implementation of the LHE Fan and Vercauteren (FV) scheme [29]. The main reasons for us to have chosen this library were, on one hand the fact that this library was used (and developed) by the authors of Cryptonets [31], on which we base a large portion of this work, and on the other hand, that Simple Encrypted Arithmetic Library (SEAL) is, to our knowledge, the only library which allows fractional numbers to be directly encoded into plaintexts. This is an important characteristic, as in most ML frameworks both the input features and the model's parameters are real numbers and turning them into integers is not straightforward, requiring specific encoding schemes. Additional details regarding this implementation, including a security review of the scheme, can be found in Appendix A.

2.2.2 Oblivious Transfer

Oblivious Transfer (OT) is a cryptographic primitive that allows two parties to securely exchange data.

The two parties in this scheme are usually labeled as the *sender* and the *receiver*. The *sender* is in possession of a dataset $\mathbf{x} = \{x_0, x_1, ..., x_n\}$ that contains a datapoint x_i that interests the *receiver*. The *sender* is willing to give x_i to the *receiver*, however, the *receiver* does not want the *sender* to know

which $x_i \in \mathbf{x}$ they're interested in. In addition, the *sender* does not want the *receiver* to learn anything else about the dataset besides x_i.

The simplest version of this protocol is 1-out-2 OT, where the *sender* is in possession of two messages, with equal probability. Using a specific cryptographic protocol, the *receiver* selects the message through an encrypted input bit *c*. The *sender* takes this encryption of the input bit and uses it to encrypt all his data in such a way that the *receiver* can only decrypt the message selected by himself. The *receiver* then obtains all messages, and decrypts them. Of the decrypted values, only the value corresponding to the the message he selected will be meaningful. In this way, the *receiver* never learns anything about the other message and the *sender* never learns which message interested the *receiver* [26]. The first form of 1-out-2 OT was proposed by Rabin, M. in 1981 [61].

It has been shown that with OT it is possible to construct any cryptographic task, as well as to construct efficient STPC and SMPC protocols [53] [18] [47].

2.2.3 Garbled Circuits

A Garbled Circuit (GC) is a cryptographic construction that allows two parties, Alice and Bob, to jointly compute a function f using their private inputs x and y. In this protocol, the user's data as well as intermediate representations are always kept secure, and only the output is revealed to one, or both parties. For this purpose, one party, whom we will assume is Alice, the *garbler*, must transform the function f into a Boolean Circuit, using double input logic gates. Additionally, Alice needs to encrypt, or garble, the values of every wire in the circuit. After this, Alice maps her inputs to their garbled counterparts, in order to mask them. She then sends the garbled truth tables of the circuit along with her garbled input to Bob. Bob, who is the *evaluator*, receives the garbled circuit and inputs, from which he cannot learn anything, and must now map his own inputs to their garbled equivalents. To do this, Bob uses 1-out-2 OT to ask Alice for the garbled versions of his inputs, guaranteeing that she does not have access to them. In possession of his garbled inputs, Alice's garbled inputs and the garbled circuit, Bob is now able to evaluate the garbled circuit's gates. After the output gate is evaluated, Bob sends the resulting value to Alice, who can, using the original mapping of the circuit, decrypt the final result $f(x, y)$, which she may share with Bob. [77]

The simplest form of the protocol guarantees security against *semi-honest* adversaries, which in this case are parties that are assumed to follow the protocol in place, but that may try to deduce information about the other parties' data. However, since it was originally proposed by Yao, several advances have been made to this protocol, in order to support other security models [84].

Although conceptually simple, this protocol has been applied to PPML in works such as [77], [73] and [54], among others.

2.2.4 Distance-preserving Hashing Techniques

2.2.4.A Locality-Sensitive Hashing

Being a method that reduces the dimensionality of high-dimensional data, Locality-Sensitive Hashing (LSH) is commonly applied to Nearest Neighbours problems, with its applications including data compression, information retrieval, near-duplicate search and machine learning [45].

LSH is based on locality-sensitive hash functions $H(.)$ that project vectors into lower-dimensional spaces. If two vectors are close enough in the input space, they will hash to the same value, or "bucket", with high probability. On the other hand, if two vectors are far apart in the input space, they will be projected onto different buckets.

More formally, given two vectors, x and y, an LSH function will be a map h from space $\mathbb{M}$ to space $\mathbb{S}$ such that, given a distance function d in the input space:

$$\begin{aligned} h(x) = h(y) \text{ with high probability, if } d(x-y) \leq r \\ h(x) \neq h(y) \text{ with high probability, if } d(x-y) \geq cr \end{aligned} \tag{2.16}$$

where r is the radius of interest and c a constant greater than 1. If two inputs have a distance smaller than r, there will be a a high probability that they will have the same hash, whereas if their distance is greater than cr there will be a high probability that they will fall into different buckets [45] [69].

Although LSHs can be used to mask data, they do not provide any information-theoretic security guarantee. As such, to ensure privacy it is necessary to employ information-theoretic secure embeddings, or HE schemes. Two such approaches are explained in the next subsections.

2.2.4.B Secure Binary Embeddings

Secure Binary Embedding (SBE) is a distance-preserving hashing technique that uses band-quantized random projections to convert real-valued vectors into bit sequences. SBEs have the property that, if the Euclidean distance between the original vectors is lower than a threshold, the Hamming distance of the hashed vectors will be proportional to it. Otherwise the Hamming distance will provide no information on the true distance between the original vectors.

SBE is based on Universal Quantization, which redesigns the scalar quantization function to have non-contiguous quantization regions.

For a vector $\mathbf{x}$ with *L* samples, the Universal Quantization process can be summarized in vector form as:

$$\mathbf{q}(\mathbf{x}) = Q\left(\mathbf{\Delta}^{-1}(\mathbf{A}\mathbf{x} + \mathbf{w})\right), \tag{2.17}$$

where $\mathbf{q}$, also referred as the hash of $\mathbf{x}$, is an *M*-bit binary vector, $\mathbf{\Delta}^{-1}$ is a diagonal matrix that defines the precision parameters, $\mathbf{A} \in \mathbb{R}^{L \times M}$ is a random matrix, where each row vector $\mathbf{a}_m$ is composed of i.i.d elements sampled from a normal distribution and $\mathbf{w} \in \mathbb{R}^M$ is the additive dither, drawn from a uniform distribution in $[0, \Delta]$. Finally, the quantization function $Q(.)$ is defined as $Q(x) = \lfloor x \ mod \ 2 \rfloor$.

Following certain conditions, the normalized, per-bit, Hamming distance d_h between the hashes of two vectors, obtained using Equation 2.17, is bounded and depends on the precision parameter Δ. Additionally, d_h is correlated with the Euclidean distance d_e between the two original vectors if d_e is smaller than a certain threshold, also dependent on Δ. After this threshold, d_h converges quickly to its upper bound losing any relevant information on d_e. Conversely, it is provable that the Hamming distance between vectors further apart than this threshold provides no information whatsoever on the true distance that separates them and as such, this embedding provides information-theoretic security beyond the radius defined by this threshold, as long as the parameters $\mathbf{A}$ and $\mathbf{w}$ are kept secret from other parties [8] [70] [69] [22].

Moreover, this technique may be combined with SVMs by modifying kernels to work with Hamming distances between hashes, allowing for the hashed representation of the input vectors to be used directly with the SVM. In this way, secure two-party MLaaS frameworks can be built on top of the two techniques, as it has been shown in [70], for both speaker verification and authentication tasks, as well as in [22], for an emotion recognition task.

When used together with SVMs, a larger Δ increases the accuracy of the classifier, as well as increasing the value of M in relation to L, the original vector length, also referred to as the number of *bits per coefficient* or *bpc* [23] [22].

2.2.4.C Secure Modular Hashing

A generalization of SBE, Secure Modular Hashing (SMH) is a modular variant of *p*-stable LSH [45]. As opposed to SBE, in SMH the hash function is defined as a random projection from $\mathbb{R}^N$ to $(\mathbb{Z}/k)^M$, where $\mathbb{Z}/k$ is the set of integers from 0 to $k-1$ and M is the number of hashes. More concretely, this is accomplished through the use of:

$$\begin{aligned} \mathbf{q}(\mathbf{x}) &= Q\left(\mathbf{\Delta}^{-1}(\mathbf{A}\mathbf{x} + \mathbf{w})\right) \\ &= \lfloor \mathbf{\Delta}^{-1}(\mathbf{A}\mathbf{x} + \mathbf{w}) \rfloor (\ mod\ k), \end{aligned} \tag{2.18}$$

where $\mathbf{w}$ is uniformly distributed between $[0, k]$ and $\mathbf{A} \sim N(0, 1)$.

Like in SBE, in SMH there are two essential parameters to be chosen by the user, Δ, the scalar variance of $\mathbf{A}$ and the number of samples M (equivalent to bit-length in SBE). In addition, SMH involves choosing a third parameter, the modulus k. Furthermore, the user must also generate the random matrix

$\mathbf{A}$ and the random vector $\mathbf{w}$, which, as in SBE must be treated as the framework's key and, therefore, must be kept secret in order to ensure its security. Similarly to SBE, the accuracy of a framework combining SMH and SVMs depends greatly on the parameters Δ and M. However, in SMH, the performance of the framework also depends on the modulus k.

2.2.4.D Leakage

All three of the approaches described above consider the possibility of information *leakage*, when two vectors are closer than a certain threshold. When this happens, it may be possible to retrieve information about specific features of the original vectors, from their hashes. In [70], *leakage* is defined as the number of hashes that have their normalized distance below a threshold in relation to the total number of hashes, or:

$$Leakage = \frac{\#leaks}{\#hashes}. \tag{2.19}$$

Increasing Δ, M and k (in the case of SMH), increases the *leakage* of the framework, as it increases the maximum distance threshold for which information can still be obtained. Thus, to ensure the framework's privacy, it is necessary to carefully select a trade-off between the classifier's performance and the information leakage [23].

2.3 Related Work

Privacy-preserving in speech is a fairly recent topic of research. One of the first strides in this direction was made by Pathak et al. [67], who adapted a GMM to work with HE, in particular with the Paillier and BGN cryptosystems, to perform both speaker verification and identification. In a different approach, Portêlo et al. [70], and later Jiménez et al. [45], applied SBEs and SMHs to speaker verification. The most recent effort in this topic was provided by Dias et al. [23], where both SMHs and HE were applied to an emotion recognition task.

Outside speech, the literature is extensive, and a very large number of different techniques has been developed for PPML, including frameworks for linear regression, K-Means Clustering, SVMs and NNs.

One of the first approaches for secure NNs was developed by Barni et al. [5]. In their work, the authors used HE to compute private inner products, while the activation functions in the network were applied by the user. However, this not only implied a large and constant communication overhead with the user, but it also made it easy for the user to obtain information about the model, although this problem was later tackled by Orlandi et al. [64].

More recently, the authors of Cryptonets [31] gave a very important contribution to PPML frameworks, by computing the prediction stage of an NN with LHE. This framework was further improved by

Chabanne et al. [14] and Hesamifard et al. [39]. However, this approach present two setbacks: first the number of layers in the network is limited by the computational overhead of the LHE cryptosystem; second, this cryptosystem does not allow comparisons between two ciphertexts, thus preventing piece-wise functions from being computed. To avoid both problems, [79] and [9] made use of a FHE scheme, that implements very fast bootstrapping, and in which all computations are performed at the binary gate level. However, they require discrete weights with very small absolute values to be efficient, which translates to either a loss in precision of the network, when one uses small weights, or on a large increase in the computational toll due to the use of large weights.

Rouhani et al. [77] have avoided the computational overhead of HE using OT and GC. This approach obtained very good results. Yet, it requires a constant presence of the client during the process, and demands that the NN be represented as a boolean circuit. Nevertheless, communication and computational costs between the server and the client are very small when compared to Cryptonets. Similarly, [58], [54] and [73] base their approaches on HE, GCs and secret sharing, improving the results of [77].

Due to the accuracy and computational losses imposed by secure frameworks, very few works perform the training stage in a privacy setting. Two notable examples are [58] and [83], which are based on OT, GC and Differential Privacy.

The works mentioned above are all based on software approaches. However, some works have based their frameworks on secure dedicated hardware, and more specifically on Intel's SGX processor. This processor allows computations to be performed on a secure *enclave*, or in other words, a program which is executed in isolation from all other software being executed on the system. VoiceGuard [11], proposes a system with which these processors can be used to perform PPML speech applications.

3

Privacy-preserving Neural Networks

Contents

In this chapter we explore how a Neural Network (NN) can be turned into an Encrypted Neural Network (ENN), using Leveled Homomorphic Encryption (LHE). We discuss the limitations of this approach and point out its potential for health related speech applications, exemplifying for the detection of a Cold, the assessment of Parkinson's Disease as well as both the detection and assessment of Depression. The chapter starts with a brief introduction on the topic in 3.1. This is followed by Section 3.2 where the adaptation of NNs to HE is discussed, as well as how different activation functions can be approximated with polynomials. Section 3.3 and 3.4 detail the experimental setup and results obtained, respectively. Finally in 3.5 we discuss and present some conclusions regarding this approach.

3.1 Introduction

Neural Networks (NNs) are nowadays the state-of-the-art for most machine learning applications. With the growing concern on user privacy, especially when considering medical applications, it is important to find ways to adapt NNs in order to protect the user's privacy. As it has been shown by [31] [14] [39], NNs are perfect candidates for adaptation to a privacy-preserving setting, using Homomorphic Encryption (HE). In these approaches all mathematical operations in the NN are performed over encrypted data, taking advantage of the properties of homomorphic encryption, which allows for computations to be performed over Ciphertexts (i.e. encrypted values) in such a way that when the Ciphertext is decrypted, it yields the same value as if the computations had been performed over a Plaintext (i.e. its unencrypted value).

In Figure 3.1 we present a scheme of how a Machine Learning as a Service (MLaaS) framework can be applied to speech, using an Encrypted Neural Network (ENN), although this framework may also be applicable to any generic ML algorithm, as long as it is compatible with HE and follows a similar pre-processing chain. The client starts by extracting a set of features from a raw audio file. This feature vector is then encrypted using an encryption key created using a set of encryption parameters, defined by the service provider. The client can now safely send the encrypted feature vector to the service provider without having to be concerned with the security of his information. The service provider takes this encrypted vector and feeds it to a previously trained ENN, which will in turn output an encrypted prediction. This prediction is sent back to the client who can decrypt it using a decryption key, created with the same set of parameters that were used for the encryption key.

In this framework, the user's information is always kept secure, as it is never seen in Plaintext form by anyone other than the user. In addition, the NN produces an encrypted prediction that can only be decrypted by the user. Both these conditions prevent the service provider and malicious third parties from learning anything about the user from either the user's data or the prediction of the ENN.

NNs require large amounts of data and expertise to be developed and trained. Using HE allows

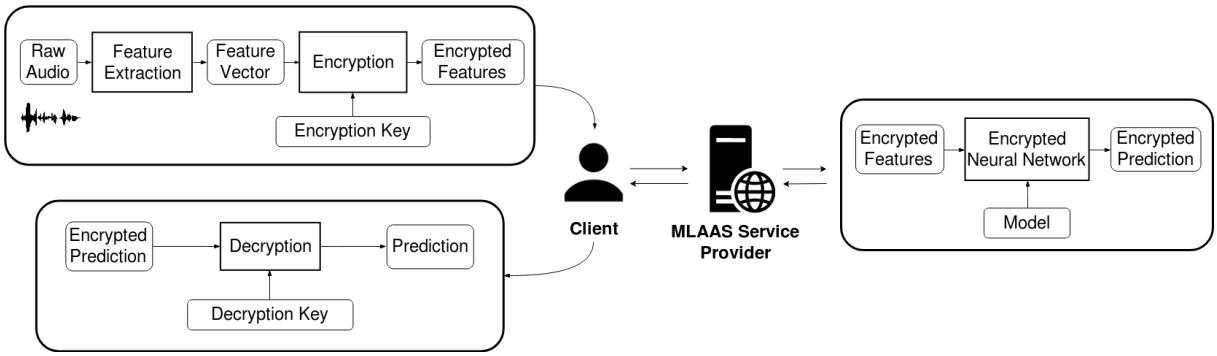


Figure 3.1: Machine Learning as a Service (MLaaS) framework using an Encrypted Neural Network (ENN).

the service provider to make predictions over the user's data locally, without ever having to distribute the NN model. If a system is already in place (i.e. the user already knows which features to extract and the required encryption parameters), only two rounds of communication are necessary between the client and the service provider, one for the client to send his data and another for the service provider to send the encrypted prediction back. This is an advantage over other approaches such as [77], [58], [73] and [54], as it does not require the constant presence of the client during the computation. In addition, this allows the server to protect its model, as opposed to the approaches referenced above, which allow the user to learn the model's architecture.

The main goal of this chapter is to show how this framework can be applied to health related paralinguistic speech tasks and in particular to evaluate its performance on the detection of a Cold, the detection and assessment of Depression, and the assessment of Parkinson's Disease.

3.2 Encrypted Neural Networks

Several works have explored how an NN can be adapted to HE, following the framework detailed in the introduction of this chapter (3.1). Cryptonets [35] [31] is the first and most important of these studies. In their work, the authors showcase how a Convolutional Neural Network (CNN) can be successfully adapted to use HE operations, with minimum accuracy degradation, and reasonable performance. This work was later extended by Chabanne et al. [14] and Hesamifard et al. [39], which focused particularly on how to deal with activation functions.

As was described in Section 2.1.2, a Neural Network (NN) or, more formally, an Artificial Neural Network (ANN), is a set of interconnected nodes, usually organized in layers in a feed-forward fashion, inspired by the structure of the human brain, which is composed of a series of interconnected neurons. The parameters in an NN can be trained, using Backpropagation (BP), to perform tasks such as regression and classification. The most common type of layer in an NN is the Fully Connected Layer (FC), which is composed of stacked perceptrons, or neurons (2.1.2). This layer is essentially a linear trans-

formation (Equation 3.1), where A is the weight matrix applied to the input of the layer, and b is the bias vector.

$$y = Ax + b \tag{3.1}$$

The FC layer is generally followed by the Activation Layer, which applies a nonlinear function to each of its outputs. The Rectified Linear Unit (ReLU) and the Sigmoid are two examples of functions typically used in Activation layers, and are displayed in equations 3.2 and 3.3, respectively.

$$y = max(0, x) \tag{3.2}$$

$$y = \frac{1}{1 + e^{-x}} \tag{3.3}$$

To adapt an NN to HE it is thus necessary to replace every operation by its HE counterpart. For FC layers this is simply a matter of replacing additions and multiplications with their HE equivalent. However, this does not hold for the nonlinear functions in activation layers and these need to be replaced with polynomials, which only use additions and multiplications. It is also worth noting that due to the noise toll of HE multiplication it is important to keep the multiplicative depth of the network to a minimum. This is particularly true for multiplications between Ciphertexts, as multiplications between a Ciphertext and a Plaintext have a much smaller toll on noise, due to the fact that Plaintexts do not have an associated noise term. For this reason, most of the approaches described bellow try to keep the degree of the polynomials used to a minimum.

On a side note, although possible, encrypted training with only one stage of encryption and decryption would be very inefficient, due to the number of operations required in each iteration of the training algorithm. For this reason, in this work we will not consider encrypted training. Having said that, secure training has been shown to be achievable in SMPC schemes, which involve several parties computing the training function over a private dataset [58] [83].

3.2.1 Polynomial Approximations of the Activation Functions

In Cryptonets [31], to avoid nonlinear functions, the authors simply replace each activation function with the square function. This has some disadvantages during the training stage, as the derivative of x^2 is unbounded, which may lead to gradients growing very large, and to over-fitting. To avert this issue, the authors of [14] proposed instead to train the network with regular Activation functions (such as the ReLU), and replace them with polynomial approximations during inference. However, these were done using Least Squares, having only a small convergence interval around the origin and diverging quickly

outside of it. For this reason, the authors proposed another important innovation, the introduction of a Batch Normalization Layer (BN) before each Activation Layer. The BN layer insures the input of the Activation Layer is close to a normal distribution, with zero mean and unit variance, and that most inputs will fall within the convergence interval of the polynomial, minimizing the accuracy drop in the inference stage, when the ReLU is replaced by its approximation. The BN layer also has the advantage of being easily adaptable to HE, with only a division operation having to be transformed into a multiplication, as can be verified in Equation 3.4 (previously described in Section 2.1.2.C).

$$y = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{3.4}$$

This approach was also used by Dias et al. [23], where the authors applied the scheme to a speech emotion recognition task.

Hesamifard et al. [39] expanded this idea by keeping the BN layer, but approximating the ReLU through its derivative, the Step function, integrating the resulting polynomial, and using it to train the network, as was previously done by Cryptonets. The motivation behind this is that if the derivative of the polynomial is close to the derivative of the ReLU, there is a smaller chance that the problems encountered by Cryptonets, regarding the gradients, will appear, and the network can be directly trained with it. Furthermore, to insure a more accurate approximation, considering the Step function is non-differentiable at zero, the authors decided to approximate a continuous function with similar shape, the Sigmoid, instead of the derivative itself. To do this, the authors used Chebyshev polynomials, which allow a trade-off between the size of the convergence interval and the precision of the approximation around zero.

In this work we applied the method described in Hesamifard et al. [39], as it obtained the best results of the three above mentioned approaches in the MNIST dataset [51]. As such the Activation functions were approximated using the method described in Hesamifard et al. [39], and the networks were trained using the approximated polynomials, preceded by BN layers.

However, the authors of [39] are omissive with regard to the constant term of the polynomial resulting from the integration step. Consequently, this prompted us to develop our own method to find it. Considering we wanted the polynomial to have a behavior as similar as possible to the ReLU, we not only wanted the polynomial to have the smallest error between itself and the ReLU, but also to have a value as close as possible to zero, when $x < 0$. To achieve this, we minimized the MSE between the approximation and the ReLU with an added regularizer that penalizes negative values in the interval in which the function is being approximated, such as the one in equation 3.5, where $p(n)$ is the polynomial, and c is the value being optimized.

$$R = \sum_{n} max(-p(n) + c, 0)^2 \tag{3.5}$$

Using this method, the derivative was approximated using Python's *numpy* package, with a high degree Chebyshev polynomial, in the interval $[-120, 120]$. The resulting polynomial was integrated, and all coefficients of degree higher than two were dropped to keep the multiplicative depth of the network to a minimum, for the reasons stated above. To determine the constant coefficient Python's *scipy.optimize* package was used, with a sample of 10000 data points, in the interval $[-50, 50]$. This resulted in the final polynomial in Equation 3.6, represented in Figure 3.2.

$$p(x) = 0.03664x^2 + 0.5x + 1.7056 \tag{3.6}$$

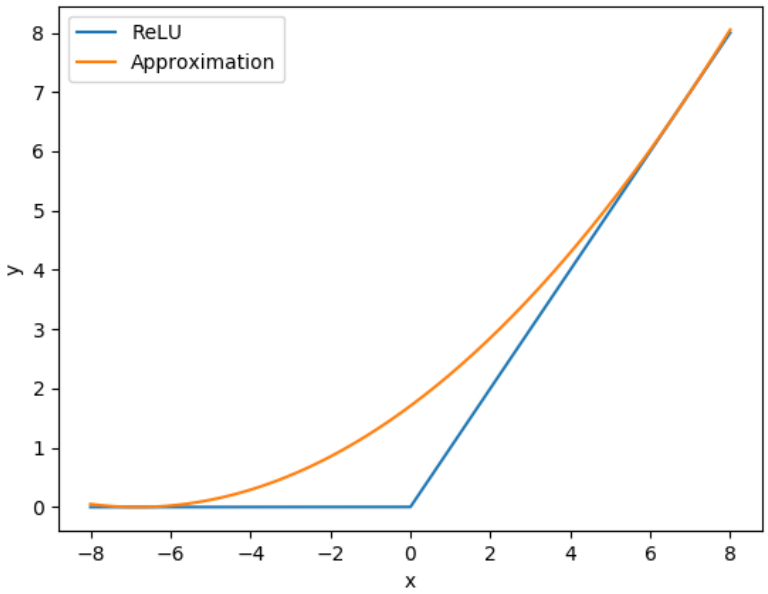


Figure 3.2: ReLU and our Polynomial Approximation

For classification tasks, NNs usually have an output activation function, with the Sigmoid being one of the most common. However, this function can be left out in the prediction stage when the network has more than one output, due to the fact that it is monotone increasing, and taking the maximum of the set of inputs will yield the same result with or without it [31]. In single output applications this does not hold, and there is a need for an alternative. Using the same process and parameters as for the ReLU, the Sigmoid was approximated with a 1^{st} degree polynomial, to keep the degree as low as possible, resulting in Equation 3.7. This proved to be a simple and effective alternative since its low slope results in most outputs being bound between 0 and 1, as is the case with the Sigmoid. However, it does not

introduce non-linearity to the network, which may cause some performance degradation.

$$y = 0.004997x + 0.5 \tag{3.7}$$

Though a higher degree polynomial could have been used, it would have had a significant impact on the computational complexity of the scheme, which motivated the use of a linear polynomial.

3.3 Experimental Setup

To compare the performance of networks with regular activation functions such as the ReLU and Sigmoid (referred to as NNs from this point forward), and their encrypted counterparts using polynomial approximations (referred to as ENNs), experiments were conducted for three different speech affecting conditions, Cold, Depression and Parkinson's Disease. For Cold we used the Upper Respiratory Tract Infection Corpus (URTIC) [48] to perform a classification task. In the experiments performed for Depression, the Distress Analysis Interview Corpus - Wizard of Oz (DAIC-WOZ) [36] was utilized, for both regression and classification. Finally for PD the Parkinson's Disease (PD) Spanish Corpus was used for a regression task [65].

Due to the fact that the datasets used were the same as the ones provided for several speech paralinguistic challenges, labels were only available for training and development sets. While these datasets could have been split to create new test sets, this was not done, so that our results could be compared with the baseline results of each challenge. A detailed description of the three datasets is provided in Appendix B.

3.3.1 Features

Two different feature sets were used in our experiments. For the experiments performed on the Depression and Cold dataset the eGeMAPS feature set was used [27]. The Geneva Minimalistic Acoustic Parameter Set (GeMAPS) was developed in an effort to standardize baseline results in paralinguistic tasks. An extension of this feature set, the extended Geneva Minimalistic Acoustic Parameter Set (eGeMAPS) includes 88 features, containing information on frequency, energy, spectral and cepstral characteristics. However, in the PD experiment, eGeMAPS did not yield significant results. Consequently, a Parkinson's Disease specific feature set was used instead. Developed by Pompili et al. [68], it contains 36 GeMAPS based features, along with 78 MFCC based features, resulting in a 114 dimensional feature vector. Both feature sets were extracted from audio files using openSMILE configuration files [28].

All features were zero-centered and normalized with unit variance using the mean and standard

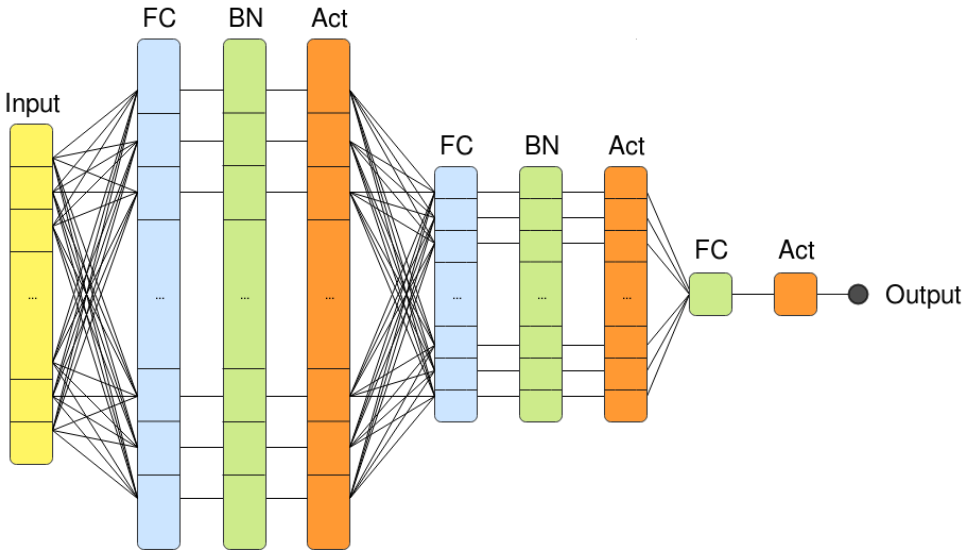


Figure 3.3: Neural Network Architecture (adapted from [23])

deviation computed from each of the training sets.

3.3.2 Network Architecture and Training

The network used in our experiments is similar to the one used by Dias et al. [23], containing two initial FC layers, each followed by a BN layer and an Activation layer, and a third output FC layer. For classification tasks an output Activation layer was included after the third FC layer. During training, a dropout layer was also inserted, before the second and third FC layers. This serves as a regularizer, to help prevent the network from overfitting [85]. The BN layer already present in the architecture, also has a regularization effect [43], apart from assuring that the inputs of the activation layer are normally distributed. The feed-forward network is displayed in Figure 3.3, with the layers labelled FC being Fully Connected layers, BN, Batch Normalization layers and Act, Activation Layers.

The size of our network was limited to three sets of layers for two reasons. The first was the fact that the datasets used for our experiments are relatively small, and using larger networks did not yield any performance gains. The second is the limitation imposed by LHE. For the reasons discussed in Section 2.2.1.D, adding another layer set (FC + BN + Activation) would require an increase in the encryption parameters, resulting in a higher computational toll, or in other words, making each prediction take a larger amount of time to be computed.

All networks were implemented and trained using Keras [17], using Tensorflow as backend. The predictions for the ENN were computed using the same networks trained on Keras with polynomial approximations, implemented in C++ using the SEAL library [50].

In all the experiments the first and second Fully Connected Layers (FCs) had 120 and 50 neurons,

respectively, while the last FC layer only had 1 neuron. The values used for dropout probability were found through random search: 0.3746 and 0.5838 for the Cold; 0.092 and 0.209 for Depression; 0.877 and 0.246 for Parkinson's Disease.

While training for classification tasks, it was noted that the Cold and Depression training partitions had a large misrepresentation of the subjects that presented the respective condition. To balance this, weights were attributed to each class. For Depression a weight of 0.8 was attributed to positive samples, and 0.2 was given to negative samples. In Cold, the difference was larger, thus we gave weights of 0.9 and 0.1 to the positive and negative samples, respectively.

All models were trained with a learning rate of 0.02, 100 epochs with early stopping, and a weight decay of 0.005, using RMSProp, an adaptive learning rate backpropagation algorithm implemented in Keras. As loss functions we used BCE for classification and Mean Squared Error (MSE) for regression. To keep track of the performance of each system during training, in regression tasks, the Mean Average Error (MAE) was computed. For classification tasks, due to class unbalance, the Unweighted Average Recall (UAR) function was implemented in Keras, and also computed during training.

3.3.3 Encryption Parameters

For the reasons stated in Section 2.2.1.D, encryption parameters define the amount of operations that can be performed on each Ciphertext, while it can still be decrypted correctly. Aside from this, encryption parameters also define the security level of the scheme. For our tests we used a polynomial modulus of 8192, and a plaintext modulus of 2^{30}. SEAL allows us to select a default coefficient modulus based on the value of the polynomial modulus, and the security level needed. For the value of this parameter we used the default value for a security level of 128 bits, and the polynomial modulus referenced above.

In addition, to encode real numbers we used SEAL's Fractional Encoder (Appendix A), with a polynomial expansion base of 3, selecting 64 coefficients for the integer part of the value and 32 coefficients for the fractional part.

With these encryption parameters our implementation of the network takes on average 750 ms to encrypt the feature vector, 4.5 seconds to compute a single prediction, and less than 5ms to decrypt the result, using 24 Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz processors, taking advantage of SEAL's multithreading capabilities. It is worth noting that the value for the plaintext modulus is very high, which takes a large toll on the noise budget. Reducing it would allow us to decrease the polynomial modulus to 4096, which would make a single prediction take about 1.2 seconds. However, having a lower plaintext modulus causes the decrypted predictions to have a larger associated error (Appendix A), which leads to accuracy degradation. For this reason, we decided to use higher parameters, leading to a slower but more accurate encrypted network.

3.4 Results

In this section, the results relative to each experiment are presented. Each table contains the results obtained for both the regular NN and the ENN with the activation functions replaced with their polynomial approximations. Additionally, each table also contains the baseline results of the challenge corresponding to each task, with the exception of the Depression task, where we have two tables, one for results at the segment level and another at the interview level. The reason for this is that the baseline results in the AVEC 2016 challenge are at the interview level, while the results for the other tasks are relative to the segment level, thus, for coherence, for Depression we present results at both levels [1]. Additionally, all results are relative to the Development set of each corpus.

Each classification task challenge uses its individual metric, be it the UAR, used in Interspeech's 2017 ComParE Challenge, or the three metrics used for 2016's AVEC, the F1 Score, Precision and Recall. All four metrics are included for both tasks, first for comparison purposes and also because having the F1 Score, Precision and Recall for each class is more comprehensive, as it allows for a better understanding of how the model is behaving. When referring to the F1 Score, Precision and Recall, the values outside of the brackets are relative to the positive class (presence of the condition), while the ones corresponding to the negative class are inside brackets. The metrics for AVEC's regression baseline are the RMSE and the MAE. For Interspeech's 2015 ComParE Parkinson's challenge the metric is Spearman's Correlation Coefficient (ρ), but we included the RMSE and MAE as well, for comparison purposes.

3.4.1 Cold

In Table 3.1 we have the results for the Cold classification task, together with the baseline for Interspeech's 2017 ComParE Challenge [81]. While both networks have a better performance than the baseline, there is a slight difference between the NN and the ENN. This is likely due to the polynomial activation functions, which have unbounded derivatives, as well as the output activation layer, which in the case of the NN is a Sigmoid, which also introduces further nonlinearity to the network, while in the ENN it is a linear polynomial, which does not introduce nonlinearity into the network.

Method	UAR(%)	F1 Score	Precision	Recall
Challenge Baseline	66.1	-	-	-
NN	66.9	.279 (.687)	.169 (.959)	.803 (.535)
ENN	66.7	.278 (.687)	.168 (.958)	.799 (.535)

Table 3.1: Baseline and results obtained for Cold classification.

[1] For disambiguation, a segment corresponds to an individual file.

3.4.2 Depression

The results of the Depression task are shown in Tables 3.2 and 3.4. In Table 3.2 we can observe that there is a slight performance degradation from the NN to the ENN; as before, in the experiments with Cold, this performance degradation is most likely to the unbounded derivatives of the polynomial activation functions in the network together with the fact that the output activation layer is a linear polynomial that does not introduce additional nonlinearities into the network. For the assessment of the degree of Depression we have our results presented in Table 3.4. In this case the ENN performs better than the NN in both metrics.

For the reasons stated in the beginning of this section we included the baseline for this challenge, 2016's AVEC [89], in separate tables, Table 3.3 and Table 3.5, where the results have been aggregated using a simple average.

In Table 3.3, contrary to what was obtained at the segment level, the ENN has a higher UAR when compared to the NN. Although this might be counter intuitive, considering the ENN has a lower UAR than the NN at the segment level, if we analyze the values of Precision and Recall for each class in Table 3.2, we can notice that for the ENN the results for each class are more balanced than in the NN, which may lead to the improvement of the global results at the interview level. In this case the ENN outperforms the baseline, and the NN has a slight performance degradation. For regression, our results are very similar to those of the baseline, with the ENN again outperforming both the baseline and the NN, and the NN slightly under-performing when compared to the baseline.

Method	UAR(%)	F1 Score	Precision	Recall
NN	60.6	.586 (.515)	.454 (.782)	.827 (.384)
ENN	60.2	.541 (.642)	.480 (.713)	.621 (.584)

Table 3.2: Results obtained for Depression classification at the Segment level.

Method	UAR(%)	F1 Score	Precision	Recall
Baseline	69.9	.462 (.682)	.316 (.938)	.857 (.540)
NN	65.2	.600 (.470)	.428 (1.00)	1.00 (.304)
ENN	74.3	.667 (.750)	.556 (.882)	.833 (.652)

Table 3.3: Results obtained for Depression classification at the Interview level.

3.4.3 Parkinson's Disease

The results regarding Parkinson's Disease are presented in Table 3.6. As in the case of the assessment of Depression, here the ENN performs better than the NN in the RMSE and MAE metrics,

Method	RMSE	MAE
NN	7.43	5.80
ENN	6.77	5.64

Table 3.4: Results obtained for Depression severity at the Segment Level.

Method	RMSE	MAE
Baseline	6.74	5.36
NN	6.84	5.50
ENN	6.37	5.35

Table 3.5: Results obtained for Depression severity at the Interview Level.

but the NN performs better than both the ENN and the baseline in the challenge's metric, Spearman's Correlation Coefficient ρ.

Method	RMSE	MAE	ρ
Baseline	-	-	0.492
NN	16.6	12.6	0.507
ENN	16.0	12.5	0.450

Table 3.6: Baseline and results obtained for Parkinson's Disease.

3.5 Discussion

In this chapter we have provided a proof-of-concept on how we can adapt an NN to HE, to create a secure framework for paralinguistic health-related speech applications.

Our results show that there's little to no performance degradation between the original network and the ENN, which proves the validity of this approach, however this framework still has some limitations. The size of the network is constrained by the computational complexity of the LHE scheme, as is the number of activation functions. To use a larger network, and more importantly, a network with more activation functions, would require higher encryption parameters, which would in turn increase the computational complexity of the scheme. Furthermore, for deeper networks it's unclear how the polynomial activation functions would affect the performance gap between the ENN and the NN. Very recent works are starting to address this problem [79] [9], using schemes in which bootstrapping is much more efficient, which in turn allows the computational complexity of the scheme to be independent of the depth of the network. Additionally these schemes allow for comparisons to be made between Ciphertexts, which enables the use of piece-wise functions, such as the Sign and the ReLU. Nevertheless, these are still very recent, and they still do not provide the same level of accuracy and versatility achievable with our approach. Finally, the framework we presented in Figure 3.1 requires the client to compute the feature

vector. This not only puts part of the computation the service provider should perform on the user's side, as it reveals some information on the model to the user. In many applications, the features used for a model required a large amount of investment, both monetarily and in terms of expertise, to develop, and allowing the client to have access to which features are being computed may present itself a drawback in the use of this framework for MLaaS providers. For this reason, in the next chapter, we will further explore this issue, and how it can be avoided.

4

Privacy-preserving End-to-End Convolutional Neural Networks

Contents

In this chapter, end-to-end approaches for secure paralinguistic speech tasks are explored. First in Sections 4.1 and 4.2 we provide the motivation and a brief overview on these methods. In Section 4.3 we explore how end-to-end networks can be adapted to HE along with techniques to aggregate predictions belonging to the same utterance. Sections 4.4 and 4.5 include the experimental setup and results obtained in our experiments, respectively. Finally, in Section 4.6 we discuss this approach and its suitability as a real world application.

4.1 Introduction

Recent years have seen the development of end-to-end Neural Network (NN) architectures for speech applications [86] [88] [37] [78]. Similar to what is accomplished in Image Processing, where raw images are given as inputs to a Convolutional Neural Network (CNN), the idea behind end-to-end schemes is to avoid using specialized feature sets and instead use raw audio, or some other low-level representation of the audio signal (i.e. Spectrograms and Log-Mel-Filterbank Energies (FBEs)), so that the system can learn the most relevant features for the task at hand on its own.

In Chapter 3 we described a framework (Section 3.1, Figure 3.1) that allows the development of secure MLaaS applications based on speech. Though useful, this framework has several shortcomings, as was discussed in Section 3.5, one of which being the amount of pre-processing that the client needs to perform before sending their encrypted data to the service provider. Aside from the computational expense, having the client to perform feature extraction discloses some information on the model. When considering a customized feature set, that was designed specifically for the task at hand, requiring a large amount of expertise and investment from the service provider in its development, it is clear how having to disclose it may dissuade service providers from using this framework.

The motivation behind this chapter is thus to determine whether end-to-end architectures can be applied in the context of HE, in order to, on one hand, reduce the amount of pre-processing the user needs to perform before encrypting the data, and on the other hand to avoid the need to develop and distribute specialized feature sets for each task.

4.2 End-to-End Convolutional Neural Networks

Taking advantage of the characteristics of Convolutional and Recurrent layers, recent works in speech processing, for tasks such as Automatic Speech Recognition (ASR) as well as in paralinguistic tasks such as emotion recognition, suggest the use of Convolutional Long Short-Term Memory Deep Neural Networks (CLDNNs) for end-to-end systems [86] [88] [78]. These networks are composed of three separate stages:

- In the first stage, a sequential input is fed to a Convolutional Neural Network (CNN). This CNN, composed by sequences of time/frequency convolutional, activation and pooling layers, can be regarded as the feature extractor. Convolutional layers are able to learn operations that produce high-level feature representations of the input, similar to those performed during hand-engineered feature extraction.

- The second stage, the Recurrent Neural Network (RNN), is composed of one or more Long Short-Term Memory Layers (LSTMs), and takes advantage of the memory-like capabilities of these layers to model time-domain characteristics of the signal. An example of this is stated in Trigeorgis et al. [86], where it is shown that the output of some units of the LSTM layer closely follow the time variation of features like pitch and loudness.

- The third and final stage is composed of one or more FC layers, that take the output of the RNN stage, and compute a prediction for each frame of the input sequence.

Nevertheless, when considering the constraints of HE, CLDNNs raise a major issue: the number of operations required by the LSTM layers. These, as all recurrent layers, encompass a large amount of operations, together with several activation functions that, if replaced by 2^{nd} degree polynomials, would result in a multiplicative depth of 4, for each pass through the layer. Furthermore, for a recurrent layer to be indeed recurrent, its output has to be fed to itself at least one time, which means that this layer would have a multiplicative depth of at least 8, the double of a network with two 2^{nd} degree polynomial activation functions, such as the one used in Chapter 3. Even though this can be done, computing a recurrent layer in a LHE setting would require very large encryption parameters that would render the network computationally unfeasible. Moreover, limiting the size of a time-sequence input to only two would discard the time-domain modelling capabilities of the LSTM. Consequently, recurrent layers create a series of limitations that make it unsuitable for LHE.

Alternatively one may drop the recurrent stage and simply use a CNN. In this way, although the network will not have a layer to deal with time-sequence inputs, as long as each input of the network contains enough time-domain contextual information, the network will still be able to make correct predictions. Nevertheless, this entails developing a solution to deal with variable length inputs. One such solution could be to cut and pad files so as to have all files with the same fixed size. Alternatively, files could be split into several smaller fixed-size frames. However, this would demand an additional strategy to combine the predictions for each frame into a final prediction for the whole utterance. Additionally, when considering a CNN with raw audio inputs, the size of each segment (or individual input) must be carefully selected. With very small segments the network will not have enough contextual information to properly train and make correct predictions, and too large segments will result in memory constraints, particularly when considering a HE framework, in which each sample is a Ciphertext. One can consider

as a short example a raw audio segment of 1 second at a sampling rate of 16 kHz. Supposing a Ciphertext encrypted with a polynomial modulus of 8192, with a coefficient modulus of 218 bits (default value in SEAL (v2.3.1) for 128-bit security), a vector of size 16000 samples will occupy 16000*8192*218 bits = 3.33 GB. Supposing we have a Convolutional layer with 16 feature maps that keeps the size of the input, the output of this layer would be 16*3.33 = 53 GB. This would not only poses memory constraints on the system running the prediction, but also is impractical from the point of view of communication between the user and the service provider. Moreover, the required amount of operations a Convolutional layer would perform on a vector of this size would greatly increase the time it would take to perform a single prediction.

Systems that receive time-frequency representations such as Spectrograms and Log-Mel-Filterbank Energies (FBEs) as inputs are thus more adequate for HE. This approach has the advantage that both the window size and shift with which these features are computed can be selected, which allows the user to determine the best trade-off between the time-frequency resolution and the time-span covered by the feature set. Although this approach is not entirely end-to-end, it still fulfills the objectives defined in the beginning of the chapter, as the input is not a specialized feature set developed specifically for the task at hand, and it only requires a minimal amount of pre-processing.

Several works have used this approach for paralinguistic tasks, including Mao et al. [55], Milde et al. [57] and Neumann et al. [60].

Mao et al. [55] trained a CNN to learn salient features for Speech Emotion Recognition (SER), using audio spectrograms, and showed how these features can be used to train an SVM, outperforming common standardized feature representations. Even though they did not train a full network that performs both feature extraction and classification, this work is important since it shows the modelling capabilities of CNNs with respect to paralinguistic speech.

Milde et al. [57] used a CNN to classify the eating condition of a given utterance. To this end, 40 FBEs were computed, using 25 ms long windows, which were then combined in sets of 11 vectors. Since each file was split into several FBEs frames, the authors proposed the use of a weighted voting strategy, based on Ridge Regression, to combine the predictions for each frame.

Neumann et al. [60] studied the performance of various feature sets (FBEs, MFCCs, a prosodic feature set and *eGeMAPS*), for emotion detection. The input of the CNN was a fixed-size set of feature vectors, that were computed with overlapping time-frames. To determine the most relevant time-frames, this study made use of an attention mechanism. However, this technique only contributed with a small performance gain. Furthermore, in this work, feature vectors were computed from 7.5 seconds long utterances, at every 10 ms, with 25 ms windows. This means that this feature matrix had a dimensionality of 750 times the length of the feature vector. Since the smallest feature vector contained 13 features, the feature matrix would have at least 9,750 features. Considering what was discussed previously regarding

the dimensionality of the inputs of the network, it is evident how this approach would also be unfeasible in HE.

For our experiments, the approach of Milde et al. [57] was chosen, as not only do FBEs require little pre-processing to be computed, but small arrays of FBEs still cover a time-span large enough for the network to be able to make correct predictions. Furthermore, using Ridge Regression allowed us to combine predictions for several frames, and consequently use smaller frames, with sizes more compatible with the restrictions of HE. Additionally, since Ridge Regression is only a linear transformation of the original features, in a HE context it is straightforward to perform, and it can be done in parallel with a prediction of a CNN.

4.3 Encrypted Convolutional Neural Networks

4.3.1 Adaptation to HE

As was explained in Chapter 3, to convert an NN into an Encrypted Neural Network (ENN) one needs to replace all mathematical operations by their HE counterparts. This means that all nonlinear functions need to be substituted by either polynomials or linear functions. For CNNs, all the techniques presented in Chapter 3 are applicable, but due to the nature of some of its most common layers, further adaptation is required.

Convolutional Layers can be considered weighted sums, requiring additions and multiplications to be performed and, therefore, it is straightforward to adapt them to HE. Additionally, CNNs, as described in Chapter 2 Section 2.1.2, often contain Pooling layers, that are used to reduce the dimensionality of their input and to highlight relevant features. The most commonly used Pooling layer is the Max Pooling Layer, which takes a part of the input feature map and selects the highest feature within it. However, this is a nonlinear operation, that requires comparisons to be made. Alternatively a more HE-friendly layer can be used, namely the Average Pooling Layer, which is essentially a weighted sum and, therefore, simple to adapt to HE.

Similar approaches were taken in works such as Cryptonets [31], Chabanne et al. [14] and Hesamifard et al. [39]. Nevertheless, while Chabanne et al. also replaced the Max Pooling Layer with an Average Pooling Layer, Cryptonets and Hesamifard et al. replaced it with a scaled sum. The scaled sum is the same sum that would be applied in average pooling but without dividing the final result by the number of features being pooled. Hesamifard et al. state that this was done to avoid the impact this operation would have on the multiplicative depth of the circuit, while Cryptonets pointed out that the scaled average is easy to compute in HE and considering that the only consequence from this is a scaled output, it will not affect the network's performance in any relevant way. Since regular average pooling only amounts in one additional HE plain multiplication for each feature, and as this layer is already implemented in most

NN libraries, we decided to use it in our networks.

4.3.2 Weighted Prediction Strategies

Following the reasons presented in 4.2, the input to a HE friendly CNN, will need to be a small sized low-level representation of the original signal. This entails that there will be several predictions for each audio segment. Since LSTMs cannot be used in this context, some other form of combining the predictions of several frames is required.

A simple way to aggregate predictions for the same file is to compute their average, and use the result as the final prediction. Another similarly straightforward method would be to use a majority vote. This method, only useful for classification tasks, consists of taking the predictions regarding each file, adding them, and then selecting the class with the highest count as the final prediction. Both methods, albeit useful, may not be enough in situations where each prediction corresponds to a very small temporal frame, and further contextual information is necessary. One alternative, for frames with different lengths, would be to use the frame's length as a weight, and perform a weighted average. Furthermore, one could try to determine the importance of each frame, for the class that it is being tested for, as was proposed in [57]. In this method, after training the CNN, a Ridge Regression model is trained for each class, using the CNN's input features of the corresponding class, with labels defined as:

$$y = \begin{cases} 0, & \text{if the prediction is incorrect} \\ 1, & \text{if the prediction is correct} \end{cases} \tag{4.1}$$

The motivation behind 4.1 is that some frames might not contain relevant information for the class at hand (i.e. might contain silences or noise) which will cause the network to output a prediction that although correct for the frame at hand, it will not match the label of the overall utterance being classified. Using labels in this manner will allow the regressor to learn the characteristics that make a frame relevant or not for each of the classes. The output of each regressor can then be used to weigh the prediction of each class. Following this step, a final Weighted Majority Vote can be computed for each class.

4.4 Experimental Setup

The models used in our experiments follow the network architecture presented in Figure 4.1, in which the acronym *Conv* corresponds to a Convolutional Layer, *BN* to a Batch Normalization Layer, *Act* to an Activation Layer, *FC* to a Fully Connected Layer and *AP* to an Average Pooling Layer. This network is composed of two Convolutional layers, each followed by a BN layer, an Activation layer and an Average Pooling Layer (AP) layer. The output of the last Convolutional layer is then turned into a vector, to enter the DNN stage of the network. This stage was composed of two FC layers, each followed by a BN and

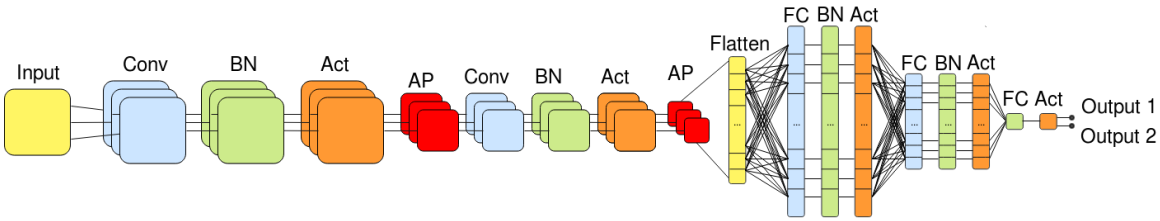


Figure 4.1: Convolutional Neural Network Architecture

an Activation layer, which in turn are followed by an output FC layer. For classification tasks, a BN and an activation layer was added after the last fully connected layer.

All experiments were implemented in Keras with Tensorflow backend and RMSprop with default parameters as the backpropagation algorithm, and were performed on the URTIC corpus [48], detailed in Appendix B.

4.4.1 Raw Audio

To establish a baseline for the CNN described in Section 4.2, a first experiment was conducted using raw audio feature vectors. Each file was split into 3-second long segments [1]. Smaller segments resulting from splitting the audio files were zero-padded. Since the recordings were performed with a sampling rate of 16 kHz, each segment includes 48,000 samples. Additionally, each segment was zero-centered and normalized to unit variance using its own mean and variance.

To aggregate predictions for each segment, a weighted average was employed, using the original size of each segment as weight. In this way, samples that include a large number of zeros due to padding contributed less than others to the final result.

The weighted voting schemes, described in Section 4.3.2, were not considered for this experiment, since they are tailored specifically to determine if a small frame is relevant or not to the prediction, whereas this experiment deals with large audio segments, that can be considered full utterances, and as such, there is no need to determine whether they are relevant or not, rendering this approach ineffective.

The network employed for this experiment was trained with Binary Cross-Entropy, a learning rate of 0.001 and 500 epochs, together with early stopping. The remaining network parameters used were as follows: the first convolutional layer included 20 one-dimensional filters of shape (1,80). The first average pooling layer is a time-dimension and filter-dimension pooling layer, having a pooling filter and a stride with shape (2,10). The second convolutional layer is composed of 40 two-dimensional filters with shape (160,2). The second average pooling layer has a pooling filter of shape (20,10), and equal stride. Each

[1] The minimum length of an audio sample in the URTIC corpus

of the following two FC layers consists of 200 units. The final FC layer is composed of only one unit. Before each FC layer, a Dropout layer was inserted during training, with a probability of dropout of 0.5.

4.4.2 Log-Mel Filterbank Energies

As explained in Sections 4.2 and 4.3, raw audio is not a suitable format for HE. For this reason, FBEs were chosen as the alternative, using the approach suggested by [57], together with the average and weighted voting strategies described in 4.3.2. Vectors of 40 log-Mel Filterbank Energies were computed from each file with 30 ms windows and 10 ms shift, using OpenSMILE [28]. From each of the resulting vectors, 11 x 40 feature maps were created with the ± 5 feature vectors around it. All data was zero-centered and normalized to unit variance using the mean and variance of the training set.

In order to combine predictions for frames of the same original file, the two majority voting techniques described in 4.3.2 were used, plus a simple average, to allow for a fair comparison between these methods.

This model was trained with Categorical Cross-Entropy as the loss function, 0.01 learning rate, 500 epochs, a weight decay of 0.001 and early stopping. The remaining parameters were as follows: the first and second convolutional layers were composed of 32 and 64 two-dimensional filters with shape (3,3) and (2,2), respectively. Furthermore each Average Pooling layer had a pool-size and strides with shape (2,2). Each of the first two FC layers was composed of 500 units and the third of two output units. Before each FC layer, a Dropout layer was inserted during training with 0.5 dropout probability.

It is important to note that contrarily to what was done in the previous chapter, here the network has two outputs, as the use of Majority Voting with Ridge Regression implies weighing the result of each class for the same frame with separate weights. Furthermore, this allows us to train the network with an output Sigmoid activation layer. When using HE this layer can simply be left out, as it will not affect which output is larger.

In addition, we assume that the prediction combination techniques can be performed by the user. Although it only requires one multiplication, computing Ridge Regression with HE would add an extra multiplicative layer to the computation, which might demand an increase in the encryption parameters. Furthermore, this computation is relatively cheap, computationally, and only leaks information about the Ridge Regressor to the user, giving him no extra information about the model.

4.4.3 Encryption Parameters

For the experiments performed using encrypted features, the networks described in the previous subsections were implemented using SEAL, with both weights and features encoded with the library's Fractional Encoding scheme, with an expansion base of 3, using 16 coefficients for both the integer

and the fractional part. The values assigned to the encryption parameters were as follows: 16,384 for the polynomial modulus, 9,147,936,743,096,321 for the plaintext modulus and finally, the library's default value for the aforementioned polynomial modulus and 128-bit security was used for the coefficient modulus. Predictions were computed using multithreading with 24 Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz processors, in a machine with 250 GB of RAM.

4.4.4 Encrypted Network Performance

In Chapter 3, Section 3.3, we stated how our network took on average 4.5 seconds to perform a prediction. However, for the network implemented in this chapter, which is almost twice as large as the one used in Chapter 3, a single prediction took 36 minutes. Considering the time it takes to compute a single prediction, and that the development partition of the URTIC corpus, with each file split into frames, is composed of a total of 2,624,391 frames, it would be unfeasible to compute predictions for each one. For this reason, all the results we present in Section 4.5 were computed with unencrypted data.

4.5 Results

In this section we present our results for the three experiments described in Section 4.4, together with a detailed discussion on the performance of the ECNNs implemented with SEAL. Regarding the experiments done with the URTIC Corpus, and unlike what was done in Chapter 3, where we displayed results for each individual class, here we give the average result for both classes, in percentage. This is done to favour the intelligibility of the results, in order to facilitate the comparison between different systems. For the reasons stated in Subsection 4.4.4, all the results presented in this section were computed using unencrypted data.

4.5.1 Raw Audio

Table 4.1 presents our results for the experiments conducted with Raw Audio, together with the baseline obtained in Chapter 3, using an NN with eGeMAPS features. Although the dimensions of the input might make it unfeasible when applied together with HE, we also present unencrypted results for a CNN, where the Activation functions were replaced by the polynomials described in Chapter 3. For coherence with the results in the other tables, we kept the same nomenclature, and named this network an ECNN, albeit the results were not computed in an encrypted setting. All results correspond to the utterance level, and were obtained by applying the Weighted Average strategy described in Section 4.4.1, to frame-level predictions.

This table shows that whilst the UAR is never as high as the one obtained in the baseline, it is very close in the case of the CNN. In addition both the average Precision and F1 Score are significantly higher. The increase in the F1 score means that although the UAR is not as high, the values for Recall and Precision are more balanced in each. For the ECNN the same conclusions apply, but there is some performance degradation over all three metrics, with Precision taking the largest toll.

Method	UAR(%)	F1 Score (%)	Precision (%)
NN	66.9	48.3	56.4
CNN	66.1	56.3	70.7
ECNN	65.0	55.9	66.7

Table 4.1: Results obtained using Raw Audio

4.5.2 Log-Mel Filterbank Energies

In Table 4.2 we present the results for the NN baseline and those obtained with CNNs that receive FBEs as inputs. Additionally, we also display the results for the methods described in Section 4.3.2, to compare the different strategies. Regarding nomenclature, SA corresponds to Simple Average, MV to Majority Vote and WMV to Weighted Majority Vote, using the weights computed with Ridge Regression. As in the previous subsection, the ECNN corresponds to the same network as the CNN, but with the Activation functions replaced by their polynomial approximations.

Our results show that, although the UAR is much lower than the baseline, and the Precision is only slightly lower, the F1 Score has a significant increase. As in the experiment performed with Raw audio, this means that the Recall and Precision for each class are much more balanced than those of the baseline. At the utterance level, while the UAR only presents a small performance increase, both the Precision and the F1 Score increase significantly when compared to the baseline and the results at the frame level. Comparing the prediction aggregation methods, there is no significant difference from the simple Majority Vote and the Simple Average, but using the Weighted Majority increases both the UAR and the F1 Score, while the Precision has a small drop.

In Table 4.3 we present the results obtained using the same CNN but with Polynomial Activation functions, together with the baseline obtained in Chapter 3, for the ENN. As in the regular CNN, there is a large degradation in the UAR, and both the F1 Score and the Precision overpass those of the baseline in all strategies. The best value for the UAR is obtained with the simple Majority Voting strategy, contrarily to the CNN, where the best value for this metric is obtained using the Weighted Voting Strategy. The F1 Score for the ECNNs is in overall higher than for the CNN, meaning the results for each class are more balanced, but both the UAR and the Precision are smaller in all strategies.

In addition, the results obtained with FBEs are never as high as the ones obtained with Raw audio. This was to be expected, since FBEs represent only a small fraction of the whole utterance, whereas Raw audio has a much higher amount of contextual information.

Method	UAR(%)	F1 Score(%)	Precision(%)
NN Baseline	66.9	48.3	56.4
CNN + SA	60.7	61.7	63.0
CNN + MV	60.8	61.7	63.0
CNN + WMV	62.6	62.2	61.8

Table 4.2: Results obtained using FBEs.

Method	UAR(%)	F1 Score(%)	Precision(%)
ENN Baseline	66.7	48.3	56.3
ECNN + SA	59.1	64.0	60.4
ECNN + MV	59.2	64.4	60.5
ECNN + WMV	57.4	63.3	58.2

Table 4.3: Results obtained using FBEs with Polynomial Activation functions.

4.6 Discussion

In this chapter we addressed the issue of implementing a secure end-to-end framework based on speech, using a CNN. This framework aimed to reduce the amount of computation to be performed on the user's side to a minimum, while preventing the user from obtaining information about the model. We showed how Raw Audio is an unsuitable format to work with HE, and why low-level representations such as FBEs and time-frequency Spectrograms are a trade-off between computational cost and dimensionality. Additionally, from the results obtained it was possible to conclude that using a CNN with FBEs yields more balanced models at frame and utterance levels. However, in the case of the experiments performed with the URTIC corpus, if we only consider the metric chosen for Interspeech's 2017 ComParE Challenge, none of the approaches in this chapter are able to reach the baseline. Regardless, CNNs entail a larger set of parameters than NNs, and thus require larger amounts of training data, which was not the case for the datasets used in this chapter. The lack of training data, however, could have been partially attenuated with the use of out-of-domain data. As was suggested by Milde et al. [57], pre-training a network with a larger dataset for a similar task would have allowed the network to learn similar high-level representations of the data, that could later be adjusted to the task at hand, by re-training it

with the dataset corresponding to it. Furthermore, our results display a clear degradation in performance for networks that use polynomial activation functions, when compared with their non-polynomial counterparts. Although the previous chapter showed that for small networks there is no noticeable degradation from NNs to ENNs, since the networks employed in the experiments of this chapter were deeper, and included more activation functions, when replacing these with polynomial functions, that have unbounded derivatives, some instability might have been introduced during training, which may have lead to this degradation.

Moreover, the encrypted predictions made with the ECNNs had most times incorrect values. Since each of the layers was tested individually, yielding correct results, we concluded that what was causing the incorrect results was the Fractional Encoding and decoding procedure. As is described in Appendix C, SEAL's Fractional Encoding scheme takes the base B expansion of each value, and encodes the integer part of the value in the lowest degree coefficients of the Plaintext polynomial, whereas the fractional part is encoded in the highest degree coefficients of the polynomial. With each multiplication the base B representation of each part of the value grows to occupy further coefficients, and if either part grows too large, it may reach the coefficients of the polynomial reserved for the other part, rendering the decoded result meaningless. Additionally, if one of the coefficients reaches a value larger than the plaintext modulus *t*, it will be reduced modulo *t*, and the result will be incorrect. Nonetheless, this phenomenon is not only present in this type of encoding, but has to be taken into account in all techniques that encode a message into a polynomial. Due to the large amount of multiplications required to compute a prediction in the ECNNs we assume that both of these reasons may have contributed to create wrong results. This could be averted with the use of the Integer Encoder also present in this library. Since it only encodes the integer part of a value, using it would remove the possibility of overlapping occurring. However, to use this encoder we would need to scale every value in order to turn it into an integer and an even larger plaintext modulus would need to be selected, to insure reduction modulo the plaintext modulus would not occur. Moreover, the growth of the scale factor would also have to be taken into consideration due to the 2^{nd} degree polynomial activation functions.

On a final note, the results presented in this chapter were obtained through the use of networks that were able to attain a reasonable performance while still being relatively small. Considering the fact that the inputs of these models can be considered small frames, and that it is necessary to combine them after each has been computed, seeing that a single encrypted prediction takes 36 minutes to be computed, the strategies discussed in this chapter do not seem feasible in a real world setting. However, if all predictions could be computed at the same time, this framework would become somewhat more viable. As it will be seen in Chapter 5, this is possible in some HE schemes, but it entails further restrictions that need to be addressed before applying it to NNs.

5

Privacy-preserving Discretized Neural Networks

Contents

In this chapter we explore how to adapt an NN to HE batching. The chapter starts with a debate over the reasons that impelled it (Section 5.1). In Section 5.2 similar works and techniques used for HE batching and recent developments in Quantized Neural Networks (QNNs) and DiNNs are exposed. Section 5.3 provides the experimental setup along with details on the implementation of the encrypted network, and the feature quantization technique used. Finally, in Sections 5.4 and 5.5 we present and discuss our results, respectively.

5.1 Introduction

In Section 4.6 we discussed how having several values be operated on at the same time could be useful in situations where several predictions are to be made regarding one single file. This is a characteristic shared by many Speech Processing tasks, where single utterances are split into several frames, due to their variable (and usually high) dimensionality, as well as to trade-off between precision and the time-span covered by the features computed from them.

Some RLWE and LWE based cryptographic HE schemes allow several values to be batched into a single ciphertext, which can be operated on as *SIMD (Single Instruction, Multiple Data)*. This technique, called *batching*, takes advantage of the fact that a ciphertext in this type of scheme is a polynomial, which, under certain conditions, can be factored into several polynomials. Using this property, it is possible to encode an independent value into each of the factored polynomials, and applying homomorphic operations to the original ciphertext will be the equivalent of applying each operation to each encoded value. As such, factoring this polynomial will yield a set of results equivalent to applying these operations individually, to each of the encoded values. Since most operations on NNs are vectorizable, batching would allow several encrypted predictions to be made in the time it takes to compute one, making it a technique especially suited for speech-based machine learning tasks.

RLWE and LWE cryptographic schemes are typically built over integer sets. Nevertheless, it is possible to encrypt real-valued integers, using encoding schemes, as was done in Chapters 3 and 4. However, these encoders are not compatible with *batching*, and for this reason, to use this technique, it is necessary that the values to be encrypted are integers. On the other hand, for most applications both the weights and inputs of an NN are real numbers, and cannot be used directly with HE. The goal of this chapter is thus to determine how a network and its inputs can be discretized, in order to make them compatible with *batching*.

5.2 Discretized Neural Networks

As stated in the previous section, to use batching, both the weights and the inputs of a network need to be integers. Works such as Cryptonets [31] and Hesamifard et al. [39] also make use of batching techniques. In the case of Cryptonets, it is stated that their network is discretized by scaling the weights up to a fixed precision. In Hesamifard et al., the authors are omissive on this matter, but it can be deduced from the encryption parameters that this is also the case.

In a different approach, some works have adapted techniques developed in a recent line of research in machine learning which is pushing towards the *quantization* of NNs. This line of work aims to reduce the amount of space and power required to run NNs, with the objective of deploying these networks in mobile devices, as well as devices with restrictions, whether they are power, memory or computational [59].

To do this, some researchers have explored ways of computing the training and inference stages of an NN using quantized weights with fixed precision, reducing the amount of temporary memory as well as the computational overhead of operating over them [52] [16]. Others, such as Binarized Neural Networks (BNN) [20] [41], have pushed quantization to the extreme where the network's weights are only a binary (0 or 1) or ternary (-1, 0 or 1) value, for both the training and inference stages. At the same time, most have used some form of network pruning (included or not in the quantization strategy), by setting weights bellow a threshold to zero, and removing neurons, effectively reducing the size of the network and the number of operations to be computed during the inference stage [16]. While some of these works compute the inference stage, and even the training stage, with integer weights, these rely on being able to quantize or reduce the precision of the output of each layer, which requires nonlinear operations, as well as divisions, which are incompatible with HE [4] [44].

In the context of HE, encrypted Binarized Neural Networks have been implemented by [79], while [9] implemented an ENN with real valued weights directly converted to integers, although this approach gives origin to a large accuracy drop, as is stated by the authors. It is important to note that both these works use a different HE scheme than the one that has been used throughout this thesis, which is a LHE scheme. In their case, a FHE scheme is used, which does not allow batching. Having the above in mind, the approach chosen for this chapter was to simply scale each weight of the network by a factor. Nonetheless this poses some constraints. Since the Activation layers being used in our networks are 2^{nd} degree polynomials, if a weight is multiplied by a scaling factor, when it is propagated through the activation layer, the factor will be squared. Moreover the output of this layer will then be multiplied by the weights of the next layer, which are also scaled by a factor, meaning that the output of the second layer will be raised to the power of 3. It is easy to see how propagating a scaled value through the network, results in a very rapid growth for the scaling factor. Furthermore, if the values grow too large, in a HE setting, they can be reduced modulo the plaintext modulus, creating incorrect results. Thus the scaling

factor, as well as how each layer is scaled, has to be selected carefully, taking into account the maximum value allowed, and to avoid precision loss during inference.

Regarding the input features, if we consider multiplying them by a factor as well, it will result in another level of scaling, making the largest absolute values in the network grow even more, in addition to contributing to a greater loss in precision. Quantization schemes on the other hand allow for a safer trade-off between maximum absolute values and precision loss, and quantized features can be used to train models beforehand; thus, these can be employed directly into ENNs, without further modifications.

5.3 Experimental Setup

From what was discussed in the previous section, to discretize a network, it is necessary to find a suitable feature quantization method, as well as to scale the weights of each layer in such a way that the scaling factor grows as little as possible while it is propagated through the network. As such, we performed several experiments to determine the ideal number of quantization channels 2^n and scaling factor s. To do this, as a first step, a baseline was trained for an NN. Using the same architecture, several models were trained with different quantized inputs. Subsequently, the model that attained the best results was used to test different scaling factors, in order to determine which one achieved the best trade-off between model performance and the largest absolute value in the network.

Using the values obtained from these experiments, the networks described in Chapter 3, Section 3.3 were retrained and re-implemented in SEAL using batching.

5.3.1 Implementation

As discussed in Section 5.2, scaling the network's weights has to be done such that the scaling factor grows as little as possible. In Cryptonets [31], the authors mention that in order to improve the performance of their network, they collapse linear layers into a single layer. For the reasons stated in Chapter 3, Section 3.2, between each pair of FC (Equation 5.1) and Activation layers, our encrypted networks include a Batch Normalization Layer (Equation 5.2):

$$y_{fc} = Ax_{in} + b \tag{5.1}$$

$$y_{bn} = \gamma \frac{x_{bn} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{5.2}$$

Since both the FC and the BN layers are linear transformations, using the idea proposed in [31], considering all operations performed in the BN layer are element-wise, we can combine them to obtain Equation 5.3.

$$y = \gamma \frac{(Ax + b) - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} Ax + (\frac{\gamma}{\sqrt{\sigma^2 + \epsilon}}(b - \mu) + \beta) = A'x + b' \tag{5.3}$$

In this way, the input of the polynomial activation layer, will only be scaled by a factor s:

$$y_{fc+bn} = sA' \cdot x + sb' = A'_I \cdot x + b'_I \tag{5.4}$$

where the subscript I denotes integer values. However, the polynomial Activation layer (Equation 5.5), includes multiplicative constants, which also need to be scaled:

$$y_{act} \ = \ c_2 y_{fc+bn}^2 + c_1 y_{fc+bn} + c_0 \tag{5.5}$$

To avoid increasing the degree of the scaling factor, we can instead pre-multiply these constants by the weights of the previous layers, avoiding another level of scaling. Since the scaling factor will be squared in the 2^{nd} degree term of the polynomial activation function, only the constant c_2 needs to be pre-multiplied, as c_0 and c_1 need only to be multiplied by a scaling factor that yields the same degree for the scaling factor as the one resulting from the 2^{nd} degree term of the equation:

$$y_{fc+bn} = s\sqrt{c_2} A' \cdot x_{in} + s\sqrt{c_2} b' = A''_I \cdot x_{in} + b''_I \tag{5.6}$$

Additionally, since y_{fc+bn} is already multiplied by $\sqrt{c_2}$, the 1^{st} degree coefficient of the equation must be multiplied by $\frac{1}{\sqrt{c_2}}$:

$$\begin{aligned} y_{act} &= y_{fc+bn}^2 + (s\frac{c_1}{\sqrt{c_2}}) y_{fc+bn} + s^2 c_0 \\ &= y_{fc+bn}^2 + c_1^I y_{fc+bn} + c_0^I \end{aligned} \tag{5.7}$$

where the superscript I denotes integer constants.

Alternatively, instead of pre-multiplying the constant coefficients, it is possible to expand the squared term of the polynomial, and pre-compute every constant, obtaining:

$$\begin{aligned} y_{fc+bn+act} &= (s\sqrt{c_0} A' \cdot x_{in})^2 + (s^2(2c_0 b' + c_1)A') \cdot x_{in} + s^2(c_0 b'^2 + c_1 b' + c_2) \\ &= (sA'' \cdot x_{in})^2 + (s^2 B'') \cdot x_{in} + s^2 c'' \\ &= (A''_I \cdot x_{in})^2 + B''_I \cdot x_{in} + c''_I \end{aligned} \tag{5.8}$$

where A and B are matrices with the same dimensions.

However, this form requires two matrix dot products. Since both matrices have the same dimensions,

this will result in the double of the number of operations required by Equation 5.7.

From these equations, we get that the degree of the scaling factor at the exit of the first FC+BN+Act layer will be 3, for the second layer it will be 6, for the third layer 14, and so on. This puts a restriction in both the value of the scaling factor as well as on the depth of the network. In SEAL, the maximum value for the plaintext modulus t is 2^{60}. If we consider a scaling factor of 100, the depth of the network can be at most 2, otherwise during the encrypted inference values will undergo reduction modulo t. Additionally, the inputs of the network have to be as small as possible, as 100^{10} is larger than 2^{60}, and thus it has to be made sure that no calculation in the network reaches this value.

5.3.2 μ-Law Quantization

To quantize input features we used the μ-Law companding transformation, displayed in Equation 5.9, whose input is constrained to $[-1, 1]$. This algorithm applies a non-linear logarithmic transformation to the input signal. It is commonly used to digitally encode speech for 8-bit PCM (Pulse Code Modulation), as the logarithmic transformation is similar to the response of the human ear to the amplitude of a sound, where a higher resolution is given to sounds with low amplitude, while louder sounds, which saturate receptors, become less distinguishable.

$$f(x) = sign(x)\frac{ln(1+\mu|x|)}{ln(1+\mu)} \tag{5.9}$$

Since Equation 5.9 maps its inputs to real values, to discretize its output it is necessary to apply a quantization function that maps these values to the interval $[0, \mu - 1] \in \mathbb{Z}$, where μ is the number of quantization channels (i.e. the maximum number of different values), usually represented by the number of bits necessary to represent it in binary, n. By replacing n in $\mu = 2^n$, one can determine the maximum possible binary value represented with n bits. To do this, we applied the quantization function in Equation 5.10.

$$Q(x) = \left\lfloor \mu\frac{f(x)+1}{2} + \frac{1}{2} \right\rfloor \tag{5.10}$$

Additionally, to promote smaller values, each feature vector was zero-centered relative to its median value.

This approach was inspired by WaveNet's [90] use of the μ-Law algorithm in order to reduce the noise and variance of the input features of the network; although in their case the network was dealing with raw audio, the same reasoning can be applied to any continuous valued feature set.[1]

[1] Our implementation of the μ-law transformation was adapted from WaveNet's implementation, available in https://github.com/ibab/tensorflow-wavenet.

5.3.3 Language Verification Task - KALAKA-2 Dataset

To ensure the validity of the conclusions taken from the results of our experiments, in this chapter we decided to use an additional dataset, together with the ones used in the previous chapters. As has already been stated, the datasets employed in our previous experiments are very small, and even though what can be inferred from these is not necessarily incorrect, a larger dataset would add strength to the results obtained.

For this purpose we selected the KALAKA-2 corpus [76], described in Appendix B, to perform a Language Verification Task. The recordings in this dataset are divided into three partitions, Training, Development and Evaluation, with those in the Development and Evaluation sets divided into 3, 10 and 30 second length files. In addition the dataset is composed of 6 languages: Spanish, Catalan, Basque, Galician, Portuguese and English.

Our goal with this dataset is to assess the performance of models in different quantization and scaling conditions. As such, we only include results regarding clean and *closed-set* conditions.

This dataset is used in a language verification task, and results are reported in terms of Accuracy together with a metric developed specifically for this evaluation, which is defined as follows:

$$\begin{aligned} C_{avg} &= \frac{1}{L}\sum_{i=1}^{L}\{C_{miss}.P_{target}.P_{miss}(i) \\ &+ \sum_{\substack{j=1 \\ j\neq i}}^{L} C_{fa}.P_{non-target}.P_{fa}(i,j) \\ &+ C_{fa}.P_{OOS}.P_{fa}(i,0)\} \end{aligned} \tag{5.11}$$

where P_{miss} is the miss rate corresponding to a target language i, $P_{fa}(i,j)$ is the false alarm rate corresponding to a target language i and an erroneously accepted language j. Both the cost of false alarm (C_{fa}) and miss rate (C_{miss}) are equal to 1, and the prior probability of target languages (P_{target}) is equal to 0.5. The prior probability of non-target languages ($P_{non-target}$) and the prior probability of out-of-set languages are defined as follows:

$$P_{OOS} = \begin{cases} 0.0, & \text{closed-set condition} \\ 0.2, & \text{open-set condition} \end{cases} \tag{5.12}$$

$$P_{non-target} = \frac{1 - P_{target} - P_{OOS}}{L-1} \tag{5.13}$$

Since all experiments were performed with a closed set, this expression can be simplified as:

$$C_{avg} = \frac{1}{2L}\sum_{i=1}^{L}\{P_{miss}(i) + \sum_{\substack{j=1 \\ j\neq i}}^{L} P_{fa}(i,j)\} \tag{5.14}$$

To compute these metrics, we used MATLAB R2017b [56], together with the FoCal v1.0 package [12].

To train models with this dataset we followed the approach described in [1], and used a feature vector of Shifted Delta Cepstra (SDC) computed from 7 Perceptual Linear Prediction coefficients with log-RelAtive SpecTrAl speech processing (PLP-RASTA). First the PLP-RASTA were computed from each segment with 25 ms windows and 10 ms shift, using openSMILE [28]. The features obtained from each segment were then zero-centered and normalized to unit variance. The SDC features were computed from the PLP-RASTA, with a 7-1-3-7 configuration, resulting in a 56 dimensional feature vector. However, preliminary experiments showed that this vector did not obtain significant results, hence, groups of 10 feature vectors were used instead, resulting in a final vector of 560 components. Additionally, low energy frames were filtered out. Since each recording yielded several feature vectors, these were combined by computing the average log-posterior estimate of each class:

$$s = \frac{1}{N}\sum_{t=1}^{N} log(p(L_l|x_t,\theta)), \tag{5.15}$$

where s is the final score, N the number of frames of the recording, and $p(L_l|x_t,\theta)$ the posterior probability of language *l*, given the feature vector for time frame *t*, x_t, and the model θ, which corresponds to the output of the network. This approach is suggested by Gonzalez-Dominguez et al. [34], who used an NN to perform a similar Language Identification (LID) task.

5.3.3.A Neural Network Architecture and Training

Due to the limitations imposed by the scaling factor, our network had to be limited to two activation functions. To compensate for this fact, the network used in our experiments with the KALAKA-2 dataset, although following the same overall architecture of the one employed in Chapter 3, Figure 3.3, included an extra FC layer after each of the original FCs layers, with the exception of the last. Since these FC layers are linear, they can be easily collapsed, in a similar fashion to what was explained in Section 5.3.1. Additionally, we decided to add a Softmax Activation function, after the last FC layer. The remaining activation functions in the network are polynomials, as defined in Chapter 3, Equation 3.6.

Since the network includes two sets of FC+BN+Polynomial Activation, the final scaling factor will be raised to the power of 7. Consequently, the outputs of the network will need to be scaled down by s^7. As such, the final network architecture was composed of two sets of two FC layers, each with 2048 nodes, followed by a polynomial activation function. The final FC layer included 6 nodes (one for each language), and was followed by a Softmax Activation layer. Before each FC layer, with the exception of

the first, a Dropout layer was inserted with 0.5 dropout probability.

Although the Softmax cannot be implemented in HE, it helps increase the performance of the network. For this reason, trading-off between the performance of the network, the computational toll on the client's side and leaking some information about the model, we decided to include this layer, assuming that it is computed by the user, along with the log-posterior probabilities. For the same reasons, for the paralinguistic classification tasks, the polynomial approximation of the Sigmoid was replaced with the Sigmoid itself, and needs to be computed by the user.

This network was implemented and trained with Keras, using Tensorflow as backend, with early stopping, a learning rate of 0.0005, 5,000 epochs, the Categorical Cross-Entropy loss function, and using Adam with default parameters as the optimizer.

5.3.4 Encryption Parameters

For batching to work correctly it is necessary that the plaintext modulus t be a prime number, congruent to $1 \ mod \ 2n$, with n being the polynomial modulus. To account for the large values that arise from scaling the network's weights we chose a plaintext modulus of 1,137,721,855,872,860,161. This value is larger than 2^{59}, and fulfills the condition defined above. The polynomial modulus was chosen to be 16384, although the network in this experiment is not deeper than the one implemented in the first chapter, a larger polynomial modulus had to be selected, as a large plaintext modulus takes a large toll on the noise budget. Thus it was necessary to increase the polynomial modulus in order to have a noise budget large enough to compute the whole network. As in the previous chapters, for the coefficient modulus, SEAL's default value for a security level of 128 bits was selected. The encrypted predictions were computed using multithreading with 24 Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz processors, in a machine with 250 GB of RAM.

5.4 Results

In this section we present the results obtained in the experiments detailed in the previous section. We begin with an overview of the performance of the network described in Section 5.3, for the KALAKA-2 corpus, followed by the an analysis of the performance of the network with quantized inputs for a varying number of quantization channels. We then present results for different weight scaling factors, obtained with the best performing network of the previous experiment. Furthermore, we present results for networks retrained with quantized features, and scaled weights, for the corpora tested in Chapter 3 (URTIC, DAIC-WOZ and Spanish PD corpus). In all tables, the acronym QNN refers to results obtained with quantized features, and Scaled QNN to results obtained with networks scaled according to the method described in the previous section. Of these scaled results, only the ones present in Table 5.3 were computed using encrypted data, due to the size and computational expense of the network used for KALAKA-2. The baseline results obtained for KALAKA-2 are presented in Table 5.1.

Metric	Development			Evaluation		
	3 (s)	10 (s)	30 (s)	3 (s)	10 (s)	30 (s)
Accuracy (%)	66.5	81.1	87.2	57.0	68.0	71.8
$100\times C_{avg}$	10.7	5.92	3.58	17.58	13.86	13.03

Table 5.1: Baseline NN results for KALAKA-2.

5.4.1 Feature Quantization

In Figure 5.1 it is possible to observe the varying performance of the NN for a growing number of quantization channels. Although none are able to achieve the baseline, the best results, for files 3 and 10 seconds long, are obtained with 4-bit quantization, while for 30 seconds long files, the best results correspond to 2-bit quantization. Counter-intuitively, performance degrades with an increasing number of quantization channels. Since the number of possible different values increases, it would be expected that the network's performance would begin to grow closer to the original. However, with a growing number of quantization channels, the maximum absolute value of each discretized feature also increases, which may lead to the degradation observed.

5.4.2 Scaling Factor

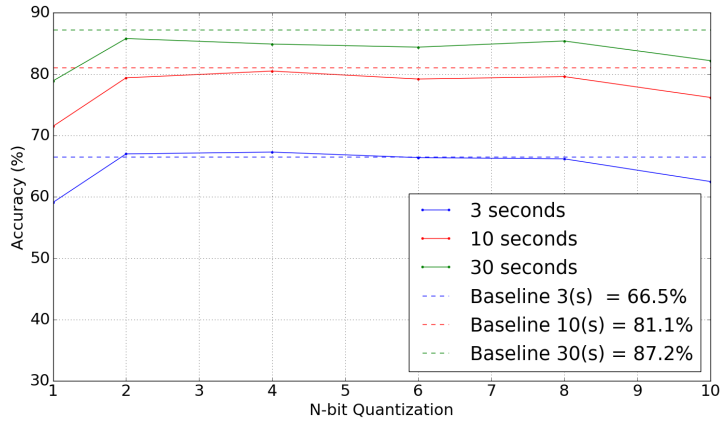
In the experiments of the previous subsection, the overall best results were obtained with 4-bit quantization. For this reason, this model was used in the experiments conducted to determine the best scaling factor. These results are presented in Figure 5.2, and were obtained with 30-seconds-long files.

In Figure 5.2(b) and Figure 5.2(a) we can see that there is an inflection point when the scaling factor s reaches 150, and that the network's performance starts stabilizing around the baseline after s reaches 200. Hence, the best trade-off scaling factor seems to be $s = 200$. However, with this value, some computations in the network reach an absolute value larger than 2^{59}, which is very close to our plaintext modulus. For this reason, and to guarantee that reduction modulo t never occurs, we decided to use 150 as a scaling factor. With this factor the largest value in the network never surpasses 2^{55}, which is 16 times smaller than the plaintext modulus, leaving room for values to grow without risking reduction modulo t.

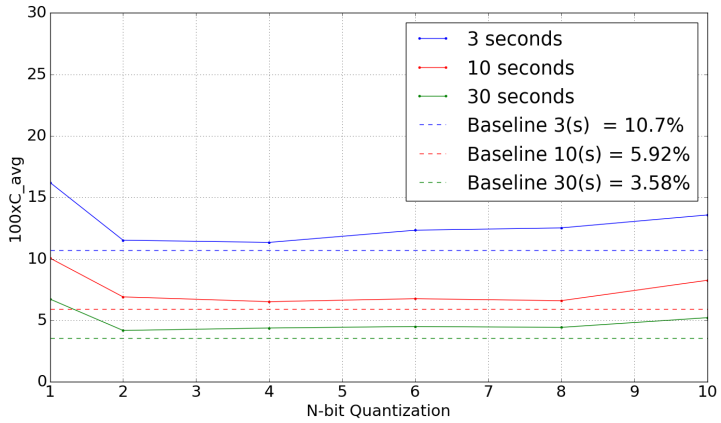
5.4.3 Computational Performance

With 4-bit quantization and a scaling factor of $s = 150$ we obtained the final results displayed in Tables 5.2 and 5.3, for the Development and Evaluation sets of the KALAKA-2 corpus, with 30 seconds-long files, as well as for the three paralinguistic datasets, for Cold, Depression and Parkinson's Disease tasks.

It is important to note that the results for the Paralinguistic tasks were obtained with Equation 5.8,



(a) Accuracy $(\%)$

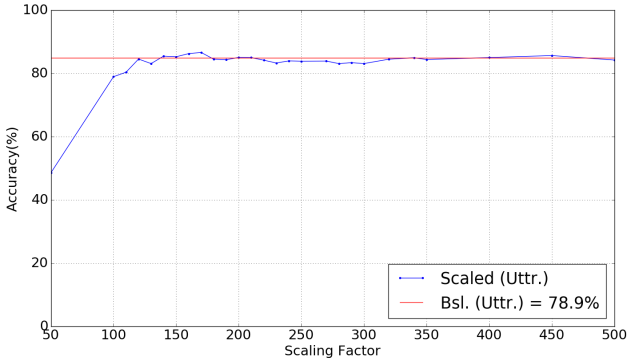


(b) $100 \times C_{AVG}$

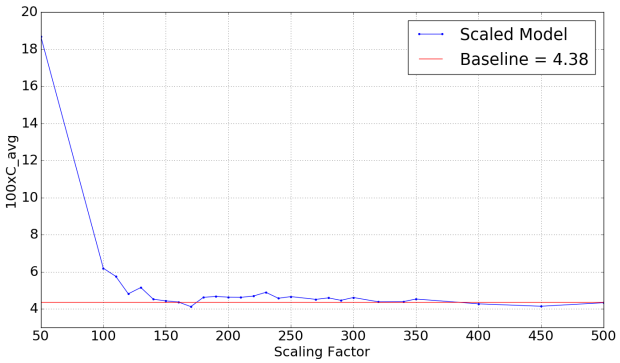
Figure 5.1: Results for different quantization bits.

whereas the results for the LV task were obtained using Equation 5.7. The reason for this is that a larger percentage of weights in the Paralinguistic task's networks were lower than 0.01. Since the scaling factor was equal to 150, most were set to 0, resulting in incorrect results, that did not occur with Equation 5.8. For the LV task, this was not the case, and both equations yielded similar results. As such, the computationally faster equation was used.

For the results in the metrics of both tables, there is not a large performance degradation. Furthermore, the encrypted implementations of these networks take 23 minutes, and 23 seconds, for KALAKA-2's scaled network and for the scaled network developed for the paralinguistic tasks, respectively. Since 16,384 predictions can be made at the same time, the effective value for a prediction is 84.23 milliseconds for the Language Recognition task and 1.40 milliseconds for the paralinguistic tasks. This



(a) Accuracy ($\%$)



(b) $100 \times C_{AVG}$

Figure 5.2: Results for different scaling factors.

represents a large computational performance gain, and, if we consider that in speech tasks, of which the LV task is a considerable example, one needs to make several predictions for a single utterance, this becomes an even more valuable technique, with only a small performance cost.

As a side note, it is interesting to observe that for the KALAKA-2's task, quantizing the input features yielded a performance gain in the evaluation set. We hypothesize that quantizing the input features removes some of their noise and variability, and thus functions as a regularization technique, which may help the model to generalize better, and to achieve better results on unseen data.

Method	Development		Evaluation	
	Accuracy (%)	$100\times C_{avg}$	Accuracy (%)	$100\times C_{avg}$
Baseline NN	87.2	3.57	71.8	13.0
QNN	84.9	4.18	75.9	10.3
Scaled QNN	85.3	4.45	73.9	11.0

Table 5.2: Results for the KALAKA-2 Corpus with 30-seconds-long utterances.

Method	Cold			Depression (Class.)			Depression (Regr.)		Parkinson's Disease		
	F1 Score	Precision	Recall	F1 Score	Precision	Recall	RMSE	MAE	RMSE	MAE	ρ
Baseline ENN	48.3	56.3	66.7	59.2	59.7	60.2	6.77	5.64	16.0	12.5	0.45
QNN	53.0	56.7	66.8	60.3	60.2	60.6	6.74	5.62	15.9	12.6	0.53
Scaled QNN	50.2	56.5	66.9	59.8	60.0	60.5	6.67	5.63	15.8	12.6	0.52

Table 5.3: Batching results for Paralinguistic Tasks

5.5 Discussion

In this chapter we described how an NN can be adapted to the constraints of batching in HE. It was shown how using μ-Law quantization and scaling enables one to discretize a network that can be implemented with batching, at the cost of some performance loss. While the computational cost of computing a single prediction increases, it decreases significantly when considering the effective time a batched prediction takes to compute. Nevertheless, the work developed in this chapter has some limitations. Disregarding the model's performance degradation, the restriction to only two polynomial activation functions is very limiting, as most tasks require much more than two activation functions to achieve reasonable results. For this reason it would be important, as future work, to find other techniques to discretize networks that would not require scaling. Furthermore, in this chapter we only studied the performance of μ-Law for quantization, however, many other quantization techniques exist that may yield better results, and that should be studied in the future as well.

6

Conclusions

Contents

6.1 Conclusions

In the beginning of this thesis (Chapter 1) we discussed the characteristics of Secure Machine Learning as a Service (SMLaaS) and Speech Pattern Recognition as well as the importance of privacy in *paralinguistic* health-related speech tasks. Throughout this work we demonstrated that it is possible to build secure frameworks for these tasks with minimal accuracy degradation at the cost of a significant computational overhead.

In Chapter 3 we provided a proof-of-concept on how an NN can be adapted to work with a LHE scheme, receiving as input an encrypted feature vector, and outputting an encrypted prediction. This network obtained results with minimal degradation when compared to a regular NN, and our implementation was able to perform a single prediction in 4.5 seconds. The security of this framework is based on the security of the LHE scheme. Since the user's input data vector is encrypted neither the server nor malicious third parties can learn anything from the user's data. Nevertheless, this scheme relies on the user to perform a feature extraction stage, before encrypting and sending the data to the server, which provides the user with some information about the model and can become a limitation for devices with low computational power.

Chapter 4 focused on minimizing the user's role on the framework. Our initial objective was to remove the feature extraction stage altogether, and use raw audio as input to our network. However, this proved unfeasible due to the dimensionality of this data format. Instead we chose to trade-off between the amount of computations on the user's side and the model's performance, selecting log-Mel Filterbank Energies as a low-level and low-dimensional representation of the input, together with a frame-level prediction aggregation scheme. Although this approach improved the overall performance of the network, a single prediction took 36 minutes to perform, and, in addition, the encrypted results yielded incorrect results. As such, this method cannot be considered suitable for a real world application. Nonetheless, the methods and reasoning used in this Chapter regarding dimensionality, and frame-posterior aggregation, can serve as a base for future related works to build upon.

In Chapter 5 we focused on transforming the network and its inputs into integer values, using feature quantization and weight scaling, in order to apply a LHE *batching* technique. Our results showed that this can be done with minimal accuracy degradation, and we were able to perform 16384 simultaneous predictions. However, this comes at the cost of having a maximum number of 2 activation layers in the network, and an increase in computational cost, as a single batch takes around 23 seconds to compute. Nevertheless, the effective time cost of a single prediction is 1.4 milliseconds as opposed to the original approach where a prediction took on average 4.5 seconds to compute. In addition this chapter provided a methodical approach for selecting the number of quantization channels and the scaling factor, together with a layer combination method for network implementations that require scaling.

6.2 Publications

Part of the work developed in this thesis was published in, and contributed to:

- ***Teixeira, F.**, Abad, A. and Trancoso, I. - Patient privacy in paralinguistic tasks. In Proceedings Interspeech (2018), pp. 3428–3432.*
- *Correia, J., Raj, B., Trancoso, I. and **Teixeira, F**. - Mining multimodal repositories for speech affecting diseases. In Proceedings Interspeech (2018), pp. 2963–296.*
- *Trancoso, I., Correia, J., **Teixeira, F.**, Raj, B. and Abad, A. - Speech analytics for medical applications. Keynote Talk at the 21st International Conference on Text, Speech and Dialogue (2018).*
- *Trancoso, I., Correia, J., **Teixeira, F.**, Raj, B. and Abad, A. - Analysing speech for clinical applications. Keynote Talk at the 16th International Conference on Statistical Language and Speech Processing (2018).*

6.3 Future Work

The frameworks used in this thesis have several advantages. The first is the relative ease with which one can implement an NN using LHE. Using already existing libraries and a previously trained model, that abides by the constrains of LHE, transforming an NN into an ENN requires only the replacement of each function with its homomorphic counterpart. The second advantage is the fact that this framework does not require the user to remain online while the computation is being performed. A third advantage is the fact that the architecture of the model is hidden, or at least partially hidden.

Nevertheless, this framework poses some limitations. Even though networks trained with small datasets would not provide better results with a larger amount of layers, the same does not hold true for tasks that have very large datasets available. As such, although this framework does not induce a direct accuracy degradation from the activation functions, it would induce it indirectly by restricting the depth of the network to a few layers. This was seen in Chapter 4, where both the size of the input and the network where subjected to strict restrictions to avoid an even larger computational cost, and in Chapter 5, where networks were limited to two activation layers.

The recent approach of [9] seems to solve the restrictions over the number of layers, and allows piece-wise functions to be computed. However, it remains to be seen how increasing the size of the NN will affect the performance of the scheme. Furthermore, this approach still lacks a weight discretization approach that does not entail a large accuracy drop.

Approaches using OT and GC [73] [54] [77] are much faster, and provide more freedom in terms of the model's architecture. However, the networks' architecture needs to be disclosed to the user.

Additionally, they require the NN to be represented as a set of binary logical gates, adding a layer of complexity in implementing the scheme. Nevertheless, it is worth to consider whether disclosing the model's architecture can be seen as a small price to pay for the large performance improvements this type of framework provides.

Throughout this thesis we did not consider the problem of privacy-preserving training. Although it is not feasible in the same setting as the networks we implemented, using LHE, it is possible to perform it in SMPC settings, using both Differential Privacy and GCs [83] [58]. These have the potential of being extremely useful in medical applications, where data is scarce due to the fact that most hospitals and clinics have severe restrictions to divulge their patient's information.

Finally, the approaches used in our work to develop and train models for speech *paralinguistic* tasks were somewhat generic (even though in some cases specific feature sets were employed), and it would be interesting to apply the frameworks of this thesis to methods developed specifically for each of the medical conditions we tested.

Taking these considerations into account, topics for future work should include:

- Researching alternative weight and feature discretization techniques. These can be applied both in the frameworks implemented in this thesis and in the approach developed by [9].
- Applying privacy-preserving model training techniques to *paralinguistic* speech tasks.
- Applying HE to other ML algorithms commonly used for speech tasks.
- Further research GC and OT based PPML frameworks, to determine if they are suitable in the context of speech *paralinguistic* tasks.
- Improving results, by developing privacy-preserving frameworks that adapt specific methods used in health-related *paralinguistic* tasks.

Further research on this topic will allow the development of more accurate and computationally efficient private frameworks, that, in the future, may become real world tools to improve the quality of life of people living with speech-affecting conditions.

Bibliography

[1] ABAD, A., KOLLER, O., AND TRANCOSO, I. The l2f language verification systems for albayzin-2010 evaluation. In *Proceedings FALA* (2010).

[2] ACAR, A., AKSU, H., SELCUK ULUAGAC, A., AND CONTI, M. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys 51* (04 2017), 79.

[3] ARMKNECHT, F., BOYD, C., CARR, C., GJØSTEEN, K., JÄSCHKE, A., REUTER, C. A., AND STRAND, M. A guide to fully homomorphic encryption. *IACR Cryptology ePrint Archive 2015* (2015), 1192.

[4] BANNER, R., HUBARA, I., HOFFER, E., AND SOUDRY, D. Scalable methods for 8-bit training of neural networks. *CoRR abs/1805.11046* (2018).

[5] BARNI, M., ORLANDI, C., AND PIVA, A. A privacy-preserving protocol for neural-network-based computation. In *MM&Sec* (2006).

[6] BOERSMA, P., AND WEENINK, D. Praat: doing phonetics by computer (version 6.0.42), 2018.

[7] BOST, R., ADA POPA, R., TU, S., AND GOLDWASSER, S. Machine learning classification over encrypted data. *NDSS* (01 2015).

[8] BOUFOUNOS, P., AND RANE, S. Secure Binary Embeddings for Privacy Preserving Nearest Neighbors. In *Information Forensics and Security (WIFS), 2011 IEEE International Workshop* (2011), IEEE, pp. 1–6.

[9] BOURSE, F., MINELLI, M., MINIHOLD, M., AND PAILLIER, P. Fast homomorphic evaluation of deep discretized neural networks. *IACR Cryptology ePrint Archive 2017* (2017), 1114.

[10] BRAKERSKI, Z., GENTRY, C., AND VAIKUNTANATHAN, V. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory 6*, 3 (July 2014), 13:1–13:36.

[11] BRASSER, F., FRASSETTO, T., RIEDHAMMER, K., SADEGHI, A.-R., SCHNEIDER, T., AND WEINERT, C. Voiceguard: Secure and private speech processing. In *Interspeech* (09 2018), pp. 1303–1307.

[12] BRUMMER, N. Focal toolkit v1.0. https://sites.google.com/site/nikobrummer/focalmulticlass, 2007.

[13] CAMPBELL, W. M., CAMPBELL, J. P., REYNOLDS, D. A., SINGER, E., AND TORRES-CARRASQUILLO, P. A. Support vector machines for speaker and language recognition. *Computer Speech & Language 20* (2006), 210–229.

[14] CHABANNE, H., DE WARGNY, A., MILGRAM, J., AND MOREL ET AL., C. Privacy-Preserving Classification on Deep Neural Network. *IACR Cryptology ePrint Archive 2017* (2017), 35.

[15] CHILLOTTI, I., GAMA, N., GEORGIEVA, M., AND IZABACHÈNE, M. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Report 2016/870, 2016.

[16] CHOI, Y., EL-KHAMY, M., AND LEE, J. Towards the limit of network quantization. *CoRR abs/1612.01543* (2016).

[17] CHOLLET, F., ET AL. Keras. https://github.com/keras-team/keras, 2015.

[18] CHOU, T., AND ORLANDI, C. The simplest protocol for oblivious transfer. In *Proceedings of the 4th International Conference on Progress in Cryptology – LATINCRYPT 2015 - Volume 9230* (2015), pp. 40–58.

[19] CHUNG, J., GULCEHRE, C., CHO, K., AND BENGIO, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[20] COURBARIAUX, M., AND BENGIO, Y. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR abs/1602.02830* (2016).

[21] CUMMINS, N., SCHMITT, M., AMIRIPARIAN, S., KRAJEWSKI, J., AND SCHULLER, B. "you sound ill, take the day off": Automatic recognition of speech affected by upper respiratory tract infection. In *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (2017).

[22] DIAS, M. Privacy-preserving speech emotion recognition. Master's thesis, Instituto Superior Técnico, 2017.

[23] DIAS, M., ABAD, A., AND TRANCOSO, I. Exploring Hashing and Cryptonet based Approaches for Privacy-preserving Speech Emotion Recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2018 IEEE International Conference* (2018), IEEE.

[24] DIBEKLIOGLU, H., HAMMAL, Z., YANG, Y., AND COHN, J. F. Multimodal detection of depression in clinical interviews. *Proceedings of the ... ACM International Conference on Multimodal Interaction. ICMI 2015* (2015), 307–310.

[25] ELGAMAL, T. A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. *IEEE Transactions on Information Theory 31*, 4 (1985), 469–472.

[26] EVEN, S., GOLDREICH, O., AND LEMPEL, A. A randomized protocol for signing contracts. *Commun. ACM 28*, 6 (June 1985), 637–647.

[27] EYBEN, F., SCHERER, K., SCHULLER, B., AND SUNDBERG ET AL., J. The Geneva Minimalistic Acoustic Parameter Set (GeMAPS) for Voice Research and Affective Computing. *IEEE Transactions on Affective Computing 7*, 2 (4 2016), 190–202. Open access.

[28] EYBEN, F., WENINGER, F., GROSS, F., AND SCHULLER, B. Recent developments in openSMILE, the Munich Open-source Multimedia Feature Extractor. In *Proceedings of the 21st ACM International Conference on Multimedia* (2013), MM '13, pp. 835–838.

[29] FAN, J., AND VERCAUTEREN, F. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive 2012* (2012), 144. Informal publication.

[30] GENTRY, C. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.

[31] GILAD-BACHRACH, R., DOWLIN, N., AND LAINE ET AL., K. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *ICML* (2016), vol. 48 of *JMLR Workshop and Conference Proceedings*, pp. 201–210.

[32] GOETZ, C. G., TILLEY, B. C., AND SHAFTMAN ET AL., S. R. Movement Disorder Society-sponsored revision of the Unified Parkinson's Disease Rating Scale (MDS-UPDRS): scale presentation and clinimetric testing results. *Mov. Disord. 23*, 15 (2008), 2129–2170.

[33] GOLDREICH, O. Secure multi-party computation, 03 1999.

[34] GONZALEZ-DOMINGUEZ, J., LOPEZ-MORENO, I., MORENO, P. J., AND GONZALEZ-RODRIGUEZ, J. Frame-by-frame language identification in short utterances using deep neural networks. *Neural Networks 64* (2015), 49 – 58. Special Issue on "Deep Learning of Representations".

[35] GRAEPEL, T., LAUTER, K. E., AND NAEHRIG, M. ML Confidential: Machine Learning on Encrypted Data. *IACR Cryptology ePrint Archive 2012* (2012), 323.

[36] GRATCH, J., ARTSTEIN, R., LUCAS, G. M., AND STRATOU ET AL., G. The Distress Analysis Interview Corpus of human and computer interviews. In *LREC* (2014), pp. 3123–3128.

[37] GRAVES, A., AND JAITLY, N. Towards End-to-end Speech Recognition with Recurrent Neural Networks. In *Proceedings of the 31st International Conference on Machine Learning - Volume 32* (2014), ICML'14, pp. II–1764–II–1772.

[38] HEMMERLING, D., OROZCO-ARROYAVE, J., SKALSKI, A., GAJDA, J., AND NÖTH, E. Automatic detection of parkinson's disease based on modulated vowels. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH* (2016), 1190–1194.

[39] HESAMIFARD, E., TAKABI, H., AND GHASEMI, M. CryptoDL: Deep Neural Networks over Encrypted Data. *CoRR abs/1711.05189* (2017).

[40] HOENIG, F., STEMMER, G., HACKER, C., AND BRUGNARA, F. Revising perceptual linear prediction (plp). In *Interspeech* (01 2005), pp. 2997–3000.

[41] HUBARA, I., COURBARIAUX, M., SOUDRY, D., EL-YANIV, R., AND BENGIO, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR abs/1609.07061* (2016).

[42] HUCKVALE, M., AND BEKE, A. It sounds like you have a cold! testing voice features for the interspeech 2017 computational paralinguistics cold challenge. In *INTERSPEECH* (08 2017), pp. 3447–3451.

[43] IOFFE, S., AND SZEGEDY, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR abs/1502.03167* (2015).

[44] JACOB, B., KLIGYS, S., CHEN, B., ZHU, M., TANG, M., HOWARD, A. G., ADAM, H., AND KALENICHENKO, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CoRR abs/1712.05877* (2017).

[45] JIMÉNEZ, A., RAJ, B., PORTÊLO, J., AND TRANCOSO, I. Secure Modular Hashing. In *Information Forensics and Security (WIFS), 2015 IEEE International Workshop* (2015), IEEE, pp. 1–6.

[46] KALIA, L. V., AND LANG, A. E. Parkinson's disease., Aug 2015.

[47] KILIAN, J. Founding crytpography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (1988), STOC '88, pp. 20–31.

[48] KRAJEWSKI, J., SCHNIEDER, S., AND BATLINER, A. Description of the Upper Respiratory Tract Infection Corpus (URTIC). In *INTERSPEECH* (2017).

[49] KROENKE, K., STRINE, T. W., SPITZER, R. L., AND WILLIAMS ET AL., J. B. The PHQ-8 as a measure of current depression in the general population. *J Affect Disord 114*, 1-3 (2009), 163–173.

[50] LAINE, K., CHEN, H., AND PLAYER, R. Simple Encrypted Arithmetic Library - SEAL v2.3.1. Tech. rep., Microsoft, December 2017.

[51] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[52] LIN, D. D., TALATHI, S. S., AND ANNAPUREDDY, V. S. Fixed point quantization of deep convolutional networks. *CoRR abs/1511.06393* (2015).

[53] LINDELL, Y. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*. 2005, pp. 1005–1009.

[54] LIU, J., JUUTI, M., LU, Y., AND ASOKAN, N. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), CCS '17, pp. 619–631.

[55] MAO, Q., DONG, M., HUANG, Z., AND ZHAN, Y. Learning salient features for speech emotion lt;newline/ gt;recognition using convolutional lt;newline/ gt;neural networks. *IEEE Transactions on Multimedia 16*, 8 (2014), 2203–2213.

[56] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[57] MILDE, B., AND BIEMANN, C. Using representation learning and out-of-domain data for a paralinguistic speech task. In *INTERSPEECH* (2015).

[58] MOHASSEL, P., AND ZHANG, Y. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)* (2017), pp. 19–38.

[59] MOONS, B., GOETSCHALCKX, K., BERCKELAER, N. V., AND VERHELST, M. Minimum energy quantized neural networks. *CoRR abs/1711.00215* (2017).

[60] NEUMANN, M., AND VU, N. T. Attentive convolutional neural network based speech emotion recognition: A study on the impact of input features, signal length, and acted speech. *CoRR abs/1706.00612* (2017).

[61] O. RABIN, M. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive 2005* (01 2005), 187.

[62] ORGANIZATION, W. H. World health organization. http://www.who.int/news-room/fact-sheets/detail/influenza-(seasonal), 2018.

[63] ORGANIZATION, W. H. World health organization. http://www.who.int/mental_health/management/depression/en, 2018.

[64] ORLANDI, C., PIVA, A., AND BARNI, M. Oblivious neural network computing via homomorphic encryption. *EURASIP J. Inf. Secur. 2007* (Jan. 2007), 18:1–18:10.

[65] OROZCO-ARROYAVE, J. R., ARIAS-LONDOÑO, J. D., BONILLA, J. F. V., AND GONZALEZ-RÁTIVA ET AL., M. C. New Spanish Speech Corpus Database for the Analysis of People Suffering from Parkinson's Disease. In *LREC* (2014), pp. 342–347.

[66] OROZCO-ARROYAVE, J. R., VDSQUEZ-CORREA, J. C., HÖNIG, F., ARIAS-LONDOÑO, J. D., VARGAS-BONILLA, J. F., SKODDA, S., RUSZ, J., AND NOTH, E. Towards an automatic monitoring of the neurological state of parkinson's patients from speech. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2016), pp. 6490–6494.

[67] PATHAK, M. A., AND RAJ, B. Privacy-Preserving Speaker Verification and Identification Using Gaussian Mixture Models. *IEEE Transactions on Audio, Speech, and Language Processing 21*, 2 (Feb 2013), 397–406.

[68] POMPILI, A., ABAD, A., ROMANO, P., AND MARTINS ET AL., I. P. Automatic Detection of Parkinson's Disease: An Experimental Analysis of Common Speech Production Tasks Used for Diagnosis. In *TSD* (2017), vol. 10415 of *Lecture Notes in Computer Science*, pp. 411–419.

[69] PORTÊLO, J. *Privacy-Preserving Frameworks for Speech Mining*. PhD thesis, Instituto Superior Técnico, 2015.

[70] PORTÊLO, J., ABAD, A., RAJ, B., AND TRANCOSO, I. Secure Binary Embeddings of Front-end Factor Analysis for Privacy Preserving Speaker Verification. In *INTERSPEECH* (2013), pp. 2494–2498.

[71] PRADHAN, A. Support vector machine-a survey. *International Journal of Emerging Technology and Advanced Engineering 2*, 8 (2012), 82–85.

[72] REYNOLDS, D. A., QUATIERI, T. F., AND DUNN, R. B. Speaker verification using adapted gaussian mixture models. *Digital signal processing 10*, 1-3 (2000), 19–41.

[73] RIAZI, M. S., WEINERT, C., TKACHENKO, O., SONGHORI, E. M., SCHNEIDER, T., AND KOUSHANFAR, F. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security* (2018), ACM, pp. 707–721.

[74] RIVEST, R. L., ADLEMAN, L., AND DERTOUZOS, M. L. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation, Academia Press* (1978), 169–179.

[75] RODRIGUEZ-FUENTES, L. J., PENAGARIKANO, M., VARONA, A., DIEZ, M., AND BORDEL, G. The albayzin 2010 language recognition evaluation. In *Proceedings of Interspeech* (01 2011), pp. 1529–1532.

[76] RODRÍGUEZ-FUENTES, L. J., PEÑAGARIKANO, M., VARONA, A., DÍEZ, M., AND BORDEL, G. Kalaka-2: a tv broadcast speech database for the recognition of iberian languages in clean and noisy environments. In *LREC* (2012).

[77] ROUHANI, B. D., RIAZI, M. S., AND KOUSHANFAR, F. Deepsecure: Scalable provably-secure deep learning. *CoRR abs/1705.08963* (2017).

[78] SAINATH, T. N., WEISS, R. J., SENIOR, A. W., WILSON, K. W., AND VINYALS, O. Learning the speech front-end with raw waveform CLDNNs. In *INTERSPEECH* (2015).

[79] SANYAL, A., KUSNER, M. J., GASCÓN, A., AND KANADE, V. Tapas: Tricks to accelerate (encrypted) prediction as a service. In *ICML* (2018).

[80] SCHULLER, B., AND BATLINER, A. *Computational Paralinguistics: Emotion, Affect and Personality in Speech and Language Processing*. 2013.

[81] SCHULLER, B., STEIDL, S., BATLINER, A., AND BERGELSON ET AL., E. The INTERSPEECH 2017 Computational Paralinguistics Challenge: Addressee, Cold & Snoring. In *Computational Paralinguistics Challenge (ComParE), Interspeech 2017* (2017).

[82] SCHULLER, B., STEIDL, S., BATLINER, A., BURKHARDT, F., DEVILLERS, L., MÖLLER, C., AND NARAYANAN, S. Paralinguistics in Speech and Language — State-of-the-art and the Challenge. *Computer Speech & Language 27*, 1 (2013), 4 – 39. Special issue on Paralinguistics in Naturalistic Speech and Language.

[83] SHOKRI, R., AND SHMATIKOV, V. Privacy-preserving deep learning. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (Sept 2015), pp. 909–910.

[84] SNYDER, P. Yao's garbled circuits: Recent directions and implementations, 2014.

[85] SRIVASTAVA, N., HINTON, G. E., KRIZHEVSKY, A., AND SUTSKEVER ET AL., I. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15* (2014), 1929–1958.

[86] TRIGEORGIS, G., RINGEVAL, F., BRUECKNER, R., MARCHI, E., NICOLAOU, M. A., SCHULLER, B., AND ZAFEIRIOU, S. Adieu features? End-to-end Speech Emotion Recognition using a Deep Convolutional Recurrent Network. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2016), pp. 5200–5204.

[87] TYRRELL, D. A. J., COHEN, S., AND SCHILARB, J. E. Signs and symptoms in common colds. *Epidemiology and Infection 111*, 01 (1993), 143–156.

[88] TZIRAKIS, P., TRIGEORGIS, G., NICOLAOU, M. A., SCHULLER, B. W., AND ZAFEIRIOU, S. End-to-End Multimodal Emotion Recognition using Deep Neural Networks. *IEEE Journal of Selected Topics in Signal Processing 11*, 8 (Dec 2017), 1301–1309.

[89] VALSTAR, M. F., GRATCH, J., SCHULLER, B. W., AND ET AL., F. R. AVEC 2016 - Depression, Mood, and Emotion Recognition Workshop and Challenge. *CoRR abs/1605.01600* (2016).

[90] VAN DEN OORD, A., DIELEMAN, S., ZEN, H., SIMONYAN, K., VINYALS, O., GRAVES, A., KALCHBRENNER, N., SENIOR, A. W., AND KAVUKCUOGLU, K. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499* (2016).

[91] VAN DIJK, M., GENTRY, C., HALEVI, S., AND VAIKUNTANATHAN, V. Fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT 2010* (2010), pp. 24–43.

[92] VÁSQUEZ-CORREA, J., OROZCO-ARROYAVE, J. R., AND NÖTH, E. Convolutional neural network to model articulation impairments in patients with parkinson's disease. In *Proc. Interspeech 2017* (2017), pp. 314–318.

A

Simple Encrypted Arithmetic Library (SEAL)

Throughout the experiments of this thesis we used the Simple Encrypted Arithmetic Library (SEAL) library to perform encrypted operations. Developed by researchers at Microsoft, this library implements the Fan-Vercauteren (FV) Leveled Homomorphic Encryption (LHE) scheme. In this Appendix we provide a brief description of this implementation, detailing the basic encryption and decryption operations, the Homomorphic noise growth in this implementation, the working principle of the Fractional Encoder, a basic description of the CRT Batching technique and a security review of the scheme. The contents of this Appendix are adapted from SEAL's v2.3.1 manual, available in [50].

A.1 Notation

Before we start to detail operations in SEAL it is first necessary to explain the notation used. In Table A.1 we list some of the most parameters we will refer to in this Appendix.

Parameter	Description
q	Modulus in the ciphertext space, or *coefficient modulus*. This value is of the form $q_1 \times ... \times q_k$, where q_i are prime.
t	Modulus in the plaintext space, or *plaintext modulus*.
n	A power of 2.
$x^n + 1$	The *polynomial modulus* that specifies the ring R.
R	The ring $\mathbb{Z}[x]/(x^2 + 1)$.
R_a	The ring $\mathbb{Z}_a[x]/(x^2 + 1)$, the same as the ring R but with coefficients reduced modulo a.
w	Base into which ciphertext elements are decomposed during relinearization.
l	Each component of each Evaluation Key is composed of $l + 1 = \lfloor log_w q \rfloor + 1$ elements.
δ	Expansion factor in the ring R ($\delta \leq n$).
$\mathbf{\Delta}$	Quotient on division of q by t, or $\lfloor q/t \rfloor$.
$r_t(q)$	Remainder on the division of q by t, i.e. $q = \mathbf{\Delta}t + r_t(q)$.
χ	Error distribution (a truncated discrete Gaussian distribution).
σ	Standard deviation of χ.
B	Bound on the distribution χ.

Table A.1: Notation used in SEAL (adapted from [50]).

A.2 Basic Operations

In FV, Plaintexts are defined in the ring structure $R_t = \mathbb{Z}_t[x]/(x^n + 1)$, i.e. polynomials of degree less than n with integer coefficients modulo t. As such, before one can encrypt a value, it is necessary to encode it first into a plaintext polynomial in R_t. In addition, Ciphertexts are defined as arrays of polynomials in R_q, that contain at least two polynomials, and that grow in size with each homomorphic multiplication.

Considering a security parameter λ, that $a \xleftarrow{\$} \mathbb{S}$ denotes that a is sampled uniformly from $\mathbb{S}$, that $[.]_a$ denotes the reduction of an integer modulo a and that $\lfloor . \rceil$ denotes the rounding operation, operations in FV are defined as:

- *SecretKeyGen*(λ): Sample $s \xleftarrow{\$} R_2$ and output $sk = s$.
- *PublicKeyGen*(sk): Set $s = sk$, sample $a \xleftarrow{\$} R_q$ and $e \leftarrow \chi$. Output $pk = ([-(as + e)]_q, a)$.
- *Encrypt*(pk, m): For $m \in R_t$, let $pk = (p_0, p_1)$. Sample $u \xleftarrow{\$} R_2$ and $e_1, e_2 \leftarrow \chi$. Compute:

$$\text{ct} = ([\Delta m + p_0 u + e_1]_q, [p_1 u + e_2]_q). \tag{A.1}$$

- *Decrypt*(sk, ct): Set $s = sk$, $c_0 = ct[0]$ and $c_1 = ct[1]$. Output:

$$\left[\left\lfloor \frac{t}{q}[c_0 + c_1 s]_q \right\rceil\right]_t \tag{A.2}$$

- *Add*(ct_0, ct_1): Output $(ct_0[0] + ct_1[0], ct_0[1] + ct_1[1])$.
- *Multiply*(ct_0, ct_1): Compute

$$c_0 = \left[\left\lfloor \frac{t}{q} ct_0[0] ct_1[0] \right\rceil\right]_t$$

$$c_1 = \left[\left\lfloor \frac{t}{q} (ct_0[0] ct_1[1] + ct_0[1] ct_1[0]) \right\rceil\right]_q$$

$$c_2 = \left[\left\lfloor \frac{t}{q} ct_0[1] ct_1[1] \right\rceil\right]_t$$

It is important to note that with homomorphic multiplication the size of the ciphertext arrays increases. However, it is possible to reduce the size of the array back to 2, using Relinearization. Although homomorphic operations can be expressed in order to work with an indefinite number of array elements, relinearization reduces the computational complexity of these algorithms.

Relinearization should be performed directly after homomorphic multiplication, and requires a set of *Evaluation Keys*, generated as follows:

- *EvaluationKeyGen*(sk, w): for $i \in 0, ..., l$, sample $a_i \xleftarrow{\$} R_q, e_i \leftarrow \chi$.
- Output: $evk = ([-(a_i s + e_i) + w^i s^2]_q, a_i)$

To perform relinearization one can apply the Evaluation Keys as:

$$\begin{aligned} c_0' &= c_0 + \sum_{i=0}^{l} evk[i][0] c_2^{(i)}, \\ c_1' &= c_1 + \sum_{i=0}^{l} evk[i][1] c_2^{(i)}, \end{aligned} \tag{A.3}$$

where c_2 is expressed in base w as $c_2 = \sum_{i=0}^{l} c_2^{(i)} w^i$, and the new ciphertext is (c_0', c_1').

In addition to these operations, SEAL also implements the negation operation and two operations called *AddPlain* and *MultPlain*. These operations allow ciphertexts to be added or multiplied to plaintexts at a much lower noise cost.

A.3 Noise

From Equation A.1 it is possible to observe that a freshly encrypted ciphertext has an associated noise term. This noise term grows with each homomorphic operation, and after a certain threshold it is impossible to correctly decrypt it. For ciphertexts in SEAL it is demonstrable that the decryption operation is successful as long as the infinity norm of the noise follows $||\nu|| < 1/2$, where ν is the ciphertext's noise. Alternatively we can choose to represent the amount of noise we have left until decryption fails. This amount can be defined as $-log_2(2||v||)$, and is referred to as the *Noise Budget*.

Although it is not possible to ascertain the exact amount of noise each operation will consume, it is nonetheless to establish upper bounds for the noise growth with each operation. In Table A.2 we present SEAL's estimates for noise growth, where N_m represents an upper bound on the number of non-zero coefficients in polynomial m.

Operation	Input	Noise Bound
Encrypt	Plaintext m	$\frac{r_t(q)}{q}\|\|m\|\|N_m + \frac{7nt}{q}min\{B, 6\sigma\}$
Add/Sub	Ciphertexts c_1, c_2	$\nu_1 + \nu_2$
AddPlain/SubPlain	Ciphertext c_t and plaintext m	$\nu + \frac{r_t(q)}{q}\|\|m\|\|N_m$
Multiply	Ciphertexts c_1, c_2 of sizes $j_1 + 1$ and $j_2 + 1$	$t\sqrt{3n}[(12n)^{j_1/2}\nu_2$ $+(12n)^{j_2/2}\nu_1 + (12n)^{(j_1+j_2)/2}]$
Multiply Plain	Ciphertext c_t and plaintext m	$N_m\|\|m\|\|\nu$
Relinearize	Ciphertext c_t of size K and target size L, such that $2 \leq L < K$	$\nu + \frac{2t}{q}min\{B, 6\sigma\}(K - L)n(l + 1)w$

Table A.2: Nose estimates for operations in SEAL (adapted from [50]).

A.4 Encoders

As stated in Section A.2, to encrypt a message m it is first necessary to represent it as a polynomial in R_t. One way to do this would be to encode it as the constant polynomial $m \in R_t$. However, if the

plaintext polynomial wraps around t it will be reduced modulo t and the decoded result may be incorrect. Although we could simply increase t, the noise growth depends strongly on t, as can be seen by the estimates of Table A.2.

Alternatively, SEAL provides two encoding schemes, an *Integer Encoder* and a *Fractional Encoder*.

The *Integer Encoder* works by expanding a message m by a base B. This can be exemplified by selecting a base $B = 2$, and having a message $-(2^n - 1) \leq m \leq 2^n - 1$. In this case, if the binary expansion of $|m|$ is represented as $a_{n-1}, ...a_1 a_0$, then m will be encoded as:

$$\text{IntegerEncode}(m, B = 2) = sign(m) \cdot (m_{n-1}x^{n-1} + ... + a_1 x + a_0) \tag{A.4}$$

The decoding process is defined as the evaluation of the polynomial at $x = 2$. In addition, it is easily provable that this encoding respects integer operations, and as such, homomorphic operations can be applied, and the decoding process will yield a correct result. Nevertheless, when using this encoder one must be careful to guarantee that with homomorphic operations, especially multiplications, the coefficients in the polynomial never reach t. If this happens, the value will be reduced modulo t, and the decoded results will be incorrect. For this reason, although B can be selected to be any positive integer, it should be as small as possible, to ensure the values of the polynomial coefficients never grow too large, and thus allowing for a smaller t to be selected, obtaining a larger noise budget.

However, many applications require real valued inputs to function properly. As such, SEAL implements an extension of the *Integer Encoder*, the *Fractional Encoder*. Repeating the process as in the previous encoder, we can consider a base $B = 2$ and a rational message $m = 5.8125$. This value has a finite binary expansion defined as:

$$5.8125 = 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4}$$

To encode this message we take the integer part of the value and encode it as in the previous example, obtaining $\text{IntegerEncode}(5, B = 2) = x^2 + 1$. On the other hand, the fractional part $2^{-1} + 2^{-2} + 2^{-4}$ is converted into a polynomial by first adding n to each exponent. If we change the base B into the variable x, and flip the coefficient's signs, we obtain $-x^{n-1} - x^{n-2} - x^{n-4}$. This transformation is denoted as $\text{FracEncode}(r, B = 2)$. The final form of the encoder is thus:

$$\text{FracEncode}(r, B = 2) = sign(r) \cdot [\text{IntegerEncode}(\lfloor |r| \rfloor, B = 2) + \text{FracEncode}(\{|r|\}, B = 2)],$$

where $\{.\}$ denotes the fractional part. Using the previous example we will then have:

$$\text{FracEncode}(5.8125, B = 2) = -x^{n-1} - x^{n-2} - x^{n-4} + x^2 + 1,$$

The decoding process works by first separating the high degree part of the plaintext polynomial and invert the signs of these terms and shift their exponents by $-n$. Afterwords the entire expression can be evaluated at $x = 2$.

Since multiplications create cross-terms, it is important to guarantee that the integer and fractional parts never overlap. To avoid this, this encoder selects a maximum number of terms for the integer part, and the miminum possible number of terms for the fractional part. In this way, the decoding process can select the original maximum number of integer terms as the integer part and the remaining coefficients as the fractional part. Nevertheless, as in the *Integer Encoder*, it is necessary to ensure the polynomial coefficients never reach t. In addition to the *Integer* and *Fractional* Encoder, SEAL also provides the *CRT (Chinese Remainder Theorem) Batching* encoder. This encoder allows that n integers modulo t to be packed into a single plaintext polynomial, and operate on these integers as *SIMD* (Single Instruction Multiple Data). This technique is also called *batching*, and it only works when the plaintext modulus t is a prime number congruent to $1(\text{mod } 2n)$.

If this is the case, it can be shown that the ring R_t can be decomposed into a series of polynomials that can be used to encode individual messages. Furthermore, operations applied to the ring R_t are isomorphic with the decomposition of the ring into polynomials. As such, by applying operations to a R_t, we are indeed applying them to all polynomials that form its decomposition. To this end, SEAL provides two operations, *Compose* and *Decompose*, that allow us to encode n integer-valued messages into a single plaintext. This can be extremely useful in vectorizable operations. However, as in the previous encoders, we need to make sure the underlying values of the messages in each slot do not wrap around t during the computation. This requires t to be chosen to be much larger than in the previous encoders, as the messages are not expanded by a base B, which results in very large values in each slot.

A.5 Security

The security of the FV scheme is based on the hardness of RLWE problem. This problem can be presented as the decision-RLWE defined as:

Decision-RLWE: *Let n be power of 2. Let* $R = \frac{\mathbb{Z}[x]}{(x^n+1)}$ *and* $R_q = \frac{\mathbb{Z}_{||}[x]}{(x^n+1)}$ *for some integer* q*. Let* s *be a random element in* R_q*, and let* χ *be a distribution on* R_q*, obtained by choosing each coefficient of the polynomial from a discrete Gaussian distribution over* $\mathbb{Z}$*. Denote by* $A_{s,\chi}$*, the distribution obtained by choosing* $a \leftarrow R_q$ *uniformly at random, choosing* $e \leftarrow \chi$ *and outputting* $(a, [a \cdots + e]_q)$*. Decision-RLWE is the problem of distinguishing between the distribution* $A_{s,\chi}$ *and the uniform distribution on* R_q^2*.*

However, the security level of this scheme depends on the encryption parameters. SEAL provides default values for the coefficient polynomial for several security levels. In Table A.3 we present the bit length of these default values for several security levels and values of n.

n	Bit-length of default q		
	128-bit Security	192-bit Security	256-bit Security
1024	27	19	14
2048	54	37	29
4096	109	75	58
8192	218	152	118
16384	438	300	237
32768	881	600	476

Table A.3: SEAL's default (n,q) pairs for $\sigma = 8/\sqrt{2\pi}$, for 128, 192 and 256-bit security levels (adapted from [50]).

A.6 Encryption Parameters and Practical Considerations

In SEAL we have to select several crucial encryption parameters.

The first value we need to choose is n, which will specify the polynomial modulus as $x^n + 1$. Since this value will affect the security of the scheme, if we intent to use SEAL's security estimates, the library allows us to select its default values for the *coefficient modulus* q through a function that receives $x^n + 1$ as input. In addition, this requires the use of the default values of σ and χ, that unless specific values are necessary, do not need to be selected.

The last value we need to select is t, the *plaintext modulus*. This will determine how large our underlying messages can grow when homomorphic operations are being performed. To perform operations with larger underlying values a larger t needs to be selected. However, with a larger t the noise growth will be higher for every operation. As such, it is necessary to trade-off between the absolute value of the underlying messages and the noise growth of the scheme. Nevertheless, when larger values and further operations are required we can increase the value of n. This will increase the size of the plaintext polynomials, and as such, the noise budget that is allowed, but it will require that a larger value of q be used, in order to keep the security level, thus resulting in larger computational cost for each operation.

If we want to utilize one of the encoding schemes, we must also select their parameters accordingly. For the *Integer Encoder* SEAL allows us to select the expansion base B, that has the default value of 2. For the *Fractional Encoder* besides B, we need to select the original amount of coefficients for the fractional and integer parts of the value. While the fractional part is expected to grow beyond its original amount of terms, the integer part must be selected with a value large enough that the integer part never grows past it, for the reasons stated in Section A.4.

B

Datasets

In this Appendix we provide a short description of the Datasets used in the experiments of this thesis.

B.1 Cold: URTIC

The Upper Respiratory Tract Infection Corpus (URTIC) [48], was provided by the Institute of Safety and Technology of the University of Wuppertal, Germany, for 2017's Interspeech Computational Paralinguistics Challenge (ComParE) [81]. The full corpus (including the test set) consists on scripted and spontaneous speech tasks, recorded in quiet rooms, performed by 630 subjects, each split including 210 subjects. These recordings are divided into segments varying between 3 and 10 seconds, amounting to a total of 45 hours. The training and development sets include 173 healthy controls and 37 presenting symptoms of a Cold, with the recordings divided into 9,505 and 9,596 segments, respectively. This corpus was used for a classification task.

B.2 Depression: DAIC-WOZ

The Distressed Analysis Interview Corpus - Wizard of Oz (DAIC-WOZ), is a subset of the DAIC corpus [36]. It contains clinical interviews designed to support the diagnosis of psychological distress conditions, conducted by a virtual interviewer, recorded in both audio and video formats. It consists in 189 sessions ranging between 7 and 33 minutes, 106 of which are included in the training partition, 34 in the development split, and the remaining 40 in the test set. Of the subjects included in the training set, 76 are healthy controls and 30 present symptoms of Depression. For the development set, 12 subjects have Depression, while 22 are healthy controls. These interviews are split into 6218 and 1862 segments, for the training and development sets, respectively. Each interview has a set of labels, classifying the subjects on whether or not they are depressed, based on their classification on the PHQ-8 scale [49], which is also provided. Since the interviews are given in full, a segmentation file is included alongside each interview recording, allowing for the removal of the interviewer's interventions and for the interview to be split into smaller segments. This corpus was provided for 2016's Audio Visual Emotion Challenge (AVEC) [89]. Both classification and regression tasks were performed with this dataset.

B.3 Parkinson's Disease

In the experiments performed for Parkinson's Disease (PD) the dataset provided for 2015's Interspeech ComParE Challenge was used. This corpus is a subset of a Spanish Corpus from Universidad de Antioquia [65]. It includes 50 patients, recorded in noise controlled conditions, and classified on the UPDRS-III scale [32], with classifications ranging between 5 and 92. Each subject performed 42 speech tasks, such as reading a text, performing a monologue, uttering isolated words and sentences, as well as the rapid repetition of the syllables /pa-ta-ka/, /pa-ka-ta/ and /pe-ta-ka/. The training split includes 35 patients, with the remaining 15 being present in the development set. This results in 1470 and 630 different files for the training and development partitions, respectively. Since the subjects are classified only on the degree of Parkinson's, this corpus was used for a regression task.

B.4 KALAKA-2

This corpus was developed for 2010's Albayzin Language Recognition Evaluation (LRE) and is composed of 6 different languages Spanish, Catalan, Basque, Galician, Portuguese and English. It includes speech recorded from TV broadcasts [75] [76]. Speech recordings in this dataset are divided into three partitions, Training, Development and Evaluation, and include speech in both clean and noisy conditions. Recordings in the Development and Evaluation sets are divided into 3, 10 and 30 second length files.

Additionally, out-of-set language recordings are also included in the Development and Evaluation sets. In the literature, tests conducted with these files are called *open-set* tests, while the ones conducted without them are called *closed-set* tests.

C

Case: Parkinson's Disease

In Chapter 1, Section 1.2, we discussed how PD primarily affects phonation, prosody and articulation. The latter has been shown to be the most predominant symptom, with several works reporting positive results when testing for the presence of the disease and its severeness, using articulatory features [38] [66] [92]. In [92] the authors proposed the use of the spectrograms of transition regions between unvoiced to voiced (voice onset) and voiced to unvoiced (voice offset) utterances, to separate parkinsonian speakers from healthy controls. The intuition behind this method was the fact that the difficulties PD patients display to start and stop the vibration of vocal folds can be modelled through the time-frequency representation of voice onset/offset transition regions [92].

Since the main goal of this thesis is to determine how HE can be applied to create secure frameworks for health-related speech tasks, we deemed it pertinent to include and test this approach, as it serves as an example of how such a framework could be useful in a real world application. Nevertheless, since it cannot be considered an end-to-end approach as it relies on first locating and computing the spectrogram of transition regions before giving it as input to the network, we decided not to include it in Chapter 4, and instead include it only as an appendix.

C.1 On/Offset Spectrograms

Following the approach detailed above, we conducted an experiment using the Spectrograms of onset/offset transition regions together with a CNN. To do this, we followed the approach suggested by [92], and used Praat [6] to determine the presence of the Fundamental Frequency (F0), computing its value at every millisecond. We defined as voice offset regions, areas of the signal where the pitch dropped from a non-zero value to 0, and vice-versa for onset regions. However, due to the presence of outliers, the amount of transition regions was very large. For this reason, the pitch vector was smoothed over with an Average Filter, with a window of 32 ms. Subsequently, we computed the magnitude Spectrogram of the $\pm$80 ms region around the onset/offset (resulting in a 160 ms frame), using the STFT (Short Time Fourier Transform) implementation of Python's *scipy.signal* package.

This network follows the same architecture as the models from Chapter 4, represented in Figure 4.1, and was trained with MSE as the loss function, a learning rate of 0.2, a weight decay of 0.0001, 100 epochs and early stopping. The first and second convolutional layers of the network both had 16 two-dimensional filters, with shape (2,2) and each Average Pooling layer had a pool-size and strides with shape (2,2). The first two FC layers had 100 units, while the last had only one output unit. Before the first and second FC layers, a Dropout layer was inserted during training with 0.1 dropout probability. Even though this method was designed to detect the presence of PD, and not its severity, since the only corpus available to us was the subset of the Spanish PD Corpus used in the 2015's Interspeech ComParE Challenge, which does not include recordings for healthy controls, we could only train the network for regression. Nevertheless, results obtained by [66], show that the severity of the disease can be determined through the energy related characteristics of each transition region, although, in their case, this was modelled using higher level features, such as MFCCs and Bark band energies.

C.2 Results

Table C.1: Results obtained for Parkinson's using On/Offset Spectrograms for the CNN

Method	RMSE	MAE	ρ
DNN Baseline	16.3	12.4	0.492
Onsets + WA	17.7	14.5	0.019
Offsets + WA	18.0	14.7	-0.12

Table C.1 and Table C.2 display the results regarding the PD experiment performed following the methods previously described. Here we used the same metrics as in the results concerning PD in the previous chapter (Chapter 3, Section 3.4). Since the network is trained with individual On/Offset Spectrograms, we include results at the utterance level, computed by averaging the predictions for each

Table C.2: Results obtained for Parkinson's using On/Offset Spectrograms for the ECNN

Method	RMSE	MAE	ρ
DNN Baseline	16.3	12.4	0.492
Onsets	17.8	14.5	0.003
Offsets	17.9	14.5	-0.008

frame. The results show that while there is not a large degradation from the baseline to the CNN and ECNN, considering the RMSE and the MAE, when taking the Spearman's Correlation Coefficient into account there is a very large degradation. The values for this coefficient show that there is no correlation between the target vector and the output of the network. Furthermore, when inspecting the value of each prediction, we noted that all fell close to the average score of the training data, with most belonging to the interval $[30, 40]$. The reason for this might be that the network is not able to differentiate between the severeness levels of PD from the on/offset spectrograms, and that these may only be suitable to distinguish parkinsonian from non-parkinsonian speakers. Nevertheless, it would be important, as future work, to clarify if the results obtained in this experiment are caused by the fact that this approach is not suitable for a regression, and re-test this approach for a classification task.

As in Chapter 4, our results could not be computed using encrypted data, due to time it took to compute a single prediction. For the network used in this experiment, a single encrypted prediction took 56 minutes to be computed, making it unfeasible to provide results with encrypted data.