KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY

DE NAYER (SINT-KATELIJNE-WAVER) CAMPUS

Secure face matching

Felix LERNER

Promotor: Prof. dr. ir. Toon Goedemé
Co-promotor: Dr. Pradip Mainali (Onespan)

Masterproef ingediend tot het behalen van de graad van master of Science in de industriële wetenschappen: Industriele Wetenschappen Electronica-ICT

Academiejaar 2019 - 2020

Voorwoord

Het voorwoord vul je persoonlijk in met een appreciatie of dankbetuiging aan de mensen die je hebben bijgestaan tijdens het verwezenlijken van je masterproef en je hebben gesteund tijdens je studie.

Samenvatting

De (korte) samenvatting, toegankelijk voor een breed publiek, wordt in het Nederlands geschreven en bevat **maximum 3500 tekens**. Deze samenvatting moet ook verplicht opgeladen worden in KU Loket.

Abstract

We are affected with machine learning in many aspects of our daily lives, applications ranges from facial recognition to enhanced healthcare to self-driving cars. As companies outsource image classification tasks to cloud computing service providers, we see a rise in privacy concerns for both the users wishing to keep their data confidential, as for the company wishing to keep their classifier obfuscated.

Keywords: Computer vision, Cryptography, machine learning, secure multiparty computation, deep learning, object detetcion, privacy preserving, MLaaS, encryption

Contents

List of Figures

List of Tables

Lijst van symbolen

Maak een lijst van de gebruikte symbolen. Geef het symbool, naam en eenheid. Gebruik steeds SI-eenheden en gebruik de symbolen en namen zoals deze voorkomen in de hedendaagse literatuur en normen. De symbolen worden alfabetisch gerangschikt in opeenvolgende lijsten: kleine letters, hoofdletters, Griekse kleine letters, Griekse hoofdletters. Onderstaande tabel geeft het format dat kan ingevuld en uitgebreid worden. Wanneer het symbool een eerste maal in de tekst of in een formule wordt gebruikt, moet het symbool verklaard worden. Verwijder deze tekst wanneer je je thesis maakt.

b	Breedte	$[mm]$
A	Oppervlakte van de dwarsdoorsnede	$[mm^2]$
c	Lichtsnelheid	$[m/s]$

Lijst van afkortingen

MPC Secure Multiparty Computation
MLaaS Machine Learning as a Service
CNN Convolutional Neural Network
ReLU Rectified Linear Unit
DNN Deep Neural Network
GPU graphics processing unit

1 Introduction

Deep learning-based object detection on images is a hot topic for researchers and interest in machine learning is steadily growing among miscellaneous businesses. Facial recognition is one of the many applications machine learning has to offer. A face recognition algorithm tries to recognise faces of the same person. Faces are very unique parts of our body, thus face matching can be used as a means to do biometric authentication. In this case, a client sends a picture containing their face to an external service which grants the client access if the face is similar to the one stored in the database.

More and more users are concerned about their privacy and the security of their data stored and processed on servers. Not only are they afraid of malicious hackers stealing their sensitive data, they also fear the servers operator will use their data for purposes other than the user agreed to. Big corporations have been found guilty of collecting user data for unethical purposes Cadwalladr and Graham-Harrison (2018).

Secure multiparty computation (MPC) is a subfield of cryptography, making it possible for a party to run an algortihm on confidential data, that is supposed to stay unknown even to the party running the algorithm. There exist different methods to perform privacy-preserving computations, MPC is the one we will use.

Secure Multiparty Computation and Machine Learning aren't new concepts, in fact they exist for over 40 years. But with the rise of big data and processing power lately, there has been an increase in research into these fields.

Because of these concerns researchers are looking for technologies to enhance the privacy of the user during the processing of it's data on a server.

Onespan[1] (formerly VASCO Data Security International, Inc.) is a global company were most of our research was done. Onespan offers a series of security and authentication products and technologies and specializes in digital identity and anti-fraud solutions. The company continues to be active in research and innovation in different fields of technology, especially cryptography and data science.

In this thesis we study the applicability of MPC protocols on deep learning-based face matching and try to implement a privacy-preserving face matching algorithm.

1.1 Problem

The use of third party MLaaS (Machine Learning as a Service) providers or any cloud computing solution, as processing power for an image classification task, raises privacy concerns as sensitive images of users need to be sent to servers running an instance of the neural network.

It's important to note that the transport of the image from the client to the server is deemed to be secure, since the parties can make use of reliable HTTPS (Hypertext Transfer Protocol Secure) connections.

The user's images, however, are stored in plaintext [2] on the server, as well as the computed output of the image.

Furthermore the whole design of the neural network including all trained parameters needs to be stored on the servers of the third party, for the image classifier to function. Both of these remote processing solutions require a considerably amount of trust in the third party. Since the third party could potentially exploit the user data for commercial purposes or even steal the intellectual property of the image classifier. Of course most cloud computing service providers are not inherently malicious. But as long as the user's data is stored in cleartext on the server, there is a risk that the service provider could turn malicious. Or even worse, hackers could break in to the server and breach the confidential user data.

In this thesis we try to tackle the need to trust a third party MLaaS provider. We want it to compute an encrypted image on an obfuscated neural network to ouput a correct encrypted result. This encrypted result shall be sent to the client, which will then decrypt it.

1.2 Hypothesis

How can we securely compute the inference of a deep learning-based facial recognition neural network?

With the use of MPC protocols we can implement methods such that we can compute a whole face recognition convolutional neural network on an encrypted image of a face. This preserves the privacy of the user while allowing the computation to be outsourced to an untrusted third party.

[1] Onespan's official website: `www.onespan.com`

[2] plaintext or cleartext are common cryptographic terms for unencrypted data.

How can we optimise the secure facial recognition task to run more efficiently?

We predict a drastic decrease in perfomance when running inference on the privacy-preserving neural network, because MPC is a protocol over a network of parties the computational time will not be the only factor to account for. We will try to find performance optimizations along the way of implementing a proof of concept by looking at existing optimisation concepts for MPC as well as optimisation solutions for neural networks.

2

Literature study

In this chapter we will take a quick look at how convolutional neural networks (CNN) function, layer by layer. We will also learn how secure multiparty computation (MPC) in general is possible. Finally, we will discuss the related work on combining these two subjects so far.

The reason we don't just train our network on encrypted images as input data. Is because machine learning is generally based on discovering statistical patterns in data and the whole point of encryption is to make sure there is no statistical patterns and with truly random data there is none (think of image noise figure 2.1). In addition there is no heuristic you can use to tell if you are getting close to a correct classification.

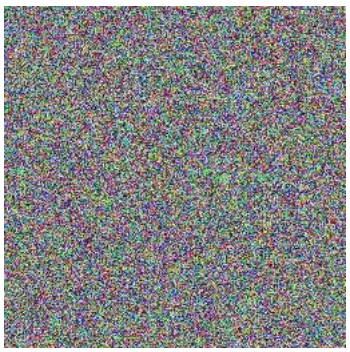

Figure 2.1: Random noise

2.1 Convolutional neural network

Convolution neural networks (CNN) is a special type of neural network used for images. The spatial properties of the pixels in the image are used during the evaluation of the input, meaning the neighbouring pixels of a central pixel impact the output to the next layer of that central pixel while pixels further away do not. CNNs are made of multiple layers. Typically, as you move further from the input layer to the output layer the dimensionality reduces, we can say the input gets mapped on a desired output manifold. Inputs that we classify as similar are supposed to be in the same region in the output manifold. Neural networks go through two phases: training and inference. A neural network needs to be trained in order to achieve good results. After the training the network will try to predict things based on the inputted data, in this phase the parameters of the network do not change.

2.1.1 Convolution layer

In this layer a discrete convolution of a kernel K shifting over an image I is performed, as shown in equation 2.1. The kernel has parameters also known as weights, so that certain features get extracted from the input.

$$(I * K)[m,n] = \sum_j \sum_k I[m-j, n-k]K[j,k] \tag{2.1}$$

The output of this layer is a convolved feature map. The parameters of the kernel are usually floating-point numbers and can be positive or negative.

2.1.2 Activation function

Since these convolutions are simple lineair operations and most image classification tasks require non-lineair classifiers, non-lineairity needs to be added to the neural network. This is achieved through adding a non-linear activation function after a convolution or fully connected layer. The most popular activation function is the rectified linear unit (ReLU) as seen in equation 2.2 and figure 2.2.

$$f(x) = max(0,x) \tag{2.2}$$

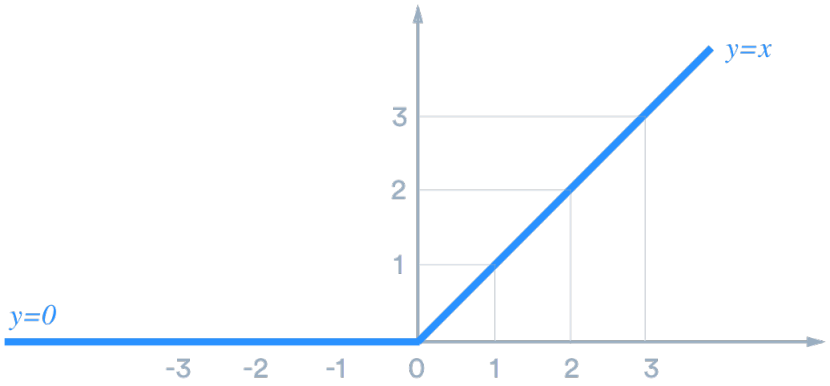

Figure 2.2: ReLu activation function

2.1.3 Pooling layer

The dimensionality reduction we talked about in the beginning of this chapter happens primarily in the pooling layer [1]. A kernel is shifted over the image. In the case of max-pooling the kernel selects the maximum value of the portion of the image it covers, to create the new dimensionality reduced image. A portion of the image value is theoretically lost, but because we only retain the maximum value in the window we sort of extract a feature from the input matrix. An example of this process can be seen in figure 2.3.

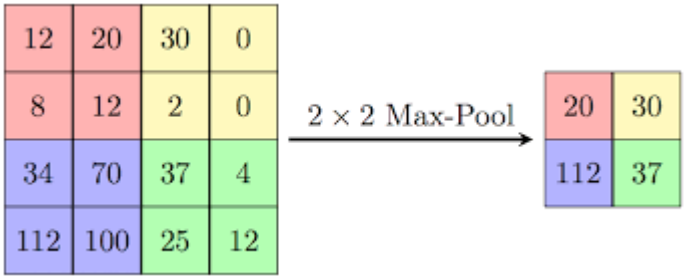

Figure 2.3: Example of 2×2 max-pooling

2.1.4 Fully connected layer

The fully connected layer is a multilayer perceptron that discriminates different object classes and identifies identical ones. All elements in vector h_{i-1}^{out} have their own bias B_i and weight W_i so that h_i^{in} can be calculated for each layer i according to equation 2.3.

$$h_i^{in} = h_{i-1}^{out} \cdot W_i + B_i \tag{2.3}$$

The fully connected layers usually come after the last convolution layer. The output of a convolution layer is a tensor, this means that the output needs to be flattened to a one-dimensional array. The

[1] Dimensionality reduction can also be combined with feature extraction in the convolution layer.

last layer of the fully connected layers is called the output layer. This output layer determines the classification.

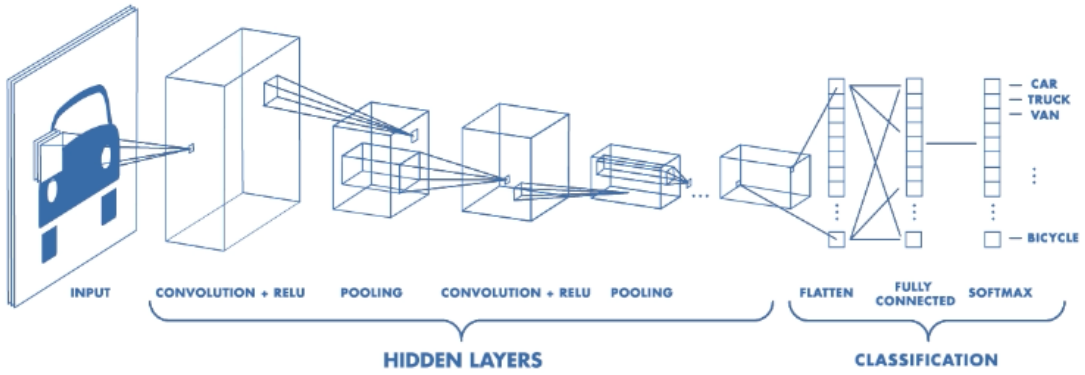

Figure 2.4: Overview of CNN

When all these layers are connected to each other, as you can see in figure 2.4, we speak of a model with a CNN architecture. And if there are enough layers between the input layer and the output layer, we say the model is a Deep Neural Network (DNN).

2.1.5 Face matching

Face matching is an algorithm that tries to match two faces of the same person. Face matching can be performed with convolutional neural networks an image gets fed to the network and produces an output vector depending on what face was in the image. Koch et al. (2015) showed that a siamese neural network makes it possible to not only recognise new data (unseen during the training) but to also recognise entirely new classes. In the case of face matching each person's face is a class. With siamese neural networks it is thus possible to recognise faces of persons which the network didn't see during training.

A siamese neural network consists of two CNN's. These CNN's are identical. A siamese neural network has to be fed two images, it then produces two output vectors. When these two images have faces that are similar the euclidean distance between the two output vectors will be small. When the two faces on the images are dissimilar, the euclidean distance between the two vectors is large. Thus to determine if two images are of the same person, we calculate the euclidean distance (equation 2.4) between the two output vectors, if the distance lies under a certain threshold we accept that the two faces belong to the same person. If one output vector of the two images is stored in a database, an application can use this algorithm to allow for biometric authentication. This output vector is also referred to as the embedding of the face or a point on a manifold. A manifold is a topological space that locally resembles the Euclidean space near each point, this means that each point of that n-th dimensional manifold has a neighborhood that is homeomorphic

to the Euclidean space of dimension n. This is great news, we can compare two embeddings on the manifold by calculating the Euclidean distance between the two points, the only condition is that the two points are in eachothers locality or neighborhood. A general overview of a siamese neural network architecture can be found in figure 2.5.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2.4}$$

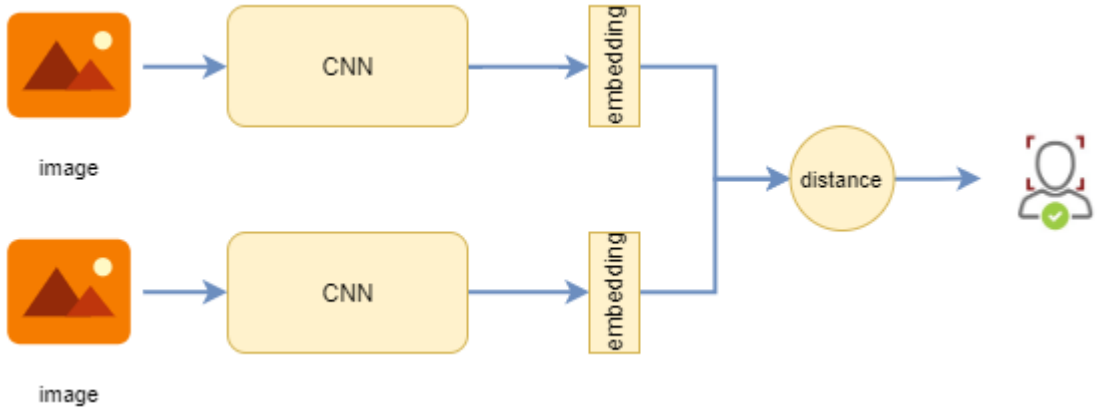

Figure 2.5: Overview of siamese neural network

2.2 Secure multiparty computation

Secure multiparty coputation is a protocol that is used between n number of parties P. Each of these parties has private data also called a secret S. With MPC it is possible for these parties to compute a public[2] function f on the secrets. Such that a party $P_i \in P$ only knows his secret $s_i \in S$ and the public securely computed output $f(s_0, s_2, s_{n-1})$ after the protocol has succesfully finished. A classic application is Yao's Millionaires' problem Yao (1982) in wich two millionaires wish to know who is richer, there is catch however. Instead of making their balances publicly known. They wish to keep their balances a secret. In this case the number of parties n is 2 the secrets s_0 and s_1 are their balances. The public function $f(s_0, s_1) = 1$ if $s_0 < s_1$ and 0 otherwise.

We categorize 2 types of parties based on their willingness to deviate from the correct predefined protocol.

- **Honest parties:** Parties do not wish to know other parties secrets and will never reveal the secret.
- **Honest but curious parties (passive):** Parties wish to know other parties secrets but will not deviate from the protocol at any time. Also called semi-honest parties.

[2]Public or global means known to all parties, while private or local means known only by the corresponding party.

- **Malicious parties (active):** Parties wish to know other parties secrets and wish to change output of computation to favourable result. Parties will deviate from the protocol to cheat and change the outcome at any time.

In practice the whole set of parties will exist of subsets of these different types of parties. Ideally every party is honest, but this is rather a naive way of thinking.

If the two millionaires are honest but curious parties, they will not deviate from the protocol and they will computer the correct output as a result they will know who is the richer millionair but they won't know how much money the other one has. In the other case one of the two millionaires is corrupt and acts maliciously, the honest millionair will follow protocol while the dishonest millionair will deviate from the protocol to change the result in his favour. In the event that the dishonest millionair is poorer he will change the outcome thus appearing richer. From now on, we assume the set of parties are a mix of honest and semi-honest parties, unless specified otherwise. We also assume the communication between the different parties to be secure.

The efficiency of an MPC protocol is defined by three metrics:

- **round complexity:** The amount of rounds needed in the protocol. In one round each party can read all messages sent to the party in the previous round as well as perform an arbitrary amount of local computation and finally send messages to all other parties.
- **communication complexity:** The amount of communication between all parties (measured in bits).
- **computational complexity:** The number of primitive operations performed by all parties.

In general, the computational complexity is very low while the communication complexity dominates the protocol's total complexity. Thus an estimate of the overall complexity can be measured by combining the round and communication complexity. This means the efficiency of MPC protocols is heavily based on the network's latency rather than the network's throughput.

2.2.1 Secret sharing

In order to do secure computing, the parties need to split their secret into secret shares. A secret sharing method can be used by the secret holder to split a secret into a number of shares. Combining these shares will reveal the secret, while individual shares alone will leak nothing about the secret. In a (t,n) threshold secret sharing sheme parties must combine at least t shares of the total n shares, to obtain the secret. We can now set a threshold t high enough, denying the secret to small curious parties and allowing to reveal the secret when a majority ($\geq t$) consensus is reached. Shamir's secret sharing scheme Shamir (1979) is based on polynomial interpolation and the essential idea is that it takes at least t points in order to define a polynomial $p(x)$ of degree $t-1$. Given a set of t points in a 2-dimensional carthesian system $(x_1,y_1),(x_2,y_2),...,(x_t,y_t)$, there exists only one polynomial of degree $t-1$. This can be proven and the mathematical construction

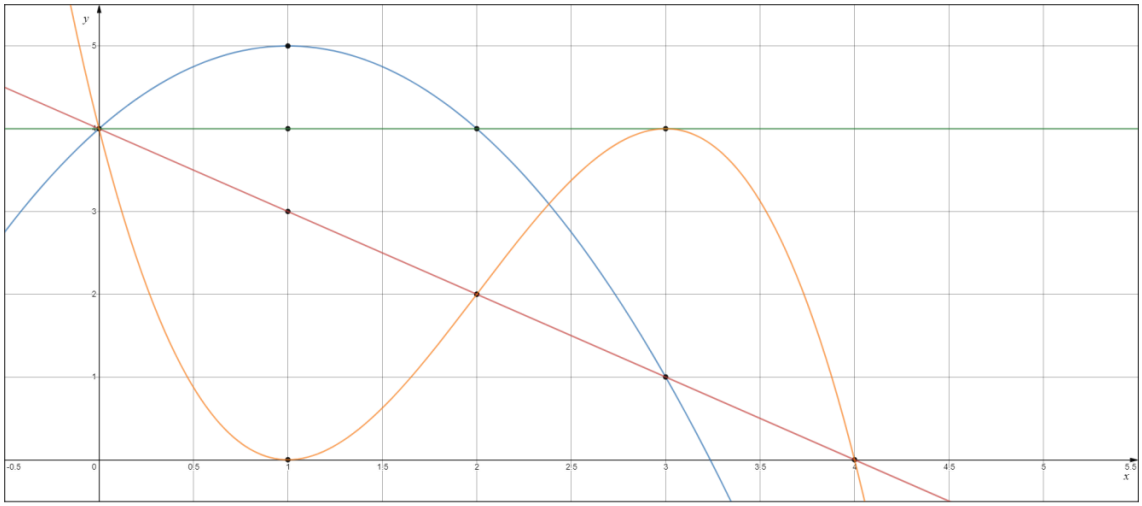

Figure 2.6: Shares for different t with secret $s = 4$

of a polynomial $p(x)$ of degree $t-1$ based on a set of t points can be calculated using Lagrange's interpolation formula 2.5.

$$p(x) = \sum_{i=1}^{t} y_i \delta_i(x) \quad \text{with} \quad \delta_i(x) = \prod_{1 \leq j < t; i \neq j} \frac{x - x_i}{x_j - x_i} \tag{2.5}$$

With this in mind, a secret dealer can now share his secret s to n parties by choosing a random $t-1$ degree polynomial $p(x) = a_0 + a_1 x + ... + a_{t-1} x^{t-1}$ in which a_0 is the secret or the number representation of the secret if the secret is not a number. The dealer now calculates n points on the polynomial starting from $x = 1$, because the secret is located at $x = 0$. Each party $P_i \in P_1, P_2, ..., P_n$ is given a different single point (x_i, y_i), at this stage the secret is shared. The convention for the notation of a shared secret is $[s]$. We can say a party P_i is holding a share in the form of a point (x_i, y_i) or shortened $[s]_i$.

To recombine the secret, the parties simply broadcast or send their shares to a central entity, if more than t shares are known, it suffices to calculate the Lagrange polynomial $p(x)$ and $s = p(0)$. In the case of not having enough shares, the Lagrange polynomial containing the secret becomes impossible to calculate since every polynomial is equally likely, thus revealing absolutely nothing about the secret.

Note that all arithmetic in this section can be done over some finite field $\mathbb{F}_q$ to speed up the algorithms.

A Generalisation of this (t, n)-Shamir secret sharing scheme from thesis de Hoogh (2012) as follows (Protocol 2.1 and Protocol 2.2):

1. **Share Generation:** To share $s \in \mathbb{F}_q$, the dealer generates random $a_1, ..., a_t \in \mathbb{F}_q$ and puts $p(x) = s + a_1 x + ... + a_t x_t$. Then the dealer computes $[s]_i = p(i)$.

2. **Share Distribution:** For each $i \in 1, ..., n$, the dealer sends $[s]_i$ to party P_i.

3. **Secret Reconstruction:** Let $D \subset 1,...,n$ be a set of size $t+1$. Each party P_i for $i \in D$ sends his share $[s]_i$ to all parties. Then, each party reconstructs the secret via Lagrange interpolation.

In figure 2.6 a visualisation of this scheme shows us that in order to get the secret value, $s = 4$ in this case, we need to know at least $t+1$ points for a given t. Two points are needed to recombine a first order polynomial (red line), three points for a second order polynomial and so on.

It's important to make sure the parties are distributed and to try minimizing the incentive to collude. Distribution is needed to lower the risk of having a malicious party taking over control of the network, this can be done by splitting the parties up to many small stakeholders instead of a couple centralized stakeholders. These stakeholders could be rival companies, local governments or even individual citizens. The only way to totally avoid collusion, is to let the secret holder partake to the MPC. However, this is not wished, as we want the computation to be outsourced.

2.2.2 Operations

A neural network can be seen as an enormous function with millions of coefficients. Lucky for us the function can be broken up into 3 different operations: addition and multiplication in fully connected and convolution layers and a relational operator for max pooling and ReLU activation function. We will now show how to securely compute each of these basic operations.

2.2.2.1 Arithmetic operators

Addition, subtraction, division and multiplication all fall under arithmetic operations. In this section we will take a look at how these operations can be performed in MPC.

linear protocols: Since Shamir's secrete sharing scheme is a linear sharing scheme, each party P_i can locally compute any linear combination of a public or secret value with their secret share $[s]_i$. This gives us following operations:

- **Addition of secret and public value ($[c] \leftarrow [a] + \beta$):** Each party P_i locally adds the public value $\beta \in \mathbb{F}_q$ to it's share $[a]_i$, resulting in the new share $[c]_i = [a]_i + \beta$. Since all parties add the same β, this value is a constant.
- **Multiplication of secret and public value ($[c] \leftarrow [a] \cdot \beta$):** Each party P_i locally multiplies the public (constant) value $\beta \in \mathbb{F}_q$ with it's share $[a]_i$, resulting in the new share $[c]_i = [a]_i \cdot \beta$.
- **Addition of multiple secrets ($[c] \leftarrow [a] + [b]$):** Each party P_i locally adds it's secret shares $[a]_i$ and $[b]_i$, resulting in the new share $[c]_i = [a]_i + [b]_i$.

The last operation is visually demonstrated in figure 2.7. In this example secret $a = 1$ (green) and secret $b = 3$ (red). After each party P_i locally computes the addition of $[a]_i$ and $[b]_i$ and stores it as a new share $[c]_i$. After broadcasting the new, computed share $[c]$, they can recombine the shares

Figure 2.7: Adding two secrets for $n = 3$ parties

via Lagrange interpolation to get the polynomial of the combined shares $[c]$ and more importantly the output of the addition of the two secrets a and b. This happens all with zero knowledge about a or b. All of the computations are done locally and no communication other than the inital share distribution and the share reconstruction is needed.

The inverse operations (subtraction and division) are possible too, and the protocol is the same.

Multiplication protocol: Multiplication of two secret values is more challenging since it isn't a linear operation. Their are multiple methods to perform secret multiplication, the method described by Ben-Or et al. (1988) also called the BGW protocol (initials of the authors) has to solve two problems along the way of computing $c = a \cdot b$.

Assume a and b are respectively encoded by $f(x)$ and $g(x)$ and $n \geq 2t + 1$. The free coefficient of $h(x) = f(x) \cdot g(x)$ is simply $h(0) = f(0) \cdot g(0)$, this means a simple multiplication of the polynomials would be sufficient to compute the multiplication of the secrets. There is however a major problem, multiplication of two polynomials of degree t yields a polynomial of degree $2t$.

While this poses no problem with interpolating $h(x)$ from it's shares $[c]$ since $n \geq 2t + 1$, further multiplications will continue to raise the degree to a level where $t > n$ making it impossible to interpolate the resulting polynomial $h(x)$. The second problem is harder to spot. The polynomial $h(x)$ as a result of the multiplication of polynomials $f(x)$ and $g(x)$ is reducible (since it's a multiplication). In other words $f(x)$ and $g(x)$ are uniformly random polynomials of degree t, but $h(x)$ as a multi-

plication of two random polynomials is not irreducible, therefore $h(x)$ is not uniformly random. A uniformly random polynomial is a polynomial with coefficients that are randomly sampled according to the uniform distribution, i.e. the coefficients are randomly sampled from a set where drawing each element is equally probable. To make sure the resulting polynomial stays of degree t and is uniformly random, the parties run a protocol to generate a random polynomial of degree $2t$.

The result of a multiplication of a uniformly random polynomials of degree t with a uniformly random polynomial of degree less than or equal to t is a polynomial of degree $2t$. Assuming the base field is Z_q, then there are q^{2t+1} such polynomials, but we would only get q^{t+1} polynomials. So the distribution is not uniform.

This protocol works as follows. Each party P_i randomly selects a private polynomial $q_i(x)$ with secret $q_i(0) = 0$ of degree $2t$ and distributes it's shares among the parties. Each party P_i now has n random shares $[q]_i^k$ (with $k : 1 \rightarrow n$). After each party adds all it's random shares they hold a secret, truly random polynomial $q(x)$ with a zero as free coefficient $q(0) = 0$. Each party P_i now computes the multiplication of the two secret values $[a]$ and $[b]$ and instead of using $[c]$ encoded as $h(x)$ we can add the random polynomial to the result, thus randomizing the coefficients and making the polynomial uniformly random. This will not mess up the result as now $a \cdot b$ is $(f(0) \cdot g(0)) + q(0)$ and $q(0)$ equals zero. This step is also called the randomization step.

The parties now run a protocol to reduce the degree of $h(x)$ to t. This protocol can be computed locally (no communication is required) by multiplying the shares of the polynomial $h(x)$ to a specific, matrix of constants. This will truncate the result $h(x)$ of degree $2t$ to a polynomial of degree t. Proof for this degree reduction step can be found in the study by Asharov and Lindell (2017).

After these two steps, the parties hold $c = a \cdot b$ encoded by $h(x)$ of degree t with coefficients uniformly distributed. They can now recombine their shares to find the polynomial $h(x)$ and the product of a and b, $h(0)$. The whole multiplication protocol requires only one additional round of communication, this happens during the distrubition of the shares of the random polynomials $q(x)_i$ in the randomization step. Note that since we conditioned the protocol to work in cases where $n \geq 2t + 1$, a majority of honest parties is needed. Opposed to the linear protocols explained earlier, where only one party needed to be honest.

A large proportion of the neural network is ready to be transformed with these secure arithmetic operators, namely the convolution layer and the fully connected layer.

2.2.2.2 Relational operators

In this section we will focus on comparison between secrets. There are two important protocols in secure comparison, equality testing and greater-than testing.

Equality testing: Suppose we have two shared secrets $[a]$ and $[b]$ and the parties want to know if $a = b$, without knowing a or b. The easy way, would be to just securely compute and reveal $c = a - b$. This would however reveal secret b if a was to be revealed, since $b = a - c$. To make sure the the output of the subtraction is irreducible and uniformly random Franklin and Haber (1996)

came up with an idea to let the parties generate a random shared non-zero secret $[r]$. Then compute and reveal $c = (a - b) \cdot r$. Since $r \neq 0$, if $c = 0$, a must be equal to b. If $c \neq 0$, a and b are not equal. If we want to compare a secret with a public value β, we just take a as the secret and use a β instead of $[b]$ in the protocol. In this case the naive method $c = (a - \beta)$ would just give away the secret value even if $a \neq \beta$, so it's absolutely required to use a random multiplier r to hide the reversible operation.

The generation of a random shared non-zero secret $[r]$ appears to be very similar to the generation of the random share in the randomization step of the multiplication protocol but there is one difference, in this case the secret value must be different than zero (invertible), while in the multiplication protocol the random share needed to be equal to zero. The idea is to generate two shared random secrets $[x]$ and $[y]$. Then the parties compute the product of the two secrets $z = x \cdot y$ and reveal z. If $z \neq 0$, both random secrets x and y are non-zero, thus applicable in the equality testing protocol. If $z = 0$ repeat the random share generation protocol with different random shared secrets and retry the multiplication with these new shares.

Greater-than testing: Often we want to know more about two secrets than just equal or different. We want a protocol comparing two secrets a and b that returns a boolean for $a > b$. There exists multiple different protocols for greater-than comparisons. The one we use in our implementation is published in Erkin et al. (2009). This protocol compares the two secrets on bit-level.

2.2.3 Number representation

When a CNN gets computed on cleartext data almost all parameters of the CNN are floating-points, it is thus important that these numbers can be transformed to representations suitable for MPC.

Floating-points can be implemented in MPC. But they usually come with a high complexity cost for addition as well as multiplication and comparison tests. The advantage of floats over other number representations like scaled integers and fixed-point numbers is that the the maximal rounding error scales with the magnitude of the number. Unlike floats fixed-points have a rounding error of a fixed size. Campmans (2018) studied the use of fixed-point numbers as an alternative for floating-point numbers in MPC protocols. He states that with fixed-point numbers no comparison needs to be done during multiplication of two secrets. While the costly comparison is needed when multiplicating two secret floating-point numbers.

A fixed point data type is essentialy an integer that is scaled by a chosen factor. Let's say our numbers in the MPC protocol never go out of the bounds $[-500, 500]$ and we want a precision of 2 fractional digits. Then we can use the integers bounded by $[-50000, 50000]$ to perform all arithmetic. The arithmetic operations will be the same as when we use integers. But we will actually be computing arithmetic operations on fixed-point numbers. This allows for an efficient approach to floating-point numbers while minimizing rounding errors. Of course we need to make sure that the operations don't overflow and that precision loss is small.

2.3 Overview

We now have seen the miscellaneous protocols MPC has to offer. From now on we will treat the seen MPC protocols like black boxes that accept inputs in the form of secret shares and compute outputs in the form of secret shares. As long as the parties do not collude to find the secret, the protocol is secure.
Overview of protocols:

- Secure addition of two secrets or a secret and a public value (F_{add}).
- Secure multiplication of secret and public value ($F_{mul}^{constant}$).
- Secure multiplication of two secrets (F_{mul}).
- Equality testing of two secrets or a secret and public value (F_{eq}).
- Greater-than testing of two secrets (F_{gt}).

This set of protocols can be used to transform any basic CNN to a secure one.

In figure 2.8 a high-level workflow of secure face recognition is shown. The MPC protocol works for three parties [3], who each have their own computing instance connected to the MPC network and the client. The workflow commences by acquiring an image of the clients face, this can be done by taking a photograph with the front-facing camera of the clients smartphone. The client then performs secret sharing on the image [4] and sends the shared secret to the participating parties. The parties receive their shares and jointly compute the output of the face recognition model on the given shared image of the face. The face recognition model can be public or secret. In the case of a secret model, the secret shares of the weights and biases need to be sent to the participating parties as well. Note that this only has to be done once, during initialisation. After the computation is finished, the parties send their resulting secret shares to the client. The client interpolates the shares to find the secret values determining if her face matches or not.

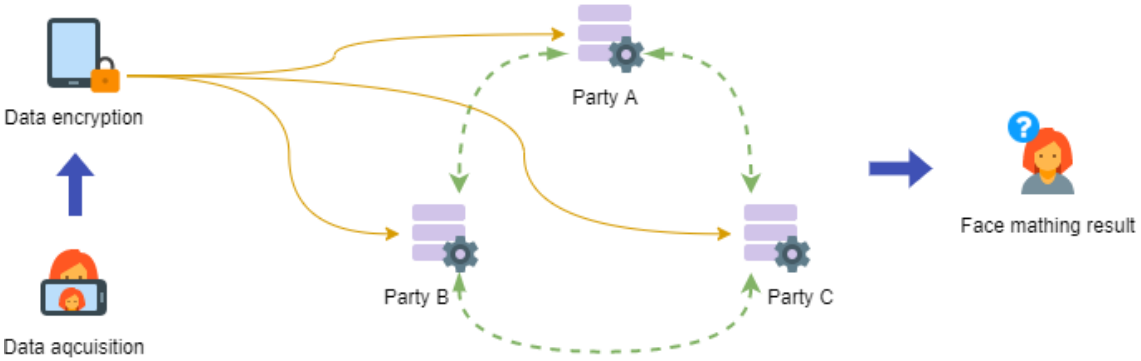

Figure 2.8: Workflow of secure face matching

[3] This is the smallest amount of parties needed in order to do secure multiplication (F_{mul}).
[4] Secret sharing is performed for each pixel of the image.

This protocol ensures that the client is the only one having knowledge about the cleartext of the image and the output of the face recognition algorithm. This protocol also offers an obfuscation of the model's parameters, the parties and the client have no knowledge about the parameters if the owner of the model uses secret sharing to send the parameters to the parties.

2.4 Related work

Erkin et al. (2009) presented for the first time a privacy-preserving face recognition alogithm using MPC, this was before the rise of machine learning. Thus they were restricted to using eigenvalues of the faces described in Turk and Pentland (1991). They show that their privacy-preserving face recognition algorithm is as reliable as the normal face recognition algorithm. Unfortunately, the computational cost for producing the eigenvalues of an image is way less than producing a result for a CNN.

Mainali and Shepherd (2019) recently published the first framework for privacy-ehancing fall detection from a body-worn inertial measurement unit using traditional machine learning and MPC. The data they work with is time-series inertial measurements this has a significantly lower dimensionality than an image. Traditional machine learning classifiers require less computations than deep neural networks. The authors hint to secure video-based fall detection as a possibility future research and to which extent secure video-based fall detection can be performed real-time with high-dimensionality.

Gilad-Bachrach et al. (2016) discuss implementing CryptoNets, a fully homomorphic encryption (FHE) based approach for privacy-preserving optical character recognition task on the Modified National Institute of Standards and Technology (MNIST) database of handwritten digits. FHE is another approach to making machine learning privacy-preserving. But since we focus on MPC in this thesis, we refer the interested reader to the paper written by Gentry et al. (2009).

Barni et al. (2006) presented an algorithm that does privacy-preserving computation on a neural network using MPC. The data owner encrypts the input, via secret sharing, and sends the secret shares to the participating MPC parties. The parties compute the inner product of the weights of the layer and the data. The product gets sent back to the data owner. The data owner then adds non-linearity via an activation function on the outputs, this is done in cleartext since the data owner can decrypt the secret shares and is allowed to see is own data. The result then gets encrypted again and is sent to the MPC parties to compute the inner product in the second layer. This protocol is done for all the parties. In this algorithm the data owner communicates multiple times with the MPC parties. Some computation of the neural network is also done by the data owner. A strong cooperation between the parties and the data owner is thus needed.

Campmans (2018) worked on optimisation of existing MPC protocols for convolutional neural network operations. He also suggested using fixed-point numbers instead of floating-point, losing some accuracy for more efficiency. He also explored the use of discrete fourier transforms as an approach to costly convolutions in MPC. The convolutions are costly because they require lots of multiplications.

Makri et al. (2019) proposed EPIC an efficient private immage classification system based on support vector machine (SVM) learning. They focus on efficiency by minimizing the load on the privacy-preserving part. Their experiments conclude that their is a tradeoff between efficiency and accuracy of the privacy-preserving image classification systems. They consider the deployment of a CNN with current MPC protocols in an active adversary MPC environment to be computationaly prohibitive. It should be noted that in our implementation, we assume the MPC environment to be made of passive adversaries.

Taigman et al. (2014) presented DeepFace, a face recognition algorithm trained on millions of images of Facebook users. It achieves a stunning 97.35% on the Labeled Faces in the Wild (LFW) dataset, which is the classic benchmark dataset for face recognition, closing the gap to human-level perfomance. The authors made use of a siamese neural network. But this alogrithm is already outdated. Deep learning-based networks use more layers than a regular neural network, Wang and Deng (2018) give an overview of the recently published alogrithms, most of them use a deep learning-based architecture. Some of the alogrithms achieve an acurracy of around 99.8% on the LFW dataset. This surpasses the ability of face recognition by humans. These neural networks often have millions if not billions of parameters. This means that the computational complexity is very high and it is unlikely that we will find a way to efficiently make this type of neural networks secure. Instead we will focus on slightly lighter neural networks with less layers and parameters.

2.5 Conclusion

It is theoretically possible to implement a privacy-preserving CNN that works just like a normal CNN on cleartext data. But in practice we expect some resistance based on the related work, the main problem is the complexity that arises when performing MPC protocols on high dimensional data like images. There is a tradeoff between precision and complexity. Our first steps shall be to implement a naive design of the MPC protocol for CNN's. Then we will try to improve the efficiency, by adding existing optimisation solutions, thus minimizing the time it takes to perform one face matching task. We will compare our results with other privacy-preserving techniques for CNN's.

3 Implementation

In this chapter we will explain how we implemented MPC for face matching algorithms. We will do this by giving a high-level overview of the system and then diving deeper in more interesting parts. With this information and the code in the appendix, you should be able to reproduce our experiments. You can also checkout our Github repository[1].

3.1 Specifications

There are two major subprojects. The first subproject (chapter 3.1.1) is making sure we can generate the appropriate parameters for the face matching network. It is important that the model is accurate enough. The second subproject (chapter 3.1.2) is about transforming the classic machine learning functions to secure ones. To add this security or privacy-preserving factor we use a MPC framework.

3.1.1 Deep Learning

A machine learning project usually includes on of the popular frameworks available to the public. Since we were already familiar with Pytorch we used this library as python package.

Pytorch [2] provides us with a deep learning research platform that provides maximum flexibility and speed. It's fairly easy to use but that doesn't mean we can't design more complex models or features. Pytorch uses tensors, tensors are multi-dimensional matrix containing elements of a single

[1] `github.com/Fluxmux/securefacematching`
[2] `pytorch.org`

data type. Designing a neural network with Pytorch is as simple as defining a class with the layers in the correct order. An example of a neural network written using pytorch can be seen in the following code 3.1

Listing 3.1: Pytorch neural network example

```
class SiameseNetwork(nn.Module):
    def __init__(self):
        super(SiameseNetwork, self).__init__()

        self.cnn = nn.Sequential(
            nn.ReflectionPad2d(1),
            nn.Conv2d(1, 16, kernel_size=5),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(16),
            nn.MaxPool2d(kernel_size=2, stride=2),
            ...
        )
```

Making an accurate face matching neural network. Is a process that involves three major steps.

First of all the design or architecture of the network gets chosen. There exist a number of different topologies used in deep neural networks. But often choosing which one to take and how many layers to use, is the most difficult task. We will cover the architecture of the model in chapter 3.2. Adding more layers is the same as adding more parameters. And a model with more parameters is more complex.

The second step is called the traing of the neural network. Training is done using a part of the dataset that is specific for training and shouldn't be used for anything else. Training a neural network takes some time, but the process can be sped up by using graphics processing units (GPU). We were lucky enough to have a dedicated GPU server at our disposal. While training the network is an easy task, we should look out for overfitting or underfitting.

Overfitting a model happens when there are too much parameters for a model or when the training was performed for too long. It will perform poorly on the validation dataset (part of dataset used to detect bad trainig behaviour) while performing excellent on the training dataset. There are two ways of overcoming overfitting. One way is to make the training dataset larger. Having more samples to train on, generalizes the learning model better. The other way is to design the model with fewer parameters, making it less complex. We use learning curves to track the training process of our face matching algortihm. An example of a more or less correct learning curve can be found in figure 3.1.

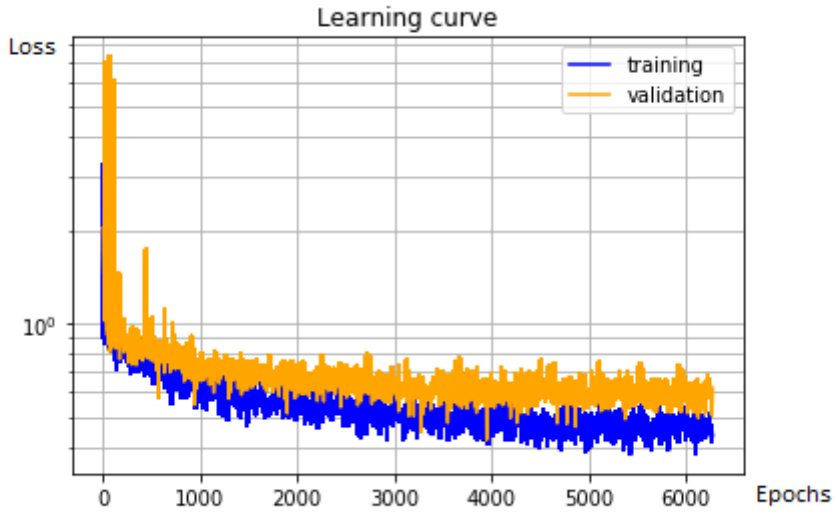

Figure 3.1: Learning curves are used to track the training of a model

Underfitting happens when a machine learning algorithm cannot capture the underlying structure of data. The model can't fit the data enough. Underfitting is more difficult to spot but easier to overcome. We can overcome this problem by making our model more complex.

As you can see by now, the architecture of a model is extremely important for it to function as wished.

The third step also called the hyperparameter tuning or hyperparameter optimisation step, is what makes a good machine learning model even better. This process is not at all logic, experience and intuition can facilitate this. Hyperparameters are all the parameters whose values are set before the learning process begins. There are different methods for optimizing hyperparameters, since we wanted to learn the model and how it behaves relating to small changes in the values of the hyperparameters we went for Manual Random Search. Note that this took us some time because this involves multiple training steps. But since we had a GPU server at our disposal we could parallelize this task. In our case this step improved our accuracy by about 5%.

We used the Database of Faces[3] to train and test our face matching algortihm. The dataset contains a set of 10 images of frontal faces with different expressions per person and a total of 40 persons. The pictures are in pgm format which is extremely easy to interact with. They have a dimension of 1 x 92 x 112. An example of a set of images from the dataset can be seen in figure 3.2. We divided the dataset in to 3 parts: 75% training, 12.5% testing and another 12.5% for validation.

[3] `cam-orl.co.uk/facedatabase.html`

Figure 3.2: Example of faces in dataset

3.1.2 Secure Functions

3.2 Design

3.3 Conclusion

4 Evaluation

4.1 Results

4.1.1 Reliability results

4.1.2 Timing results

4.2 Discussion

4.3 Conclusion

5

Conclusion

Bibliography

Asharov, G. and Lindell, Y. (2017). A full proof of the bgw protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151.

Barni, M., Orlandi, C., and Piva, A. (2006). A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia and security*, pages 146–151. ACM.

Ben-Or, M., Goldwasser, S., and Wigderson, A. (1988). Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM.

Cadwalladr, C. and Graham-Harrison, E. (2018). Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The guardian*, 17:22.

Campmans, H. (2018). *Optimizing Convolutional Neural Networks in Multi-Party Computation*. PhD thesis, Master's thesis, Dept of Mathematics and Computer Science, TU Eindhoven.

de Hoogh, S. J. A. (2012). Design of large scale applications of secure multiparty computation: secure linear programming.

Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., and Toft, T. (2009). Privacy-preserving face recognition. In *International symposium on privacy enhancing technologies symposium*, pages 235–253. Springer.

Franklin, M. and Haber, S. (1996). Joint encryption and message-efficient secure computation. *Journal of Cryptology*, 9(4):217–232.

Gentry, C. et al. (2009). Fully homomorphic encryption using ideal lattices. In *Stoc*, volume 9, pages 169–178.

Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. (2016). Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210.

Koch, G., Zemel, R., and Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2.

Mainali, P. and Shepherd, C. (2019). Privacy-enhancing fall detection from remote sensor data using multi-party computation. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, page 73. ACM.

Makri, E., Rotaru, D., Smart, N. P., and Vercauteren, F. (2019). Epic: efficient private image classification (or: learning from the masters). In *Cryptographers' Track at the RSA Conference*, pages 473–492. Springer.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708.

Turk, M. A. and Pentland, A. P. (1991). Face recognition using eigenfaces. In *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591. IEEE.

Wang, M. and Deng, W. (2018). Deep face recognition: A survey. *arXiv preprint arXiv:1804.06655*.

Yao, A. C. (1982). Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE.

A

Uitleg over de appendices

Bijlagen worden bij voorkeur enkel elektronisch ter beschikking gesteld. Indien essentieel kunnen in overleg met de promotor bijlagen in de scriptie opgenomen worden of als apart boekdeel voorzien worden.

Er wordt wel steeds een lijst met vermelding van alle bijlagen opgenomen in de scriptie. Bijlagen worden genummerd het een drukletter A, B, C,...

Voorbeelden van bijlagen: Bijlage A: Detailtekeningen van de proefopstelling Bijlage B: Meetgegevens (op USB)

FACULTY OF ENGINEERING TECHNOLOGY
DE NAYER (SINT-KATELIJNE-WAVER) CAMPUS
Jan De Nayerlaan 5
2860 SINT-KATELIJNE-WAVER, België
tel. + 32 16 30 10 30
fet.denayer@kuleuven.be
www.fet.kuleuven.be