T.C.

ALTINBAŞ UNIVERSITY

Institute of Graduate Studies

Electrical and Computer Engineering

PRODUCING SECURE MULTIMODAL BIOMETRIC DESCRIPTORS USING ARTIFICIAL NEURAL NETWORKS

Raghad Saeed Hasan

Doctor of Philosophy

Supervisor

Asst. Prof. Dr. Doğu Çağdaş ATİLLA

Istanbul, 2021

PRODUCING SECURE MULTIMODAL BIOMETRIC DESCRIPTORS USING ARTIFICIAL NEURAL NETWORKS

by

Raghad Saeed Hasan

Electrical and Computer Engineering

Submitted to the Institute of Graduate Studies
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

ALTINBAŞ UNIVERSITY

2021

This thesis titled "PRODUCING SECURE MULTIMODAL BIOMETRIC DESCRIPTORS USING ARTIFICAL NEURAL NETWORKS" prepared and presented by "Raghad Saeed HASAN" was accepted as Doctor of Philosophy Thesis in Electrical and Computer Engineering.

Asst. Prof. Dr. Çağatay AYDIN	Asst. Prof. Dr. Doğu Çağdaş ATİLLA
Co-Supervisor	Supervisor

Thesis Defense Jury Members:

Asst. Prof. Dr. Doğu Çağdaş ATİLLA	School of Engineering and Architecture, Altinbaş University	_____
Asst. Prof. Dr. Cahit KARAKUŞ	School of Engineering and Architecture, Esenyurt University	_____
Assoc. Prof. Dr. Yasa EKŞİOĞLU ÖZOK	School of Engineering and Architecture, Altinbaş University	_____
Asst. Prof. Dr. Aytug Boyaci	National Defence University	_____
Asst. Prof. Dr. Oğuz ATA	School of Engineering and Architecture, Altinbaş University	_____

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

_____

Assoc. Prof. Dr. Oğuz BAYAT

Approval Date of Graduate School of

Science and Engineering: ____/____/____

Director

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Raghad Saeed Hasan

DEDICATION

To my Parents

The reason of what I become today.

To my sister & two brothers

To my Dear Husband Baraa Saad

You have been my inspiration, and my soul mate.

To my lovely daughter

And

A special thanks to Asst. Prof. Dr. Doğu Çağdaş ATİLLA

For his support, worth notes and close follow up.

ABSTRACT

PRODUCING SECURE MULTIMODAL BIOMETRIC DESCRIPTORS USING ARTIFICIAL NEURAL NETWORKS

HASAN, Raghad Saeed

Ph.D., Electrical and Computer Engineering, Altınbaş University

Supervisor: Asst. Prof. Dr. Doğu Çağdaş ATİLLA

Co. Supervisor: Asst. Prof. Dr. Çağatay AYDIN

Date: December / 2021

Pages: 106

With the growing importance and sensitivity of information and digital services being provided for different users, the need for better protection schemes has surfaces, according to the sensitivity of traditional secret-based techniques to simple attacks, such as shoulder surfing and guessing. Biometric authentication has emerged as a solution to these limitations, according to the high robustness of biometric features and their high availability. Combined with anti-spoofing methods that the biometric collection sensors are equipped with, biometric authentication has been able to provide better security to the protected systems. Furthermore, multimodal biometric authentication has been widely investigated in recent years to improve the accuracy of biometric authentication, by using multiple biometric templates to recognize a candidate.

Despite the significant improvement of recognition accuracy, multimodal biometric authentication does not contribute towards securing the vulnerabilities of these systems. Several of these attacks occur during the transportation of the information over the network, which encouraged the proposal of methods that deny such attacks. However, existing methods require additional processing to detect these attacks, which increases the complexity of the system. Hence, a new method is proposed in this study to produce a fixed-size descriptor for each candidate based on their face and fingerprint templates. The proposed method also takes into consideration the timestamp in which the template is collected and a system identifier, so that, the descriptor becomes invalid after a predefined period of time

and a descriptor generated at a certain system fails to authenticate of another, even if these descriptors are generated at the same timestamp.

The proposed method uses a Convolutional Neural Network (CNN) to generated a 512-value descriptor, which is used to authenticate the candidate by simply measuring the Euclidean distance to the model descriptor. The embedding of the timestamp and system identifier in the descriptor allows the validation of these parameters in the same measurement, i.e. no additional processing is required. In addition to the high recognition accuracy achieved by the proposed method, with 99.41% and 99.32% accuracies using two different setups, the proposed method has also been able to maintain the privacy of the users, based on the one-way computations in the CNN.

Keywords: Artificial Neural Network, Biometric Authentication, Security, Template Protection.

TABLE OF CONTENTS

Pages

LIST OF TABLES

Pages

LIST OF FIGURES

Pages

LIST OF ABBREVIATIONS

PIN : Personal Identification Number

QoS : Quality of Service

LOG : Laplacian of Gaussian

DOG : Derivative of Gaussian

DWT : Dynamic Wavelet Transform

SIFT : Scale Invariant Feature Transform

SURF : Speeded Up Robust Features

ANN : Artificial Neural Network

ZMM : Zernike Moment

RBF : Radial Basis Function

TanH : Hyperbolic Tangent

ReLU : Rectified Linear Unit

SSE : Sum of Squared Errors

OSL : One-Shot Learning

MTCNN : Multi-Task Convolutional Neural Network

MNS : Non-Maximum Suppression

LBPH : Local Binary Pattern Histogram

SID : System Identifier

1. INTRODUCTION

The wide use of digital services to access and accomplish different types of tasks has imposed the need to store and process personal, sensitive and private information. Restricting access to such information has become of huge interest of many researchers, especially with the rapidly developing methods that are being used to gain unauthorized access to such information. Accordingly, several techniques that rely on predefined secrets, e.g. patterns, Personal Identification Number (PIN) and passwords have become obsolete, based on their sensitivity to simple attacks such as shoulder sniffing[1, 2]. Additionally, the tendency of humans to use easy-to-remember secrets, which, accordingly, are easy-to-predict, has made these methods vulnerable to other types of simple attacks, such as guessing and dictionary attacks [3, 4].

Based on the limitations of these methods and their sensitivity to such simple attacks, significant attention has been brought towards the use of biometric features in the authentication process. Based on the way these biometric features are extracted, these features can be classified into two main categories, physical and behavioral. Physical biometric features are directly extracted from a body part of the user being authenticated, e.g. fingerprint and iris, whereas behavioral features rely on the unique behaviors of humans when executing certain tasks, such as walking. However, according to the high similarity among humans when executing such tasks, distinguishing differences among different users is a more challenging task, compared to the use of physical biometrics. The features in physical biometrics are more robust and easier to access, according to their high availability and the ease of collecting their templates and extracting the required features [5].

Despite the relatively better performance achieved by physical biometrics, biometric authentication systems that rely on these features still suffer from certain vulnerabilities that are common among biometric authentication systems. These vulnerabilities can be exploited by attackers to gain unauthorized access to the protected information and services or block legitimate users from accessing them. In both scenarios, the reliability of the systems and quality of services provided by them can be dramatically reduced when affected by such attacks, which has brought the attention toward securing these vulnerabilities. When secured ad combined with the robustness of physical biometric features, services and information can be secured and the Quality of Service (QoS) is maintained high, by denying any unauthorized access to the system [6].

Attackers exploit the vulnerabilities on different parts of a biometric authentication system, as illustrated in the generic system shown in Figure 1.1. In such a system, the attacks may be executed against the components that are designated for certain tasks or by attacking the information being exchanged between two of these components. Ratha et al. [7] summarize these attacks as follows:

i. **Sensors Attacks:** In this type of attacks, the attackers, i.e. intruders, attempt to provide false information to the sensor in order to produce false positives and gain access to the system. For instance, a fingerprint template of a legitimate user is forged and provided to the system by the attacker. Biometric authentication systems normally use anti-spoofing methods, such as liveness detection, in order to immune such vulnerability.
ii. **Replay Attack:** In this type of attacks, the intruders attempt to capture the information being sent by the sensor to the next stage and replay it whenever an unauthorized access to the system is required. This type of attacks is one of the widely executed attacks, according to the ease of application and high success rate of this attack. Existing methods [8-10] use a separate stage to detect the replay attack or rely on coordination between the sensor and server to extract certain parts of the templates per each scan, which become invalid for next scans.
iii. **False Features Production:** When this type attack is executed, the intruder replaces the features extracted by the features extraction stage by those defined by the attackers, directly of by manipulating the templates that are being used to extract these features. Accordingly, unauthorized access is gained or access of legitimate users is denied. The production of false features affects the decision made by the system, as this decision is based on the matching results that measure the similarity based on the features descriptor provided by the features extraction stage. An example of this type of attacks is the use of trojan horses at the feature extraction stage to manipulate the values outputted by the features extractor before being sent.
iv. **Tampering Features Descriptors:** Instead of manipulating the values of the descriptor at the feature extractor, the attackers in this type of attacks manipulate the information being sent between the features extraction stage and the matcher. Although the execution of this type of attacks can be easier than manipulating these descriptors at the feature extraction stage, recognizing the values that are appropriate for the required goal is a challenging task, as these values are extracted by the feature extraction stage and are difficult to be interpreted by humans.

Hence, attacks normally use similar systems to produce the required descriptors and replace them during the communications.

v. **Modifying Matcher's Score:** Similar to the false features production, the attackers exploit this attack by manipulating the output of the matching stage, either by manipulating its inputs to produce the required output or by intercepting the output and change its values before being sent to the next state. This type of attacks is secured by securing the computer that is being used for this stage, so that, no Trojan horses or unauthorized network access can be gained.

vi. **Tampering Model Templates:** In order to authenticate a user, it is important to store a model template that can be used to match the biometric collected from that user in the future. Accordingly, by replacing the store model template by another generated for a different user, the legitimate user loses access to the intended services and information, whereas the user with the templates that are used in the replication gains the access. In addition to securing the server from unauthorized access, it is important to encrypt the templates stored in the database, so that; new templates for unauthorized users cannot be generated, even if the security of the server is compromised.

vii. **Tampering Communicated Model Templates:** When the database management server is secured and attacker are unable to gain access to the data stored on that server, attackers use this type of attacks to manipulate the templates extracted from the database while being sent to the matching stage. This attack emphasizes the importance of encrypting the templates, so that, they cannot be generated by the attackers. Hence, even if an attacker gains the ability to manipulate the templates during communications, no unauthorized access is gained to the system.

viii. **Decision Manipulation:** Authentication decision are made based on the matching score between the templates collected during authentication and those of the legitimate user stored in the database. In some biometric authentication system, complex techniques are being used to make the appropriate decision, especially when multiple biometric templates are involved in the matching operation. Hence, simplifying the decision-making procedure can reduce the probability of manipulating the decision made by the system.

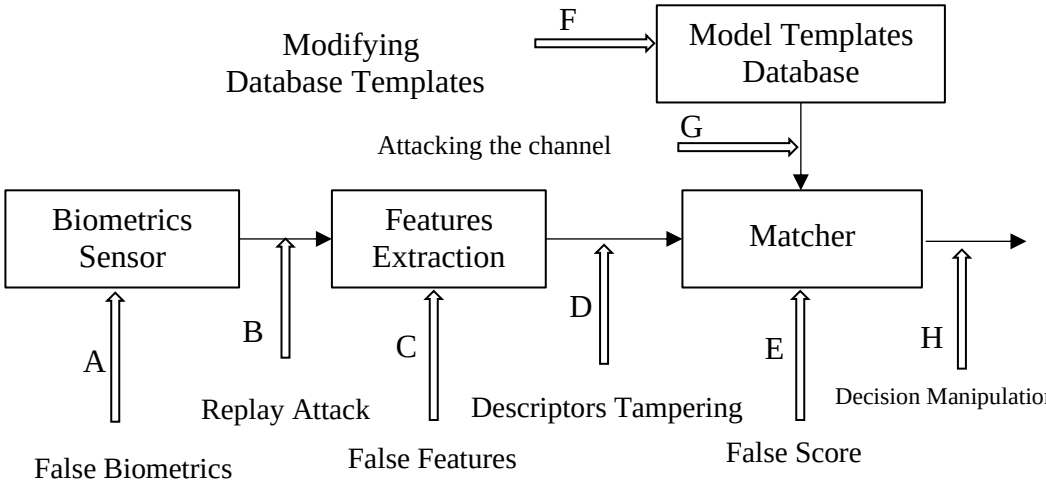

Figure 1.1: A generic biometric authentication system and its vulnerabilities [7].

As the templates collected from the same body part of the same candidate are different every time a template is collected, feature extraction is a challenging task, as the extracted features are required to be robust regardless of the variable parameters that affect the template collection, e.g. different lighting, positioning and alignment conditions. However, the features extracted from these templates are required to be robust and invariant to these conditions, so that, by matching these features, a candidate can be authenticated into the system. If the extracted features lack the required robustness, multiple attempts are required from the candidate in order to authenticate into the system, according to the high probability of false negatives, whereas the lack of uniqueness of these values can produce false positives, which may result in an unauthorized access.

The use of traditional computer vision techniques to extract features from biometric templates relies on generating descriptors based on key points with predefined characteristics, e.g. blobs and edges. Regardless of the significance of these points in the required output the generated descriptors contain information about all the features that include the hand-crafted characteristics. Hence, false positives and negatives can occur frequently in these methods based on the existence of less important features, such as glasses in face recognition, and their absence, compared to the stored template of that user. Several methods are being used to extract such descriptors from the image that contains the biometric credentials, based on the type of the collected biometric. For instance, the patterns in the iris are robust and certain features, such as the pupil, can be used to align the image every time it is collected. Accordingly, traditional feature extraction methods such as Laplacian of Gaussian (LOG) and Derivative of Gaussian (DOG) [11], Dynamic Wavelet Transform (DWT) [12] and Gabor filter [13], are being

used to extract iris descriptors. Additionally, methods that rely on hand-crafted features can also be applied to this such a type of biometric credentials, such as the use of Scale Invariant Feature Transform (SIFT) [14], Speeded Up Robust Features (SURF) [14] and Haar-like features [15].

To avoid the extraction and use of non-decisive features that do not contribute the required decision, several of the recent methods rely on the use of Artificial Neural Networks (ANNs). The features that are detected using these networks are discovered based on the output required from these inputs, so that, more weight is applied to the more important features, whereas features that do not participate into reaching the required output are neglected. Accordingly, the use of the descriptors that are produced using ANNs has shown significantly better performance, by reducing the false positives and negatives, i.e. improve the accuracy.

For further improvement of the authentication accuracy, recent biometric authentication methods rely on using multiple biometrics for the same candidate to be authenticated in the system. However, despite the improvement in the accuracy, the vulnerabilities that are illustrated earlier are still persist, i.e. the use of multimodal biometric authentication improves only the recognition accuracy but does not protect the system from attackers. Additionally, the need to store and communicate multiple biometric templates imposes the need for more network bandwidth and computer storage capacity. Despite the benefits of using digital watermarking techniques to protect the templates and reduce the space and bandwidth requirements [16-22], the need to capture, store and communicate the entire biometric template pose privacy concerns towards using these systems.

1.1. PROBLEM STATEMENT

Biometric authentication has emerged as a solution to secure access to information and services being provided digitally, according to the better performance they provide compared to the use of classic secret-based methods. This improvement is achieved by the use of the robust and unique biometric features that can be extracted from humans, especially physical biometric features. Moreover, to further improve the performance of such systems, recent attention has been brought towards the use of multiple biometric templates during the authentication of candidates, so that, false decisions made based on one of the templates are corrected when the other template is compared. Despite the improvement in the recognition accuracy, the use of multimodal biometric authentication has not been able to handle the vulnerabilities in biometric authentication systems, in general.

Digital watermarking techniques are being used to secure the information being exchanged among the different parts of the biometric authentication system [18, 19, 23]. In addition to the security features, these techniques allow the embedding of one biometric template over the other, which improves the efficiency of bandwidth and storage. However, the storage and communication of the entire biometric templates rises concerns about the privacy of the candidates authenticating into the system, as such information must be hidden even from system administrators. Thus, it is important to reduce the size of data being stored or communicated among the parts of the biometric authentication system and protect the privacy of the candidates by producing a descriptor that can be used for authentication but can never reveal the original templates that are used to create the descriptor.

Despite the existence of multiple feature-level fusion methods [11-15], the descriptors generated by these methods, which contain the features that are extracted from both templates, the ability of using these descriptors repeatedly require additional steps to secure the system against replay attacks. The generation of a descriptor that becomes invalid after a period can reduce the complexity of the system and allows a single decision to be made within a single comparison, between the descriptor received from the candidate templates and the model ones. This feature can also improve the security of the system, as the matching and decision-making stages can be fused together and a decision can be made via a single comparison between the descriptors.

1.2. THE AIM OF THE STUDY

To improve the performance of multimodal biometric authentication systems, a new method is proposed in this study to produce time-sensitive fixed-size multimodal descriptors that contain the features of both fingerprint and face templates. The proposed method uses a convolutional neural network to fuse all the information in a single descriptor, so that, an attacker cannot manipulate a part of the descriptor to manipulate the features of a certain template. Accordingly, the attackers cannot gain access to the protected information even if they gain the ability to compromise the communications among the different parts of the biometric authentication system.

The proposed method uses an ANN to generate the required descriptors, so that, the produced descriptors are efficient and contain only significant features that allow the biometric authentication system to reach correct decisions. Moreover, to further secure the system, the proposed method considers a unique value that is configured by the system administrator, so that, a descriptor generated by a certain system is not suitable for

another. Such descriptors add another layer of protection as another system cannot be used to execute an attack by the intruder. Hence, as the descriptors become invalid after a certain period of time, i.e. fail to match even to the model template of the same user, and as a template of the same user generated on a different system at the same time instance also fails to authenticate, the proposed method can significantly reduce the false positives that can allow intruders access to the protected services and information.

As the generated descriptor contains information about the timestamp, system and both the face and fingerprint templates, and according to the use of artificial neural networks to generate these descriptors, the original biometric templates cannot be retrieved from the generated descriptor. Hence, the privacy of the users of the system is maintained even against the administrators of the systems, as each value generated by the proposed method can vary based on any of the inputs.

1.3. STRUCTURE OF THE THESIS

The remainder of this thesis is structured as follows:

- **i.** Literature related to the topic of the thesis and techniques that are employed in the proposed method is reviewed in Chapter Two.
- **ii.** Detailed description of the proposed method is provided in Chapter Three, in addition to the motivations behind selecting each technique employed at each stage.
- **iii.** The performance of the proposed method is evaluated using a set of experiments that are described alongside their results in Chapter Four.
- **iv.** A summary of the experimental results, discussion and comparison to existing methods are provided in Chapter Five.
- **v.** The conclusions of this study and suggestions for future work are summarized in Chapter Six.

2. LITERATURE REVIEW

2.1. INTRODUCTION

To provide an integrated overview of biometric authentication, its challenges and how they are being handled currently, a review of the recent literature is provided first in this chapter. Then, the techniques that are employed by the proposed method to achieve the intended task are described in details, in addition to the features of each technique and the reasons behind selecting each technique for each task. Finally, state-of-the-art methods that are being used for multimodal biometric authentication, their advantages and limitation are reviewed in order to illustrate the importance of the proposed method.

2.2. AUTHENTICATION

Authentication is the process of distinguishing legitimate users from intruders, by comparing the authentication credentials provided by the user to those stored for the legitimate users. Authentication techniques are categorized into different categories, based on the credentials provided by the user to authenticate into the device, or the service, which are [24]:

a. **Something you know:** it is known as well, secret-based authentication, it usage involve predefined secrets for the legitimate users, such as Personal Identification Number (PIN) or patterns. PINs are numerical secrets predefined by the user, so that if the number provided to the authentication technique matches the one stored for the legitimate user, the user is considered as a legitimate user and authenticated into the device. Passwords are alphanumerical values, which may consist of number, letters or both [25]. The main drawback of secret-based authentication is its vulnerability to simple attacks, such as shoulder surfing, especially that users tend to use easy-to-remember secrets, where such secrets are easy to predict as well [26].

b. **Something you have:** in this method of authentication, the authenticity of the legitimate user is proven by providing physical objects belonging to the legitimate user such as an ATM card, Smartcards or electronic tokens

c. **Something you are:** this authentication technique rely on biometric features of the user, such as the fingerprint and handwriting [27, 28], which can also categorized into two categories:

i. Physiological biometric: Physical biometrics features are extracted from the physical characteristics of a certain part of the human body, such as the iris and

fingerprint. Although physical biometric features have shown higher resistance to simple attacks, which has encouraged many manufacturers to embed sensors that extract these features from the users, many of the users have shown concerns about their privacy while using such authentications techniques. Some of the widely used physical biometric features are the fingerprint, iris and facial recognition, where even knowing the type of features used by the authentication technique does not allow the intruder to access the device, as these features are unique per each person

ii. Behavioral biometric: behavioral biometric features do not rely on physical characteristics of the user; they are extracted by collecting information about the unique way that a user performs a certain task. Although the behavior of one individual is different from the behavior on another, the behavior of a certain individual is not as static as the characteristics of the body of that individual, which are used to extract physiological biometric features. Thus, direct comparisons of the features extracted from the individual's behavior and the template features, extracted for the same individual upon setting up the authentication, are not applicable as they are in physiological features. For this reason, it is important to use machine learning techniques in order to make the decisions whether the features extracted from the authentication attempt are for the same individual that registered the template features during the setup of the authentication method [29, 30].

2.2.1. Multimodal Biometric Authentication

With the existence of false positives and false negatives in biometric authentication, significant attention has been brought toward multimodal biometric authentication, in which the candidate is authenticated based on templates collected for multiple biometrics. Accordingly, an error that occurs according to the features extracted from one template can be corrected by using the other template. However, with additional features, additional challenges are introduced, which in the case of multimodal biometric authentication is the need for additional templates during storage, communications and processing. Additionally, multimodal biometric authentication only handles the problem of false positives and false negatives, but does not propose any solution to the vulnerabilities of a biometric authentication system.

Despite the availability of many biometric templates to be collected from a human body, many of the recent research mostly investigate iris, face, palm and fingerprint templates [31, 32]. The preferably of these biometrics is based on the robust features that these biometrics can provide, in addition to their high availability, i.e. can be easily captured when needed. Moreover, significant attention has been brought toward the use of fingerprint and face templates in the authentication process of a multimodal system, according to the ability of collecting both templates simultaneously, as the face image can be passively collected [16, 20-22].

When multiple templates are used in the authentication process, a fusion is required to produce a single decision based on various templates. The method proposed by [33] fuses the features extracted from the face and fingerprint templates in order to produce a single descriptor that can be used to enroll the candidates. This method incorporates the Zernike Moment (ZM) and Radial Basis Function (RBF) neural network to produce the descriptors, with a maximum length of 72 features, and enroll candidates. Despite the ability of this method to improve the recognition accuracy of the candidates, with 96.55% compared to 73.20% and 92.89% when the face and fingerprint templates are used solely, such accuracy has not been able to outperform the performance of the methods that use the entire templates in the matching stage, instead of the descriptors. The methods that use the entire biometric templates for matching [18, 19, 23] have been able to achieve significantly better accuracy. However, in addition to the privacy concerns, the use of the entire biometric suffers from two main disadvantages, which are the larger size of the data, which requires additional bandwidth and storage to communicate and store the templates, as well as, the ability to tamper with these templates [34].

Existing multimodal biometric authentication systems that use the entire biometric templates [18, 19, 23] use fragile watermarking for tamper detection. These methods use one of the biometric templates as a watermark over the other, so that, the bandwidth required communicating both templates is similar to that required to communicate the cover template. Moreover, according to the use of fragile watermarking, the cover images cannot be compressed, as the watermark information is lost when the image is compressed. Additionally, these methods still pose a privacy risk, as the entire biometric template is being communicated, and are not resistant to replay attacks, in which an intruder replays the messages, sent by the sensors during the enrollment of a legitimate candidate, to gain access to the system [35]. Such an attack is secured in [10] by measuring the similarity using different regions of the biometric template every time a

candidate is authenticating into the system. As these regions are selected by the server, replaying messages from a previous transmission fails to authenticate the intruder. However, as this method uses a single biometric template, i.e. the iris, and relies only on regions of the template, instead of the entire template, this method has shown lower accuracy, compared to the other methods, which use face and fingerprint templates.

Candidates cannot be enrolled by directly matching the pixel values of the images collected for their biometrics, such as the face, iris and fingerprint, as these images are different from each other even for the same subject. Such a difference can occur for a wide range of reasons, such as the positioning of the part that the template is being collected from with respect to the sensor that is collecting the template or to the dynamic nature of these biometrics, e.g. illumination, skin wounds and scratches when collecting fingerprint or the facial hair and accessories when collecting facial images [36-38]. Instead, a descriptor is created based on the distinctive features that exist in the template, so that, matching can be conducted based on the difference between these descriptors [39, 40].

Several methods are being used to extract such descriptors from the image that contains the biometric credentials, based on the type of the collected biometric. For instance, the patterns in the iris are robust and certain features, such as the pupil, can be used to align the image every time it is collected. Accordingly, traditional feature extraction methods such as Laplacian of Gaussian (LOG) and Derivative of Gaussian (DOG) [11], Dynamic Wavelet Transform [12] and Gabor filter [13], are being used to extract iris descriptors. Additionally, methods that rely on hand-crafted features can also be applied to this such a type of biometric credentials, such as the use of Scale Invariant Feature Transform (SIFT) [14], Speeded Up Robust Features (SURF) [14] and Haar-like features [15].

According to the variations that can occur in other types of biometric credentials, such as the fingerprint and face, the use of hand-crafted features for descriptors generation can produce high numbers of false positives, compared to the number of distinctive features. These false positives affect the accuracy of the authentication system, as they increase the matching scores [38, 41]. Hence, more-complex methods are being used to generate such descriptors for this type of biometric credentials. For instance, Gottschlich et al. [42] propose an orientation field detection method to recognize the correct orientation of a fingerprint, regardless of the actual orientation of the finger on the sensor. Alternatively, the minutiae-based method proposed in [43] considers each pair of key points instead of one for descriptor generation, to reduce the possibility of false positives.

In addition to the more efficient use of bandwidth and storage space to communicate and store biometric credentials when using these descriptors, this methodology can also improve the privacy measures of the system. For this purpose, non-invertible feature transformation methods are being used, so that, the original biometric credentials can never be retrieved from the stored descriptor [44, 45]. Instead, the credentials collected at the sensors are passed through the same transformation function, so that, they can be matched with those retrieved from the database. Alternatively, biometric cryptosystems can also be used to protect the privacy of the users by encrypting the biometric templates. The keys that are used to decrypt the biometric templates are either bounded to the template [46, 47] or generated based on the features detected in that template [48-50]. Despite the ability of these methods to protect the privacy of the users, the need for additional computation requires either a more powerful processor or a longer execution time, as the overall complexity of the system is increased.

2.3. ARTIFICIAL NEURAL NETWORKS

The mathematical operations in artificial neural networks are inspired by the human brains, in which an enormous number of nerve cells exchange electrical signals based on information collected from the external world. In average, a human's brain contains 10^{11} biological neurons that are interconnected to each other to create complex networks. As shown in Figure 2.1, the body of the neuron's cell receives adjusted electrical signals through its dendrites. Each dendrite may collect thousands of inputs, which may be collected from other neurons or other parts of the body, such as vision nerves in the eyes and smell sensors in the nose. The influence of the input on the output of the body cell is adjusted by adjusting the conductivity of the electrochemical solution in the synapsis correspondent to that input. However, the type of the influence that an input can cause, i.e., excitatory or inhibitory, is decided by the body cell, where excitatory inputs induce output from the body cell and such an output is prevented by inhibitory inputs. The output of a neuron can finally be provided as input to another neuron or interpreted into a decision in the human body, e.g., a movement [51-53]. Accordingly, the magnitude of the electrical signal, known as membranes, which are short-lived impulses, is controlled by the conductivity of the synapsis and its influence on the output is decided by the cell body.

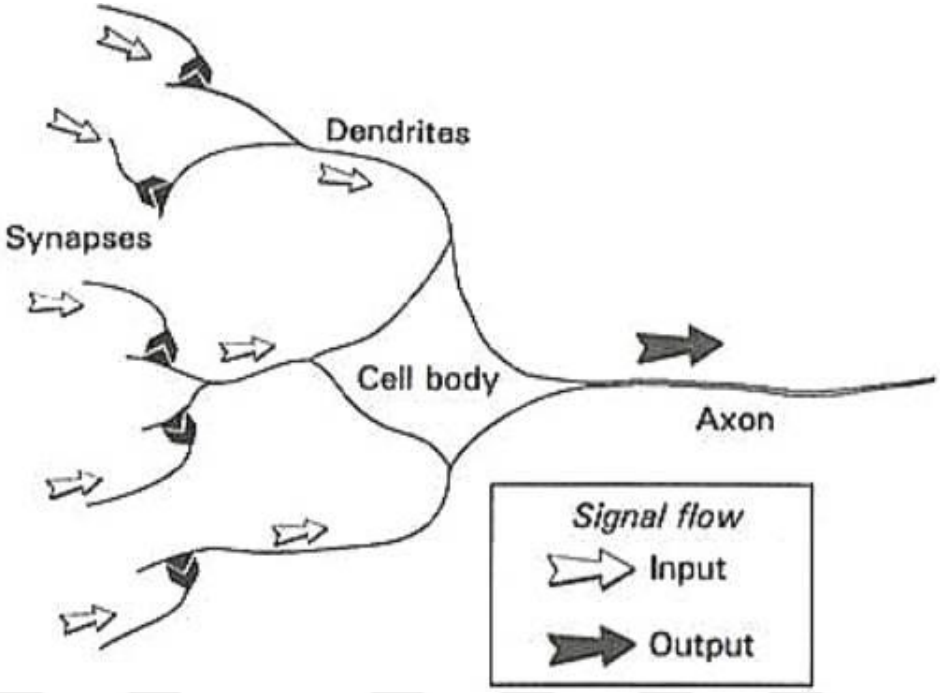

Figure 2.1: A neuron's Essential Components [51].

To mathematically represent these neurons and the networks that they create, the basic component of an artificial neural network, the artificial neuron, also adjusts the influence of each input on its output by using a parameter known as the weight. The value of the weight adjusts the influence of the corresponding input, in terms of how significant is that input, whereas the sign of that weight adjusts the type of the influence, i.e., excitatory or inhibitory. Then, the output of the neuron is calculated by summing the weighted inputs, as shown in Equation 2.1, where *S* is the summation result of the inputs *x*, each weighted by the corresponding weight value, *w*. Additionally, the neuron uses a bias value *b* to improve the flexibility of computations [54]. These computations are also illustrated visually in Figure 2.2, which shows another important component of the neurons, which is the activation function.

$$S_j = \sum_i x_i w_{ij} + b_j \tag{2.1}$$

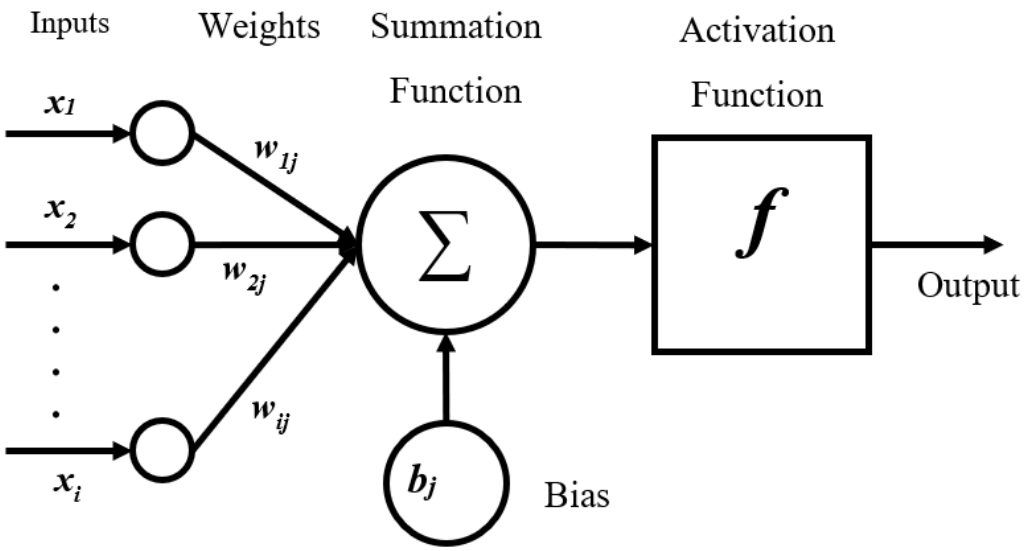

Figure 2.2: Visual Illustration of the Neuron's Computations [54].

According to Equation 2.1, the output of a neuron is linear to its inputs, regardless of the values of the weights and bias. To allow these networks to handle and solve non-linear problems, activation functions are being applied to the output of the neurons, after summing the weighted inputs and the bias value. This non-linearity provides the neuron with better ability to craft the features it can detect, which is a certain pattern in the input values. As shown in Figure 2.3, there are several activation functions that are being used with artificial neural networks, which has been able to improve the performance of these networks. Additionally, limiting the values that flow in, or outputted by, the neural network is a significant feature of activation functions, where for instance, the Hyperbolic Tangent (TanH) function limits the output in the interval [-1, 1], whereas sigmoid limits it in the interval [0, 1]. These functions are widely used in the output layer, based on the range of values required from the neural network [55, 56], whereas the use of the Rectified Linear Unit (ReLU) in hidden layers has shown significant improvement in the performance of the artificial neural networks in different applications [57, 58].

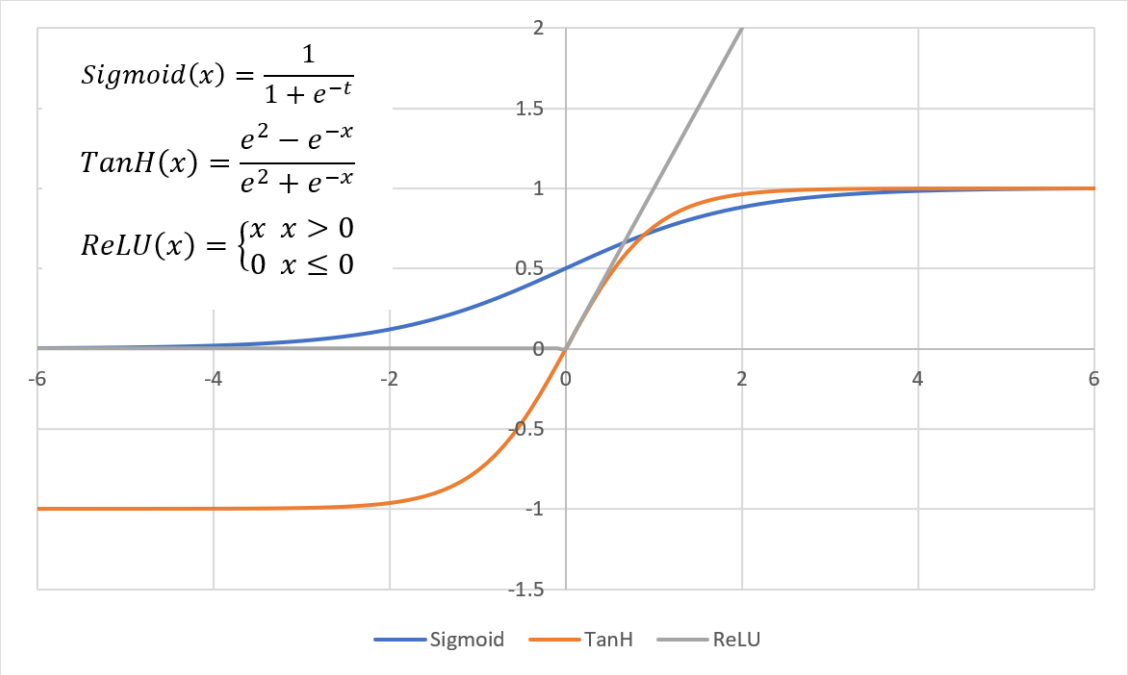

Figure 2.3: Activation Function for Neurons [57, 59].

Despite the ability to produce complex artificial neural networks, the neurons in a standard neural network are distributed in layers, so that, the neurons in a certain layer process the outputs of the ones in the previous layer. This distribution allows the neurons to detect complex features by combining the features detected in the previous layer, which may also be a combination of the features detected in a previous layer of detected directly in the input. In addition to the layers that are used to detect these features, which are known as hidden layers, every artificial neural network has an input layer, which gathers the input from the external environment, and an output layer that presents the computed values to the environment [60]. Figure 2.4 presents a sample feedforward neural network with four inputs, two outputs, which govern the number of neurons in the input and output layers, in addition to two hidden layers. To increase the complexity of the patterns, or features, that the neural network can detect, the number of hidden layers is increased. However, this increment increases the complexity of the required computations, according to the increased number of weights, biases and neurons. Moreover, to increase the number of patterns that the neural network is capable of detecting at a certain level of complexity, the number of neurons at the corresponding layer is increased, which also increases the complexity of the computations. Hence, a balanced topology is required to detect the features that allow the neural network to achieve the required task with minimum possible complexity.

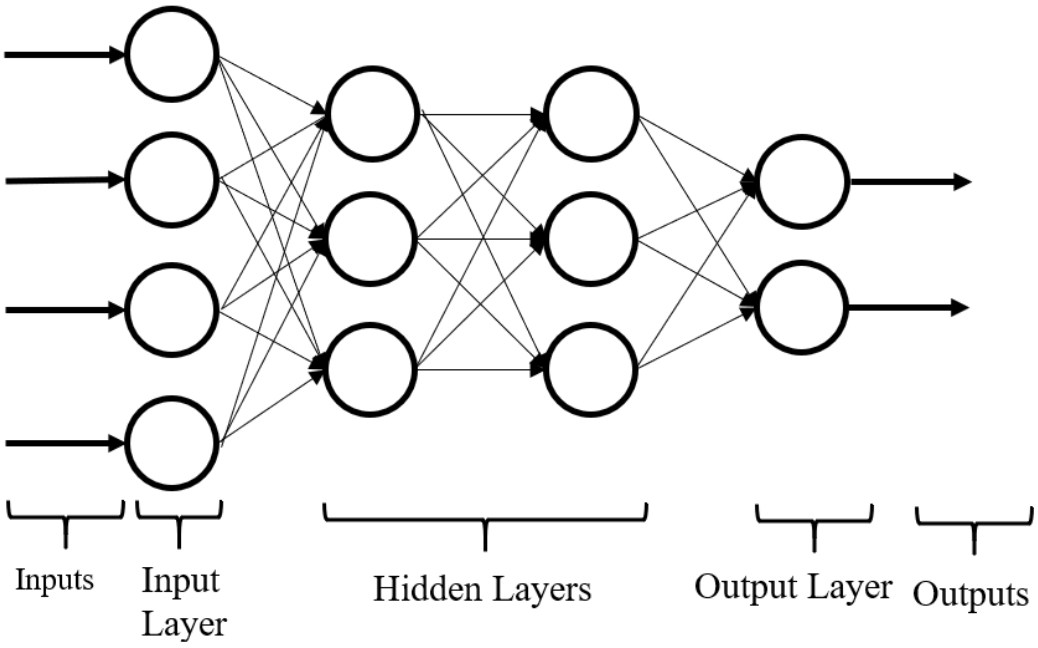

Figure 2.4: Sample Feed-Forward Artificial Neural Network.

According to the computations that are being executed by each neuron and the distribution of the neurons in an artificial neural network, the patterns that each neuron can detect, in order to calculate the output required from the neural network, are governed by the weights and bias value of each neuron. These values are optimized to minimize the difference between the values outputted from the neural network and the values required from the network per each input, which is known as the loss. Any optimization algorithm can be used for this purpose; however, the enormous number of parameters to be optimized in the neural network has imposed the need to use efficient optimization algorithms that has the ability to optimize such a number of parameters shortly. Accordingly, gradient descent optimization algorithms are being used for this purpose, which require a convex loss function, as shown in Figure 2.5. This type of optimization algorithms relies on calculating the partial derivative of each parameter to the loss, which indicates the direction in which changing the value increases the loss. Hence, the optimization algorithm changes the value in the opposite direction, to minimize the loss, which represents the difference between the required and actual values. Accordingly, this type of optimization algorithms can get easily stuck in local minimums, which non-convex loss functions are used [61, 62].

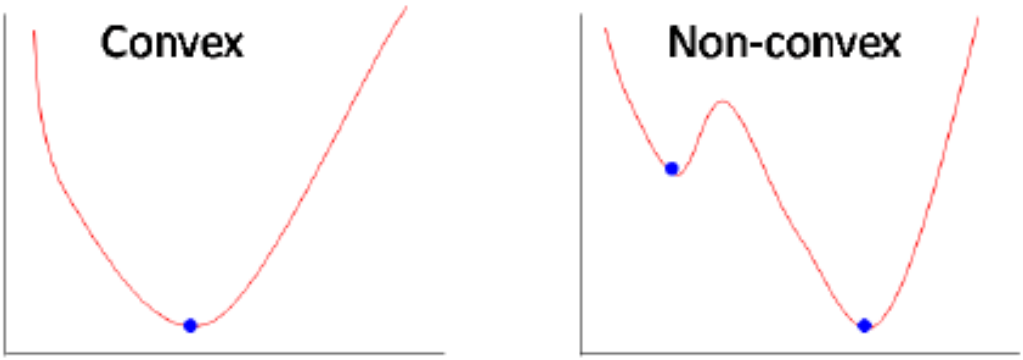

Figure 2.5: Convex versus non-convex function [63].

One of the widely used loss, also known as cost, functions in training neural networks for regression applications is the Sum of Squared Errors (SSE), shown in Equation 2.2. The output value *y* is calculated by passing the input values, which are beyond the control of the neural network, through the neurons in the layers of the neural network. Then, the output values are provided to the loss function, as well as the actual values $\hat{y}$, which are required from the neural network. Accordingly, the neural network has the only option of adjusting the weights and biases of the network, in order to minimize the loss and provide the values required from the neural network [64].

$$L = \sum_i (y_i - \hat{y}_i)^2 \tag{2.2}$$

There are two passes of computations in the neural network, a forward pass to calculate the output of the neural network for a certain input, and back propagation, in which the weights and biases are updated. The use of reverse order to update the weights and biases of the neural network is to eliminate any redundancy in computations, as the partial derivatives are being calculated with respect to the loss, which is at the output of the neural network. Hence, the computations of the partial derivatives for the weights and biases in the neurons of the output layer, when being calculated with respect to the loss, are used to calculate the partial derivatives of the weights and biases of the neurons in the last hidden layer, instead of repeating the same computations. Considering the bias as a weight or an input with a constant value of one, the same results can be collected from the neuron and the bias value can be updated alongside all other weight values [65].

The new weight value $\widehat{w}$ is calculated as shown in Equation 2.3, where *L* is the value of the loss, and *E* is the value of the learning rate. As the value of $\frac{\partial w}{\partial L}$ indicates the gradient ascent, this value is subtracted from the original weight value to reduce the loss, i.e.,

match the required output values. Moreover, the learning rate E is used to avoid huge updates to the weight values, which can cause the optimizer to miss the global minimum of the loss and limits the performance of the neural network. Accordingly, this process is repeated for a predefined number of time, i.e., epochs, or until a certain condition is met, e.g., reaching a certain loss value [66-68].

$$\widehat{w} = w - \frac{\partial w}{\partial L} \times E \times L \tag{2.3}$$

When the input is represented by a multi-dimensional array, processing such inputs requires the detection of local multi-dimensional features. A sample of such inputs is images, which consists of three-dimensional arrays, in which the first two dimensions represent the width and height of the image, whereas the third dimension represents the number of color channels in each pixel [69, 70]. To illustrate the importance of convolutional layers, consider a neuron that is required to recognize the feature shown in Figure 2.6(a), in the images shown in Figure 2.6(b) and 2.6(c). Flattening these inputs, as shown in Figure 2.7, enables a fully-connected layer to process these inputs. However, such a flattening loses the relativity among the input values, which limits the performance of the fully-connected layer in such an application.

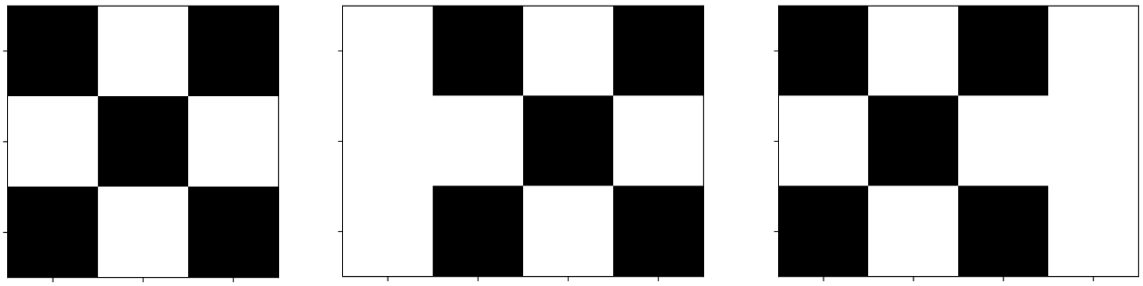

Figure 2.6: Sample multi-dimensional feature and inputs. (a): The required feature; (b) First sample input; (c) Second sample input.

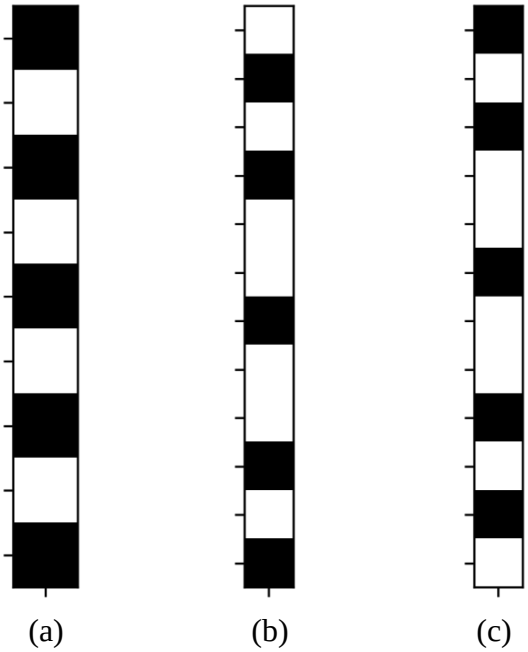

Figure 2.7: Flattened sample feature and inputs. (a) Flattened feature; (b) Flattened first sample input. (c) Flattened second sample input.

Instead of flattening the image, convolutional neural networks use multi-dimensional filters to investigate the existence of the required feature in a limited region of the input, which has the same dimensions of the filter. The values in the filter are collected and processed by the neuron, in order to calculate its output. This output is placed in a multi-dimensional array, which results from convoluting the filter of the input array. Hence, by using a 3×3 filter to detect the feature shown in Figure 2.8(a), in the sample images in Figure 2.8(b) and 2.8(c), the values outputted by the convolutional neuron are as shown in Figure 2.8, considering the filter outputs a value of 1 when 100% match is found in the filter values.

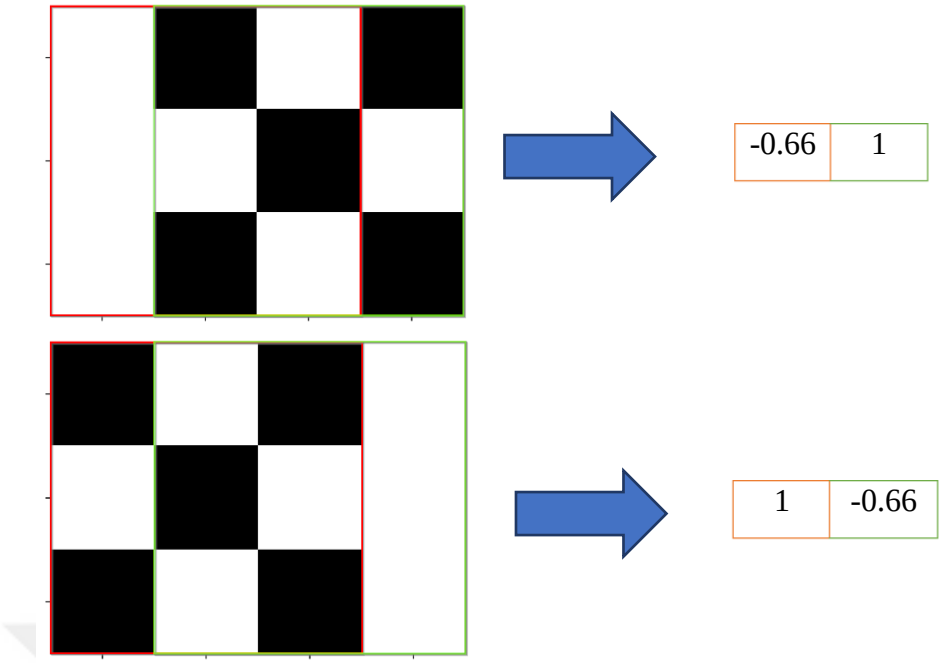

Figure 2.8: Output of a sample convolutional neuron.

The movement of the filter on the input array is governed by a parameter known as the strides, which defines the number of values the filter skips per each convolution. Moreover, as shown in Figure 2.8, the dimensions of the output array are smaller than the input array, which is not a desired behavior in certain applications. Accordingly, when the output of the convolutional layer is required to be identical to the input image, the input array is padded with zeros, based on the required shape of the output array. This approach also allows the detection of partial matches to the feature required by the neuron, e.g., at the corners and edges of the input [70, 71].

Abstraction convolutional layers, also known pooling layers, are also used alongside the convolutional neurons. However, these layers do not have any parameters and their only purpose is to reduce the dimensionality of the data being processed while maintaining the information that are required for the purpose of the neural network. The filters in this type of convolutional layers use abstraction functions, e.g., average or maximum, to summarize all the values in the filter to a single value, as shown in Figure 2.9. These layers also require specifying the size of the filters that they use, the strides that govern the convolution of the filter and the type of the padding that is used [72, 73].

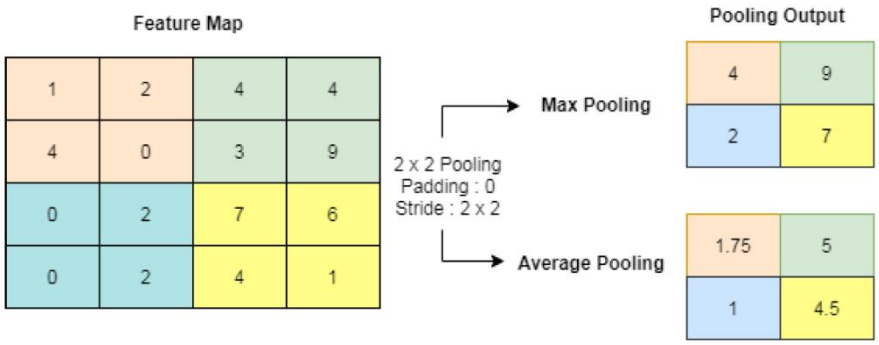

Figure 2.9: Illustration of convolutional pooling layers.

As shown in Figure 2.10, a convolutional neural network may contain several types of layers, e.g., convolutional, abstraction and fully-connected layers. However, in certain applications, fully convolutional networks are used, in which the output is multi-dimensional. In such a type of networks all layers use convolutional neurons, which allow achieving such output. Moreover, Figure 2.10 also shows the importance of abstraction layers to reduce the dimensionality of the arrays, which reduce the complexity of the computations required to process these inputs [74].

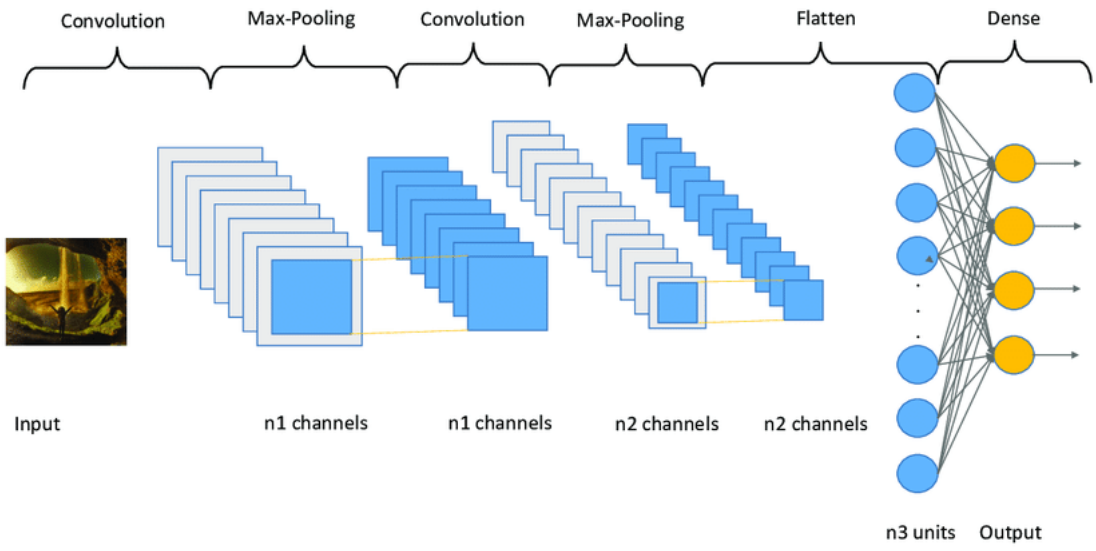

Figure 2.10: A sample convolutional neural network [74].

2.4. ONE-SHOT LEARNING AND SIAMESE NEURAL NETWORKS

In order to train the neural network, the difference between the values outputted from the network and those that are actually required, i.e. the loss, must be calculated. When the neural network is used for a classification task, the number of neurons in the output layer is set according to the number of classes, so that, each neuron outputs the probability of the input to be belong to the corresponding class [75, 76]. Hence, the loss can be

calculated between the values outputted at the neurons and the actual probabilities of the input, which is to be of 1 at the class that it actually belongs to [77, 78].

Despite the outstanding performance of these networks in classifications tasks, according to their ability to distinguish and emphasize distinctive features in the inputs and relate them to their labels, classification approach can limit the application of the neural networks in certain scenarios. Designating a neuron for each required output in complex domains requires significantly more-complex computations, according to the enormous number of neurons that is required in the output layer. Moreover, even if such complexity can be handled, another problem rises up when using neural networks in classification approaches, which is the need to train the neural network whenever a new class is added to the neural network, or one is removed from it. To better illustrate this problem, consider face recognition as a classification problem, in which samples of all the faces that the system is required to know must be collected and used during the training. A new training is required whenever a new face is added to the database, so that, the system can recognize the corresponding candidate [79, 80].

The performance of an ANN improves significantly by increasing the number of samples, i.e. training data that are provided, during the training. This increment allows the neural network to recognize the distinctive features, i.e. inter-class variation, from non-distinctive, i.e. intra-class features. Then, by emphasizing inter-class and neglecting intra-class variations, the neural network can improve its performance and provide accurate predictions [81]. However, in some applications, such as biometric authentication, it is not possible to collect an enormous number of sample templates from each candidate. Instead, the system is required to rely on a few, or may be a single, sample template to enroll the candidate. This type of learning is known as One-Shot Learning (OSL), in which the number of samples provided the method is limited [79, 82].

To overcome this limitation, Siamese neural networks are used during the training in order to allow ANNs to handle OSL problems [79, 83, 84]. As shown in Figure 2.11, two identical networks are used with identical weights and biases, which are updated simultaneously. Instead of outputting the class, i.e. the candidate, that the input belongs to, this approach computes the Euclidean distance between the outputs of the two networks. This distance is passed through a sigmoid function and outputted from the neural network. According to this hierarchy, the values at $G_w(X_1)$ and $G_w(X_2)$ are identical for identical inputs and the output of the neural network is zero.

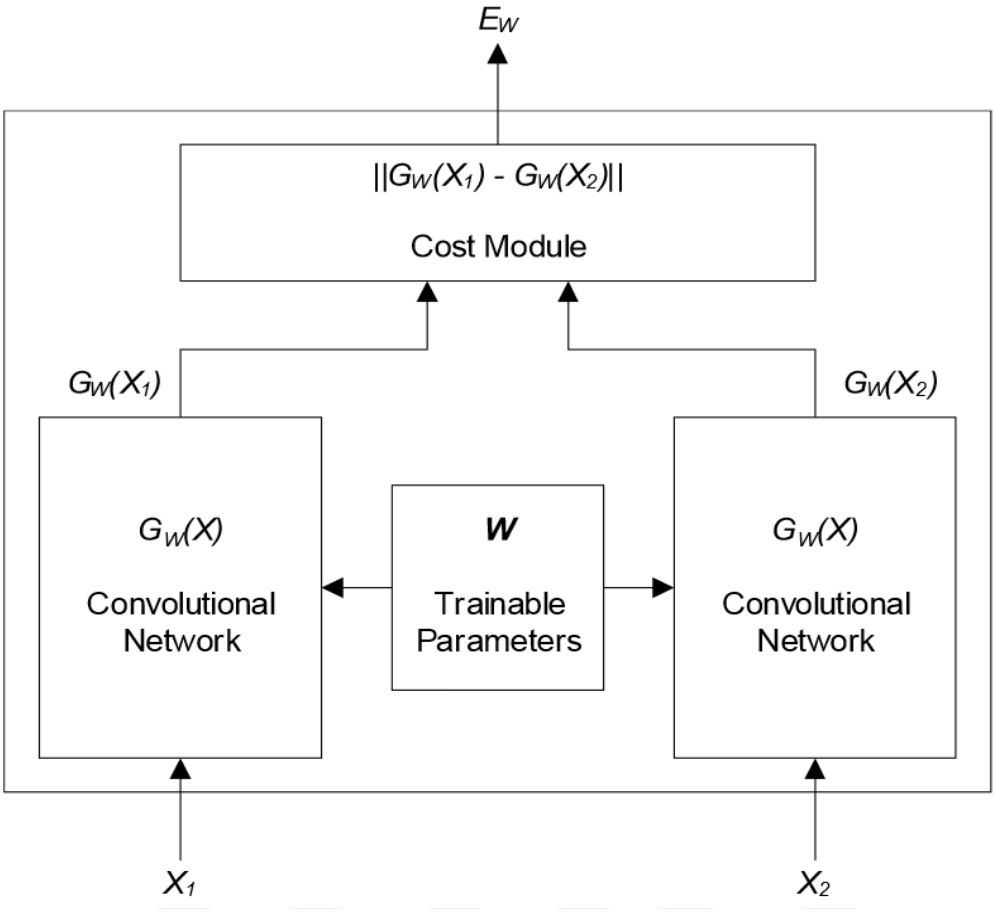

Figure 2.11: Architecture of Siamese neural networks for similarity measurement [83].

By using the triples loss approach, shown in Figure 2.12, Siamese neural networks are trained to neglect intra-class variation and emphasize inter-class ones. This is achieved by providing two different samples for the same candidate and assigns a value of zero to be outputted from the neural network. Accordingly, the neural network is required to provide very similar values at $G_w(X_1)$ and $G_w(X_2)$. Alternatively, a value of one is assigned with the negative samples, i.e. samples from different candidates, which requires the neural network to provide significantly different values at $G_w(X_1)$ and $G_w(X_2)$, so that, a value of one is outputted by the neural network after passing the Euclidean distance between these vectors through the sigmoid function [85].

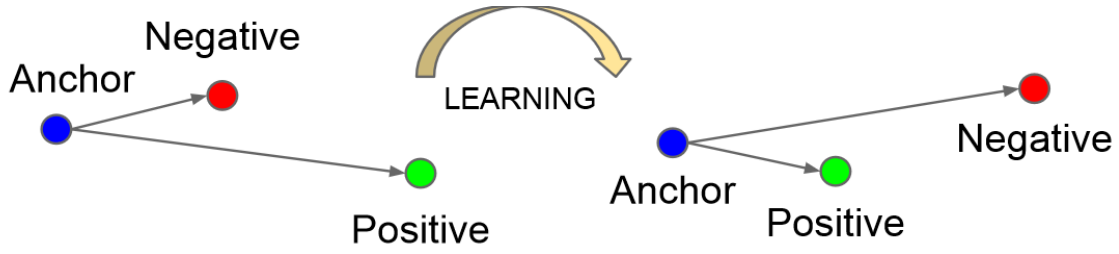

Figure 2.12: Illustration of the distances between the anchor image and the positive and negative samples after training using triplet loss [85].

When the training is complete, one of the neural networks, which are identical, is used to provide descriptor values for each input. Based on the training and the features that the neural network has learned to distinguish distinctive from non-distinctive features, the provided descriptors are expected to be very similar when compared to those collected from positive samples and significantly different from those that are collected from negative samples. Then, by comparing the Euclidean distances between descriptors provided for two inputs, the similarity between these inputs can be measured and a decision whether these inputs are of a positive or a negative sample can be made [86, 87].

2.5. TRANSFER LEARNING

As the training of the neural network relies of the loss between the required values and those outputted by the neural network, which is used to adjust the weights and biases on the neurons, these values are updated from starting from the output towards the input layers. Moreover, as the rate of change of these weights and biases for neurons closer to the input layer is significantly lower than those closer to the output layer, the detection of the low-level features requires more training epochs, which requires huge computational resources, especially when taking into consideration the need to damp the changes by using the learning rate to avoid missing the global minimum loss, i.e., best performance of the neural network [88, 89].

Primitive features, which are normally detected by neurons in the layers closer to the input, are normally similar among different applications. For example, the basic features than a CNN use to recognize a fingerprint or a face in an image, which mainly consists of simple horizontal, vertical and diagonal short lines, are similar. Then, these features are combined in later layers to detect the require object, by accumulating the detected features to recognize more-complex ones. Hence, combined with the slower update to such features according to their closer position to the input layer, providing the neural network with weights and biases that allows it to detect these features can significantly accelerate the training process, as the biases and weights of neurons closer to the output layer are updated faster. This behavior is exploited in transfer learning, in which a pretrained neural network, which is normally trained for a similar, but may not be identical, application is fine-tuned for the required application [90, 91].

2.6. FACE DETECTION

According to the different positioning and distance between the camera that is used to collect the face image and the candidate, the exact position of the face in the captured image cannot be guaranteed. Hence, the captured image must have larger dimensions in

order to ensure that the required full image of the face is captured. Then, any additional regions that do not contain the required face template are removed and only the face image is cropped from the captured image. This approach allows the elimination of any features that may exist in the image and may affect the matching stage [92-94].

One of the face detection methods that is widely used in different applications is based on detecting and recognizing the Haar features, shown in Figure 2.13, that compose the face image and define the region that contains it. Using this approach, the keypoints of the human face, such as the eyes, mouth and nose, are defined using these Haar-like features, so that, the method detects the positioning of these keypoints. Then, the relative positioning of these points is also detected, using the same principle of features, at a different scale level, in order to detect the overall positioning of the face [95, 96]. Despite the good performance of this method when used with frontal-view face images, it has shown limited performance when used with images that contain faces rotates to the left or the right, even with slight rotations [97].

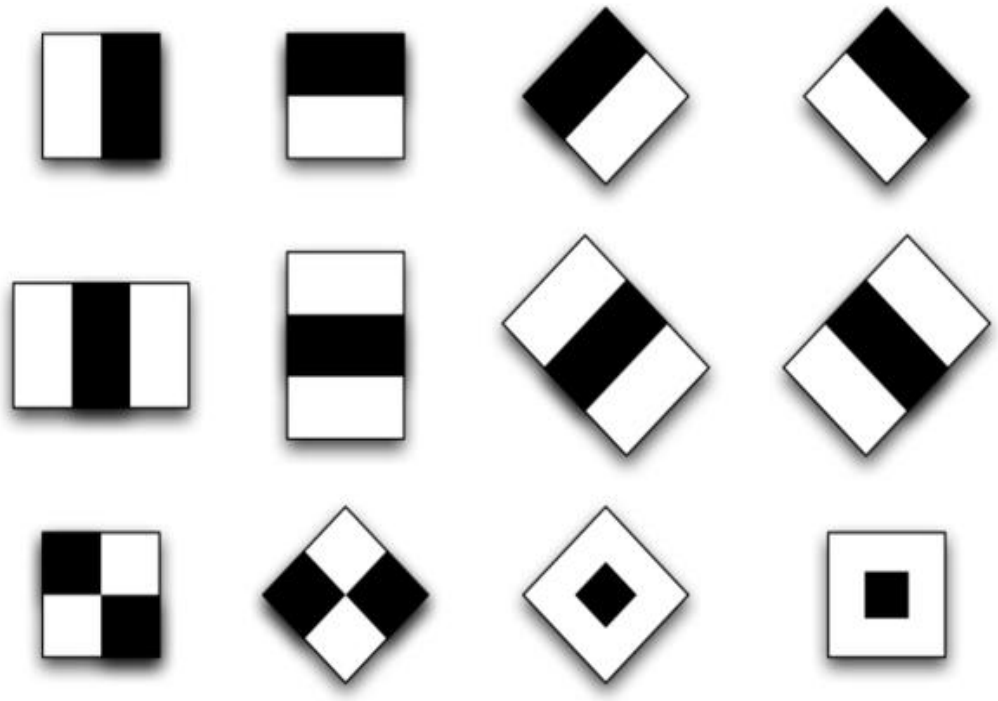

Figure 2.13: Haar-like features [96].

Based on CNNs, a more recent approach is proposed in [98], by using a Multi-Task Convolutional Neural Network (MTCNN). This neural network consists of three sub networks, the Proposal network (P-net), Refine (R-net) and the Output network (O-net), as shown in Figure 2.14. After obtaining the predicted bounding boxes from the P-net, Non-Maximum Suppression (MNS) is used to refine these boxes. Then, these boxes are

provided alongside the image to the R-net, which refines the detected faces based on the proposed boxes. This procedure is also repeated for the bounding boxes outputted by the R-net, by the O-net, and the output is provided by the neural network.

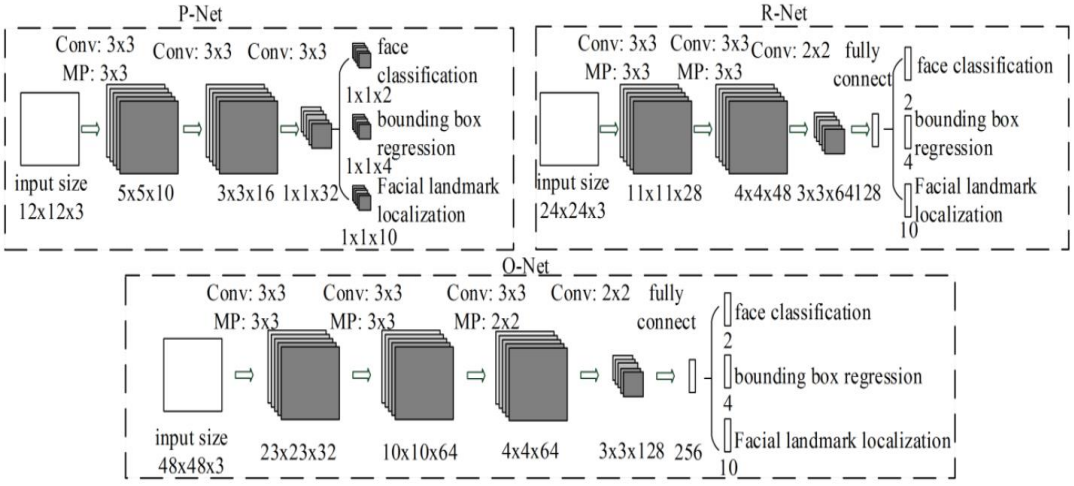

Figure 2.14: Structure of the MTCNN used for face detection [98].

The output of the neural network defines the region of the face by providing the *x* and *y* coordinates of the detected faces, in addition to the width and height of each region that contains the face. Additionally, the MTCNN provides five key points for each region that is predicted to contain a face, as shown in Figure 2.15. According to the ability of ANNs to handle complex decisions, based on complex relations among the detected features, this method has shown significantly better performance, compared to other types of face detection methods, which has encouraged the use of this approach in many face detection applications [99].

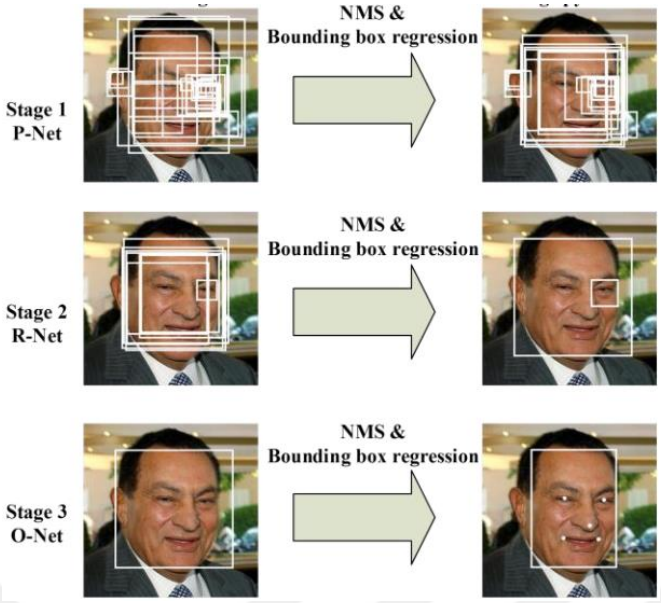

Figure 2.15: The outputs of the face detection MTCNN's networks [98].

2.7. FACE RECOGNITION

Many of the recent real-life applications, such as entertainment, law enforcement, information security and criminal investigation, have started to require the recognition of humans based on the visual features in their faces, similar to the human approach of recognizing people. These methods are required to provide accurate decision despite the changing environmental conditions, such as lighting, and the changing nature of human faces, such as the use of accessories and differences in hair styles. Accordingly, for efficient face recognition, it is important to produce descriptors that are irrelevant to these conditions, so that, the produced values are always similar for the same individual [100-102].

In early stages of human recognition based on their facial features, the distances between the key points are measured and used to produce a descriptor, manually, even before the employment of computers in such a task [103]. This method relies on generating a 16-value descriptor for each face, which is then used to match it with any other descriptor by simply measuring the Euclidean distance between the descriptors. These descriptors are considered for the same individual if the calculated distance is found less than a threshold value, otherwise, the templates are considered from different individuals, i.e. do not match.

The method proposed by Wiskott et al. [104] uses Gabor wavelet transform to generate a labeled graph for each face image, based on the key parts of the face image in the template.

Compared to other methods from the same era, e.g. [105], this method has shown significantly better performance by producing very accurate descriptors. However, with the rapid developing in computers and the use of computer vision techniques, the need for manual annotation for each template, i.e. positioning of the key parts, has become a huge limitation to this method, according to the ability of modern computer-vision techniques to automatically locate these parts.

Several computer-vision-based techniques have been proposed in the literature for face recognition. Different researchers have evaluated different computer vision methods for face recognition, such as Local Binary Pattern Histogram (LBPH) [106], Speeded-Up Robust Feature (SURF) and Scale-Invariant Feature Transform (SIFT) [107-110]. However, the use of hand-crafted features imposes the production of several false features that are irrelevant to the recognition task. On the other hand, the methods that are proposed based on neural networks have achieved significantly better performance, with accuracy of 99.13% by [111], using a convolutional neural network.

Schroff et al. [85] propose and train Siamese neural network that is used for face recognition, denoted as FaceNet. The structure of the neural network that is used in this method, shown in Figure 2.16, illustrates that each face image is assigned with a descriptor that contains 128 values. Although the values in these descriptors do not reflect any interpretable information, the values provided for a candidate are expected to be similar, regardless of the variable conditions that the captured image can contain. This behavior is ensured by including a huge number of training images that include all possible parameters and their variations, so that, the neural network learns and emphasizes the distinctive features in a face template. In addition to outstanding performance of FaceNet in face recognition, compared to other neural networks and techniques, this neural network has shown the ability to achieve remarkable performance when used with fingerprint recognition, i.e. using transfer learning, in several researches [112, 113]. Based on the size of the input template, several versions of FaceNet exist in the literature, which can be used based on the size of the extracted face template [113].

Figure 2.16: Structure of the FaceNet [85].

type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

2.8. IMAGE PREPROCESSING

2.1.1 Histogram Equalization

Real life images are taken under different illumination conditions, which may affect the distinguish ability of the patterns and features in the image. Thus, it is important to adjust the intensity of each pixel in the image. In other words, instead of having an image with pixels' intensity concentrated in a certain range of values, a new image is created with intensities vary from the minimum to the maximum possible intensity values. To achieve this task, histogram equalization is used [114-116].

The histogram equalization method computes the frequency of each intensity value in the existing image and creates a cumulative probability table. The probability of an intensity value is equal to the frequency of that value in the image, to the total number of intensity values in that image. As the created table is a cumulative table of probabilities, the last value in that table is always equal to one. Thus, by multiplying the computes probabilities in the table with the maximum possible intensity value, depending on the size of data reserved per pixel, new intensity values are computed. The old intensity values are replaced with the new ones to equalize the intensity histogram of the image [117]. Histogram equalization algorithm is shown in Figure 2.17.

Input: Grayscale image.
Output: Equalized histogram grayscale image.

Begin
Step 1: Read the input grayscale image.
Step 2: Calculate the frequency of each possible value appears in the image pixels.
Step 3: Calculate the probability of unique value.
Step 4: Calculate the cumulative probability for these values.
Step 5: Multiply the maximum possible value by the cumulative probabilities.
Step 6: Replace each original value with corresponding value calculated in Step 5.
Step 7: Return the equalized image.
End.

Figure 2.16: Histogram Equalization Algorithm.

2.1.2 Resizing Images

When the dimensions of an image are changed, it is important to maintain the information of the image as similar to the original image as possible. One of the widely used image resizing techniques is the linear interpolation [118, 119]. Linear interpolation created a new image with the required dimensions, then, the value of each pixel in the new image is mapped from the original image. If a pixel is mapped from the new image to an exact pixel from the original image, then the value of that pixel is cloned as it. However, if a pixel is not mapped on an exact pixel in the original image, the nearest pixel to the position that the pixel is mapped in are used to compute the new value. The new value v is computed by creating a line between the two adjacent values of v_1 and v_2 located at (x_1, y_1) and (x_2, y_2) respectively, and then select the pixel value based on its position (x, y) between these pixels, using Equation 2.4 and Equation 2.5, where Equation 2.4 is used when the nearest pixels are in the same row of the mapped pixel and Equation 2.5 is used when the nearest pixels are in the same column with the mapped pixels. Because the mapping is done from the destination image to the original image, the computations in this process are known as destination to source computations. Image resizing algorithm is shown in Figure 2.18.

$$v = \frac{(v_2 - v_1) * (x - x_1)}{(x_2 - x_1)} + v_1 \tag{2.4}$$

$$v = \frac{(v_2 - v_1) * (y - y_1)}{(y_2 - y_1)} + v_1 \tag{2.5}$$

where v is the new value calculated for the resized image, while v_1 *and* v_2 are the values in the pixels nearest to the mapped position in the original image.

Input: Image and required dimensions.
Output: resized image.

Begin
Step 1: Read the original image and required dimensions.
Step 2: Create a blank image with the required dimensions.
Step 3: For each pixel in the blank image
    Map the pixel in the original image.
    If the mapped position falls on a pixel in the original image.
        Clone the pixel value from original image
    Else
        Calculate a new value using Equations 2.4 and 2.5.
Step 4: Return resized image
End

Figure 2.17: Image Resizing Algorithm.

3. METHODOLOGY

3.1. OVERVIEW

To satisfy the aim of this thesis, a new multimodal biometric authentication system is proposed. The proposed system produces fixed-length descriptors, one for each set of inputs. This descriptor is then used for decision making in a single distance measurement action. However, in order to ensure that the proposed method is applicable, it is important to satisfy the following requirements:

i. **Time-Sensitivity:** The values in the descriptors generated for the same input templates must be different when generated after a certain interval, so that, an intruder fails to authenticate into the system by replaying the descriptor of a legitimate candidate. This feature is achieved by including the timestamp when the descriptor is generated.

ii. **Uniqueness:** The values in the descriptors generated for the same templates are different per each system, so that, a descriptor cannot be generated even if the intruder gains access to the neural network implemented for this purpose. This feature is achieved by including a configurable identification number that is set by the administrator of the system.

iii. **Fusion:** All the information collected from the features detected in the biometric templates, timestamp and system identifier are fused into a single descriptor, so that the candidate can be enrolled in a single operation based on the values in these descriptors.

iv. **Privacy:** The values in the descriptor cannot be used to retrieve the original biometric templates of the user. Hence, even the administrator of the system cannot regenerate these templates.

According to these requirements, the inputs to the neural network must include the face image, fingerprint, timestamp and system identifier. The face templates are cropped from the image collected from the camera based on the region recognized by the MTCNN and resized to 160×160 pixels, to match the requirement of the FaceNet model [120, 121] that is employed in the proposed neural network. Similarly, the fingerprint template is also resized to the same dimensions before inputted to the proposed neural network. According to the difficulty of recognizing slight differences in linear values by artificial neural networks, the system identifier and timestamp are provided to the proposed method in binary format. The size of the System Identifier (SID) is set to 32 bits, which are set by the administrator, while the length of the timestamp is set to 47 bits. The bits in the

timestamp are distributed as shown in Table 1, which shows that the proposed method can achieve a time resolution up to one millisecond. However, depending on the communication speed among the different components of the system, the timestamp can be provided in lower resolutions, so that, the required sensitivity can be achieved.

Table 3.1: Structure of the timestamp provided to the proposed neural network.

Value	Size (bits)	Range	Remark
Year	11	0-2047	Timestamps provided by computers start from the year 1970, i.e. the actual range of the years is 1970-4017.
Month	4	0-15	
Day	5	0-31	
Hour	5	0-31	
Minute	6	0-63	
Second	6	0-63	
Milliseconds	10	0-1023	

3.2. NEURAL NETWORK STRUCTURE AND TRAINING

To satisfy the requirements defined in the previous section, the proposed neural network is required to accept the face and fingerprint templates, as well as, the timestamp and SID at both the sensors and database sides. Accordingly, two Inception-ResNet models are required to process the biometric templates; each model processes two templates of the same type. Thus, the inception_resnet_v1 model shown in Figure 3.1 processes the face template collected from the sensor (sface) and the face template from the database (dface). Similarly, the FPinception_resnet_v1 model processes the sensor (sfing) and database (dfing) fingerprint templates. Then, the outputs of the models for each side of the system, i.e. the sensors and database, are concatenated together, in addition to the SID value, in a single vector with 288 values in the layers ls288 and ld288, for the sensors and database sides sequentially. The concatenated vector is then passed through a dense layer, which outputs a vector of 512 values that are then passed through another dense layer to produce 384-value vectors. The inclusion of the SID at this level and the use of the dense layers is to scramble and combine the significant features detected in the biometric templates, according to the SID selected for the system. Accordingly, the values in the descriptors outputted by the layer ld384 are different when different SIDs are used to generate them,

so that, a descriptor stored at a certain system cannot be used to enroll a candidate in another system.

The outputs of the layers ls348 and ld348, for the sensors and database sides, are then concatenated with the stimes and dtimes timestamps, for the correspondent sensors and database sides. The output of each of these layers is then passed through a dense layer, to also scramble and combine the values of the timestamp in the descriptor, at the lse and lde for the sensors and database sides, respectively. Finally, the Euclidean distance is calculated between the outputs of the lse and lde layers at the layer lambda, which is then passed through the Hyperbolic Sigmoid activation function, to limit the output of the neural network in the range [0, 1].

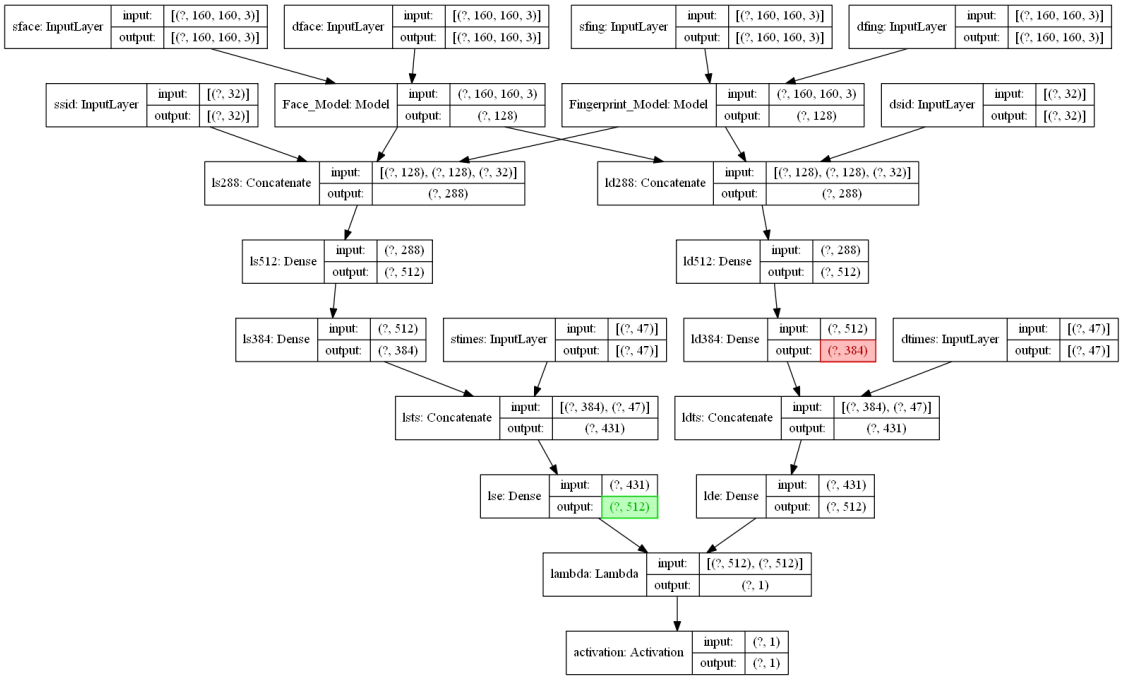

Figure 3.1: The implemented and trained neural network to produce the required

The purpose of each layer or model shown in Figure 3.1 is summarized in Table 3.2.

Table 3.2: Summary of the layers and models in the proposed neural network.

Layers and Models	Purpose

sface & dface	Collect the face templates from the sensor and database, respectively.
sfing & dfing	Collect the fingerprint templates from the sensor and database, respectively.
ssid & dsid	Collect the SID at the sensor and database sides, respectively.
Face_Model & Fingerprint_Model	Pretrained FaceNet models, using resnet neural network, to extract 128-value descriptors for the face and fingerprint templates, respectively.
ls288 & ls288	Concatenate the SIDs with the descriptors generated by the FaceNets.
ls512 & ld512	Dense layers that extract 512 features from ls288 and ld288 to merge the SIDs with the FaceNet descriptors.
ls384 & ld384	Compress the 512-value outputs of ls512 and ld512 to further embed the SID into the descriptors.
stimes & dtimes	Collect the timestamp for the sensor and database sides, respectively.
lsts & ldts	Concatenate the timestamps with the outputs of the ls384 and ld384, respectively.
lse & lde	Generate 512-value descriptors that merge the timestamps into the corresponding descriptors.
Lambda	Measures the Euclidean distances between the outputs of lse and lde.
Activation	Passes the output of the lambda layer into the TanH activation function.

The use of the Euclidean distance and TanH activation function ensures that the output of the output of the neural network is zero when the distance between the vectors generated at layers lse and lde is zero, whereas the maximum value outputted from the neural network is one, regardless of how huge the Euclidean distance, as shown in the output of the TanH function in Figure 3.2. Hence, the neural network can produce significantly

different values in the vectors being fed to the lambda layer, i.e. the layer that calculates the Euclidean distance, by simply using the value 1 during training. Moreover, training the neural network with the value zero as the label forces it to produce identical values in the vectors inputted to the lambda layer. Thus, as shown in Table 3, the inputs that are required to produce identical vectors are labeled with zeros, whereas those required to produce different values are labeled with ones. As there are seven scenarios in which the descriptors are required to be different, which are when the face or fingerprint templates are different, significant or huge difference between the timestamps, difference in one or more bits in the SIDs, seven inputs that require identical descriptors are added for balanced training.

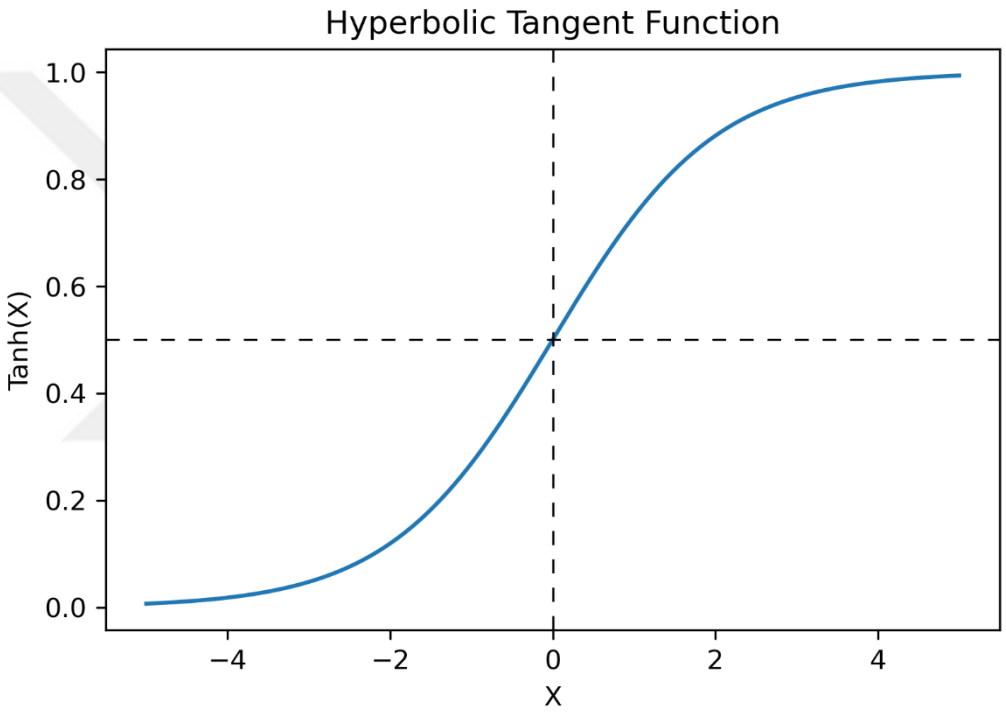

Figure 3.2: Output of TanH activation function.

Table 3.3 Characteristics of the instances in the training dataset and their labels.

#	Face templates	Fingerprint Templates	Timestamp	System Identifier	Label
1	Same subject	Same subject	Identical	Identical	0
2	Same subject	Same subject	Different in one or both of the two Least Significant Bits (LSBs)	Identical	0
3	Same subject	Same subject	Identical	Identical	0
4	Same subject	Same subject	Different in on or both of the two LSBs	Identical	0
5	Same subject	Same subject	Identical	Identical	0
6	Same subject	Same subject	Different in on or both of the two LSBs	Identical	0
7	Same subject	Same subject	Identical	Identical	0
8	Same subject	Different subject	Identical	Identical	1
9	Different subject	Same subject	Identical	Identical	1
10	Different subject	Different subject	Identical	Identical	1
11	Same subject	Same subject	Different in only one bit, other than the two LSBs.	Identical	1

12	Same subject	Same subject	Different in multiple bits, other than the two LSBs.	Identical	1
13	Same subject	Same subject	Identical	Different in a single bit.	1
14	Same subject	Same subject	Identical	Different in multiple random numbers of bits.	1

3.2.1. Deployment of the Trained Neural Network

After completing the training of the neural network, it is decomposed into three models, one for the sensors, and two for the database sides. The Face_Model and Fingerprint_Model, which are Inception_ResNet models, in the neural networks that are used in the sensors and database sides are identical, i.e. the exact same model is used twice. As the neural network that is deployed in the sensors' side is required to generate descriptors for the templates collected from the sensors and to be sent to the server for instant enrollment, the timestamp is instantly inputted to the neural network. Thus, the required 512-value descriptor is instantly generated and sent to the server for authentication. Hence, the part shown in Figure 3.3 is extracted from the trained neural network and used at the sensors' side to generate a 512-value descriptor, marked in red, per each set of inputs, marked in green.

Unlike the sensors' side, the descriptors generated for a subject at the database can be used at any time instance. Hence, it is not possible to store the timestamped descriptor, as the usage of such a descriptor at a different time instance fails the enrolment of the candidate. Alternatively, two parts are extracted from the trained neural network, one generates the descriptors to be stored in the database and the other generates timestamped descriptors before sending them to the matching, i.e. authentication, stage. As shown in Figure 3.4, the first part generates a 348-value vector for the biometric templates of the subjects, using the provided SSID. Then, the remaining part of the neural network at the database side is applied to these vectors upon the request of the descriptors, prior to sending them to the matching stage.

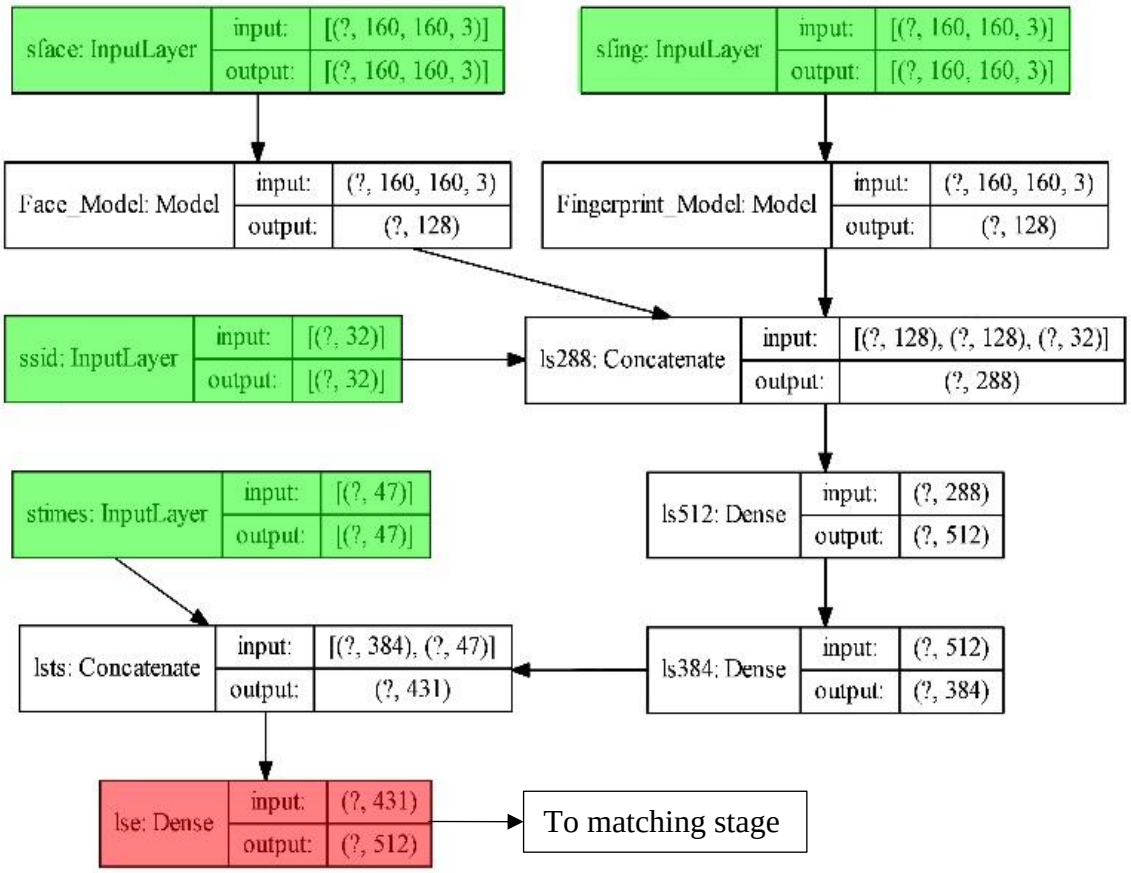

Figure 3.3: The neural network that is deployed at the sensor side

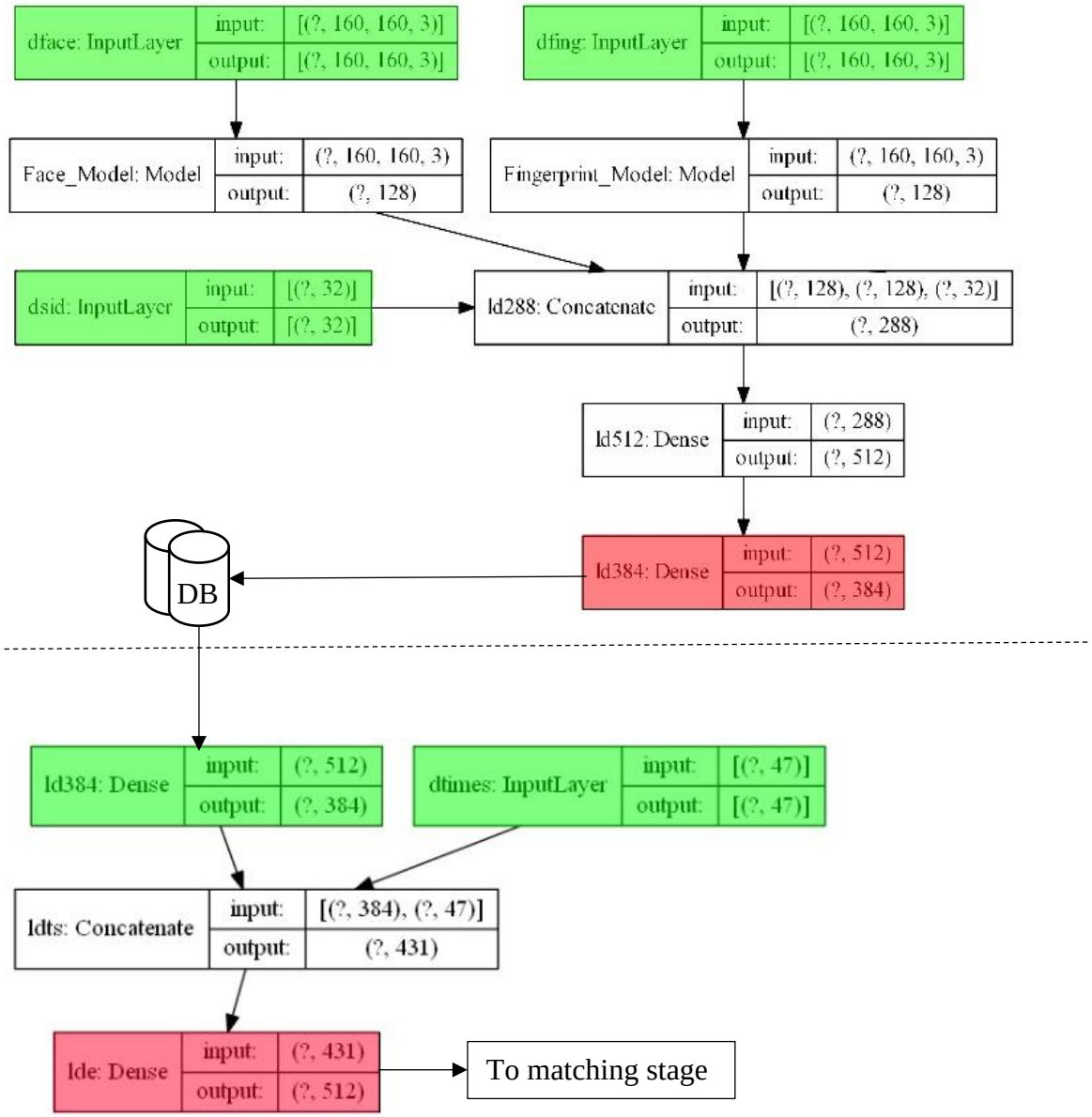

Figure 3.4: Database side neural networks. Top: generating database vectors; Bottom: Generating subjects' descriptors from database vectors.

4. EXPERIMENTAL RESULTS

4.1. EXPERIMENTAL SETUP AND NEURAL NETWORK TRAINING

The proposed neural network is trained using the FERET faces dataset [122], which contains 3528 face images that are collected from 269 subjects. Each image has a size of 512×768 pixels, which are cropped based on the predictions of the MTCNN and the cropped image is resized to 160×160 pixels to match the required input size. Per each subject, five face images are randomly selected and paired with the five fingerprint images from the FVC2000 DB3 setup B fingerprints dataset [123]. This fingerprint dataset contains 400 samples collected from 80 subjects, five samples per each subject. Two different setups are used for the evaluation, using datasets that are not used in the training. The first setup (A) uses the Indian [124] and FEI [125] Faces Datasets and the FVC2002 DB3 set-B [123] and FVC2004 DB4 set-B [126] fingerprint datasets. Similar to the benchmarks used by [18, 19], face images of 110 subjects from the FEI faces dataset are merged with the face images of 50 subjects from the Indian faces dataset and paired with the 160 total subjects in the fingerprint dataset. The second setup (B) uses face images of 110 subjects from the FRGC v2.0 faces dataset [127] paired with the fingerprint images from the FVC2002 dataset, similar to the setup used by [23]. Table 4.1 summarizes the datasets used for each of the training and evaluation setups.

Table 4.1: Training and evaluation datasets description.

Setup	Face Images Dataset	Fingerprint Images Dataset	Number of Subjects.
Training	FERET	FVC2000 DB3	80
Evaluation A	Indian FEI	FVC2002 DB3 set-B FVC2004 DB4 set-B	160
Evaluation B	FRGC v2.0	FVC2002	110

The neural network is trained for 100 epochs, which has been able to achieve 99.48% accuracy on the training dataset, as shown in Figure 4.1, which shows the loss and accuracy of the neural network versus the training epochs. Moreover, Figure 4.1 also shows that the training of the proposed method does not start from extremely low accuracy, i.e. high loss, similar to when random weights and biases are used to initialize the neural network. This behavior illustrates the importance of transfer learning, as such a huge neural network, shown in Appendix A, has been able to achieve 99.48% training accuracy using only 100 training epochs.

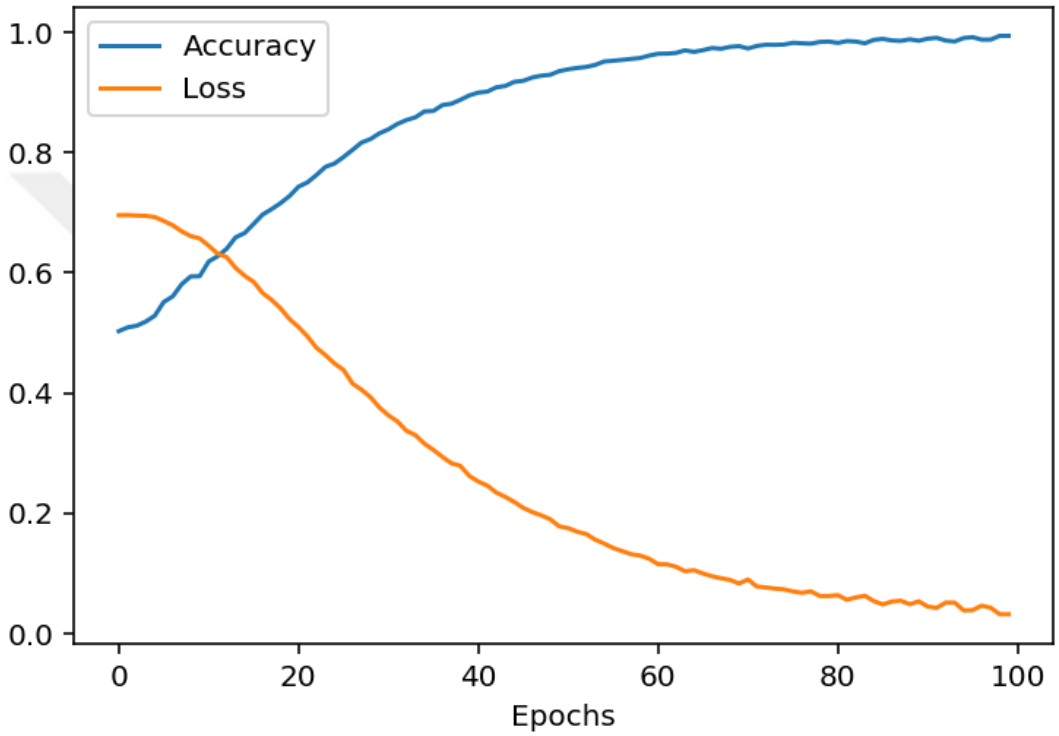

Figure 4.1: Accuracy and loss values versus training epochs.

4.2. AUTHENTICATION ACCURACY

Initially, the performance of the proposed method is evaluated by measuring the accuracy of the enrollment decisions that are made based on the descriptors outputted from the trained neural network in optimal operation conditions, i.e. no attacks are executed. Hence, only one or both of the LSBs in the timestamps are allowed to be different when inputted to the neural network, while the SID is maintained identical throughout the experiment. As shown in Figure 4.2, the proposed method has been able to achieve an accuracy of 99.41% at a threshold of 7.72, which has achieved the Equal Error Rate (EER) of 0.59%. Moreover, Fig 4.2 also shows that the proposed method has achieved EER of 0.68%, i.e. 99.32% accuracy, when the threshold value is set to 8.82.

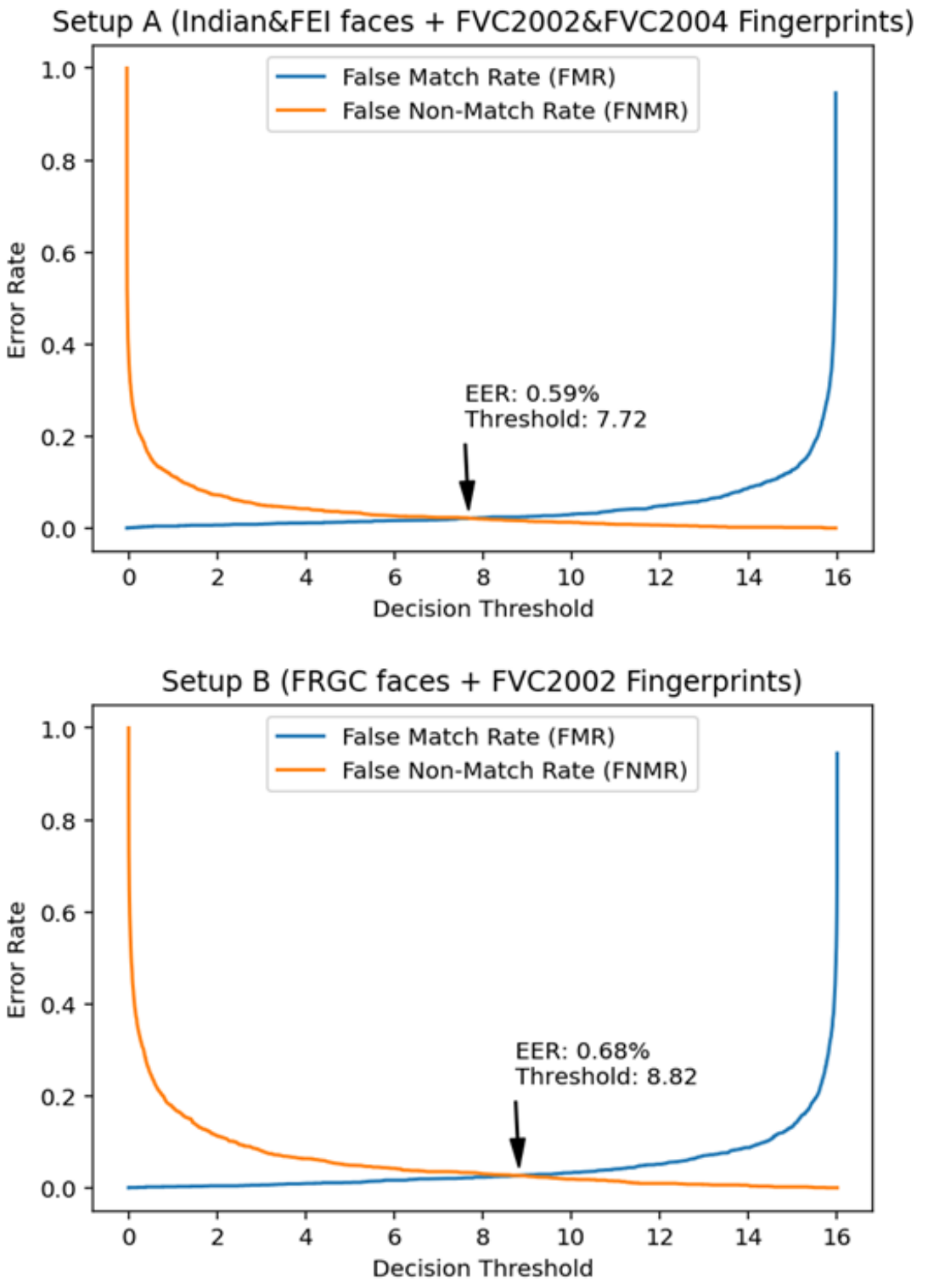

Figure 4.2: FMR and FNMR versus threshold value of the proposed method for the evaluation setups. Top: using the Indian and FEI faces dataset with the FVC2002 DB3 set-B and FVC2004 DB3 set-B fingerprints; Bottom: using the FRGC faces dataset with the FVC2002 finge

4.3. SECURITY AND PRIVACY ANALYSIS

To evaluate the influence of the timestamps on the produced descriptors, biometric templates from the same candidates are used with different timestamps. The value of each bit is changed and the Euclidean distances between the produced descriptors are

measured, as shown in Figure 4.3. The results show a significant difference when the two LSBs are changed with respect to changing other bits. Such sensitivity is a result of the efficient representation of the timestamp in binary format, instead of continuous decimal values. Moreover, the two LSBs allow the consideration of delay in the delivery of the data collected by the sensors, so that, the proposed method can detect a minimum of 3 milliseconds. For systems that require a longer time to deliver the data, it is possible to manipulate the representation of the timestamp according to the required resolution. Thus, the proposed method is resistant to replay attacks and requires no additional processing, as the timestamps are embedded in the same descriptors that are used to authenticate the candidate.

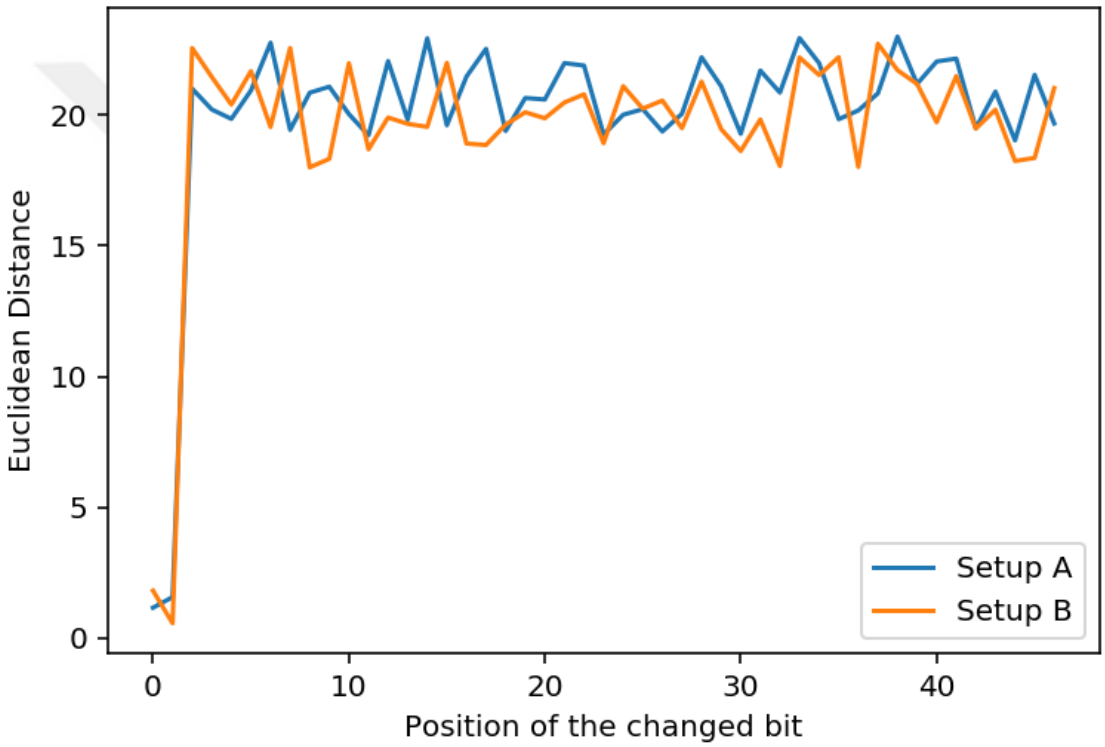

Figure 4.3: The influence of each bit in the timestamp over the distance between the produced descriptors.

Another parameter that is included in the proposed method is the SID, which aims to ensure that a descriptor generated by a certain system is different from another generated using a different system, even if both systems use the proposed method. These differences improve the security of the proposed method and the privacy of the candidate being enrolled, according to the high robustness of the features extracted by the neural network. Using this parameter, the enrolling candidate cannot be recognized by sniffing the data being transmitted by the sensors, even if they are compared to a descriptor generated for the same candidate at the same timestamp. As shown in Figure 4.4, changing any of the bits of the SID during the generation of the descriptor produces a different descriptor,

illustrated by the larger Euclidean distance. Unlike the timestamps, the results show that all the bits in the SID have a similar impact on the produced descriptors.

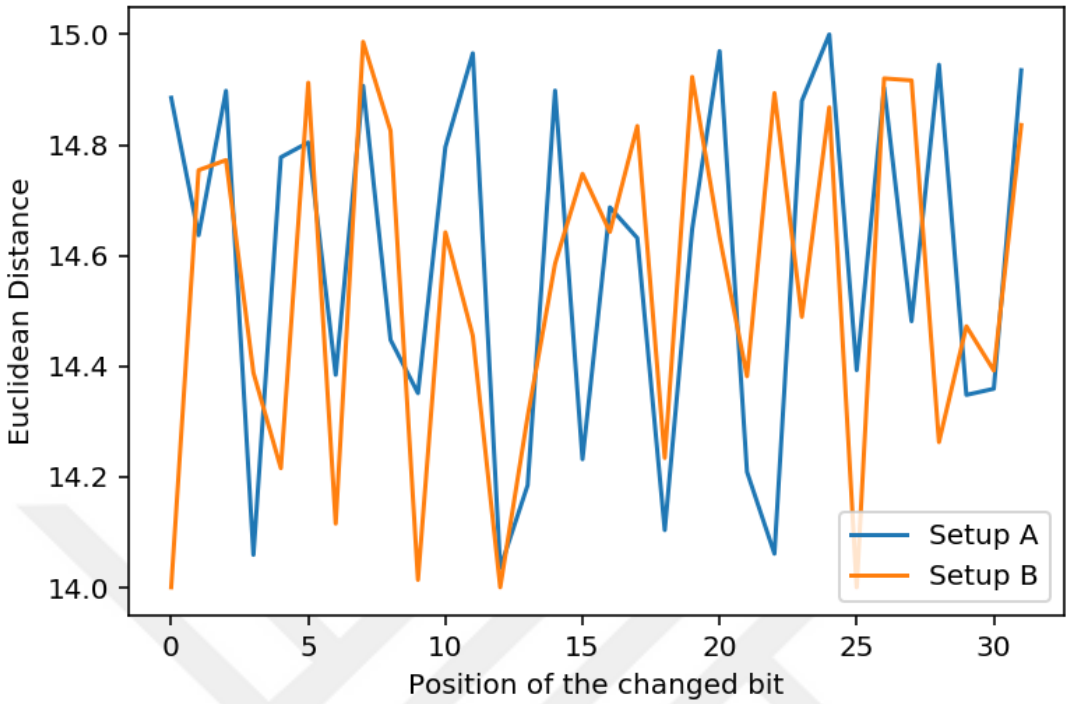

Figure 4.4: The influence of each bit in the SID over the distance between the produced descriptors.

Despite the ability of the proposed method to compare the binary values of the SID, the existence of only 32 values limits the number of possible values to 2^{32} when only binary values are considered. Such a limited number of possibilities allow attackers to execute brute-force attacks, in which a descriptor is generated per each possible SID. Alternatively, according to the ability of neural networks to process continuous values, the proposed method is evaluated by providing continuous SID values. Thus, instead of changing a single value, as executed in the previous experiment, an experiment in which several values are changed is also conducted. In this experiment, each value in the SID is randomly selected from the interval [0, 1]. The Euclidean distance between the generated SIDs is calculated and compared to the Euclidean distance of the descriptors generated for each SID. Similar to the previous experiment, all the remaining inputs are maintained identical to ensure an unbiased evaluation. Figure 4.5 shows the relation between the measured distances.

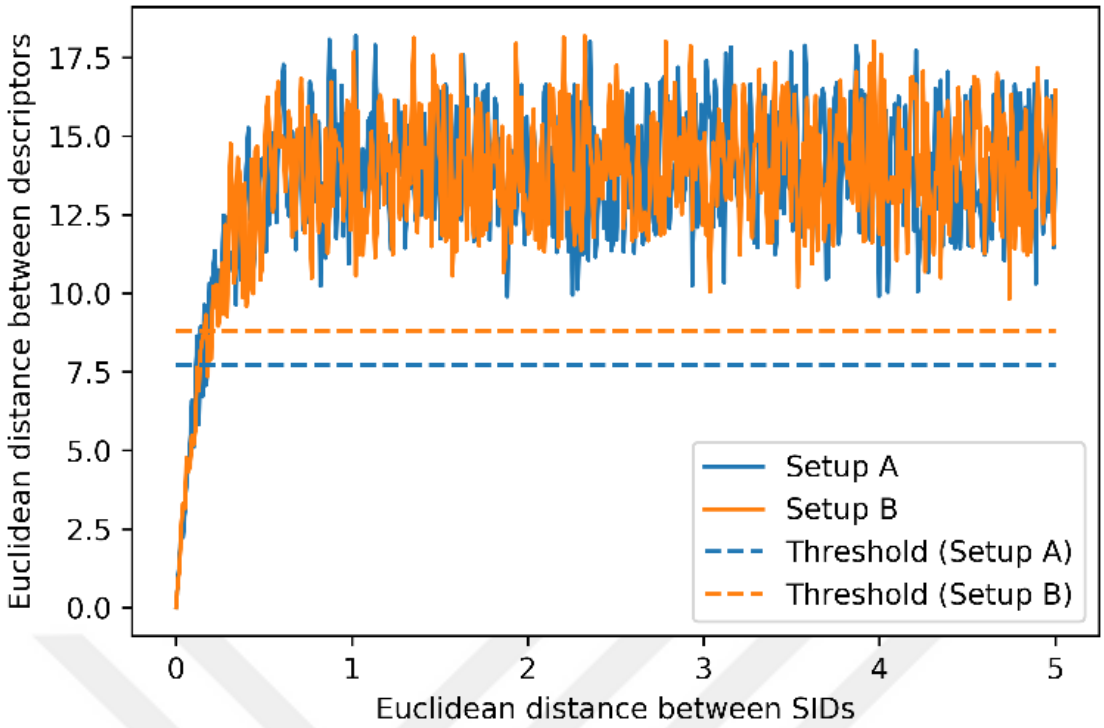

Figure 4.5: Euclidean distances between descriptors versus distances between input SIDs.

The results show that the use of continuous values in the SID has achieved better performance, as the authentication threshold is exceeded before reaching a Euclidean distance between the SIDs of one. Such improvement is according to the amplification of the differences between multiple values in the SID, instead of amplifying the difference in a single value, which is easier to saturate. This amplification is according to the multiplication of the detected differences by the weights of the neural network. Hence, the use of continuous values in the SIDs denies the use of a brute-force approach by attackers, according to the infinite number of possible SIDs, and denies the use of a descriptor that is generated at a certain system in another.

4.4. PRIVACY PROTECTION ANALYSIS

The use of activation functions in the neural network provides the nonlinearity required to improve the performance of these networks. However, such functions deny the ability of retrieving the inputs using the outputs of the network, similar to unidirectional functions. For instance, if the output of a ReLU activation function if zero, it is impossible to retrieve the value inputted to the function, i.e. the output of the neuron cannot be predicted. Hence, most of the input values cannot be retrieved when conducting a reverse pass using the same neural network.

However, as the descriptors generated based on the biometric templates must contain the significant features in the templates; a separate neural network can be trained to retrieve

these inputs in a hierarchy similar to auto encoders. As the biometric templates of the subjects must be protected even from the system administrator's access and considering training an auto encoding neural network presents the best chance to retrieve these templates, such a neural network is implemented and used to retrieve the biometric templates from the vectors stored in the database. The output of the auto encoder for sample inputs collected from the evaluation setups are shown in Figure 4.6.

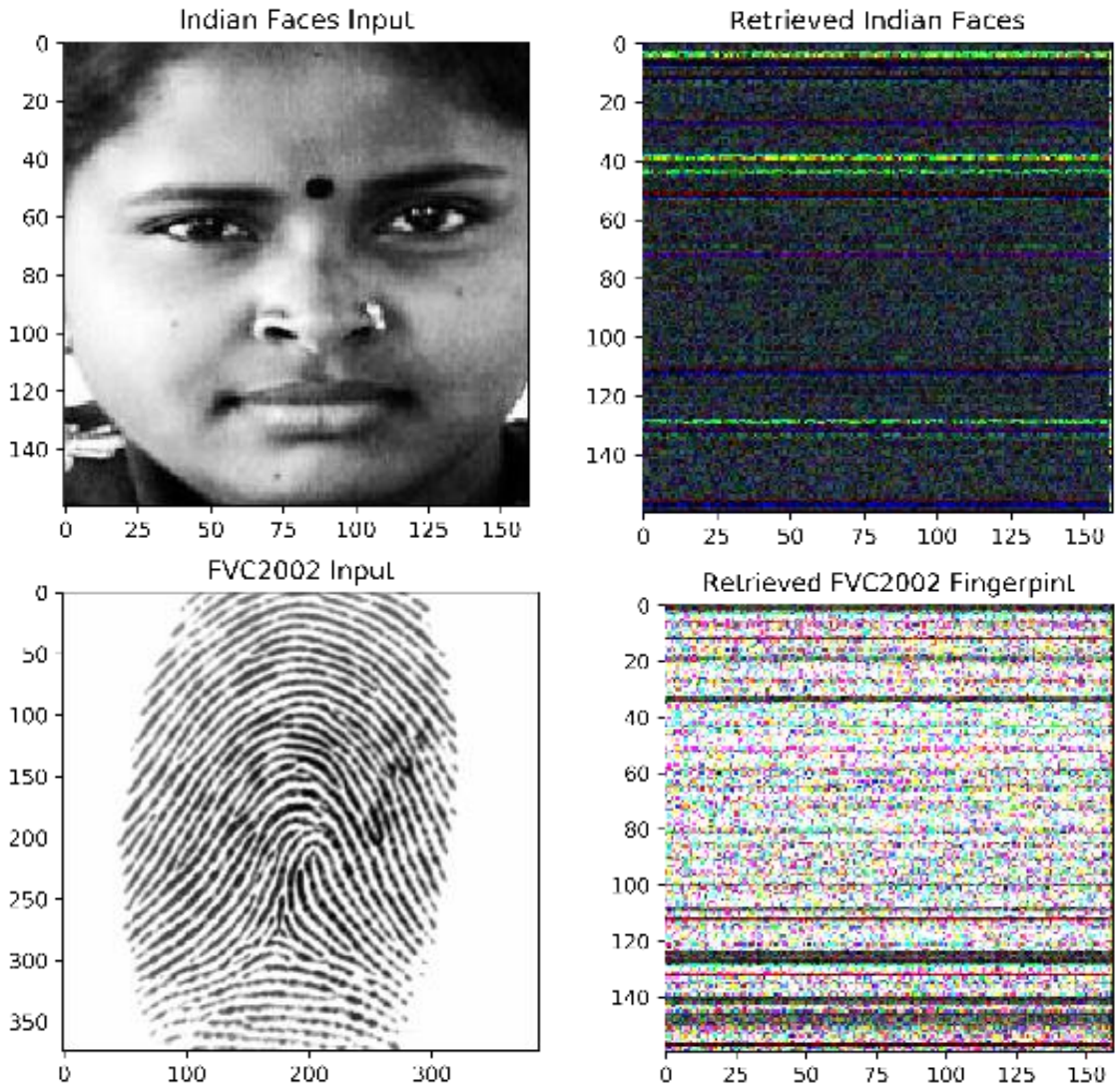

Figure 4.6: Samples of the inputted and retrieved biometric templates using the autoencoder neural network.

These results show that the original biometric credentials in the templates cannot be retrieved from the vectors stored in the database. Mainly, the storage of the features in a one-dimensional vector loses the relativity required to reconstruct a two-dimensional image. This is also the main reason behind the existence of the distinguishable horizontal lines in the images, as these lines are results of extending the values of the original descriptor. Thus, even with the knowledge of the SID and the absence of the timestamp in the vectors stored in the database, the nonlinearity of the activation functions of the

neural network has denied the ability to retrieve the biometric templates from these vectors. Thus, even the system administrators do not have the ability to retrieve the biometric templates of the subjects registered in the system to protect their privacy.

4.5. TIME ANALYSIS

Unlike traditional image processing techniques that are used to extract image descriptors the time required to process input using a neural network is constant, regardless of the number of the key points in the image. The proposed method requires three separate processes, one to create a descriptor for the templates collected at the sensors' side, another to generate the database vectors and one for timestamping the database vectors before executing the authentication process. The experiments are conducted using a Windows computer with a 2.80GHz processor and 16GB of memory. The Nvidia GTX-1080Ti Graphical Processing Unit (GPU) is used to accelerate the computations required by the neural network, by parallelizing these computations instead of the serial execution in the processor.

The average time required to generate the descriptor at the sensor side is 172mS, with a similar time to generate the database vectors. This similarity is according to the minor difference between the neural networks that are used for these tasks. In average, 68.1*u*S is consumed to produce a timestamped descriptor from the database vector. Finally, 5.23*u*S is consumed to measure the Euclidean distance between the descriptors, i.e. authenticate the candidate. As each vector in the database is generated and stored once, the average time required to enrol a candidate after the arrival of the descriptor from the sensor side is 73.33*u*S, which is the summation of the time required to timestamp the database vector and measuring the Euclidean distance. These measurements neglect the time required to retrieve the vectors from the database, as such time is beyond the control of the proposed method.

5. DISCUSSION

With the rapidly growing need for more secure and simple biometric authentication systems, the use of the proposed method can provide significant contribution to these systems. These features are achieved by combining the time sensitivity with the uniqueness of the values in the descriptors generated for a candidate. With this time sensitivity, a descriptor that is captured by an attacker who is listening to the communications cannot be used to authenticate into the system, as soon as the predefined interval is finished. Although such an approach can be achieved using different approaches, e.g. [8, 9], the embedding of such feature within the descriptors generated for the candidates, and the elimination of the need for communicating with the matching server, simplifies the authentication procedure, as all the variables are authenticated in a single distance measurement.

Another important security aspect is the immunity of the proposed method against attacks executed using descriptors that are generated using the same methodology, i.e. neural network. To avoid such limitation, the proposed method relies on the use of a unique identification number that is set for a certain system. When the values of that ID are different from those used to generate the templates in the models' database, the values produced in the descriptor are different from those generated for the same user using a different system, as shown in Figure 4.7. Hence, the authentication in such scenarios fail, as the Euclidean distance is larger than the threshold values. Moreover, the embedding of these values in the descriptors also maintains the simplicity of the system, as no additional stages are required for this task.

In addition to these features, the proposed method maintains the privacy of the users by preventing the original biometric templates to be retrieved from the descriptors. This feature is maintained even when the other parameters, i.e. the system ID and timestamp that are used for the generation of the descriptor are known. Hence, even the administrators of the systems that use the proposed method lack the ability to retrieve these templates. This feature is a result of using ANNs to generate the descriptors, which emphasizes the ability of these networks to act as one-way functions, similar to the results shown in [128-130]. Even the use of another neural network to attack these descriptors by using the inputs are training outputs has failed to retrieve the original images, which is also a result of the use of Siamese networks, in which the features are fused in all values.

With these achievements, it is important to evaluate the performance of the proposed method in terms of acts ability to accurately recognize and authenticate candidates, compared to existing state-of-the-art methods. Despite the use of the entire biometric templates in the methods proposed in [18, 19, 23], the proposed method has been able to achieve better performance, in terms of authentication accuracy. As shown in Figure 5.1, the proposed method has achieved higher are under the ROC curve, which indicates better accuracy, compared to [18, 19]. Moreover, the Cumulative Match Characteristic (CMC) curve, shown in Figure 5.2, shows that the Euclidean distances between the descriptors generated for the biometric templates are very low when for the same subject, as they are ranking in the first three ranks. Additionally, the proposed method has been able to achieve a higher identification rate even at lower ranks, compared to the method proposed by Wang et al. [23]. Thus, the proposed method has been able to outperform existing methods in terms of authentication accuracy, while lowering the size of the data being exchanged between the different parts of the system.

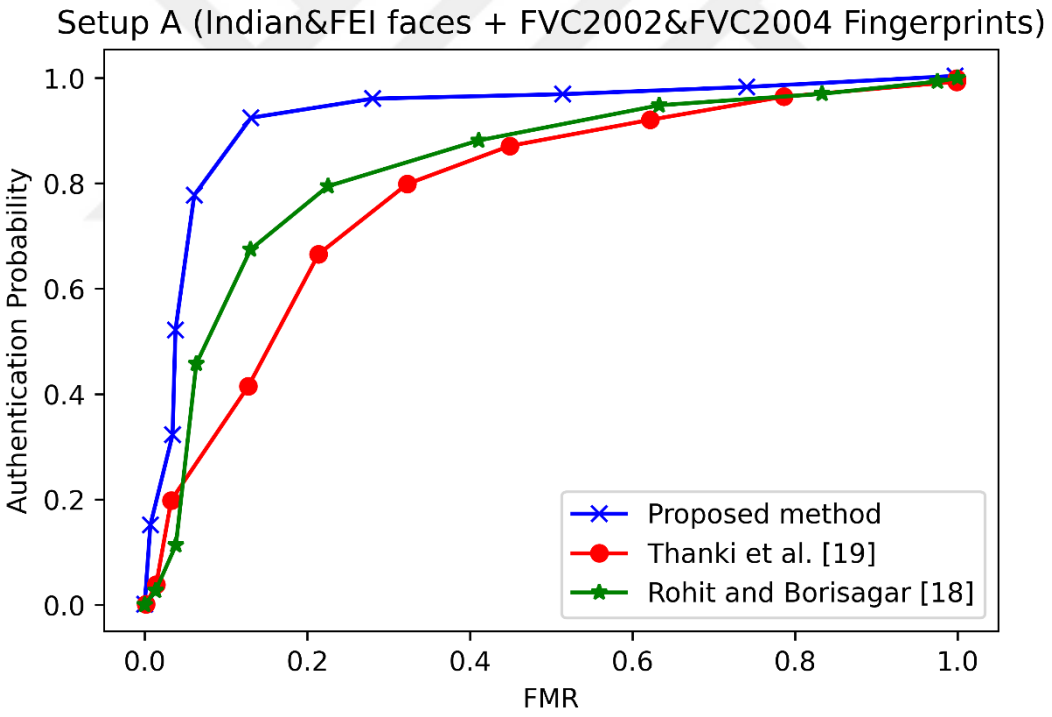

Figure 5.1: ROC curves of the proposed and existing methods using Setup A.

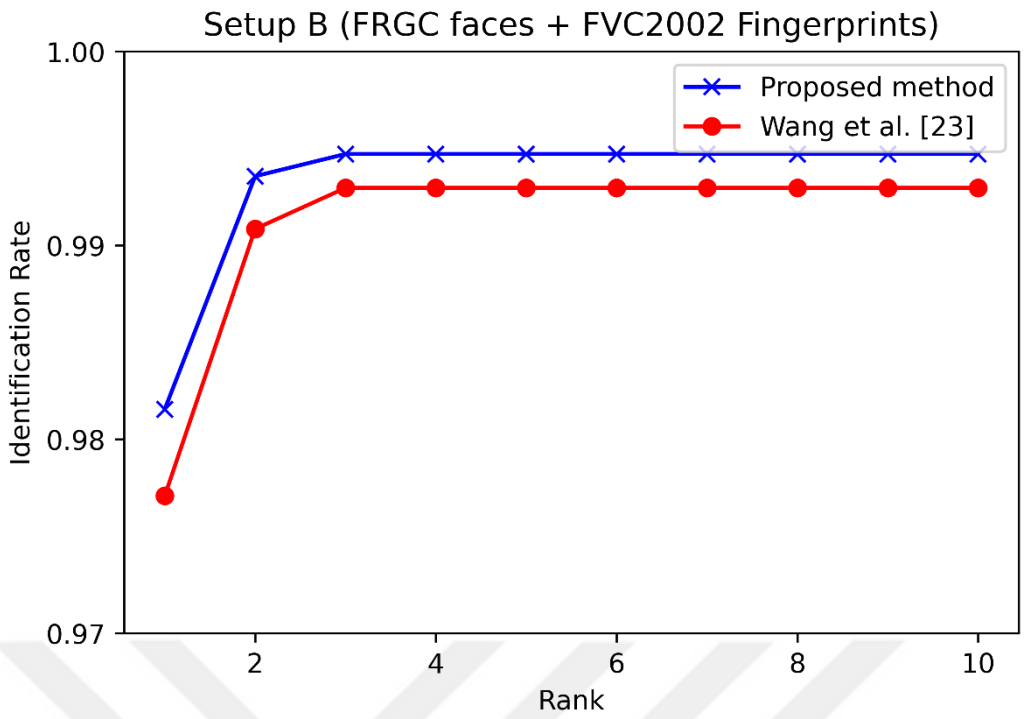

Figure 5.2: CMC curves of the proposed and existing methods using Setup B.

6. CONCLUSION

The importance and sensitivity of private information and services being provided digitally require secure authentication methods to restrict access to them, so that, only legitimate users can access their intended services and information. However, the attacking techniques that are being used by intruders to gain access to these systems or deny legitimate users from getting into them are also developing rapidly. This development is triggered by the importance of this information and services, such as banking services, and the huge benefits that attackers can gain from accessing them. Accordingly, traditional secret-based authentication methods, e.g. passwords and patterns, have become of limited usage, according to their sensitivity to simple attacks, such as shoulder surfing and guessing attacks.

Biometric authentication has emerged as a solution to handle these limitations, according to the high availability and uniqueness of biometric features, especially physical ones. Moreover, with the integration of anti-spoofing methods in the sensors that collect the biometric templates, such as liveness detection, limits the ability to gain access to the system even if the biometric template of a legitimate user is replicated. These features have brought significant attention toward using biometric authentication in different applications. However, according to the changing nature of humans and the interaction with the vitals, the collected templates can be guaranteed to be identical when collected from the same candidate. Thus, feature extraction methods are required to handle these variations and extract the same features, so that, the candidate can be matched to the model template and authenticated into the system.

As the features extracted from these templates can vary according to many conditions, such as lighting and posture in face recognition, pressure and positioning in fingerprint, false positives and false negatives can occur during the authentication. However, such false decisions can have dramatic influence on the reliability of the system, especially with the sensitivity of the information and services protected by these systems. Hence, several researchers have relied on multimodal biometric authentication systems, which rely on multiple templates to authenticate a candidate. Accordingly, if a false decision occurs based on one of the templates, this error is rectified based on the other template, which has shown significant improvement in the accuracy of the biometric authentication systems.

In addition to the challenges imposed by the use of multimodal biometrics, which is the increased need for computer storage and network bandwidth to store and communicate data, the use of this type of biometric authentication does not contribute towards handling the security vulnerabilities of biometric authentication systems, in general. Despite the existence of methods that can immune biometric authentication systems against most of their vulnerabilities, securing some of these vulnerabilities requires a separate stage of data processing, which increases the complexity of the system. Additionally, users have shown significant concerns about their privacy when provided their biometric templates, according to the consequences of misusing such information even by system administrators.

In this study, a new method is proposed to generate descriptors for multimodal biometric authentication systems. The proposed method uses transfer learning to fine tune convolutional neural network to extract the distinctive features in face and fingerprint templates. Then, these features are fused together alongside a timestamp and a system identifier to produce a fixed-size descriptor that can be used to authenticate a candidate by comparing it to the one stored in the model templates database. However, as the templates stored in the database are required to be retrieved at different times, the timestamp is not appended to these descriptors when stored. Only the SID is appended to the stored descriptor, whereas the timestamp is added when the descriptor is retrieved and used in the matching process.

The use of the artificial neural networks allows the learning, extraction and use of only distinctive features, i.e. inter-class feature, whereas intra-class features are neglected. This behavior significantly reduces the false positives and false negatives that can be produced when extracted common, non-distinctive, features from the templates, based on the extraction of hand-crafted features. The results of the experiments show that the values of the features in the produced vectors remain similar when only one or both of the two LSBs of the timestamp are changed, which allows considering the time required communicating the descriptors in the network. However, when any of the other bits in the timestamp is changed, the similarity between the descriptors falls below the threshold value, which denies any replay attacks. Additionally, a change in any of the bits of the SID value has resulted in a similarity that is also below the threshold value, i.e. large Euclidean distance, so that, a descriptor generated at a certain system cannot be used to authenticate into another.

In addition to these features, the descriptors generated using the proposed method has been able to achieve high authentication accuracy, 99.41% and 99.32% using two different setups, similar to those achieved by earlier methods, with 99.37% and 96.27% using the same setups, which use the entire templates for feature extraction at the similarity measurement stage. With such performance measures and the fixed size of the generated descriptors, the proposed method can significantly improve the performance of the multimodal biometric authentication systems. This improvement is achieved by reducing the size of the information being stored and communicated in the system, securing the system against replay attacks and maintaining the privacy of the users while achieving slightly better authentication accuracy, compared to existing state-of-the-art biometric authentication systems, which rely on the images of the entire biometric credentials during the matching process. The lower storage and bandwidth consumption are according to the higher efficiency of the produced descriptors, which is according to the ability of deep neural networks to handle inter- and intra-class variations. By neglecting the intra-class variations, the neural network outputs similar values for inputs template from the same candidate, whereas different values are produced by emphasizing inter-class variations. The security against replay attacks is achieved by fusing the timestamp in which the descriptor is generated with the values in the produced descriptor. Finally, the privacy of the users is preserved by ensuring that the original credentials cannot be retrieved from the produced descriptors by using a unique identifier for each system and fusing them into the generated descriptors, so that; even information about the enrolling candidates cannot be retrieved.

In future work, the ability to apply this method on other biometric features is going to be investigated. The iris biometrics can be used instead of the fingerprints', according to the high robustness of those features despite the high cost of the sensors that collect biometric features from the iris.

REFERENCES

[1] H.-M. Sun, S.-T. Chen, J.-H. Yeh, and C.-Y. Cheng, "A shoulder surfing resistant graphical authentication system," *IEEE Transactions on Dependable and Secure Computing,* vol. 15, pp. 180-193, 2018.

[2] M. Nagatomo, Y. Kita, K. Aburada, N. Okazaki, and M. Park, "Implementation and user testing of personal authentication having shoulder surfing resistance with mouse operations," *IEICE Communications Express,* vol. 7, pp. 77-82, 2018.

[3] M. A. S. Gokhale and V. S. Waghmare, "The shoulder surfing resistant graphical password authentication technique," *Procedia Computer Science,* vol. 79, pp. 490-498, 2016.

[4] M. Jiang, A. He, K. Wang, and Z. Le, "Two-Way Graphic Password for Mobile User Authentication," in *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*, 2015, pp. 476-481.

[5] S. A. Chaudhry, K. Mahmood, H. Naqvi, and M. K. Khan, "An improved and secure biometric authentication scheme for telecare medicine information systems based on elliptic curve cryptography," *Journal of Medical Systems,* vol. 39, p. 175, 2015.

[6] D. Dasgupta, A. Roy, and A. Nag, *Advances in User Authentication*: Springer, 2017.

[7] N. K. Ratha, J. H. Connell, and R. M. Bolle, "Enhancing security and privacy in biometrics-based authentication systems," *IBM systems Journal,* vol. 40, pp. 614-634, 2001.

[8] Q. Gui, W. Yang, Z. Jin, M. V. Ruiz-Blondet, and S. Laszlo, "A residual feature-based replay attack detection approach for brainprint biometric systems," in *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, 2016, pp. 1-6.

[9] A. A. Khan, V. Kumar, and M. Ahmad, "An elliptic curve cryptography based mutual authentication scheme for smart grid communications using biometric approach," *Journal of King Saud University-Computer and Information Sciences*, 2019.

[10] R. Gupta and P. Sehgal, "Non-deterministic approach to allay replay attack on iris biometric," *Pattern Analysis and Applications,* vol. 22, pp. 717-729, 2019.

[11] C.-T. Chou, S.-W. Shih, W.-S. Chen, V. W. Cheng, and D.-Y. Chen, "Non-orthogonal view iris recognition system," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20, pp. 417-430, 2009.

[12] K. M. A. Alheeti, "Biometric iris recognition based on hybrid technique," *International Journal on Soft Computing,* vol. 2, p. 1, 2011.

[13] P. Radu, K. Sirlantzis, W. G. J. Howells, S. Hoque, and F. Deravi, "Optimizing 2D gabor filters for iris recognition," in *2013 Fourth International Conference on Emerging Security Technologies*, 2013, pp. 47-50.

[14] H. Mehrotra, G. Badrinath, B. Majhi, and P. Gupta, "An efficient dual stage approach for iris feature extraction using interest point pairing," in *2009 IEEE Workshop on Computational Intelligence in Biometrics: Theory, Algorithms, and Applications*, 2009, pp. 59-62.

[15] N. Ahmadi and M. Nilashi, "Iris texture recognition based on multilevel 2-D Haar wavelet decomposition and Hamming distance approach," *Journal of Soft Computing and Decision Support Systems*, vol. 5, pp. 16-20, 2018.

[16] O. Nafea, S. Ghouzali, W. Abdul, and E.-u.-H. Qazi, "Hybrid multi-biometric template protection using watermarking," *The Computer Journal*, vol. 59, pp. 1392-1407, 2016.

[17] R. Thanki and K. Borisagar, "Multibiometric Template Security Using CS Theory–SVD Based Fragile Watermarking Technique," *WSEAS Transactions on Information Science and Applications*, vol. 12, pp. 1-10, 2015.

[18] R. Thanki and K. Borisagar, "Biometric watermarking technique based on cs theory and fast discrete curvelet transform for face and fingerprint protection," in *Advances in signal processing and intelligent recognition systems*, ed: Springer, 2016, pp. 133-144.

[19] R. M. Thanki, V. J. Dwivedi, and K. R. Borisagar, "Multibiometric Watermarking Technique Using Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT)," in *Multibiometric Watermarking with Compressive Sensing Theory*, ed: Springer, 2018, pp. 91-113.

[20] W. Wojtowicz and M. R. Ogiela, "Digital images authentication scheme based on bimodal biometric watermarking in an independent domain," *Journal of Visual Communication and Image Representation*, vol. 38, pp. 1-10, 2016.

[21] M. R. M. Isa, S. Aljareh, Z. J. M. T. Yusoff, and Applications, "A watermarking technique to improve the security level in face recognition systems," vol. 76, pp. 23805-23833, 2017.

[22] B. Ma, Y. Wang, C. Li, Z. Zhang, D. J. M. t. Huang, and applications, "Secure multimodal biometric authentication with wavelet quantization based fingerprint watermarking," vol. 72, pp. 637-666, 2014.

[23] B. Ma, Y. Wang, C. Li, Z. Zhang, and D. Huang, "Secure multimodal biometric authentication with wavelet quantization based fingerprint watermarking," *Multimedia tools and applications*, vol. 72, pp. 637-666, 2014.

[24] W. Meng, D. S. Wong, S. Furnell, and J. Zhou, "Surveying the development of biometric user authentication on mobile phones," *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 1268-1293, 2015.

[25] A. Mahfouz, I. Muslukhov, and K. Beznosov, "Android users in the wild: Their authentication and usage behavior," *Pervasive and Mobile Computing*, vol. 32, pp. 50-61, 2016.

[26] L. Sobrado and J.-C. Birget, "Graphical passwords," *The Rutgers Scholar, an electronic Bulletin for undergraduate research*, vol. 4, pp. 12-18, 2002.

[27] R. Spolaor, Q. Li, M. Monaro, M. Conti, L. Gamberini, and G. Sartori, "Biometric Authentication Methods on Smartphones: A Survey," *PsychNology Journal*, vol. 14, 2016.

[28] M. Harbach, A. De Luca, and S. Egelman, "The anatomy of smartphone unlocking: A field study of android lock screens," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 4806-4817.

[29] F. Monrose and A. D. Rubin, "Keystroke dynamics as a biometric for authentication," *Future Generation computer systems*, vol. 16, pp. 351-359, 2000.

[30] H. Saevanee and P. Bhatarakosol, "User authentication using combination of behavioral biometrics over the touchpad acting like touch screen of mobile device," in *Computer and Electrical Engineering, 2008. ICCEE 2008. International Conference on*, 2008, pp. 82-86.

[31] A. Natarajan and N. Shanthi, "A Survey on Multimodal Biometrics Authentication and Template Protection," in *2018 International Conference on Intelligent Computing and Communication for Smart World (I2C2SW)*, 2018, pp. 64-71.

[32] S. N. Garg, R. Vig, and S. Gupta, "A survey on different levels of fusion in multimodal biometrics," *Indian Journal of Science and Technology*, vol. 10, 2017.

[33] T. B. Long and T. Hanh, "Multimodal biometric person authentication using fingerprint, face features," in *Pacific Rim International Conference on Artificial Intelligence*, 2012, pp. 613-624.

[34] N. Hezil and A. Boukrouche, "Multimodal biometric recognition using human ear and palmprint," *IET Biometrics*, vol. 6, pp. 351-359, 2017.

[35] T. A. T. Nguyen and T. K. Dang, "Privacy preserving biometric-based remote authentication with secure processing unit on untrusted server," *IET Biometrics*, vol. 8, pp. 79-91, 2018.

[36] P. Aithal, "A Critical Study on Fingerprint Image Sensing and Acquisition Technology," 2017.

[37] G. Lovisotto, S. Eberz, and I. Martinovic, "Biometric Backdoors: A Poisoning Attack Against Unsupervised Template Updating," *arXiv preprint arXiv:1905.09162*, 2019.

[38] A. Vij and A. Namboodiri, "Learning minutiae neighborhoods: A new binary representation for matching fingerprints," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 64-69.

[39] S. Haware and A. Barhatte, "Retina based biometric identification using SURF and ORB feature descriptors," in *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*, 2017, pp. 1-6.

[40] M. H. Hamd and S. K. Ahmed, "Biometric system design for iris recognition using intelligent algorithms," *International Journal of Modern Education and Computer Science*, vol. 10, p. 9, 2018.

[41] D. Valdes-Ramirez, M. A. Medina-Pérez, R. Monroy, O. Loyola-González, J. Rodríguez-Ruiz, A. Morales, *et al.*, "A Review of Fingerprint Feature Representations and Their Applications for Latent Fingerprint Identification: Trends and Evaluation," *IEEE Access*, vol. 7, pp. 48484-48499, 2019.

[42] C. Gottschlich, B. Tams, and S. Huckemann, "Perfect fingerprint orientation fields by locally adaptive global models," *IET Biometrics*, vol. 6, pp. 183-190, 2016.

[43] T.-Y. Jea and V. Govindaraju, "A minutia-based partial fingerprint recognition system," *Pattern recognition*, vol. 38, pp. 1672-1684, 2005.

[44] A. Sardar, S. Umer, C. Pero, and M. Nappi, "A Novel Cancelable FaceHashing Technique Based on Non-Invertible Transformation With Encryption and Decryption Template," *IEEE Access*, vol. 8, pp. 105263-105277, 2020.

[45] J. B. Kho, J. Kim, I.-J. Kim, and A. B. Teoh, "Cancelable fingerprint template design with randomized non-negative least squares," *Pattern Recognition*, vol. 91, pp. 245-260, 2019.

[46] E. Maiorana and P. Campisi, "Fuzzy commitment for function based signature template protection," *IEEE signal processing letters*, vol. 17, pp. 249-252, 2009.

[47] C. Li and J. Hu, "A security-enhanced alignment-free fuzzy vault-based fingerprint cryptosystem using pair-polar minutiae structures," *IEEE Transactions on Information Forensics and Security*, vol. 11, pp. 543-555, 2015.

[48] N. Riaz, A. Riaz, and S. A. Khan, "Biometric template security: an overview," *Sensor Review*, 2018.

[49] C. Herder, L. Ren, M. Van Dijk, M.-D. Yu, and S. Devadas, "Trapdoor computational fuzzy extractors and stateless cryptographically-secure physical unclonable functions," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, pp. 65-82, 2016.

[50] T. A. T. Nguyen and T. K. Dang, "Combining fuzzy extractor in biometric-kerberos based authentication protocol," in *2015 International Conference on Advanced Computing and Applications (ACOMP)*, 2015, pp. 1-6.

[51] K. Gurney, *An introduction to neural networks*: CRC press, 2014.

[52] O. Niculaescu, "Neural networks and how machines learn meaning," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 25, pp. 64-66, 2019.

[53] O. Eluyode and D. T. Akomolafe, "Comparative study of biological and artificial neural networks," *European Journal of Applied Engineering and Scientific Research*, vol. 2, pp. 36-46, 2013.

[54] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, 2015, pp. 1117-1125.

[55] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International Conference on Machine Learning*, 2013, pp. 1058-1066.

[56] S. Sharma, "Activation functions in neural networks," *towards data science,* vol. 6, 2017.

[57] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853,* 2015.

[58] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence,* vol. 39, pp. 1137-1149, 2017.

[59] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems,* vol. 1, pp. 111-122, 2011.

[60] M. Sheinfeld, S. Levinson, and I. Orion, "Highly accurate prediction of specific activity using deep learning," *Applied Radiation and Isotopes,* vol. 130, pp. 115-120, 2017.

[61] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," in *International Conference on Machine Learning*, 2019, pp. 1675-1685.

[62] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, *et al.*, "Learning to learn by gradient descent by gradient descent," *arXiv preprint arXiv:1606.04474,* 2016.

[63] J. He, J. Rexford, and M. Chiang, "Design for optimizability: Traffic management of a future internet," in *Algorithms for Next Generation Networks*, ed: Springer, 2010, pp. 3-18.

[64] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[65] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics,* 2010, pp. 249-256.

[66] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, ed: Elsevier, 1992, pp. 65-93.

[67] I. K. Sethi and A. K. Jain, *Artificial neural networks and statistical pattern recognition: old and new connections* vol. 11: Elsevier, 2014.

[68] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks,* vol. 61, pp. 85-117, 2015.

[69] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458,* 2015.

[70] H. H. Aghdam and E. J. Heravi, "Guide to convolutional neural networks," *New York, NY: Springer*, vol. 10, pp. 978-973, 2017.

[71] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[72] V. Christlein, L. Spranger, M. Seuret, A. Nicolaou, P. Král, and A. Maier, "Deep generalized max pooling," in *2019 International Conference on Document Analysis and Recognition (ICDAR)*, 2019, pp. 1090-1096.

[73] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 111-118.

[74] M. T. García-Ordás, J. A. Benítez-Andrades, I. García-Rodríguez, C. Benavides, and H. Alaiz-Moretón, "Detecting respiratory pathologies using convolutional neural networks and variational autoencoders for unbalancing data," *Sensors*, vol. 20, p. 1214, 2020.

[75] I. N. Da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, "Artificial neural network architectures and training processes," in *Artificial neural networks*, ed: Springer, 2017, pp. 21-28.

[76] Ö. F. Ertuğrul, "A novel type of activation function in artificial neural networks: Trained activation function," *Neural Networks*, vol. 99, pp. 148-157, 2018.

[77] R. Salloum and C.-C. J. Kuo, "ECG-based biometrics using recurrent neural networks," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 2062-2066.

[78] R. D. Labati, E. Muñoz, V. Piuri, R. Sassi, and F. Scotti, "Deep-ECG: convolutional neural networks for ECG biometric recognition," *Pattern Recognition Letters*, vol. 126, pp. 78-85, 2019.

[79] O. Vinyals, C. Blundell, T. Lillicrap, and D. Wierstra, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, pp. 3630-3638, 2016.

[80] T. S. Mohammed and N. I. Al-Taie, "Artificial neural networks as decision-makers for stereo matching," *GSTF Journal on Computing (JoC)*, vol. 1, 2020.

[81] L. Tawfiq and M. Tawfiq, "The Effect of Number of Training Samples for Artificial Neural Network," *Ibn AL-Haitham Journal For Pure and Applied Science*, vol. 23, 2017.

[82] A. Shaban, S. Bansal, Z. Liu, I. Essa, and B. Boots, "One-shot learning for semantic segmentation," *arXiv preprint arXiv:1709.03410*, 2017.

[83] M. Khalil-Hani and L. S. Sung, "A convolutional neural network approach for face verification," in *2014 International Conference on High Performance Computing & Simulation (HPCS)*, 2014, pp. 707-714.

[84] D. Chicco, "Siamese neural networks: An overview," *Artificial Neural Networks*, pp. 73-94, 2020.

[85] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815-823.

[86] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, 2015.

[87] S.-C. Hsiao, D.-Y. Kao, Z.-Y. Liu, and R. Tso, "Malware image classification using one-shot learning with Siamese networks," *Procedia Computer Science*, vol. 159, pp. 1863-1871, 2019.

[88] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, ed: IGI global, 2010, pp. 242-264.

[89] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International conference on artificial neural networks*, 2018, pp. 270-279.

[90] H. Ravishankar, P. Sudhakar, R. Venkataramani, S. Thiruvenkadam, P. Annangi, N. Babu, *et al.*, "Understanding the mechanisms of deep transfer learning for medical images," in *Deep learning and data labeling for medical applications*, ed: Springer, 2016, pp. 188-196.

[91] X. Yin, X. Yu, K. Sohn, X. Liu, and M. Chandraker, "Feature transfer learning for face recognition with under-represented data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5704-5713.

[92] U. Mahbub, V. M. Patel, D. Chandra, B. Barbello, and R. Chellappa, "Partial face detection for continuous authentication," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 2991-2995.

[93] G. B. de Souza, J. P. Papa, and A. N. Marana, "On the learning of deep local features for robust face spoofing detection," in *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, 2018, pp. 258-265.

[94] S. W. Cho, N. R. Baek, M. C. Kim, J. H. Koo, J. H. Kim, and K. R. Park, "Face detection in nighttime images using visible-light camera sensors with two-step faster region-based convolutional neural network," *Sensors*, vol. 18, p. 2995, 2018.

[95] G. Kątek, A. Holik, T. Zabłocki, and P. Dobrzyńska, "Face recognition using the Haar classifier cascade and face detection based on detection of skin color areas," *I ELEKTRONIKA*, p. 29, 2016.

[96] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, pp. 137-154, 2004.

[97] K. D. Pandya, "Face Detection–A Literature Survey," *Int. J. Comput. Tech*, vol. 3, pp. 67-70, 2016.

[98] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, pp. 1499-1503, 2016.

[99] M. Ma and J. Wang, "Multi-View Face Detection and Landmark Localization Based on MTCNN," in *2018 Chinese Automation Congress (CAC)*, 2018, pp. 4200-4205.

[100] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM computing surveys (CSUR)*, vol. 35, pp. 399-458, 2003.

[101] E. Jose, M. Greeshma, M. H. TP, and M. Supriya, "Face recognition based surveillance system using facenet and mtcnn on jetson tx2," in *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, 2019, pp. 608-613.

[102] M. He, J. Zhang, S. Shan, M. Kan, and X. Chen, "Deformable face net: Learning pose invariant feature with pose aware feature alignment for face recognition," in *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*, 2019, pp. 1-8.

[103] T. Kanade, "Picture processing system by computer complex and recognition of human faces," 1974.

[104] L. Wiskott, J.-M. Fellous, N. Krüger, and C. Von Der Malsburg, "Face recognition by elastic bunch graph matching," in *International Conference on Computer Analysis of Images and Patterns*, 1997, pp. 456-463.

[105] P. J. Phillips, S. Z. Der, P. J. Rauss, and O. Z. Der, *FERET (face recognition technology) recognition algorithm development and test results*: Army Research Laboratory Adelphi, MD, 1996.

[106] K. Shrivastava, S. Manda, P. Chavan, T. Patil, and S. Sawant-Patil, "Conceptual Model for Proficient Automated Attendance System based on Face Recognition and Gender Classification using Haar-Cascade, LBPH Algorithm along with LDA Model," *International Journal of Applied Engineering Research*, vol. 13, pp. 8075-8080, 2018.

[107] G. Du, F. Su, and A. Cai, "Face recognition using SURF features," in *MIPPR 2009: Pattern Recognition and Computer Vision*, 2009, p. 749628.

[108] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer vision and image understanding*, vol. 110, pp. 346-359, 2008.

[109] L. Juan and O. Gwun, "A comparison of sift, pca-sift and surf," *International Journal of Image Processing (IJIP)*, vol. 3, pp. 143-152, 2009.

[110] B. Anand and M. P. K. Shah, "Face Recognition using SURF Features and SVM Classifier," *International Journal of Electronics Engineering Research. ISSN*, pp. 0975-6450, 2016.

[111] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *BMVC*, 2015, p. 6.

[112] F. Zhang, S. Xin, and J. Feng, "Combining global and minutia deep features for partial high-resolution fingerprint matching," *Pattern Recognition Letters*, vol. 119, pp. 139-147, 2019.

[113] R. Ramachandra, M. Stokkenes, A. Mohammadi, S. Venkatesh, K. Raja, P. Wasnik, *et al.*, "Smartphone Multi-modal Biometric Authentication: Database and Evaluation," *arXiv preprint arXiv:1912.02487*, 2019.

[114] D. J. Ketcham, "Real-time image enhancement techniques," in *Image processing*, 1976, pp. 120-125.

[115] S. F. Tan and N. A. M. Isa, "Exposure based multi-histogram equalization contrast enhancement for non-uniform illumination images," *IEEE Access*, vol. 7, pp. 70842-70861, 2019.

[116] W. Wang, Z. Chen, X. Yuan, and X. Wu, "Adaptive image enhancement method for correcting low-illumination images," *Information Sciences*, vol. 496, pp. 25-41, 2019.

[117] M. Jordanski, A. Arsic, and M. Tuba, "Dynamic recursive subimage histogram equalization algorithm for image contrast enhancement," in *Telecommunications Forum Telfor (TELFOR), 2015 23rd*, 2015, pp. 819-822.

[118] Y.-S. Wang, C.-L. Tai, O. Sorkine, and T.-Y. Lee, "Optimized scale-and-stretch for image resizing," in *ACM Transactions on Graphics (TOG)*, 2008, p. 118.

[119] V. Patel and K. Mistree, "A review on different image interpolation techniques for image enhancement," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, pp. 129-133, 2013.

[120] M. R. Golla, P. Sharma, and J. Madarkar, "Face Recognition Algorithm for Low-Resolution Images," in *Social Networking and Computational Intelligence*, ed: Springer, 2020, pp. 349-362.

[121] D. Sandberg, "FaceNet," *GitHub Repos*, 2017.

[122] P. J. Phillips, H. Moon, S. A. Rizvi, and P. J. Rauss, "The FERET evaluation methodology for face-recognition algorithms," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, pp. 1090-1104, 2000.

[123] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, and A. K. Jain. (2002). *FVC2002*. Available: http://bias.csr.unibo.it/fvc2002/

[124] V. J. a. A. Mukherjee. (2002). *The Indian Face Databa*. Available: http://www.cs.umass.edu/~vidit/IndianFaceDatabase

[125] C. E. Thomaz. (2006). *FEI Face Database*. Available: https://fei.edu.br/~cet/facedatabase.html

[126] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, and A. K. Jain, "FVC2004: Third fingerprint verification competition," in *International Conference on Biometric Authentication*, 2004, pp. 1-7.

[127] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265-283.

[128] M. Turčaník and M. Javurek, "Hash function generation by neural network," in *2016 New Trends in Signal Processing (NTSP)*, 2016, pp. 1-5.

[129] M. Turčaník, "Hash function generation based on neural networks and chaotic maps," in *2017 Communication and Information Technologies (KIT)*, 2017, pp. 1-5.

[130] N. Abdoun, S. El Assad, R. Assaf, O. Déforges, M. Khalil, and S. Belghith, "Design and implementation of robust Keyed Hash functions based on Chaotic Neural Network," 2018.

Appendix A
Structure of the FaceNet Neural Network

Layer Name	Layer Type	Output Shape	Param#	Connected To
input_1	InputLayer	(None, 160, 160, 3)	0	
Conv2d_1a_3x3	Conv2D	(None, 79, 79, 32)	864	input_1
Conv2d_1a_3x3_BatchNorm	BatchNormalization	(None, 79, 79, 32)	96	Conv2d_1a_3x3
Conv2d_1a_3x3_Activation	Activation	(None, 79, 79, 32)	0	Conv2d_1a_3x3_BatchNorm
Conv2d_2a_3x3	Conv2D	(None, 77, 77, 32)	9216	Conv2d_1a_3x3_Activation
Conv2d_2a_3x3_BatchNorm	BatchNormalization	(None, 77, 77, 32)	96	Conv2d_2a_3x3
Conv2d_2a_3x3_Activation	Activation	(None, 77, 77, 32)	0	Conv2d_2a_3x3_BatchNorm
Conv2d_2b_3x3	Conv2D	(None, 77, 77, 64)	18432	Conv2d_2a_3x3_Activation
Conv2d_2b_3x3_BatchNorm	BatchNormalization	(None, 77, 77, 64)	192	Conv2d_2b_3x3
Conv2d_2b_3x3_Activation	Activation	(None, 77, 77, 64)	0	Conv2d_2b_3x3_BatchNorm
MaxPool_3a_3x3	MaxPooling2D	(None, 38, 38, 64)	0	Conv2d_2b_3x3_Activation
Conv2d_3b_1x1	Conv2D	(None, 38, 38, 80)	5120	MaxPool_3a_3x3
Conv2d_3b_1x1_BatchNorm	BatchNormalization	(None, 38, 38, 80)	240	Conv2d_3b_1x1
Conv2d_3b_1x1_Activation	Activation	(None, 38, 38, 80)	0	Conv2d_3b_1x1_BatchNorm
Conv2d_4a_3x3	Conv2D	(None, 36, 36, 192)	138240	Conv2d_3b_1x1_Activation
Conv2d_4a_3x3_BatchNorm	BatchNormalization	(None, 36, 36, 192)	576	Conv2d_4a_3x3
Conv2d_4a_3x3_Activation	Activation	(None, 36, 36, 192)	0	Conv2d_4a_3x3_BatchNorm
Conv2d_4b_3x3	Conv2D	(None, 17, 17, 256)	442368	Conv2d_4a_3x3_Activation
Conv2d_4b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 256)	768	Conv2d_4b_3x3
Conv2d_4b_3x3_Activation	Activation	(None, 17, 17, 256)	0	Conv2d_4b_3x3_BatchNorm
Block35_1_Branch_2_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 32)	8192	Conv2d_4b_3x3_Activation
Block35_1_Branch_2_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_1_Branch_2_Conv2d_0a_1x1
Block35_1_Branch_2_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_1_Branch_2_Conv2d_0a_1x1_BatchNorm
Block35_1_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 32)	8192	Conv2d_4b_3x3_Activation
Block35_1_Branch_2_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_1_Branch_2_Conv2d_0a_1x1_Activation

Block35_1_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_1_Branch_1_Conv2d_0a_1x1
Block35_1_Branch_2_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_1_Branch_2_Conv2d_0b_3x3
Block35_1_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_1_Branch_1_Conv2d_0a_1x1_BatchNorm
Block35_1_Branch_2_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_1_Branch_2_Conv2d_0b_3x3_BatchNorm
Block35_1_Branch_0_Conv2d_1x1	Conv2D	(None, 17, 17, 32)	8192	Conv2d_4b_3x3_Activation
Block35_1_Branch_1_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_1_Branch_1_Conv2d_0a_1x1_Activation
Block35_1_Branch_2_Conv2d_0c_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_1_Branch_2_Conv2d_0b_3x3_Activation
Block35_1_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_1_Branch_0_Conv2d_1x1
Block35_1_Branch_1_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_1_Branch_1_Conv2d_0b_3x3
Block35_1_Branch_2_Conv2d_0c_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_1_Branch_2_Conv2d_0c_3x3
Block35_1_Branch_0_Conv2d_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_1_Branch_0_Conv2d_1x1_BatchNorm
Block35_1_Branch_1_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_1_Branch_1_Conv2d_0b_3x3_BatchNorm
Block35_1_Branch_2_Conv2d_0c_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_1_Branch_2_Conv2d_0c_3x3_BatchNorm
Block35_1_Concatenate	Concatenate	(None, 17, 17, 96)	0	Block35_1_Branch_0_Conv2d_1x1_Activation, Block35_1_Branch_1_Conv2d_0b_3x3_Activation, Block35_1_Branch_2_Conv2d_0c_3x3_Activation
Block35_1_Conv2d_1x1	Conv2D	(None, 17, 17, 256)	24832	Block35_1_Concatenate
lambda	Lambda	(None, 17, 17, 256)	0	Block35_1_Conv2d_1x1
add	Add	(None, 17, 17, 256)	0	Conv2d_4b_3x3_Activation, lambda
Block35_1_Activation	Activation	(None, 17, 17, 256)	0	add

Block35_2_Branch_2_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_1_Activation
Block35_2_Branch_2_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_2_Branch_2_Conv2d_0a_1x1
Block35_2_Branch_2_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_2_Branch_2_Conv2d_0a_1x1_BatchNorm
Block35_2_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_1_Activation
Block35_2_Branch_2_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_2_Branch_2_Conv2d_0a_1x1_Activation
Block35_2_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_2_Branch_1_Conv2d_0a_1x1
Block35_2_Branch_2_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_2_Branch_2_Conv2d_0b_3x3
Block35_2_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_2_Branch_1_Conv2d_0a_1x1_BatchNorm
Block35_2_Branch_2_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_2_Branch_2_Conv2d_0b_3x3_BatchNorm
Block35_2_Branch_0_Conv2d_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_1_Activation
Block35_2_Branch_1_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_2_Branch_1_Conv2d_0a_1x1_Activation
Block35_2_Branch_2_Conv2d_0c_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_2_Branch_2_Conv2d_0b_3x3_Activation
Block35_2_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_2_Branch_0_Conv2d_1x1
Block35_2_Branch_1_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_2_Branch_1_Conv2d_0b_3x3
Block35_2_Branch_2_Conv2d_0c_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_2_Branch_2_Conv2d_0c_3x3
Block35_2_Branch_0_Conv2d_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_2_Branch_0_Conv2d_1x1_BatchNorm
Block35_2_Branch_1_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_2_Branch_1_Conv2d_0b_3x3_BatchNorm

Block35_2_Branch_2_Conv2d_0c_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_2_Branch_2_Conv2d_0c_3x3_BatchNorm
Block35_2_Concatenate	Concatenate	(None, 17, 17, 96)	0	Block35_2_Branch_0_Conv2d_1x1_Activation, Block35_2_Branch_1_Conv2d_0b_3x3_Activation, Block35_2_Branch_2_Conv2d_0c_3x3_Activation
Block35_2_Conv2d_1x1	Conv2D	(None, 17, 17, 256)	24832	Block35_2_Concatenate
lambda_1	Lambda	(None, 17, 17, 256)	0	Block35_2_Conv2d_1x1
add_1	Add	(None, 17, 17, 256)	0	Block35_1_Activation, lambda_1
Block35_2_Activation	Activation	(None, 17, 17, 256)	0	add_1
Block35_3_Branch_2_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_2_Activation
Block35_3_Branch_2_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_3_Branch_2_Conv2d_0a_1x1
Block35_3_Branch_2_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_3_Branch_2_Conv2d_0a_1x1_BatchNorm
Block35_3_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_2_Activation
Block35_3_Branch_2_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_3_Branch_2_Conv2d_0a_1x1_Activation
Block35_3_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_3_Branch_1_Conv2d_0a_1x1
Block35_3_Branch_2_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_3_Branch_2_Conv2d_0b_3x3
Block35_3_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_3_Branch_1_Conv2d_0a_1x1_BatchNorm
Block35_3_Branch_2_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_3_Branch_2_Conv2d_0b_3x3_BatchNorm
Block35_3_Branch_0_Conv2d_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_2_Activation
Block35_3_Branch_1_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_3_Branch_1_Conv2d_0a_1x1_Activation
Block35_3_Branch_2_Conv2d_0c_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_3_Branch_2_Conv2d_0b_3x3_Activation

Block35_3_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_3_Branch_0_Conv2d_1x1
Block35_3_Branch_1_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_3_Branch_1_Conv2d_0b_3x3
Block35_3_Branch_2_Conv2d_0c_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_3_Branch_2_Conv2d_0c_3x3
Block35_3_Branch_0_Conv2d_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_3_Branch_0_Conv2d_1x1_BatchNorm
Block35_3_Branch_1_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_3_Branch_1_Conv2d_0b_3x3_BatchNorm
Block35_3_Branch_2_Conv2d_0c_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_3_Branch_2_Conv2d_0c_3x3_BatchNorm
Block35_3_Concatenate	Concatenate	(None, 17, 17, 96)	0	Block35_3_Branch_0_Conv2d_1x1_Activation, Block35_3_Branch_1_Conv2d_0b_3x3_Activation, Block35_3_Branch_2_Conv2d_0c_3x3_Activation
Block35_3_Conv2d_1x1	Conv2D	(None, 17, 17, 256)	24832	Block35_3_Concatenate
lambda_2	Lambda	(None, 17, 17, 256)	0	Block35_3_Conv2d_1x1
add_2	Add	(None, 17, 17, 256)	0	Block35_2_Activation, lambda_2
Block35_3_Activation	Activation	(None, 17, 17, 256)	0	add_2
Block35_4_Branch_2_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_3_Activation
Block35_4_Branch_2_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_4_Branch_2_Conv2d_0a_1x1
Block35_4_Branch_2_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_4_Branch_2_Conv2d_0a_1x1_BatchNorm
Block35_4_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_3_Activation
Block35_4_Branch_2_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_4_Branch_2_Conv2d_0a_1x1_Activation
Block35_4_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_4_Branch_1_Conv2d_0a_1x1
Block35_4_Branch_2_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_4_Branch_2_Conv2d_0b_3x3

Block35_4_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_4_Branch_1_Conv2d_0a_1x1_BatchNorm
Block35_4_Branch_2_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_4_Branch_2_Conv2d_0b_3x3_BatchNorm
Block35_4_Branch_0_Conv2d_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_3_Activation
Block35_4_Branch_1_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_4_Branch_1_Conv2d_0a_1x1_Activation
Block35_4_Branch_2_Conv2d_0c_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_4_Branch_2_Conv2d_0b_3x3_Activation
Block35_4_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_4_Branch_0_Conv2d_1x1
Block35_4_Branch_1_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_4_Branch_1_Conv2d_0b_3x3
Block35_4_Branch_2_Conv2d_0c_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_4_Branch_2_Conv2d_0c_3x3
Block35_4_Branch_0_Conv2d_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_4_Branch_0_Conv2d_1x1_BatchNorm
Block35_4_Branch_1_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_4_Branch_1_Conv2d_0b_3x3_BatchNorm
Block35_4_Branch_2_Conv2d_0c_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_4_Branch_2_Conv2d_0c_3x3_BatchNorm
Block35_4_Concatenate	Concatenate	(None, 17, 17, 96)	0	Block35_4_Branch_0_Conv2d_1x1_Activation, Block35_4_Branch_1_Conv2d_0b_3x3_Activation, Block35_4_Branch_2_Conv2d_0c_3x3_Activation
Block35_4_Conv2d_1x1	Conv2D	(None, 17, 17, 256)	24832	Block35_4_Concatenate
lambda_3	Lambda	(None, 17, 17, 256)	0	Block35_4_Conv2d_1x1
add_3	Add	(None, 17, 17, 256)	0	Block35_3_Activation, lambda_3
Block35_4_Activation	Activation	(None, 17, 17, 256)	0	add_3
Block35_5_Branch_2_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_4_Activation
Block35_5_Branch_2_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_5_Branch_2_Conv2d_0a_1x1

Block35_5_Branch_2_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_5_Branch_2_Conv2d_0a_1x1_BatchNorm
Block35_5_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_4_Activation
Block35_5_Branch_2_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_5_Branch_2_Conv2d_0a_1x1_Activation
Block35_5_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_5_Branch_1_Conv2d_0a_1x1
Block35_5_Branch_2_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_5_Branch_2_Conv2d_0b_3x3
Block35_5_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_5_Branch_1_Conv2d_0a_1x1_BatchNorm
Block35_5_Branch_2_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_5_Branch_2_Conv2d_0b_3x3_BatchNorm
Block35_5_Branch_0_Conv2d_1x1	Conv2D	(None, 17, 17, 32)	8192	Block35_4_Activation
Block35_5_Branch_1_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_5_Branch_1_Conv2d_0a_1x1_Activation
Block35_5_Branch_2_Conv2d_0c_3x3	Conv2D	(None, 17, 17, 32)	9216	Block35_5_Branch_2_Conv2d_0b_3x3_Activation
Block35_5_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_5_Branch_0_Conv2d_1x1
Block35_5_Branch_1_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_5_Branch_1_Conv2d_0b_3x3
Block35_5_Branch_2_Conv2d_0c_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 32)	96	Block35_5_Branch_2_Conv2d_0c_3x3
Block35_5_Branch_0_Conv2d_1x1_Activation	Activation	(None, 17, 17, 32)	0	Block35_5_Branch_0_Conv2d_1x1_BatchNorm
Block35_5_Branch_1_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_5_Branch_1_Conv2d_0b_3x3_BatchNorm
Block35_5_Branch_2_Conv2d_0c_3x3_Activation	Activation	(None, 17, 17, 32)	0	Block35_5_Branch_2_Conv2d_0c_3x3_BatchNorm
Block35_5_Concatenate	Concatenate	(None, 17, 17, 96)	0	Block35_5_Branch_0_Conv2d_1x1_Activation, Block35_5_Branch_1_Conv2d_0b_3x3_Activation,

				Block35_5_Branch_2_Conv2d_0c_3x3_Activation
Block35_5_Conv2d_1x1	Conv2D	(None, 17, 17, 256)	24832	Block35_5_Concatenate
lambda_4	Lambda	(None, 17, 17, 256)	0	Block35_5_Conv2d_1x1
add_4	Add	(None, 17, 17, 256)	0	Block35_4_Activation, lambda_4
Block35_5_Activation	Activation	(None, 17, 17, 256)	0	add_4
Mixed_6a_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 17, 17, 192)	49152	Block35_5_Activation
Mixed_6a_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 17, 17, 192)	576	Mixed_6a_Branch_1_Conv2d_0a_1x1
Mixed_6a_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 17, 17, 192)	0	Mixed_6a_Branch_1_Conv2d_0a_1x1_BatchNorm
Mixed_6a_Branch_1_Conv2d_0b_3x3	Conv2D	(None, 17, 17, 192)	331776	Mixed_6a_Branch_1_Conv2d_0a_1x1_Activation
Mixed_6a_Branch_1_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 17, 17, 192)	576	Mixed_6a_Branch_1_Conv2d_0b_3x3
Mixed_6a_Branch_1_Conv2d_0b_3x3_Activation	Activation	(None, 17, 17, 192)	0	Mixed_6a_Branch_1_Conv2d_0b_3x3_BatchNorm
Mixed_6a_Branch_0_Conv2d_1a_3x3	Conv2D	(None, 8, 8, 384)	884736	Block35_5_Activation
Mixed_6a_Branch_1_Conv2d_1a_3x3	Conv2D	(None, 8, 8, 256)	442368	Mixed_6a_Branch_1_Conv2d_0b_3x3_Activation
Mixed_6a_Branch_0_Conv2d_1a_3x3_BatchNorm	BatchNormalization	(None, 8, 8, 384)	1152	Mixed_6a_Branch_0_Conv2d_1a_3x3
Mixed_6a_Branch_1_Conv2d_1a_3x3_BatchNorm	BatchNormalization	(None, 8, 8, 256)	768	Mixed_6a_Branch_1_Conv2d_1a_3x3
Mixed_6a_Branch_0_Conv2d_1a_3x3_Activation	Activation	(None, 8, 8, 384)	0	Mixed_6a_Branch_0_Conv2d_1a_3x3_BatchNorm
Mixed_6a_Branch_1_Conv2d_1a_3x3_Activation	Activation	(None, 8, 8, 256)	0	Mixed_6a_Branch_1_Conv2d_1a_3x3_BatchNorm
Mixed_6a_Branch_2_MaxPool_1a_3x3	MaxPooling2D	(None, 8, 8, 256)	0	Block35_5_Activation
Mixed_6a	Concatenate	(None, 8, 8, 896)	0	Mixed_6a_Branch_0_Conv2d_1a_3x3_Activation, Mixed_6a_Branch_1_Conv2d_1a_3x3_Activation, Mixed_6a_Branch_2_MaxPool_1a_3x3

Block17_1_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 128)	114688	Mixed_6a
Block17_1_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_1_Branch_1_Conv2d_0a_1x1
Block17_1_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_1_Branch_1_Conv2d_0a_1x1_BatchNorm
Block17_1_Branch_1_Conv2d_0b_1x7	Conv2D	(None, 8, 8, 128)	114688	Block17_1_Branch_1_Conv2d_0a_1x1_Activation
Block17_1_Branch_1_Conv2d_0b_1x7_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_1_Branch_1_Conv2d_0b_1x7
Block17_1_Branch_1_Conv2d_0b_1x7_Activation	Activation	(None, 8, 8, 128)	0	Block17_1_Branch_1_Conv2d_0b_1x7_BatchNorm
Block17_1_Branch_0_Conv2d_1x1	Conv2D	(None, 8, 8, 128)	114688	Mixed_6a
Block17_1_Branch_1_Conv2d_0c_7x1	Conv2D	(None, 8, 8, 128)	114688	Block17_1_Branch_1_Conv2d_0b_1x7_Activation
Block17_1_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_1_Branch_0_Conv2d_1x1
Block17_1_Branch_1_Conv2d_0c_7x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_1_Branch_1_Conv2d_0c_7x1
Block17_1_Branch_0_Conv2d_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_1_Branch_0_Conv2d_1x1_BatchNorm
Block17_1_Branch_1_Conv2d_0c_7x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_1_Branch_1_Conv2d_0c_7x1_BatchNorm
Block17_1_Concatenate	Concatenate	(None, 8, 8, 256)	0	Block17_1_Branch_0_Conv2d_1x1_Activation, Block17_1_Branch_1_Conv2d_0c_7x1_Activation
Block17_1_Conv2d_1x1	Conv2D	(None, 8, 8, 896)	230272	Block17_1_Concatenate
lambda_5	Lambda	(None, 8, 8, 896)	0	Block17_1_Conv2d_1x1
add_5	Add	(None, 8, 8, 896)	0	Mixed_6a, lambda_5
Block17_1_Activation	Activation	(None, 8, 8, 896)	0	add_5
Block17_2_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_1_Activation
Block17_2_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_2_Branch_1_Conv2d_0a_1x1

Block17_2_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_2_Branch_1_Conv2d_0a_1x1_BatchNorm
Block17_2_Branch_1_Conv2d_0b_1x7	Conv2D	(None, 8, 8, 128)	114688	Block17_2_Branch_1_Conv2d_0a_1x1_Activation
Block17_2_Branch_1_Conv2d_0b_1x7_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_2_Branch_1_Conv2d_0b_1x7
Block17_2_Branch_1_Conv2d_0b_1x7_Activation	Activation	(None, 8, 8, 128)	0	Block17_2_Branch_1_Conv2d_0b_1x7_BatchNorm
Block17_2_Branch_0_Conv2d_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_1_Activation
Block17_2_Branch_1_Conv2d_0c_7x1	Conv2D	(None, 8, 8, 128)	114688	Block17_2_Branch_1_Conv2d_0b_1x7_Activation
Block17_2_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_2_Branch_0_Conv2d_1x1
Block17_2_Branch_1_Conv2d_0c_7x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_2_Branch_1_Conv2d_0c_7x1
Block17_2_Branch_0_Conv2d_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_2_Branch_0_Conv2d_1x1_BatchNorm
Block17_2_Branch_1_Conv2d_0c_7x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_2_Branch_1_Conv2d_0c_7x1_BatchNorm
Block17_2_Concatenate	Concatenate	(None, 8, 8, 256)	0	Block17_2_Branch_0_Conv2d_1x1_Activation, Block17_2_Branch_1_Conv2d_0c_7x1_Activation
Block17_2_Conv2d_1x1	Conv2D	(None, 8, 8, 896)	230272	Block17_2_Concatenate
lambda_6	Lambda	(None, 8, 8, 896)	0	Block17_2_Conv2d_1x1
add_6	Add	(None, 8, 8, 896)	0	Block17_1_Activation, lambda_6
Block17_2_Activation	Activation	(None, 8, 8, 896)	0	add_6
Block17_3_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_2_Activation
Block17_3_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_3_Branch_1_Conv2d_0a_1x1
Block17_3_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_3_Branch_1_Conv2d_0a_1x1_BatchNorm
Block17_3_Branch_1_Conv2d_0b_1x7	Conv2D	(None, 8, 8, 128)	114688	Block17_3_Branch_1_Conv2d_0a_1x1_Activation

Block17_3_Branch_1_Conv2d_0b_1x7_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_3_Branch_1_Conv2d_0b_1x7
Block17_3_Branch_1_Conv2d_0b_1x7_Activation	Activation	(None, 8, 8, 128)	0	Block17_3_Branch_1_Conv2d_0b_1x7_BatchNorm
Block17_3_Branch_0_Conv2d_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_2_Activation
Block17_3_Branch_1_Conv2d_0c_7x1	Conv2D	(None, 8, 8, 128)	114688	Block17_3_Branch_1_Conv2d_0b_1x7_Activation
Block17_3_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_3_Branch_0_Conv2d_1x1
Block17_3_Branch_1_Conv2d_0c_7x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_3_Branch_1_Conv2d_0c_7x1
Block17_3_Branch_0_Conv2d_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_3_Branch_0_Conv2d_1x1_BatchNorm
Block17_3_Branch_1_Conv2d_0c_7x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_3_Branch_1_Conv2d_0c_7x1_BatchNorm
Block17_3_Concatenate	Concatenate	(None, 8, 8, 256)	0	Block17_3_Branch_0_Conv2d_1x1_Activation, Block17_3_Branch_1_Conv2d_0c_7x1_Activation
Block17_3_Conv2d_1x1	Conv2D	(None, 8, 8, 896)	230272	Block17_3_Concatenate
lambda_7	Lambda	(None, 8, 8, 896)	0	Block17_3_Conv2d_1x1
add_7	Add	(None, 8, 8, 896)	0	Block17_2_Activation, lambda_7
Block17_3_Activation	Activation	(None, 8, 8, 896)	0	add_7
Block17_4_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_3_Activation
Block17_4_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_4_Branch_1_Conv2d_0a_1x1
Block17_4_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_4_Branch_1_Conv2d_0a_1x1_BatchNorm
Block17_4_Branch_1_Conv2d_0b_1x7	Conv2D	(None, 8, 8, 128)	114688	Block17_4_Branch_1_Conv2d_0a_1x1_Activation
Block17_4_Branch_1_Conv2d_0b_1x7_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_4_Branch_1_Conv2d_0b_1x7
Block17_4_Branch_1_Conv2d_0b_1x7_Activation	Activation	(None, 8, 8, 128)	0	Block17_4_Branch_1_Conv2d_0b_1x7_BatchNorm

Block17_4_Branch_0_Conv2d_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_3_Activation
Block17_4_Branch_1_Conv2d_0c_7x1	Conv2D	(None, 8, 8, 128)	114688	Block17_4_Branch_1_Conv2d_0b_1x7_Activation
Block17_4_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_4_Branch_0_Conv2d_1x1
Block17_4_Branch_1_Conv2d_0c_7x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_4_Branch_1_Conv2d_0c_7x1
Block17_4_Branch_0_Conv2d_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_4_Branch_0_Conv2d_1x1_BatchNorm
Block17_4_Branch_1_Conv2d_0c_7x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_4_Branch_1_Conv2d_0c_7x1_BatchNorm
Block17_4_Concatenate	Concatenate	(None, 8, 8, 256)	0	Block17_4_Branch_0_Conv2d_1x1_Activation, Block17_4_Branch_1_Conv2d_0c_7x1_Activation
Block17_4_Conv2d_1x1	Conv2D	(None, 8, 8, 896)	230272	Block17_4_Concatenate
lambda_8	Lambda	(None, 8, 8, 896)	0	Block17_4_Conv2d_1x1
add_8	Add	(None, 8, 8, 896)	0	Block17_3_Activation, lambda_8
Block17_4_Activation	Activation	(None, 8, 8, 896)	0	add_8
Block17_5_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_4_Activation
Block17_5_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_5_Branch_1_Conv2d_0a_1x1
Block17_5_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_5_Branch_1_Conv2d_0a_1x1_BatchNorm
Block17_5_Branch_1_Conv2d_0b_1x7	Conv2D	(None, 8, 8, 128)	114688	Block17_5_Branch_1_Conv2d_0a_1x1_Activation
Block17_5_Branch_1_Conv2d_0b_1x7_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_5_Branch_1_Conv2d_0b_1x7
Block17_5_Branch_1_Conv2d_0b_1x7_Activation	Activation	(None, 8, 8, 128)	0	Block17_5_Branch_1_Conv2d_0b_1x7_BatchNorm
Block17_5_Branch_0_Conv2d_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_4_Activation
Block17_5_Branch_1_Conv2d_0c_7x1	Conv2D	(None, 8, 8, 128)	114688	Block17_5_Branch_1_Conv2d_0b_1x7_Activation

Block17_5_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_5_Branch_0_Conv2d_1x1
Block17_5_Branch_1_Conv2d_0c_7x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_5_Branch_1_Conv2d_0c_7x1
Block17_5_Branch_0_Conv2d_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_5_Branch_0_Conv2d_1x1_BatchNorm
Block17_5_Branch_1_Conv2d_0c_7x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_5_Branch_1_Conv2d_0c_7x1_BatchNorm
Block17_5_Concatenate	Concatenate	(None, 8, 8, 256)	0	Block17_5_Branch_0_Conv2d_1x1_Activation, Block17_5_Branch_1_Conv2d_0c_7x1_Activation
Block17_5_Conv2d_1x1	Conv2D	(None, 8, 8, 896)	230272	Block17_5_Concatenate
lambda_9	Lambda	(None, 8, 8, 896)	0	Block17_5_Conv2d_1x1
add_9	Add	(None, 8, 8, 896)	0	Block17_4_Activation, lambda_9
Block17_5_Activation	Activation	(None, 8, 8, 896)	0	add_9
Block17_6_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_5_Activation
Block17_6_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_6_Branch_1_Conv2d_0a_1x1
Block17_6_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_6_Branch_1_Conv2d_0a_1x1_BatchNorm
Block17_6_Branch_1_Conv2d_0b_1x7	Conv2D	(None, 8, 8, 128)	114688	Block17_6_Branch_1_Conv2d_0a_1x1_Activation
Block17_6_Branch_1_Conv2d_0b_1x7_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_6_Branch_1_Conv2d_0b_1x7
Block17_6_Branch_1_Conv2d_0b_1x7_Activation	Activation	(None, 8, 8, 128)	0	Block17_6_Branch_1_Conv2d_0b_1x7_BatchNorm
Block17_6_Branch_0_Conv2d_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_5_Activation
Block17_6_Branch_1_Conv2d_0c_7x1	Conv2D	(None, 8, 8, 128)	114688	Block17_6_Branch_1_Conv2d_0b_1x7_Activation
Block17_6_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_6_Branch_0_Conv2d_1x1
Block17_6_Branch_1_Conv2d_0c_7x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_6_Branch_1_Conv2d_0c_7x1

Block17_6_Branch_0_Conv2d_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_6_Branch_0_Conv2d_1x1_BatchNorm
Block17_6_Branch_1_Conv2d_0c_7x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_6_Branch_1_Conv2d_0c_7x1_BatchNorm
Block17_6_Concatenate	Concatenate	(None, 8, 8, 256)	0	Block17_6_Branch_0_Conv2d_1x1_Activation, Block17_6_Branch_1_Conv2d_0c_7x1_Activation
Block17_6_Conv2d_1x1	Conv2D	(None, 8, 8, 896)	230272	Block17_6_Concatenate
lambda_10	Lambda	(None, 8, 8, 896)	0	Block17_6_Conv2d_1x1
add_10	Add	(None, 8, 8, 896)	0	Block17_5_Activation, lambda_10
Block17_6_Activation	Activation	(None, 8, 8, 896)	0	add_10
Block17_7_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_6_Activation
Block17_7_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_7_Branch_1_Conv2d_0a_1x1
Block17_7_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_7_Branch_1_Conv2d_0a_1x1_BatchNorm
Block17_7_Branch_1_Conv2d_0b_1x7	Conv2D	(None, 8, 8, 128)	114688	Block17_7_Branch_1_Conv2d_0a_1x1_Activation
Block17_7_Branch_1_Conv2d_0b_1x7_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_7_Branch_1_Conv2d_0b_1x7
Block17_7_Branch_1_Conv2d_0b_1x7_Activation	Activation	(None, 8, 8, 128)	0	Block17_7_Branch_1_Conv2d_0b_1x7_BatchNorm
Block17_7_Branch_0_Conv2d_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_6_Activation
Block17_7_Branch_1_Conv2d_0c_7x1	Conv2D	(None, 8, 8, 128)	114688	Block17_7_Branch_1_Conv2d_0b_1x7_Activation
Block17_7_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_7_Branch_0_Conv2d_1x1
Block17_7_Branch_1_Conv2d_0c_7x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_7_Branch_1_Conv2d_0c_7x1
Block17_7_Branch_0_Conv2d_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_7_Branch_0_Conv2d_1x1_BatchNorm
Block17_7_Branch_1_Conv2d_0c_7x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_7_Branch_1_Conv2d_0c_7x1_BatchNorm

Block17_7_Concatenate	Concatenate	(None, 8, 8, 256)	0	Block17_7_Branch_0_Conv2d_1x1_Activation, Block17_7_Branch_1_Conv2d_0c_7x1_Activation
Block17_7_Conv2d_1x1	Conv2D	(None, 8, 8, 896)	230272	Block17_7_Concatenate
lambda_11	Lambda	(None, 8, 8, 896)	0	Block17_7_Conv2d_1x1
add_11	Add	(None, 8, 8, 896)	0	Block17_6_Activation, lambda_11
Block17_7_Activation	Activation	(None, 8, 8, 896)	0	add_11
Block17_8_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_7_Activation
Block17_8_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_8_Branch_1_Conv2d_0a_1x1
Block17_8_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_8_Branch_1_Conv2d_0a_1x1_BatchNorm
Block17_8_Branch_1_Conv2d_0b_1x7	Conv2D	(None, 8, 8, 128)	114688	Block17_8_Branch_1_Conv2d_0a_1x1_Activation
Block17_8_Branch_1_Conv2d_0b_1x7_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_8_Branch_1_Conv2d_0b_1x7
Block17_8_Branch_1_Conv2d_0b_1x7_Activation	Activation	(None, 8, 8, 128)	0	Block17_8_Branch_1_Conv2d_0b_1x7_BatchNorm
Block17_8_Branch_0_Conv2d_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_7_Activation
Block17_8_Branch_1_Conv2d_0c_7x1	Conv2D	(None, 8, 8, 128)	114688	Block17_8_Branch_1_Conv2d_0b_1x7_Activation
Block17_8_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_8_Branch_0_Conv2d_1x1
Block17_8_Branch_1_Conv2d_0c_7x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_8_Branch_1_Conv2d_0c_7x1
Block17_8_Branch_0_Conv2d_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_8_Branch_0_Conv2d_1x1_BatchNorm
Block17_8_Branch_1_Conv2d_0c_7x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_8_Branch_1_Conv2d_0c_7x1_BatchNorm
Block17_8_Concatenate	Concatenate	(None, 8, 8, 256)	0	Block17_8_Branch_0_Conv2d_1x1_Activation, Block17_8_Branch_1_Conv2d_0c_7x1_Activation
Block17_8_Conv2d_1x1	Conv2D	(None, 8, 8, 896)	230272	Block17_8_Concatenate

lambda_12	Lambda	(None, 8, 8, 896)	0	Block17_8_Conv2d_1x1
add_12	Add	(None, 8, 8, 896)	0	Block17_7_Activation, lambda_12
Block17_8_Activation	Activation	(None, 8, 8, 896)	0	add_12
Block17_9_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_8_Activation
Block17_9_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_9_Branch_1_Conv2d_0a_1x1
Block17_9_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_9_Branch_1_Conv2d_0a_1x1_BatchNorm
Block17_9_Branch_1_Conv2d_0b_1x7	Conv2D	(None, 8, 8, 128)	114688	Block17_9_Branch_1_Conv2d_0a_1x1_Activation
Block17_9_Branch_1_Conv2d_0b_1x7_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_9_Branch_1_Conv2d_0b_1x7
Block17_9_Branch_1_Conv2d_0b_1x7_Activation	Activation	(None, 8, 8, 128)	0	Block17_9_Branch_1_Conv2d_0b_1x7_BatchNorm
Block17_9_Branch_0_Conv2d_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_8_Activation
Block17_9_Branch_1_Conv2d_0c_7x1	Conv2D	(None, 8, 8, 128)	114688	Block17_9_Branch_1_Conv2d_0b_1x7_Activation
Block17_9_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_9_Branch_0_Conv2d_1x1
Block17_9_Branch_1_Conv2d_0c_7x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_9_Branch_1_Conv2d_0c_7x1
Block17_9_Branch_0_Conv2d_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_9_Branch_0_Conv2d_1x1_BatchNorm
Block17_9_Branch_1_Conv2d_0c_7x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_9_Branch_1_Conv2d_0c_7x1_BatchNorm
Block17_9_Concatenate	Concatenate	(None, 8, 8, 256)	0	Block17_9_Branch_0_Conv2d_1x1_Activation, Block17_9_Branch_1_Conv2d_0c_7x1_Activation
Block17_9_Conv2d_1x1	Conv2D	(None, 8, 8, 896)	230272	Block17_9_Concatenate
lambda_13	Lambda	(None, 8, 8, 896)	0	Block17_9_Conv2d_1x1
add_13	Add	(None, 8, 8, 896)	0	Block17_8_Activation, lambda_13
Block17_9_Activation	Activation	(None, 8, 8, 896)	0	add_13

Block17_10_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_9_Activation
Block17_10_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_10_Branch_1_Conv2d_0a_1x1
Block17_10_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_10_Branch_1_Conv2d_0a_1x1_BatchNorm
Block17_10_Branch_1_Conv2d_0b_1x7	Conv2D	(None, 8, 8, 128)	114688	Block17_10_Branch_1_Conv2d_0a_1x1_Activation
Block17_10_Branch_1_Conv2d_0b_1x7_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_10_Branch_1_Conv2d_0b_1x7
Block17_10_Branch_1_Conv2d_0b_1x7_Activation	Activation	(None, 8, 8, 128)	0	Block17_10_Branch_1_Conv2d_0b_1x7_BatchNorm
Block17_10_Branch_0_Conv2d_1x1	Conv2D	(None, 8, 8, 128)	114688	Block17_9_Activation
Block17_10_Branch_1_Conv2d_0c_7x1	Conv2D	(None, 8, 8, 128)	114688	Block17_10_Branch_1_Conv2d_0b_1x7_Activation
Block17_10_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_10_Branch_0_Conv2d_1x1
Block17_10_Branch_1_Conv2d_0c_7x1_BatchNorm	BatchNormalization	(None, 8, 8, 128)	384	Block17_10_Branch_1_Conv2d_0c_7x1
Block17_10_Branch_0_Conv2d_1x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_10_Branch_0_Conv2d_1x1_BatchNorm
Block17_10_Branch_1_Conv2d_0c_7x1_Activation	Activation	(None, 8, 8, 128)	0	Block17_10_Branch_1_Conv2d_0c_7x1_BatchNorm
Block17_10_Concatenate	Concatenate	(None, 8, 8, 256)	0	Block17_10_Branch_0_Conv2d_1x1_Activation, Block17_10_Branch_1_Conv2d_0c_7x1_Activation
Block17_10_Conv2d_1x1	Conv2D	(None, 8, 8, 896)	230272	Block17_10_Concatenate
lambda_14	Lambda	(None, 8, 8, 896)	0	Block17_10_Conv2d_1x1
add_14	Add	(None, 8, 8, 896)	0	Block17_9_Activation, lambda_14
Block17_10_Activation	Activation	(None, 8, 8, 896)	0	add_14
Mixed_7a_Branch_2_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 256)	229376	Block17_10_Activation
Mixed_7a_Branch_2_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 256)	768	Mixed_7a_Branch_2_Conv2d_0a_1x1

Mixed_7a_Branch_2_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 256)	0	Mixed_7a_Branch_2_Conv2d_0a_1x1_BatchNorm
Mixed_7a_Branch_0_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 256)	229376	Block17_10_Activation
Mixed_7a_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 8, 8, 256)	229376	Block17_10_Activation
Mixed_7a_Branch_2_Conv2d_0b_3x3	Conv2D	(None, 8, 8, 256)	589824	Mixed_7a_Branch_2_Conv2d_0a_1x1_Activation
Mixed_7a_Branch_0_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 256)	768	Mixed_7a_Branch_0_Conv2d_0a_1x1
Mixed_7a_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 8, 8, 256)	768	Mixed_7a_Branch_1_Conv2d_0a_1x1
Mixed_7a_Branch_2_Conv2d_0b_3x3_BatchNorm	BatchNormalization	(None, 8, 8, 256)	768	Mixed_7a_Branch_2_Conv2d_0b_3x3
Mixed_7a_Branch_0_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 256)	0	Mixed_7a_Branch_0_Conv2d_0a_1x1_BatchNorm
Mixed_7a_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 8, 8, 256)	0	Mixed_7a_Branch_1_Conv2d_0a_1x1_BatchNorm
Mixed_7a_Branch_2_Conv2d_0b_3x3_Activation	Activation	(None, 8, 8, 256)	0	Mixed_7a_Branch_2_Conv2d_0b_3x3_BatchNorm
Mixed_7a_Branch_0_Conv2d_1a_3x3	Conv2D	(None, 3, 3, 384)	884736	Mixed_7a_Branch_0_Conv2d_0a_1x1_Activation
Mixed_7a_Branch_1_Conv2d_1a_3x3	Conv2D	(None, 3, 3, 256)	589824	Mixed_7a_Branch_1_Conv2d_0a_1x1_Activation
Mixed_7a_Branch_2_Conv2d_1a_3x3	Conv2D	(None, 3, 3, 256)	589824	Mixed_7a_Branch_2_Conv2d_0b_3x3_Activation
Mixed_7a_Branch_0_Conv2d_1a_3x3_BatchNorm	BatchNormalization	(None, 3, 3, 384)	1152	Mixed_7a_Branch_0_Conv2d_1a_3x3
Mixed_7a_Branch_1_Conv2d_1a_3x3_BatchNorm	BatchNormalization	(None, 3, 3, 256)	768	Mixed_7a_Branch_1_Conv2d_1a_3x3
Mixed_7a_Branch_2_Conv2d_1a_3x3_BatchNorm	BatchNormalization	(None, 3, 3, 256)	768	Mixed_7a_Branch_2_Conv2d_1a_3x3
Mixed_7a_Branch_0_Conv2d_1a_3x3_Activation	Activation	(None, 3, 3, 384)	0	Mixed_7a_Branch_0_Conv2d_1a_3x3_BatchNorm
Mixed_7a_Branch_1_Conv2d_1a_3x3_Activation	Activation	(None, 3, 3, 256)	0	Mixed_7a_Branch_1_Conv2d_1a_3x3_BatchNorm

Mixed_7a_Branch_2_Conv2d_1a_3x3_Activation	Activation	(None, 3, 3, 256)	0	Mixed_7a_Branch_2_Conv2d_1a_3x3_BatchNorm
Mixed_7a_Branch_3_MaxPool_1a_3x3	MaxPooling2D	(None, 3, 3, 896)	0	Block17_10_Activation
Mixed_7a	Concatenate	(None, 3, 3, 1792)	0	Mixed_7a_Branch_0_Conv2d_1a_3x3_Activation, Mixed_7a_Branch_1_Conv2d_1a_3x3_Activation, Mixed_7a_Branch_2_Conv2d_1a_3x3_Activation, Mixed_7a_Branch_3_MaxPool_1a_3x3
Block8_1_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 3, 3, 192)	344064	Mixed_7a
Block8_1_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_1_Branch_1_Conv2d_0a_1x1
Block8_1_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_1_Branch_1_Conv2d_0a_1x1_BatchNorm
Block8_1_Branch_1_Conv2d_0b_1x3	Conv2D	(None, 3, 3, 192)	110592	Block8_1_Branch_1_Conv2d_0a_1x1_Activation
Block8_1_Branch_1_Conv2d_0b_1x3_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_1_Branch_1_Conv2d_0b_1x3
Block8_1_Branch_1_Conv2d_0b_1x3_Activation	Activation	(None, 3, 3, 192)	0	Block8_1_Branch_1_Conv2d_0b_1x3_BatchNorm
Block8_1_Branch_0_Conv2d_1x1	Conv2D	(None, 3, 3, 192)	344064	Mixed_7a
Block8_1_Branch_1_Conv2d_0c_3x1	Conv2D	(None, 3, 3, 192)	110592	Block8_1_Branch_1_Conv2d_0b_1x3_Activation
Block8_1_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_1_Branch_0_Conv2d_1x1
Block8_1_Branch_1_Conv2d_0c_3x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_1_Branch_1_Conv2d_0c_3x1
Block8_1_Branch_0_Conv2d_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_1_Branch_0_Conv2d_1x1_BatchNorm
Block8_1_Branch_1_Conv2d_0c_3x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_1_Branch_1_Conv2d_0c_3x1_BatchNorm
Block8_1_Concatenate	Concatenate	(None, 3, 3, 384)	0	Block8_1_Branch_0_Conv2d_1x1_Activation, Block8_1_Branch_1_Conv2d_0c_3x1_Activation
Block8_1_Conv2d_1x1	Conv2D	(None, 3, 3, 1792)	689920	Block8_1_Concatenate

lambda_15	Lambda	(None, 3, 3, 1792)	0	Block8_1_Conv2d_1x1
add_15	Add	(None, 3, 3, 1792)	0	Mixed_7a, lambda_15
Block8_1_Activation	Activation	(None, 3, 3, 1792)	0	add_15
Block8_2_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 3, 3, 192)	344064	Block8_1_Activation
Block8_2_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_2_Branch_1_Conv2d_0a_1x1
Block8_2_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_2_Branch_1_Conv2d_0a_1x1_BatchNorm
Block8_2_Branch_1_Conv2d_0b_1x3	Conv2D	(None, 3, 3, 192)	110592	Block8_2_Branch_1_Conv2d_0a_1x1_Activation
Block8_2_Branch_1_Conv2d_0b_1x3_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_2_Branch_1_Conv2d_0b_1x3
Block8_2_Branch_1_Conv2d_0b_1x3_Activation	Activation	(None, 3, 3, 192)	0	Block8_2_Branch_1_Conv2d_0b_1x3_BatchNorm
Block8_2_Branch_0_Conv2d_1x1	Conv2D	(None, 3, 3, 192)	344064	Block8_1_Activation
Block8_2_Branch_1_Conv2d_0c_3x1	Conv2D	(None, 3, 3, 192)	110592	Block8_2_Branch_1_Conv2d_0b_1x3_Activation
Block8_2_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_2_Branch_0_Conv2d_1x1
Block8_2_Branch_1_Conv2d_0c_3x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_2_Branch_1_Conv2d_0c_3x1
Block8_2_Branch_0_Conv2d_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_2_Branch_0_Conv2d_1x1_BatchNorm
Block8_2_Branch_1_Conv2d_0c_3x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_2_Branch_1_Conv2d_0c_3x1_BatchNorm
Block8_2_Concatenate	Concatenate	(None, 3, 3, 384)	0	Block8_2_Branch_0_Conv2d_1x1_Activation, Block8_2_Branch_1_Conv2d_0c_3x1_Activation
Block8_2_Conv2d_1x1	Conv2D	(None, 3, 3, 1792)	689920	Block8_2_Concatenate
lambda_16	Lambda	(None, 3, 3, 1792)	0	Block8_2_Conv2d_1x1
add_16	Add	(None, 3, 3, 1792)	0	Block8_1_Activation, lambda_16
Block8_2_Activation	Activation	(None, 3, 3, 1792)	0	add_16
Block8_3_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 3, 3, 192)	344064	Block8_2_Activation
Block8_3_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_3_Branch_1_Conv2d_0a_1x1

Block8_3_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_3_Branch_1_Conv2d_0a_1x1_BatchNorm
Block8_3_Branch_1_Conv2d_0b_1x3	Conv2D	(None, 3, 3, 192)	110592	Block8_3_Branch_1_Conv2d_0a_1x1_Activation
Block8_3_Branch_1_Conv2d_0b_1x3_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_3_Branch_1_Conv2d_0b_1x3
Block8_3_Branch_1_Conv2d_0b_1x3_Activation	Activation	(None, 3, 3, 192)	0	Block8_3_Branch_1_Conv2d_0b_1x3_BatchNorm
Block8_3_Branch_0_Conv2d_1x1	Conv2D	(None, 3, 3, 192)	344064	Block8_2_Activation
Block8_3_Branch_1_Conv2d_0c_3x1	Conv2D	(None, 3, 3, 192)	110592	Block8_3_Branch_1_Conv2d_0b_1x3_Activation
Block8_3_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_3_Branch_0_Conv2d_1x1
Block8_3_Branch_1_Conv2d_0c_3x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_3_Branch_1_Conv2d_0c_3x1
Block8_3_Branch_0_Conv2d_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_3_Branch_0_Conv2d_1x1_BatchNorm
Block8_3_Branch_1_Conv2d_0c_3x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_3_Branch_1_Conv2d_0c_3x1_BatchNorm
Block8_3_Concatenate	Concatenate	(None, 3, 3, 384)	0	Block8_3_Branch_0_Conv2d_1x1_Activation, Block8_3_Branch_1_Conv2d_0c_3x1_Activation
Block8_3_Conv2d_1x1	Conv2D	(None, 3, 3, 1792)	689920	Block8_3_Concatenate
lambda_17	Lambda	(None, 3, 3, 1792)	0	Block8_3_Conv2d_1x1
add_17	Add	(None, 3, 3, 1792)	0	Block8_2_Activation, lambda_17
Block8_3_Activation	Activation	(None, 3, 3, 1792)	0	add_17
Block8_4_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 3, 3, 192)	344064	Block8_3_Activation
Block8_4_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_4_Branch_1_Conv2d_0a_1x1
Block8_4_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_4_Branch_1_Conv2d_0a_1x1_BatchNorm
Block8_4_Branch_1_Conv2d_0b_1x3	Conv2D	(None, 3, 3, 192)	110592	Block8_4_Branch_1_Conv2d_0a_1x1_Activation
Block8_4_Branch_1_Conv2d_0b_1x3_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_4_Branch_1_Conv2d_0b_1x3

Block8_4_Branch_1_Conv2d_0b_1x3_Activation	Activation	(None, 3, 3, 192)	0	Block8_4_Branch_1_Conv2d_0b_1x3_BatchNorm
Block8_4_Branch_0_Conv2d_1x1	Conv2D	(None, 3, 3, 192)	344064	Block8_3_Activation
Block8_4_Branch_1_Conv2d_0c_3x1	Conv2D	(None, 3, 3, 192)	110592	Block8_4_Branch_1_Conv2d_0b_1x3_Activation
Block8_4_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_4_Branch_0_Conv2d_1x1
Block8_4_Branch_1_Conv2d_0c_3x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_4_Branch_1_Conv2d_0c_3x1
Block8_4_Branch_0_Conv2d_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_4_Branch_0_Conv2d_1x1_BatchNorm
Block8_4_Branch_1_Conv2d_0c_3x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_4_Branch_1_Conv2d_0c_3x1_BatchNorm
Block8_4_Concatenate	Concatenate	(None, 3, 3, 384)	0	Block8_4_Branch_0_Conv2d_1x1_Activation, Block8_4_Branch_1_Conv2d_0c_3x1_Activation
Block8_4_Conv2d_1x1	Conv2D	(None, 3, 3, 1792)	689920	Block8_4_Concatenate
lambda_18	Lambda	(None, 3, 3, 1792)	0	Block8_4_Conv2d_1x1
add_18	Add	(None, 3, 3, 1792)	0	Block8_3_Activation, lambda_18
Block8_4_Activation	Activation	(None, 3, 3, 1792)	0	add_18
Block8_5_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 3, 3, 192)	344064	Block8_4_Activation
Block8_5_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_5_Branch_1_Conv2d_0a_1x1
Block8_5_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_5_Branch_1_Conv2d_0a_1x1_BatchNorm
Block8_5_Branch_1_Conv2d_0b_1x3	Conv2D	(None, 3, 3, 192)	110592	Block8_5_Branch_1_Conv2d_0a_1x1_Activation
Block8_5_Branch_1_Conv2d_0b_1x3_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_5_Branch_1_Conv2d_0b_1x3
Block8_5_Branch_1_Conv2d_0b_1x3_Activation	Activation	(None, 3, 3, 192)	0	Block8_5_Branch_1_Conv2d_0b_1x3_BatchNorm
Block8_5_Branch_0_Conv2d_1x1	Conv2D	(None, 3, 3, 192)	344064	Block8_4_Activation
Block8_5_Branch_1_Conv2d_0c_3x1	Conv2D	(None, 3, 3, 192)	110592	Block8_5_Branch_1_Conv2d_0b_1x3_Activation
Block8_5_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_5_Branch_0_Conv2d_1x1

Block8_5_Branch_1_Conv2d_0c_3x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_5_Branch_1_Conv2d_0c_3x1
Block8_5_Branch_0_Conv2d_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_5_Branch_0_Conv2d_1x1_BatchNorm
Block8_5_Branch_1_Conv2d_0c_3x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_5_Branch_1_Conv2d_0c_3x1_BatchNorm
Block8_5_Concatenate	Concatenate	(None, 3, 3, 384)	0	Block8_5_Branch_0_Conv2d_1x1_Activation, Block8_5_Branch_1_Conv2d_0c_3x1_Activation
Block8_5_Conv2d_1x1	Conv2D	(None, 3, 3, 1792)	689920	Block8_5_Concatenate
lambda_19	Lambda	(None, 3, 3, 1792)	0	Block8_5_Conv2d_1x1
add_19	Add	(None, 3, 3, 1792)	0	Block8_4_Activation, lambda_19
Block8_5_Activation	Activation	(None, 3, 3, 1792)	0	add_19
Block8_6_Branch_1_Conv2d_0a_1x1	Conv2D	(None, 3, 3, 192)	344064	Block8_5_Activation
Block8_6_Branch_1_Conv2d_0a_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_6_Branch_1_Conv2d_0a_1x1
Block8_6_Branch_1_Conv2d_0a_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_6_Branch_1_Conv2d_0a_1x1_BatchNorm
Block8_6_Branch_1_Conv2d_0b_1x3	Conv2D	(None, 3, 3, 192)	110592	Block8_6_Branch_1_Conv2d_0a_1x1_Activation
Block8_6_Branch_1_Conv2d_0b_1x3_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_6_Branch_1_Conv2d_0b_1x3
Block8_6_Branch_1_Conv2d_0b_1x3_Activation	Activation	(None, 3, 3, 192)	0	Block8_6_Branch_1_Conv2d_0b_1x3_BatchNorm
Block8_6_Branch_0_Conv2d_1x1	Conv2D	(None, 3, 3, 192)	344064	Block8_5_Activation
Block8_6_Branch_1_Conv2d_0c_3x1	Conv2D	(None, 3, 3, 192)	110592	Block8_6_Branch_1_Conv2d_0b_1x3_Activation
Block8_6_Branch_0_Conv2d_1x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_6_Branch_0_Conv2d_1x1
Block8_6_Branch_1_Conv2d_0c_3x1_BatchNorm	BatchNormalization	(None, 3, 3, 192)	576	Block8_6_Branch_1_Conv2d_0c_3x1
Block8_6_Branch_0_Conv2d_1x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_6_Branch_0_Conv2d_1x1_BatchNorm
Block8_6_Branch_1_Conv2d_0c_3x1_Activation	Activation	(None, 3, 3, 192)	0	Block8_6_Branch_1_Conv2d_0c_3x1_BatchNorm
Block8_6_Concatenate	Concatenate	(None, 3, 3, 384)	0	Block8_6_Branch_0_Conv2d_1x1_Activation,

				Block8_6_Branch_1_Conv2d_0c_3x1_Activation
Block8_6_Conv2d_1x1	Conv2D	(None, 3, 3, 1792)	689920	Block8_6_Concatenate
lambda_20	Lambda	(None, 3, 3, 1792)	0	Block8_6_Conv2d_1x1
add_20	Add	(None, 3, 3, 1792)	0	Block8_5_Activation, lambda_20
AvgPool	GlobalAveragePooling2D	(None, 1792)	0	add_20
Dropout	Dropout	(None, 1792)	0	AvgPool
Bottleneck	Dense	(None, 128)	229376	Dropout
Bottleneck_BatchNorm	BatchNormalization	(None, 128)	384	Bottleneck
normalize	Lambda	(None, 128)	0	Bottleneck_BatchNorm

Appendix B
Structure of the Proposed Neural Network

Layer Name	Layer Type	Output Shape	Param#	Connected To
sface	InputLayer	(None, 160, 160, 3)	0	
sfing	InputLayer	(None, 160, 160, 3)	0	
dface	InputLayer	(None, 160, 160, 3)	0	
dfing	InputLayer	(None, 160, 160, 3)	0	
Face_Model	Functional	(None, 128)	22808144	sface, dface
Fingerprint_ Model	Functional	(None, 128)	22808144	sfing, dfing
ssid	InputLayer	[(None, 32)]	0	
dsid	InputLayer	[(None, 32)]	0	
ls288	Concatenat e	(None, 288)	0	Face_Model, Fingerprint_Model, ssid
ld288	Concatenat e	(None, 288)	0	Face_Model, Fingerprint_Model, dsid
ls512	Dense	(None, 512)	147968	ls288
ld512	Dense	(None, 512)	147968	ld288
ls384	Dense	(None, 384)	196992	ls512
stimes	InputLayer	[(None, 47)]	0	
ld384	Dense	(None, 384)	196992	ld512
dtimes	InputLayer	[(None, 47)]	0	
lsts	Concatenat e	(None, 431)	0	ls384, stimes
ldts	Concatenat e	(None, 431)	0	ld384, dtimes
lse	Dense	(None, 512)	221184	lsts
lde	Dense	(None, 512)	221184	ldts
lambda_42	Lambda	(None, 1)	0	lse, lde
activation	Activation	(None, 1)	0	lambda_42