UNCERTAINTY ASSESSMENT FOR SPEAKER VERIFICATION SYSTEMS USING A BAYESIAN APPROACH

A Thesis

by

Çağıl Süslü

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
January 2021

UNCERTAINTY ASSESSMENT FOR SPEAKER VERIFICATION SYSTEMS USING A BAYESIAN APPROACH

Approved by:

Assoc. Prof. Dr. Cenk Demiroğlu, Advisor
Department of Electrical and Electronics Engineering
Özyeğin University

Assoc. Prof. Dr. Hasan Sözer
Department of Computer Science
Özyeğin University

Professor Dr. Ümit Güz
Department of Electrical and Electronics Engineering
Işık University

Date Approved: 18 January 2021

To My Family

ABSTRACT

The Automatic Speaker Verification (ASV) systems are developed to discriminate the genuine speakers from the spoofing attacks and they are also used as a security application in various industries (e.g., Banking and telephone-based systems). The spoofing countermeasure systems (SCS) are important for the ASV systems to protect themselves against spoofing attacks. In general, the SCSs are developed using the cross entropy loss function and the softmax classification layer to perform the best classification scores. Even though the softmax function is popularly used as a classification layer for the deep neural network tasks, it increases the uncertainty of the estimated class probabilities by squishing the probabilistic predictions of the predictive models. The aim of this work was to decrease uncertainty of the conventional cross entropy metrics and softmax function SCS by using the Bayesian approach. To accomplish this, multiple SCSs were developed to outperform the base system of the Automatic Speaker Verification Spoofing and Countermeasures 2017 Challenge. The Bayesian approach was applied to the best model (e.g., the model which performed the lowest EER score) to decrease the uncertainty of the conventional cross entropy metrics and softmax function SCS. The uncertainty of the both systems were compared with the probability distribution function, AUC value and the ROC curve. As it can be observed from the ROC curve, the Bayesian network decreased the uncertainty of the conventional cross entropy metrics and softmax function SCS by increasing AUC value 14%. Also the Bayesian network has provided the lowest EER score (16.79%) by outperforming the base system of the ASV spoof 2017 challenge.

ÖZETÇE

Otomatik Konuşmacı Doğrulama (OKD) sistemleri, gerçek konuşmacıları, sahte konuşmacılardan ayırt edebilmek için ve aynı zamanda bazı sektörlerde (Örn; Bankacılık, telekominikasyon) güvenlik sistemi oluşturma amacı ile kullanılmaktadır. Sahte konuşmacı önleme (SKÖ) sistemleri, OKD sitemlerinin kendilerini sahte konuşmacılara karşı koruyabilmeleri açısından büyük önem taşımaktadır. Genellikle SKÖ sistemleri, en iyi sınıflandırma performansını elde edebilmek için, çapraz entropi kaybı fonksiyonu ve softmax sınıflandırma fonksiyonu kullanarak geliştirilirler. Softmax sınıflandırma fonksiyonu derin öğrenme alanında birçok defa kullanılmasına rağmen, SKÖ sistemlerinin ürettiği tahmini olasılık değerlerinde sıkıştırılmaya sebep olarak, olasılık değerlerinin belirsizlik ölçümlerinde artışa sebep olabilmektedir. Bu araştırmanın asıl amacı, Bayes teorisini kullanarak geleneksel çapraz entropi kaybı ve softmax fonksiyonu SKÖ sistemlerindeki softmax kaynaklı belirsizlik değerlernin düşmesini sağlamaktır. Bunu yapabilmek için, Otomatik Konuşmacı Doğrulama ve Sahte Konuşmacı Önleme (OKDSKÖ 2017) 2017 yarışmasına birden fazla SKÖ sistemi geliştirilmiştir ve bu sistemlerin arasından en yüksek sınıflandırma başarısını gösterebilen sisteme Bayes teorisi uygulanmıştır. Sistemlerdeki belirsizlik ölçümlerini karşılaştırmak amacı ile Olasılık Dağılım Fonksiyonu (ODF), Eğri Altında Kalan Alan (EAKA) ve Alıcı Operasyon Karakteristiği (AOK) eğrisi kullanılmıştır. Araştırma sonucunda, Bayes teorisi ile yapılandırılan sistemin, geleneksel çapraz entropi kaybı ve softmax fonksiyonu SKÖ sisteminin EAKA değerini 14% yükselterek, geleneksel sistemin belirsizlik değerini düşürdüğü gözlemlenmiştir. Aynı zamanda, Bayes sistemi en başarılı Eşit Hata Oranı (EHO) değerini üreterek (16.79%), OKDSKÖ 2017 yarışmasında kullanılan temel sistemden daha üstün bir başarı elde etmiştir.

ACKNOWLEDGEMENTS

I would like to thank everyone who played a role in my academic achievements and support me during my Master's period.

At first, I would like to express my deep sense of thanks and gratitude to my thesis advisor Assoc. Prof. Cenk Demiroğlu for guiding, supporting and motivating me during my Master's studies.

I owe a deep sense of gratitude to Assoc. Prof. Murat Şensoy for his endless support and contributions to me during every period of my research.

I would like to thank my committee members Assoc. Prof. Hasan Sözer and Professor Ümit Güz for generously offering their time, support and guidance during my thesis dissertation.

I would like to thank my family for their valuable support during my Master's period.

I would also like to thank the Artificial Intelligent (AI) and Speech Processing (SP) laboratories for their kind help and support during my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

ASC Audio Signal Classification

ASV Automatic Speaker Verification

CDF Cumulative Distribution Function

$CQCC$ Constant Q Cepstral Coefficients

DNN Deep Neural Network

EER Equal Error Rate

GMM Gaussian Mixture Model

HMM Hidden Markov Model

JFA Joint Factor Analysis

KL Kullback-Leibler Divergence

$LCNN$ Light Convolutional Neural Network

$MFCC$ Mel-Frequency Cepstral Coefficients

MFM Max Feature Map

MSE Mean Square Error

PDF Probability Density Function

RA Replay Attack

RNN Recurrent Neural Network

SNR Signal to Noise Ratio

SS Speech Synthesis

TTS Text To Speech

VC Voice Conversion

CHAPTER I

INTRODUCTION

The Speaker Verification (SV) systems are developed to verify the claimed identity of the person based on their speech recordings [1]. As the Artificial Intelligent (AI) technology advances, the SV systems are automated by using machines to be more effective and efficient. Even though the Automatic Speaker Verification (ASV) systems achieves remarkable accomplishments, the vulnerabilities against spoofing attacks are still considered as a problem.

In general, the ASV systems (text-independent or text-dependent)[1] develops with the standard Gaussian Mixture Models (GMM) [1]. The work in [2], develops the claim that the Joint Factor Analysis (JFA) and the GMM-UBM systems are able to detect replay attacks when recordings of the target speakers were acquired by the far-field microphone and replayed to the system by using a loud telephone speaker. However, as mentioned in [3], the Hidden Markov Model (HMM) based systems were able to outperform the GMM based systems. Also, the Support Vector Machine (SVM) models, which were designed with a channel pattern noise-based approach, were able to reduce the Equal-Error-Rate (EER) score of the target system by 30% [4]. As shown in [5], the SVM models were able to detect replay spoofing attacks, which formed by combining the short recordings (e.g., a word) to generate a full sentence to deceive the AVS systems and they were achieved an EER score of less than 10%. These types of recordings were crucial for security applications, especially for the banking industry (e.g., accessing the personal account). The work in [6], investigated

[1] In the text-dependent system, the recognition phrases were known before, and they do not change afterwards. In a text-independent system, speakers are allowed to use any words or sentences.

the vulnerabilities of the ASV systems, which were developed with the HMM-UBM and GMM-UBM models. Both of the models showed vulnerabilities against replay spoofing attacks.

Even though the machine learning models (e.g., GMM, HMM) were able to improve the robustness of the countermeasure systems, the realistic replay spoofing attacks (e.g., almost identical with the genuine samples) were still an important thread for the ASV systems.

The Automatic Speaker Verification Spoofing and Countermeasures 2017 Challenge (ASV spoof 2017) was organized to detect these vulnerabilities and it was intended to develop robust spoofing countermeasure systems (SCS) by using the deep neural network models to secure the ASV systems against realistic replay attacks [7]. The objective of the ASV spoof 2017 was to design robust SCSs that were able to handle the *generalization* problem occurring in the ASV spoof 2017 dataset by decreasing the EER score of the base model (CQCC-GMM) of the challenge. The ASV spoof 2017 dataset version 2.0[2] [8] was especially composed of challenging replay spoof attacks that were developed to simulate "out in the wild" conditions. Also, another ASV spoof dataset was introduced to analyze the vulnerability of the ASV systems against the spoofing attacks (replay attacks, speech synthesis and voice conversion) [9].

In this work, the four diverse SCSs were developed to outperform the EER score of the base system of the ASV spoof 2017 challenge and the SCSs were trained on the ASV spoof 2017 dataset version 2.0.

[2]https://datashare.is.ed.ac.uk/handle/10283/3055

Most of the SCSs were developed by using the cross entropy loss and softmax function. The softmax function causes an uncertainty problem as a classification layer in the predictive models [10]. For the industries, which security is the main concern (e.g., Banking, call-centers), improving the confidence of the estimated class probabilities will enhance the durability of their systems against spoofing attacks. Therefore, an additional contribution is made to the ASV systems with the application of the Bayesian approach which was introduced in [10].

To do this, the best performing (e.g., lowest EER) model was selected as the conventional cross entropy metrics and softmax function SCS. The novel approach was made by applying the Bayesian approach to the conventional cross entropy metrics and softmax function SCS to decrease the uncertainty of the system. By the fact that, two different approaches were in sight for the best performing SCS: the Bayesian system, and the conventional cross entropy metrics and softmax function system. The best epoch of the both systems were used to compare the uncertainty measures. The probability distribution function, AUC value and ROC curve were employed as performance metrics for this work. According to the AUC and ROC curve metrics, the Bayesian approach was able to reduce the uncertainty of the conventional cross entropy metrics and softmax function SCS successfully. In addition, the Bayesian network has produced the overall best EER score among the other networks, which were developed in this study.

CHAPTER II

PROBLEM STATEMENT

The softmax classification layer squashes the probabilistic outputs of the predictive models to make their sum equal to 1. By reason of the fact that, it increases the uncertainty of the predicted class probabilities of the spoofing countermeasure systems. This causes the ASV systems to be more uncertain about their predicted opinions, which are produced for the spoof and genuine samples. In this thesis, the main concern was to decrease the uncertainty of the conventional cross entropy metrics and softmax function SCS. To do that, four SCSs were developed and the best performed model was employed for the Bayesian approach. The probability distribution function and ROC curve metrics are used for a measure of uncertainty in the systems.

CHAPTER III

BACKGROUND

3.1 Automatic Speaker Verification Technology

The *Speaker Verification* (SV) [11] technology advancements started decades ago. The technological evaluation allows SV to be interpreted with artificial intelligence techniques and making the process automated. The *Automatic Speaker Verification* (ASV) [12, 13] technology was aspired to detect *spoofing attacks* by developing a spoofing countermeasure systems. The general steps of the ASV systems are referred to as; the acquisition of the speech data, the feature extraction process, the learning process, and the decision process (e.g., accept or reject the input sample) [11]. The speaker verification system decides if a speaker is who they claim to be. The SV technology is used to distinguish the non-synthetic or non-recorded human voices. On the other hand, the speaker identification (SI) system aims to verify whether a speaker is a specific person or not. The ASV technology can be formed as *text-dependent* [14] and *text-independent* systems. In the text-dependent system, the recognition phrases were known before, and they do not change afterwards [12]. In text-independent systems, speakers are allowed to use any words or sentences [12]. Consequently, the train and test datasets can be completely different from each other. Therefore, between the two of them, the text-independent systems are more challenging compared to the other technology [12].

Ever since the discovery of the ASV system, there has been great advancements in the technology. The *efficiency*, *practicality*, and *utilizability* of the ASV make it seem remarkable to relevant sectors (e.g., Banking and call centers). The advanced

technology causes highly effective spoofing attack possibilities nowadays. That leads to an urge to efficacious ASV systems to secure those industries, and also it leads speaker verification systems to advance in time [15]. The other reason why researchers are so interested in such systems can be explained by the unique formation of the human speech [16]. The *speaking style*, *voice frequencies*, and *spectral magnitudes* varies according to each person [16]. Therefore, the speaker verification and identification systems draw attention from some enterprises since the human voice is convenient for the *biometric authentication* [17] and can be used as a personal signature. The formation of speech signals provides opportunities to develop down-to-earth systems for the banking industry, call centers, and telephone-based systems [18]. Even dough, ASVs are *low-cost* and *practicable*, there is still an important amount of potential for ASV systems to be threatened by *spoofing* attacks [13, 19]. In response to the threats, scientists have sought to advance effective approaches to anti-spoofing like *spoofing countermeasure systems*.

3.2 Spoofing

The spoofing attacks aim to deceive the automatic speaker verification system. The spoofing attacks specifically designed to compromise reliability of the ASVs [20]. Each type of spoofing attack was formed by a different kind of technique. To prevent those attacks, a countermeasure system should be developed to consider the aspects of each spoofing attack. Spoof attacks can be formed by a professional playback device (e.g., replay attack) or a synthetic speech to impersonate a specific person *synthetic human-voice*. In the following section, a detailed information is provided for some of the spoofing attacks.

3.2.1 Voice Conversion

Voice conversion (VC) is a spoofing attack that is used to transform one speakers' voice into another speaker [21, 13, 22]. Those voices originated from a real person,

and this makes them undetectable by countermeasure systems. State-of-the-art voice conversion algorithms have proved themselves about fooling the ASV systems and they even succeed to deceive the human listeners. The aim of the voice conversion attacks is, to convert the voice of the original speaker $(\boldsymbol{X})$ to the target speaker $(\boldsymbol{Y})$ by using the conversion function $\boldsymbol{F}$ with conversion parameters $\mathbf{Q}$ [21, 23].

$$\boldsymbol{Y} = \boldsymbol{F}(\boldsymbol{X}, \boldsymbol{Q}) \tag{1}$$

3.2.2 Impersonation

Impersonation is the most commonly used spoofing attack. It aims to imitate the voice of the target person and attempt to fool the ASVs by mimicking the vocal of the target speaker [1]. Impersonation, generally imitates some characteristics of the human voice; *pitch register*, *the quality of the voice*, *dialect*, *prosody*, and *the style of the speech* [23]. The voice of the real speaker cannot imitate identically by an impersonating attack. Therefore, this type of attacks deceives human listeners more than the state-of-the-art ASVs[23, 24].

3.2.3 Speech Synthesis

In general, the speech synthesis (SS) uses a text-to-speech (TTS) technique to produce spoofing attacks. This technique aims to produce clear, realistic, and natural-sounding speech samples from any designated text example. Those synthetic speeches can use in digital vocalization like navigation systems or electronic books and dictionaries [1]. The main constituents of speech synthesis are text analysis and speech waveform [25, 26] techniques. This study [27] is an attempt to address the issue of imposture against speaker verification systems using synthetic speech.

3.2.4 Replay Audio

The replay audio attacks (RA) were produced from the prerecorded speech samples of a genuine speaker and these attacks were formed by concatenating the shorter segments of continuous speech recordings [19]. The accessibility of high-quality and low-cost technological devices, like smartphones or studio recording equipment, allows replay attacks to become a significant threat and also, the professional recording devices can cause these attacks to be hardly detectable by the ASV [1, 19]. RA also used in the Automated Speaker Verification Spoofing and Countermeasure Challenge 2017 (ASVspoof 2017) challenge to produce spoofing attacks [28].

3.3 Spoofing Countermeasure Systems

The *spoofing countermeasure systems* (SCS), additionally known as a *presentation attack detection (PAD)* systems. The SCS is a support system for the detection of spoofing attacks that were presented to the ASV systems [20]. The attention for the SCSs had increased in recent years because of the popularity of the spoofing attacks [1, 29, 30]. The SCSs aims to detect imposter recordings among genuine speech signals and they were advanced to protect the automatic speaker verification systems from harmful intentions. The countermeasure systems lowered Equal Error Rare (EER) successfully according to many studies [1]. Especially, specific industries like *telephone-based systems*, *voice mail* [11], and *forensics* [31, 32, 33, 34] needs a successful SCSs to prevent their systems form the spoofing attacks. In the future, when we consider the success of spoofing countermeasure systems, SCSs can be used as a replacement for the *human-operated* telephone services.

As the spoofing attacks evolve and become hardly detectable by the ASVs, these techniques become more accessible for people as well. Therefore, the urge to develop

an effective SCS becomes more significantly important [1]. Some professional recording devices, as they used in ASVspoof 2017 Challenge evaluation dataset, are capable of recording a human voice without any unrelated background noises.

In the INTERSPEECH 2013 conference, to support the evaluation of SCSs, they have aimed to determine a common dataset, protocols, and metrics to be used [35] for the future researches [18]. More details can be found in the study [18] that refers to the key-milestones of the ASVs. The SCSs are generally developed by the combination of an effective feature extractor and a robust training model. The training model can be constituted by a deep neural network or machine learning techniques like the Gaussian Mixture Model (GMM) or Support Vector Machines (SVM) [2]. According to the most recent studies, the focus inclines more on developing SCSs with deep learning methods rather than using machine learning approaches.

3.3.1 Vulnerability of the Spoofing Countermeasure System Against Replay Audio Attacks

Even though the ASVs perform remarkable accomplishments, the vulnerabilities against spoofing attacks are still considered as a problem. The first study to uncover the vulnerability of the speaker verification system conducted in 1999 [36]. The study used Hidden Markov Model (HMM) as a base system and they constituted their database from the telephone recordings. In [2], the study develops the claim that the Joint Factor Analysis (JFA) based system are able to detect replay attacks when recordings of the target speaker acquired by the far-field microphone and replayed by using a loud telephone speaker. The findings of the study [37] lend support to the claim that tampering attacks, like nasalization or using a handkerchief to alter your voice to sound like someone else, are able to lower the speaker verification scores. Another study [4] stated that the playback attack detector systems designed by a channel

pattern noise-based approach and trained with Support Vector Machine reduce EER by 30%. In [9], the ASVspoof dataset introduced to analyze the vulnerability of the ASV systems against replay, speech synthesis, and voice conversion attacks. As the study stated by Jesus et al. (2011) [5], the replay attacks that were produced from the abridged recordings, imitates the specified sentence to deceive the AVS systems. These types of recordings are referred to as crucial for security applications, especially for the banking industry. In the study, they used an SVM model to achieve an EER score of less than 10%. Some independent studies have been conducted to confirm the threat of the replay attack spoofing techniques against the ASVs [38, 6, 39].

3.4 Automatic Speaker Verification Spoofing and Countermeasure 2017 Challenge

The first Automatic Speaker Verification Spoofing and Countermeasure Challenge held in 2015. The ask in ASVspoof 2015 [40] challenge was to develop an SCS to discriminate genuine speech from the *voice conversion* attacks. The replay attacks indicated as the most dangerous threats due to their accessibility [7]. Accordingly, the *ASVspoof*2017 challenge is aimed to detect replay attacks amongst genuine samples [28]. The challenge also requests participants to advance an automatic speaker verification system by using deep neural network [41] models and outperform the base model of the ASV challenge. The base system of the 2017-Challenge constituted with the constant Q cepstral coefficients (CQCC) features and Gaussian Mixture Model (GMM) [42].

The primary objective of the ASVspoof 2017 challenge was to advance spoofing countermeasure systems for the *'out in the wild'* conditions. The challenge aims to experiment with the deep neural networks' classification performances against *unseen-cases* and it aims to research the SCSs that confronted with the *generalization* problem caused by the prepared dataset [28]. To achieve that, the text-dependent *RedDots*

corpus [43] used to produce the *replayed* [8] version (Section 6.4) to act as replay spoofing attack simulation.

The rules of the challenge were developing a spoofing countermeasure system by using the predetermined train, development, and evaluation datasets. At the early stage, the train and development subsets were available to participants. The evaluation dataset shared during the score declaration period. During the training session, in necessary conditions, the train and development subsets were allowed to use together, or participants may prefer to train their model by using the train dataset alone. The equal error rate determined as an evaluation metric and only the EER score of the evaluation dataset considered in the challenge. The base model of the challenge is released to guide participants during the competition. In [28], a detailed explanation of the evaluation plan of the challenge presented.

In this thesis, focus was to develop an a successful SCS for the text-dependent Automatic Speaker Verification Spoofing and Countermeasure 2017 Challenge and outperform performance (e.g., EER score) of the base system (e.g., CQCC-GMM) presented by the challenge.To achieve that, diverse neural network architectures advanced to use as an SCS with an effective feature extractor.

CHAPTER IV

METHODOLOGY

4.1 The Replay Attack Detector System

The *replay attack detector* (RAD) systems were developed to detect the replay spoofing attacks, which occurred in the ASV systems, and they were built around the spoofing countermeasure systems. RAD systems were formed of an effective feature extraction technique and a robust spoofing countermeasure system to identify the replay attacks. In this study, the replay attacks were detected from prerecorded speech signals that were used to gain unauthorized access from the Automatic Speaker Verification systems. Therefore, this study involves in-depth research about developing a robust spoofing countermeasure system with an suitable feature extractor for the noisy audio signals.

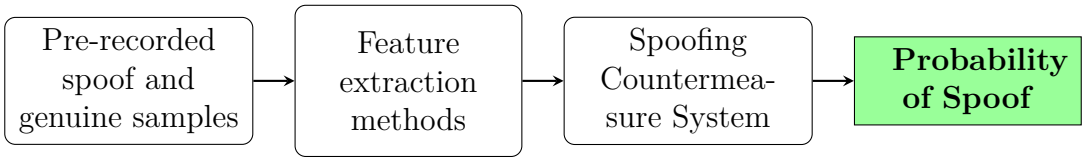

Figure 1: The Illustration of the Replay Attack Detector System

The Figure (1) represents the replay attack detector system used in this work. The system was fed with the **pre-recorded** speech signals that were obtained from the ASV spoof 2017 challenge. The ASV spoof dataset was gathered from the real speakers (e.g., RedDost corpus) and it was replayed to the ASV system's microphone to produce spoofing attacks (e.g., the details of the simulation is introduced in Section 5.1.2.1) to be verified by the SCSs [8]. The **feature extraction method** was applied

to the audio recordings to obtain a normalized binary dataset. To achieve this, the mel-frequency cepstral coefficients (MFCC) technique was applied with the zero-mean and unit-variance normalization method. The **spoofing countermeasure systems** are the backbone of the replay attack detector systems. In this work, four different SCSs were developed and the best-performed system (e.g., lowest EER score) among these neural nets were employed as the major architecture for the RAD system.

In this work, the major architecture was applied with two different forms to investigate the uncertainty in the ASV systems: the conventional cross entropy metrics and softmax function SCS (e.g., SCS_{CE}) and the Bayesian approach (Section 4.3.2) (e.g., SCS_{EDL}). The outputs of the SCSs were accepted as the estimated class probabilities for the binary classification task and they were used to classify each input sample as a replay attack (e.g., spoof) or genuine.

4.2 The Spoofing Countermeasure Systems Developed for the ASV Spoofing 2017 Challenge

4.2.1 Fully Connected Neural Network Architecture

The Fully Connected Neural Network (FNN) architecture contains the highest number of the parameters among other three neural nets with 4.9M. As presented in Table (1), the 1-dimensional max-pooling layer [44] was employed with a 3 fully-connected layers (FC) of size $m \times n$ and the output of the last fully-connected layer accepted as the estimated class probabilities of the genuine and spoof classes, which denoted as $C_i = [1, 0]$. The model was fed by 3-dimensional input samples denoted as $\mathbf{I} \in \mathbb{R}^{B \times T \times F}$ where B, T, and F denotes to the numbers of *batch size*, *time*, and *feature*. B, T, and F were set as 32, 126, and 72. The optimization was performed with the *Adam* [45] gradient optimizer with the following parameters: $epsilon = 0.1$, and $learning\ rate = 0.001$. In this architecture, the cross entropy loss function is applied with the softmax function (Section 4.3.1).

Table 1: THE FULLY CONNECTED NEURAL NETWORK ARCHITECTURE

Type	Kernel size/Stride	prob	Output	#Params
MaxPool1	6 / 2	-	$32 \times 126 \times 34$	-
FC1	-	-	32×1024	4.3M
ReLU	-	-	-	-
Drop	-	0.70	-	-
FC2	-	-	32×512	528.8K
ReLU	-	-	-	-
Drop	-	0.70	-	-
FC3	-	-	32×128	65.6K
ReLU	-	-	-	-
Drop	-	0.70	-	-
FC4	-	-	32×2	258
Total	-	-	-	**4.9M**

4.2.1.1 1-dimensional max-pooling layer

To prevent the overfitting problem in the neural network systems, the max-pooling operation is used by many researchers. The operation works by applying a *sliding window* to the input signals with a specific kernel size and then obtaining the maximum elements in the window to downsample the input matrix. For the FNN system, the max-pooling operation was applied with a 6-dimensional sliding window on the **I** and the maximum element in the window matrix was taken to form the output matrix. To test the performance of the neural network architecture with only using fully-connected layers, any convolutional layer before the max-pooling layer was not applied.

As defined in Equation (2)[1], the 1-dimensional max-pooling layer was applied to downsample the **I** and it was separated **I** matrix into 1D sub-vectors that were denoted as $\mathbf{I} \in \mathbb{R}^d$, where $d = \{n_1, n_2, ..., n_f\}$. The operation was applied with a 6-dimensional kernel vector (e.g., sliding window) to acquire the elements, which have

[1]The equation was retrieved from official Pytorch documentation.

the maximum numeric value along with the F-dimensional feature vector.

$$P_{out}(B_i, T_j, f) = \max_{m=0, kernel\_size-1} input\,(B_i, T_j, stride \times f + m) \tag{2}$$

Where, *stride* was denoted to the number of pixel shifts over the F-dimension of $\mathbf{I}$. When the stride is s, the algorithm moves the kernel matrix to s pixel at a time. The output shape of the max-pooling layer P_{out} was defined in Equation (3)[2].

$$F_{out} = \left[\frac{F_{in} + 2 \times padding - dilation \times (kernel\_size - 1)}{stride} + 1\right] \tag{3}$$

Where, *dilation* was set to 1, and it was used to control the spacing between the kernel points. The *padding* parameters p were employed to add implicit negative infinity padding on the both sides of the input matrix $\mathbf{I}$, where $p = \{0 < p \leq kernel\ size\}$. The P_{out} with size $B \times T \times F_{out}$ was flattened into 2-dimensional matrix $m \times n$ where $m = B$, and $n = T \cdot F$, before it was fed into fully-connected layers.

4.2.1.2 Dropout layer

The *dropout* [46] is an operation, which is employed to prevent the overfitting problem in the neural network systems. The operation is used to randomly set some of the elements in the feature matrices into 0 and by that it prevents *co-adapting* in the neural network systems. In this work, the dropout operation was used to eliminate some of the elements in the layer outputs by setting them into 0 and doing so it prevents the neural network exposure to the noisy features that occur in the input matrix. Throughout the training process, the output of the dropout was scaled by a factor $\frac{1}{1-p}$, where p was denoted to the *probability* parameter. As shown in Figure (2)[3], the standard neural network model (left) was transformed into the form of (right)

[2]The equation was retrieved from official Pytorch documentation.

[3]Retrieved from Deep learning for complete beginners: Using convolutional nets to recognize images, Cambridge Coding Academy

after the dropout algorithm was applied. The red crosses in the Figure refers to the canceled nodes. In the FNN system, the dropout algorithm was applied to each fully-connected layer except the last one. In the training process, the probability parameter p was set to 0.70, and during validation, p was set to 1.

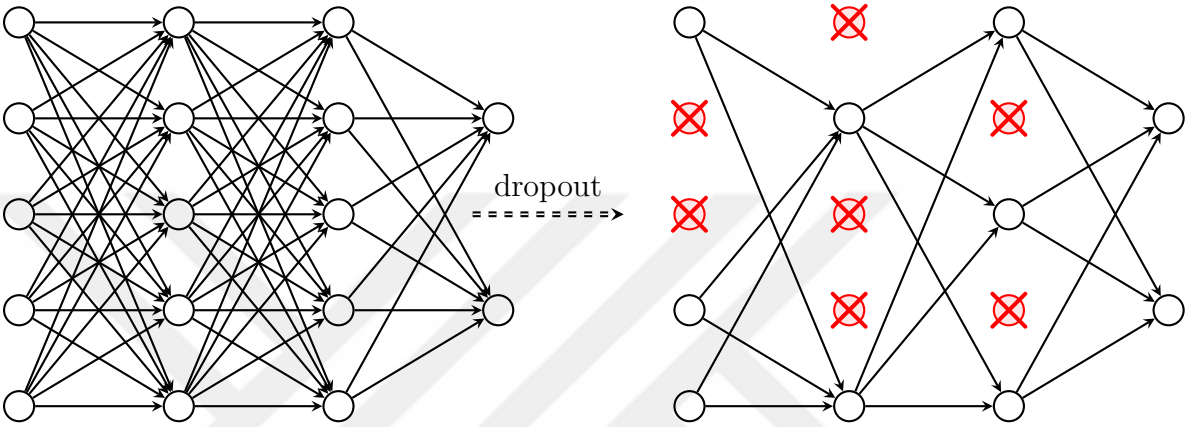

Figure 2: Illustration of the Dropout Operation

4.2.1.3 Weight and Bias Initialization

The weight matrix $\mathbf{w} \in \mathbb{R}^{B \times n}$ was initialized by the *Xavier Initialization* algorithm and it was employed from the *Pytorch* library. The values of the weight matrices were initialized by $\mathbf{w} \sim \mathcal{N}(0, std^2)$, where *mean* was set to $\mu = 0$ and the *variance* was denoted as $\sigma^2 = std^2$. The *std* was defined by the Equation (4)[4].

$$std = gain \times \sqrt{\frac{2}{\text{fan\_in} + \text{fan\_out}}} \tag{4}$$

Where, *gain* was denoted to the optional scaling factor and it was set to 1. The fan_in and fan_out parameters were computed by the multiplication of the receptive fields' size with the input/output-dimension of $\mathbf{w}$. The bias $\mathbf{b} \in \mathbb{R}^n$ was initialized by the *Constant Initialization* algorithm and it was set to 0. The outputs of the *FC* layers were defined in the Equation (5).

[4]The equation was retrieved from official Pytorch documentation.

$$y = h(\mathbf{w}_i x_i + \mathbf{b}) \tag{5}$$

Where, x_i was referred to the outputs of the specified layer, where i was enumerated the each input sample, and $h(.)$ was denoted as *rectified linear unit* (ReLU)[47] activation function, which was applied each of the fully-connected layers and it was computed based on the Equation (6).

$$h(x) = max(0, x) \tag{6}$$

According the Equation (6), if the input $x \in \mathbb{R}$ is negative, ReLU activation function outputs 0. If it is positive then the activation function outputs x matrix itself.

4.2.2 Recurrent Neural Network Architecture

The Recurrent Neural Network (RNN) system was applied to test the classification performance of the RNN architectures for the ASV spoof 2017 dataset. The complexity of the FNN architecture was decreased by reducing the parameters into 1.2M. The effectiveness of the RNN architecture was a suitable inductive bias for the audio segments that were obtained from the ASV spoof 2017 dataset. Thereby, the FNN architecture was redesigned with the *bi-directional long short-term memory* ($Bi\text{-}LSTM$) layer, which was used as a feature extractor for the system. In the following, the output of the $Bi\text{-}LSTM$ layer fed into the 3 fully-connected layers which was similar to the FNN architecture. As shown in Table (2), each of the FC_{out} was fed into the *1-dimensional batch normalization* and dropout layers. The Adam optimizer was applied with the parameters as following: $epsilon = 0.1$, and $learning\ rate = 0.001$. Weight and bias values were initialized similar to the Section 4.2.1.3.The cross entropy loss and the softmax function were used in the system.

Table 2: THE RECURRENT NEURAL NETWORK ARCHITECTURE

Type	num_layer/hidden_size	prob	Output	#Params
Bi-LSTM	4 / 8	-	$32 \times 126 \times 16$	5.6K
FC1	-	-	32×512	1M
BatchNorm	-	-	512	-
ReLU	-	-	-	-
Drop	-	0.50	-	-
FC2	-	-	32×256	131.3K
BatchNorm	-	-	256	-
ReLU	-	-	-	-
Drop	-	0.50	-	-
FC3	-	-	32×128	32.8K
BatchNorm	-	-	128	-
ReLU	-	-	-	-
Drop	-	0.50	-	-
FC4	-	-	32×2	258
Total	-	-	-	**1.2M**

4.2.2.1 Long Short-term Memory (LSTM) Cell Architecture

The *Long Short-term Memory* (LSTM) [48] architecture was introduced by Hochreiter in 1997. The structure of the LSTM cell comprises the functions which are called gates. The LSTM cell is computed by Equations (7) - (12)[5].

$$i_t = \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right), \tag{7}$$

$$f_t = \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\right), \tag{8}$$

$$g_t = \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}\right), \tag{9}$$

$$o_t = \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}\right), \tag{10}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t, \tag{11}$$

$$h_t = o_t \odot \tanh(c_t) \tag{12}$$

Where, x_t was denoted to the input matrix $\mathbf{I}$ at time t. The h_t was defined as a

[5]The equation retrieved from official Pytorch documentation.

hidden state and the h_{t-1} was denoted as the previous hidden state at time $t-1$. The c_t was denoted to the cell state at time t and the previous cell state was denoted as c_{t-1} at time $t-1$. The input gate layer, forget gate layer, cell gate layer and output gate layer were denoted as i_t, f_t, g_t, and o_t. The sigmoid function was denoted as σ, and $\odot$ referred to Hadamard product.

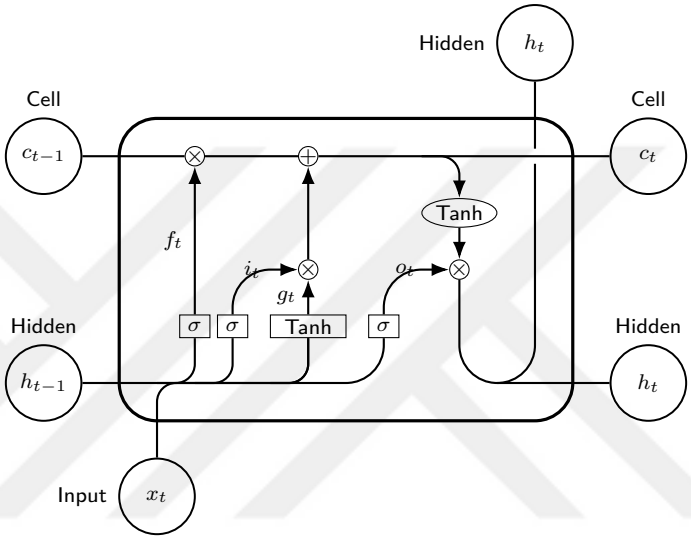

Figure 3: Illustration of a LSTM Cell architecture

Figure 3 shows the detailed illustration of the LSTM cell architecture. The forget gate f_t layer (e.g., Equation (8)) decides on the information which will be ignored by the cell and it outputs 0 to ignore the information and it outputs 1 to keep it at the previous cell state c_{t-1}. The responsibility of the input gate i_t layer (e.g., Equation (7)) is to determine the values that are necessary to update. The cell gate g_t (e.g., Equation (9)) produces new candidate information. The new information that is going to be stored in the new cell state is determined by the forget gate, previous cell state, input gate, and cell gate (e.g., Equation (11)). The new hidden state forms using the output gate o_t and new cell state (e.g., Equation (12)).

4.2.2.2 Bi-directional LSTM (Bi-LSTM) layer

In this model, the *bidirectional lstm* layer was applied to use as a feature extractor. The bidirectional LSTM layer is able to get the information from the past (backwards) and the future (forward) states of each LSTM cell simultaneously. As described in Figure (4), x_t was referred to the input matrix $\mathbf{I}$ at time t and it was fed into the $LSTM$ cell which was denoted as $cell_i$ where, i was enumerated the number of the cells. The parameters were passed between the forward and backward layer cells during the training process which was denoted as P_t, where $P_t = \{h_t, c_t\}$. The $hidden\_size$ parameter was denoted to the number of the features in the hidden state h and it was set to 8. The number of the recurrent layers are denoted with the parameter num_layers and it was set to 4. The dropout was not applied to any of the recurrent layers of the $Bi\text{-}LSTM$ layer. The output of the $Bi\text{-}LSTM$ layer with the dimension $(num\_layers \times directions = 2) \times B \times hidden\_size$ flatten into 2-dimension $B \times (T \times hidden\_size)$ to fed into FC layer. Weights and biases of the $Bi\text{-}LSTM$ layer were initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{hidden\_size}$.

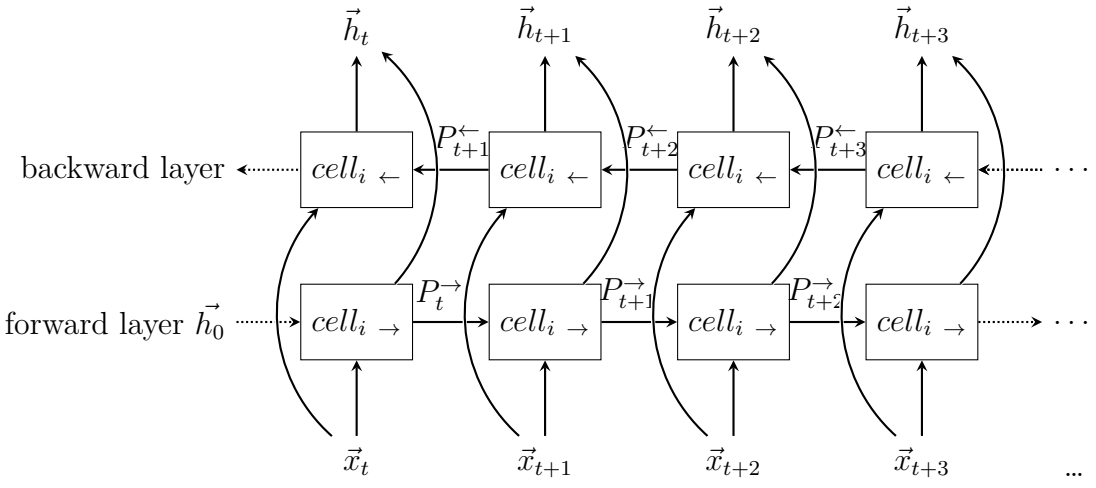

Figure 4: Illustration of a Bidirectional LSTM model

4.2.2.3 1-Dimensional Batch Normalization

The *batch normalization* [49] algorithm is used to normalize the neural network layer outputs and by that it stabilizes the training process. The 1-dimensional batch normalization algorithm was applied before the activation function to stabilize the outputs of each layer in the system. The batch normalization algorithm was calculated by the Equation (13)[6].

$$\mathbf{y} = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta \tag{13}$$

Where, the *mean* γ and the standard deviation β parameters were the learnable parameter vectors with C-dimension and they were set to 1 and 0, where C is the input size. The values over a mini-batch were denoted as $\mathcal{B} = \{x_1, ..., x_m\}$ and $\mu_{\mathcal{B}} = E[x]$ and $\sqrt{\sigma^2}_{\mathcal{B}} = \sqrt{Var[x]}$ parameters were calculated per-dimension for each of the mini-batches. The β was calculated by the biased estimator which was equivalent to the *torch.var* function that returns the variance of all the elements in the given matrix. The equation which was defined before the $*$ symbol was applied to normalize the input data. Afterwards, normalized output was multiplied by γ to scale and sum with μ to shift the output value. *Epsilon* ϵ and *momentum* parameters were set to 0.001 and 0.1. The ϵ parameter was added to the denominator to provide numerical stability. The *momentum* parameter was used as an update rule for the *running_mean* and *running_var* computation and it was applied to the input matrix by the Equation (14)[7].

$$\hat{x}_{new} = (1 - momentum) \times \hat{x} + momentum \times x_t \tag{14}$$

Where, x_t was denoted to the new observed value and $\hat{x}$ was denoted to the estimated statistics.

[6]The equation was retrieved from official Pytorch documentation.
[7]The equation was retrieved from official Pytorch documentation.

4.2.3 Residual Neural Network (ResNet) Architecture

The ReNet architectures are designed to avoid from the vanishing/exploding gradient [50] problems that occur in the deep neural network models. The *ResNet* architecture contains a *block* layer, which provides successful classification performances even as the depth of the network increases [51]. The ResNet model has 44.1K parameters and it is the least complex architecture among the other three neural network models. In this work, the ResNet architecture is inspired from the $MFCC\text{-}ResNet$ system, which was used in [52]. As shown in Table (3), the ResNet architecture is formed by the *convolutional* and *ResNetBlock* layers and it is followed by a *2-dimensional max-pooling*, *2-dimensional batch normalization*, and dropout operations. The Adam optimizer was applied with the parameters $epsilon = 0.1$, and $learning \quad rate = 0.001$. Similar to the other networks, the cross entropy loss was applied with the softmax function. Weight and bias initialization algorithms are the same as the other neural network models.

Table 3: ILLUSTRATION OF THE RESIDUAL NEURAL NETWORK ARCHITECTURE

Type	**Filter/Stride**	**prob**	**Output**	**#params**
Conv1	3×3 / 1×1	-	$32 \times 8 \times 126 \times 72$	160
Block1	3×3/	-	$32 \times 16 \times 42 \times 24$	4.6K
MaxPool1	2×2 / 3×3	-	$32 \times 16 \times 15 \times 9$	-
Block2	3×3/	-	$32 \times 32 \times 5 \times 3$	18.5K
MaxPool2	2×2 / 1×1	-	$32 \times 32 \times 6 \times 4$	-
Block3	3×3/	-	$32 \times 16 \times 2 \times 2$	11.5K
MaxPool2	2×2 / 1×1	-	$32 \times 16 \times 3 \times 3$	-
Dropout	-	0.70	-	-
FC1	-	-	32×64	9.2K
ReLU	-	-	-	-
FC2	-	-	64×2	130
Total	-	-	-	**44.1K**

4.2.3.1 Convolutional layer

The convolutional neural network is designed for image-based systems, and it generally uses pixel-wise classification. Convolutional layers are capable of extracting high-level features out of the raw inputs easily by courtesy of the pixel-wise analyze opportunity. Since the audio signals have similar structures as the image, the convolutional layers presented high-level evaluation performance against the noisy audio recordings. The input matrix $\mathbf{I}$ with size $B \times T \times F$, expanded to dimension $B \times c \times T \times F$ to fit into the convolutional layer. Since audio signals are grey-scale, the channel size parameter was denoted as c and it was set to 1. The computation of the convolutional layer is presented in Equation(15)[8].

$$conv_{out}(B_i, c_{out_j}) = \text{bias}(c_{out_j}) + \sum_{k=0}^{c_{in}-1} \text{weight}(c_{out_j}, k) \star \text{input}(B_i, k) \tag{15}$$

Where, B_i was denoted to the number of batches where, i was enumerated the batch size and c_{out_j} parameter was denoted to the output channel size where, $j = \{0, .., out\_channels\}$. The c_{in} represented the input channel size. The *weight* and *bias* were the learning parameters and the *input* matrix was indicated to the $\mathbf{I}$ along with the B-dimension. $\star$ symbol was indicated to the 2D cross-correlation operation.

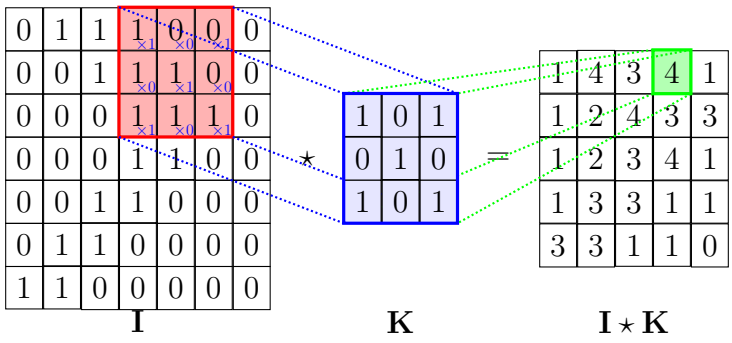

Figure 5: Illustration of the Convolutional layer

[8]The equation was retrieved from official Pytorch documentation.

As illustrated[9] in Figure (5)[10], the filter matrix, which was used as feature identifier, denoted as $\mathbf{K}$ (e.g., weight(c_{out_j}, k)) with d = out_channels $\times \frac{in\_channels}{groups} \times$ kernel_size[0] $\times$ kernel_size[1] where, $\mathbf{K} \in \mathbb{R}^d$ while *groups* set to 1. The input matrix was denoted as $\mathbf{I}$ (e.g., input(B_i, k) along with the B-dimension). To obtain the output of the convolutional layer, which was denoted as $conv_{out} = \mathbf{I} \star \mathbf{K}$, the $\mathbf{K}$ learnable parameter are slide over the $\mathbf{I}$ matrix spatially to compute dot products. Computed dot products were summed together and they constitute each element of the output feature matrix. The parameter *stride* was used to control the stride of the cross-correlation operation, and *padding* was applied to manage the amount of zero-padding on the bout sides of the matrix for each dimension. Padding was used for keeping dimensions the same after cross-correlation operation that was applied to the input matrix. *Dilation* parameter was the spacing between kernel elements and it was set to 1. The learnable weight and bias of the convolution layer was initialized by $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{groups}{c_{in} * \prod_{i=0}^{1} kernel\_size[i]}$.

4.2.3.2 2-dimensional max-pooling

The 2-dimensional max-pooling algorithm is similar to the 1-dimensional version (Section 4.2.2.3). In this case, the 2-dimensional max-pooling layer was fed with the output of the convolutional layer denoted as $conv_{out} \in \mathbb{R}^{B \times c \times T \times F}$ where c referred to channel size. The calculation for the 2-dimensional max-pooling algorithm was indicated in the Equation (16)[11].

$$P_{out}(B_i, c_j, t, f) = \max_{m=0,\ldots,kT-1}, \max_{n=0,\ldots,kF-1} input\,(B_i, c_j, stride[0] \times t + m, stride[1] \times f + n) \quad (16)$$

[9]The illustration was referred to the one channel dimension.

[10]Deep learning for complete beginners: Using convolutional nets to recognise images, Cambridge Coding Academy

[11]The equation retrieved from official Pytorch documentation.

Where, P_{out} was denoted as max-pooling output layer. The $kT \times kF$ was denoted to the *kernel_size*. The *stride* was set to 3, and the parameter implementation idea was the same that used for the 1-dimensional version (Section 5.2.1.1). The difference from the 1-dimensional version was the max-pooling operation was applied to $\mathbf{I}$ along with the B-dimension with size $T \times F$ for each of the channel c. The output dimensions of the T and F are presented in Equation (17).

$$\begin{aligned} T_{out} &= \frac{T_{in} + 2 * padding[0] - dilation[0] \times (kernel\_size[0] - 1) - 1}{stride[0]} + 1, \\ F_{out} &= \frac{F_{in} + 2 * padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \end{aligned} \tag{17}$$

Where, T_{out} and F_{out} parameters were denoted to the output size of the time T and feature F dimensions. The *padding* was applied to add implicit zero padding on the bout sides of the input matrix $\mathbf{I}$. In this case, it was set to 1. The *dilation* was set to 1.

4.2.3.3 2-dimensional batch normalization

The 2-dimensional batch normalization algorithm was calculated with the same formulation that was used for the 1-dimensional version (Section 4.2.2.3) and it is referred to in Equation (13). The only difference was the expected input matrix should be in the form of $B \times c \times T \times F$ where, c was denoted to the channel size.

4.2.3.4 ResNetBlock Layer

The ResNet block architecture was designed to prevent the vanishing/exploding gradient problem, which occurs when the number of neural network layers are increased. In a neural network model, n hidden layers multiplies with n derivatives and if the derivatives of the weight matrices are large, the gradients will increase exponentially as we backpropagate down the model, and that will cause the *exploding gradient* problem. If the derivatives are small, the gradients will decrease exponentially, and

the *vanishing gradient* problem will occur in the neural nets. In the traditional neural network architectures, each layer feeds into the next layer. Whereas, in the residual block architecture, the *skip connection* [51] concept was introduced to prevent the vanishing/exploding gradient problem. It stacks the convolution layers as 2-3 layers before, and concurrently it adds the original input to the output of the convolution block.

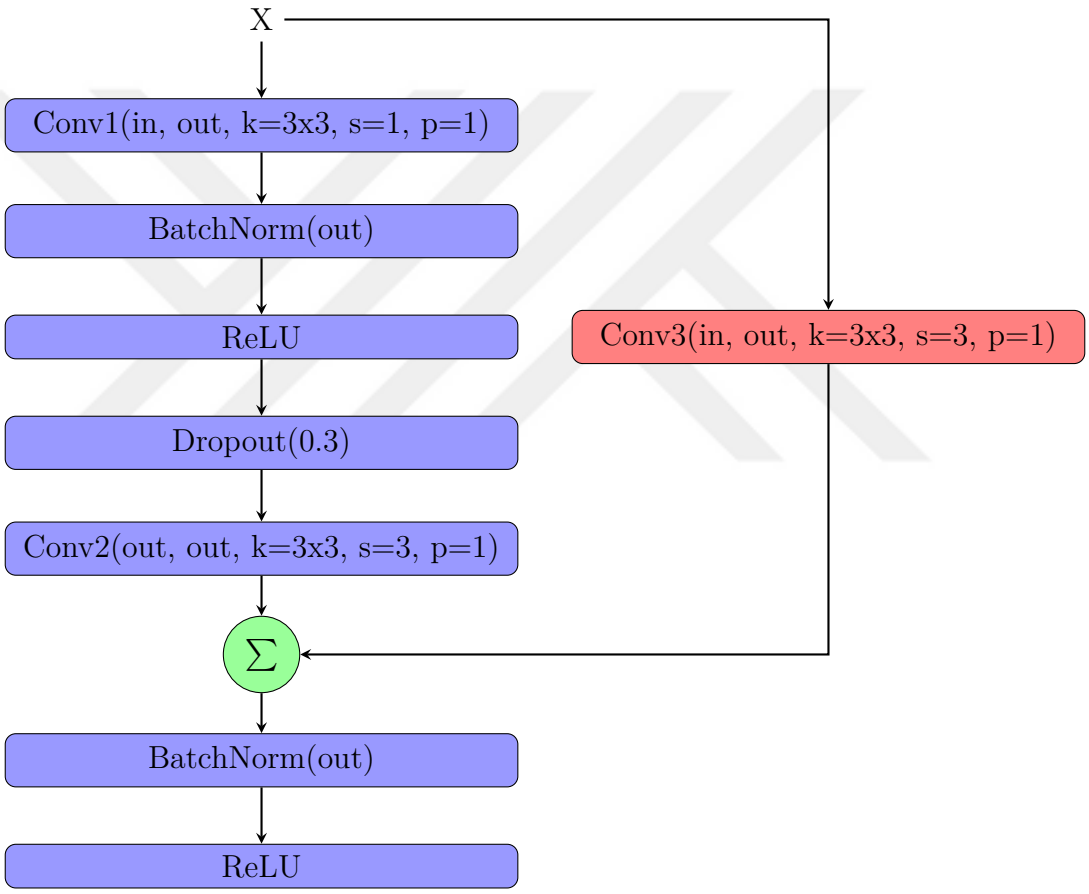

Figure 6: Detailed architecture of the ResNet block with residual connection

Figure (6) represents the ResNetBlock architecture used in this work. The parameters of the convolutional layer, which were *in*, *out*, k, s, and p, were denoted as *input dimension*, *output dimension*, *kernel size*, *stride*, and *padding*. The output layer of the ResNet block was denoted as $R_{out} = h(\text{BN}(conv_n + conv_{n-2}))$, where BN(.) was represented batch normalization operation, $h(.)$ was denoted to the ReLU activation

function and n was denoted as the number of convolutional layers.

4.2.4 Light Convolutional Neural Network Architecture

The Light Convolutional Neural Network (LCNN) architecture was designed to deal with the large-scale data, which has massive noisy samples. The study [53] which was conducted by Wu X, et al. (2017), introduced a variation of max out activation function called *Max-Feature-Map* (MFM). The traditional convolutional layer was reinterpreted by the MFM activation function to separate the noisy and informative signals from each other. The network was cautiously developed to upgrade the evaluation performance of the models while reducing their number of parameters with the MFM operation. The LCNN model was specially designed for computer vision-based tasks. Since audio signals have similar attributes with the images, the LCNN architecture became a logical option to be used with the ASV spoof 2017 dataset which was formed by a high amount of noisy samples. The LCNN model is the replicated version of the $LCNN$-9 architecture that was used in the study [53].

As illustrated detail in Table (4), the system was fed by the input matrix denoted as $\mathbf{I} \in \mathbb{R}^{B \times c \times T \times F}$, where channel size c set to 1, and it was followed by the 2-dimensional max-pooling and the dropout operations. The weight and bias are initialized by the Xavier and constant initialization algorithms mentioned. The MFM operation was developed to act as an activation function. The Adam Optimizer with parameters *epsilon=0.1*, and *learning_rate= 0.001* were used. Similar to the other architectures, the cross entropy loss and softmax functions were applied to the network.

4.2.4.1 Max Feature Map

The MFM activation function was advanced to perform as a feature filter selector for noisy datasets [53]. The MFM operation was developed to be an alternative for the

Table 4: THE ARCHITECTURES OF THE LIGHT CNN MODEL

Type	**Filter/Stride**	**prob**	**Output**	**#params**
Conv1	5×5 / 1×1	-	$32 \times 16 \times 126 \times 72$	416
MFM1	-	-	$32 \times 8 \times 126 \times 72$	-
MaxPool1	$2 \times 2/2 \times 2$	-	$32 \times 8 \times 63 \times 36$	-
Conv2a	$3 \times 3/1 \times 1$	-	$32 \times 16 \times 63 \times 36$	1.1K
MFM2a	-	-	$32 \times 8 \times 63 \times 36$	-
Conv2	$3 \times 3/1 \times 1$	-	$32 \times 32 \times 63 \times 36$	2.3K
MFM2	-	-	$32 \times 16 \times 63 \times 36$	-
MaxPool2	$2 \times 2/2 \times 2$	-	$32 \times 16 \times 32 \times 18$	-
Conv3a	$3 \times 3/1 \times 1$	-	$32 \times 32 \times 32 \times 18$	4.4K
MFM3a	-	-	$32 \times 16 \times 32 \times 18$	-
Conv3	$3 \times 3/1 \times 1$	-	$32 \times 64 \times 32 \times 18$	9.2K
MFM3	-	-	$32 \times 32 \times 32 \times 18$	-
MaxPool3	$2 \times 2/2 \times 2$	-	$32 \times 32 \times 16 \times 9$	-
Conv4a	$3 \times 3/1 \times 1$	-	$32 \times 64 \times 16 \times 9$	18.4K
MFM4a	-	-	$32 \times 32 \times 32 \times 18$	-
Conv4	$3 \times 3/1 \times 1$	-	$32 \times 48 \times 32 \times 18$	13.8K
MFM4	-	-	$32 \times 24 \times 32 \times 18$	-
Conv5a	$3 \times 3/1 \times 1$	-	$32 \times 48 \times 16 \times 9$	10.4K
MFM5a	-	-	$32 \times 24 \times 16 \times 9$	-
Conv5	$3 \times 3/1 \times 1$	-	$32 \times 48 \times 16 \times 9$	10.4K
MFM5	-	-	$32 \times 24 \times 16 \times 9$	-
MaxPool4	$2 \times 2/2 \times 2$	-	$32 \times 24 \times 8 \times 5$	-
FC1	-	-	32×512	492K
MFM5	-	-	32×256	-
Dropout1	-	0.5	-	-
FC	-	-	32×2	514
Total	-	-	-	**563.3K**

ReLU (Equation (6)) and it is a particular form of the *maxout* [54] activation function [53]. In each neural network layer, MFM operation represses the low-activation neurons to provide better classification performances over noisy signals, and by that, the function induces LCNN architecture to be robust against replay attacks. The feature maps of the convolution layers were denoted as $conv_{out}$, and $\mathbf{I} \in \mathbb{R}^{B \times c \times T \times F}$ was accepted as the input matrix. The MFM activation function was fed by $conv_{out}$ layer. The Equations (19) are referred to the mathematical formulation of the MFM operation described in [53].

$$conv_{out}^{n} = \text{conv}(in\_filter, out\_filter * 2) \tag{18}$$

$$\hat{x}_{ij}^{k} = max(x_{ij}^{k}, x_{ij}^{k+N}) \tag{19}$$

The input matrix $\mathbf{I}$ of the convolutional layer was denoted as $x^n \in \mathbb{R}^{T \times F}$, where $n = \{1, ..., 2N\}$, and $2N$ was referred to the channel size of the input convolution layer that was denoted as $(out\_filter * 2)$ in Equation (18). The conv(.) was denoted to the convolutional layer operation and the parameters in_filter and out_filter were referred to the input and output channel size. The $conv_{out_m} \in \mathbb{R}^{N \times T \times F}$ where, $m = \{1, 2\}$, was splitted into two along for the N-dimension and they were integrated with each of the feature maps to output the element-wise maximum pixels $\hat{x}_{ij}^{k}$ that were denoted in Equation (19), where $1 \leq k \leq N$, $1 \leq i \leq T$, $1 \leq j \leq F$. The MFM operation facilitates the system to obtain 50% of the informative neurons from the input feature maps by decreasing the complexity of the system.

4.3 The Application of the Bayesian Approach

In this section, the discussion will point to the uncertainty problem encountered in the prediction of the class probabilities caused by the softmax function. In the following, the Bayesian approach that was investigated to outperform the conventional cross

entropy metrics and softmax function SCS was explained in detail.

4.3.1 The Uncertainty Problem of the Softmax Classifier

The softmax function is popularly used as a classification layer for the deep neural network models. However, the softmax has deficiencies when it comes to the uncertainty measure. The work in [10] was conducted to prevent the uncertainty problem caused by the softmax function. As defined in the Equation (20), a neural network system can be defined as a multinomial distribution of the class probabilities produced by the neural network models [10].

$$Pr(y|\mathbf{x},\theta) = \text{Mult}(y|\sigma(f_1(x,\theta)),...,\sigma(f_K(x,\theta))) \tag{20}$$

Where K was denoted to the number of classes and Mult(.) was referred to the multinomial mass function. $f_j(\mathbf{x},\theta)$ was denoted to the jth output of a neural network layer $f(.)$ parameterized by θ. The $\sigma(x)$ was referred to the softmax function.

The problem of softmax function is, it squashes the estimated class probabilities of the neural network systems to form them. Thus, the output of the $\sigma(f(\mathbf{x},\theta))$, where the logits were defined as $\mathbf{x} \in \mathbb{R}^{m\times K}$, is forced to present K-dimensional outputs that are biased because of the squashed probabilities, and it inflates the estimated probability of the predicted classes [10]. Using softmax function with the cross entropy loss to minimize its squishing effects on the class probabilities is still not a practical way to prevent from these effects [10].

In [10], the LeNet system was developed by two approaches; the LeNet-Softmax (e.g., cross entropy loss) and LeNet-EDL (e.g., Bayesian approach). The LeNet-Softmax model has failed to classify the digit 1 when rotated 20%. However, the LeNet-EDL model has produced high uncertainty measures when the digit 1 was

rotated more than 20%. In this work, the Bayesian approach, which was used in work [10], was applied to decrease the uncertainty of the conventional cross entropy metrics and softmax function SCS by preventing the negative effects of the softmax function from the estimated class probabilities.

4.3.2 The Bayesian Approach

Each possible class probabilities produced by a neural network can be defined as *frame*, and it can be represented by *belief mass* [10]. The *belief* can be explained as following; the truth can be any of the possible class labels. In the Bayesian network, the summation of the total belief masses b_k over K-dimensional class labels (e.g., $\sum_{k=1}^{K} b_k$) and the *uncertainty* u were expected to equal 1, where $u \geq 0$ and $b_k \geq 0$ for $k = \{1, ...K\}$ [10]. In this work, the belief mass and the uncertainty were computed as $b_k = e_k/S$ and $u = K/S$, which is the same with the work in [10]. The S was computed as $S = \sum_{i=1}^{K}(e_i + 1)$, where i enumerates each of the evidences e_i in each sample and S indicated to the strength of the Dirichlet distribution [10].

As shown in [10], the *evidence* e was formed by replacing the softmax classification layer with an activation layer f(.) to convert neural network outputs into positive values. The e was generated using logits x as follows $e = f(x)$, where f(.) denoted to the activation layer. In this study, the activation layer f(.) was selected based on the EER performance of the SCS, and f(.) was decided as $\exp^{x_i}$.

In the early states of the learning, the neural network might fail to provide any opinion (e.g., belief mass = 0). Since uncertainty u is inversely proportional with the total evidence, if the evidence is not exist in the belief mass (e.g, $b_k = \{0, ..., 0\}$ and $\sum_{k=1}^{k} e_k = 0$), the uncertainty u equals to 0 [10]. To prevent this, similarly in [10], the Dirichlet parameters α were computed as $\alpha_k = (e_k + 1)$. Therefore, if no evidence

was produced by the neural network, the uncertainty will be equal to 1 [10].

The *support* of the Dirichlet distribution represents the sum of all estimated class probabilities that were produced by a neural network and it is expected to be equal to 1. In [10], different from the habitual terminology of the Bayesian modeling, the support of the Dirichlet distribution represented by the K-dimensional evidence e_k. The belief mass b_k was computed from the parameters of the Dirichlet distribution by $b_k = (\alpha - 1)/S$ where, $S = \sum_{i=1}^{K} \alpha_i$ [10]. As the study [10] stated, the standard neural network outputs the possible probabilities of the classes for each input sample. However, the Dirichlet distribution that is parameterized over evidence e, is able to represent the uncertainty of the belief masses from the *density* of the class probabilities.

4.3.2.1 Dirichlet Distribution

The Dirichlet distribution that was parameterized over $\boldsymbol{\alpha}$ is computed as in Equation (21) [10].

$$D(\mathbf{p}|\boldsymbol{\alpha}) = \begin{cases} \dfrac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} P_i^{\alpha_i - 1}, & \text{for } \mathbf{p} \in S_K, \\ 0, & \text{otherwise,} \end{cases} \tag{21}$$

Where, $B(\boldsymbol{\alpha})$ was denoted as K-dimensional multinomial beta function, and $\mathbf{p}$ was denoted to the probability mass function (pmf) [10]. The support of the Dirichlet distribution was defined as $S_K = \{\mathbf{p} | \sum_{i=1}^{K} p_i = 1\}$, and $0 \leq p_1, ...p_K \geq 1$. In the study [10], the mean of the Dirichlet distribution was referred to the expected probability of the each k-dimensional class value. The mean $\hat{p}_k$ and the variance $\text{Var}(\alpha_k)$ of the Dirichlet distribution were presented with Equation (22) [10].

$$\hat{p}_k = \frac{\alpha_k}{S} \qquad \text{Var}(\alpha_k) = \frac{\alpha_k(S-\alpha_k)}{S^2(S+1)} \tag{22}$$

Where, alpha α was parameterized by the Dirichlet distribution and computed as the following $\alpha_i = f(\mathbf{x}_i|\Theta)+1$ where, Θ denoted to the network parameters [10]. The y_i was referred to the ground truth label in the one-hot vector encoding form where $y_{i,j} = 1$ and $y_{i,j} = 0$ for all $k \neq j$.

4.3.2.2 EDL Loss Function

Similar to the work in [10], the EDL loss function was defined as Mean Square Error $||y_i - p_i||^2$ and its Bayes risk was computed with respect to the class predictor $\mathbf{p}_i$ and presented in Equation (23).

$$\begin{aligned} L_i(\Theta) &= \int ||y_i - p_i||^2 \frac{1}{B(\alpha_i)} \prod_i^K \mathbf{p}_{ij}^{\alpha_{ij}-1} d\mathbf{p}_i \\ &= \sum^K \mathbb{E}\left[y_{ij}^2 - 2y_i p_{ij} + p_{ij}^2\right] = \sum_{j=1}^K \left(y_{ij}^2 - 2y_{ij}\mathbb{E}[p_{ij}] + \mathbb{E}[p_{ij}^2]\right) \end{aligned} \tag{23}$$

The interpretable form of the Equation (23), was achieved by the decomposition of the first and second moments of the Equation (24) [10].

$$\mathbb{E}[p_{ij}^2] = \mathbb{E}[p_{ij}]^2 + \text{Var}(p_{ij}) \tag{24}$$

$$\begin{aligned} L_i(\Theta) &= \sum_{j=1}^K (y_{ij} - \mathbb{E}[p_{ij}])^2 + \text{Var}(p_{ij}) \\ &= \sum_j^K \underbrace{(y_{ij} - \frac{\alpha_{ij}}{S_i})}_{L_{ij}^{err}} + \underbrace{\frac{\alpha_{ij}(S_i - \alpha_{ij})}{S_i^2(S_i+1)}}_{L_{ij}^{var}} \end{aligned} \tag{25}$$

The form of the EDL loss that presented in Equation (25) is achieved by using the mean and the variance of the Dirichlet distribution (Equation (22)). As referred in

[10], the EDL loss was aimed to minimize the prediction error L_{ij}^{err} and the variance L_{ij}^{var} of the Dirichlet parameters for each of the samples, which were produced by the model.

In our work, the cross entropy loss function was replaced by the interpretable form presented in the Equation (25) to replicate the work in [10].

4.3.2.3 Kullback-Leibler (KL) Regularization

As the work in [10] indicated, the neural network models might be deceived by the similar patterns, which were recognized in each of the training samples (e.g., the similar shapes of digit zero and digit six in the MNIST dataset). As a result, the deceived network might end up producing similar evidence results for similar samples [10]. Therefore, the loss function was penalized with the Kullback-Leibler (KL) divergence term by using the estimated class probability values that did not correspond to the true label but others. By virtue of the fact that, the total evidence will shrink to zero for each sample determined as misclassified by the system. The EDL loss function with the KL regularization term is stated in Equation (26) [10].

$$L(\Theta) = \sum_{i=1}^{N} L_i(\Theta) + \lambda_t \sum_{i}^{N} KL[\tilde{\boldsymbol{\alpha}}_{\boldsymbol{i}}] \tag{26}$$

Where, $\lambda_t = min(1.0, (t/10))$ was denoted as *annealing coefficient* and t was referred to each epoch [10]. The Dirichlet parameters were computed as $\tilde{\boldsymbol{\alpha}}_{\boldsymbol{i}} = (1 - y_i) \odot (\tilde{\boldsymbol{\alpha}}_{\boldsymbol{i}} - 1) + 1$ to obtain only the misleading evidence scores [10]. The KL divergence term was computed by the Equation (27) [10].

$$\begin{aligned} KL[\tilde{\boldsymbol{\alpha}}_{\boldsymbol{i}}] = \Gamma\left(\sum_{k=1}^{K} \tilde{\alpha}_{ik}\right) - \sum_{k=1}^{K} \Gamma(\tilde{\alpha}_{ik}) + \sum_{k=1}^{K} \Gamma(\beta_{ik}) - \Gamma\left(\sum_{k=1}^{K} \beta_{ik}\right) + \\ \sum_{k=1}^{K} (\tilde{\alpha}_{ik} - 1) \left[\psi(\tilde{\alpha}_{ik}) - \psi\left(\sum_{j=1}^{K} \tilde{\alpha}_{ij}\right)\right] \end{aligned} \tag{27}$$

Where, $\psi(.)$ was referred to the *digamma* function and $\Gamma(.)$ was denoted to the *gamma* function. Beta was stated as $\beta_{ik} \in \mathbb{R}^{1\times K}$.

CHAPTER V

EXPERIMENTS

5.1 General Information About the ASVspoof Challenge Dataset

5.1.1 The RedDots Dataset and the Replayed Version

The intention of the *RedDots project* was to research speaker recognition under conditions where test utterances have a *short duration* and of *variable phonetic content* [43]. The RedDots dataset was prepared from the samples of the native and non-native English speakers [43]. The collection was specially designed to seize the *speaker-extrinsic* and the *speaker-intrinsic* variations of the speakers. In order to collect the corpus, the speakers recorded once a week over a year and the RedDots corpus expected to gather important amount of *inter-speaker* and *intra-speaker* variations [43] for the collection. The participants were asked to read a similar set of sentences in each session, and the recordings were collected by using Android smartphones.

The ASV challenge organizers have introduced the *replayed version* of the RedDots dataset to analyse the vulnerabilities of the SCSs against replay attacks. The dataset was developed to improve the replay SCSs and also, to protect the text-dependent ASV systems from replay attacks against various challenging recording and playback conditions [8]. The replay attack conditions were simulated by using the re-recordings of the original RedDots dataset. One of the concerns of SCS is that it learns the trained dataset extensively and fails to *generalize* the different test cases. Thereby, the RedDots Replayed dataset, specially prepared to achieve *'out in the wild'* conditions. The main objective was to generate a dataset that can not be generalized by deep neural network systems. To achieve this, the spoofed utterances

were produced by replaying and recording the genuine samples (e.g., RedDots corpus) in various acoustic environments and by using heterogeneous devices.

In this thesis, the ASVspoof 2017 corpus version 2.0 (ASVspoof-2017-V2) [20] dataset is used. The dataset is composed of three subsets; train, development, and evaluation. Each of the datasets are prepared using different recording and replay devices in different environments to simulate the replay attacks that may occur in the real world. Detection difficulty of the replay attacks are determined based on the acoustic environment, replay devices, and recording devices. Some of the high-end loudspeakers are used to develop replay attacks to provide various challenging samples for the SCSs. Detailed information about the acoustic environments, recording devices, and replay devices are described in the following section.

5.1.2 Meta-data of the Replay System

5.1.2.1 Replay Attack Simulation

As stated in [8, 7], the *ASVspoof*2017 challenge considers the worst-case scenario where the attackers have the digital copy of the genuine samples. In Figure 7 [8] the replay spoofing framework is illustrated according to these scenarios. In the initial step, the attacker has acquired **covert recordings** of the target speaker without their approval and then the recordings were replayed in the existing room by a **playback device**. The replay attacks were obtained by the **recording device** and they were directed to the ASV system. Subsequently, the **authentic recording device** has captured the real speaker's speech from another physical space. In this case, the genuine samples were obtained from the RedDots dataset. Finally, the ASV systems' microphone received genuine and spoof samples to verify them.

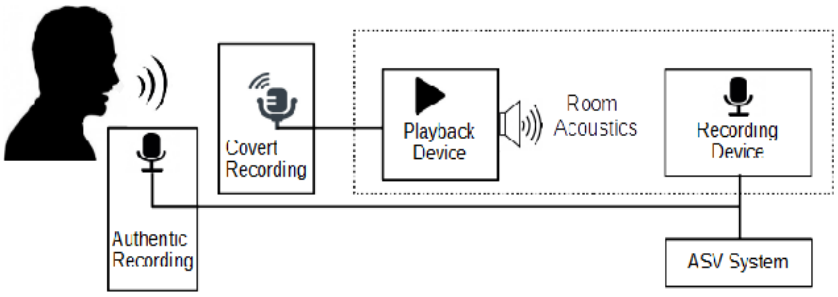

Figure 7: The Automatic Speaker Verification System used by the ASV spoof 2017 challenge illustrated

5.1.2.2 Data Partitions

The ASV spoof 2017 challenge consists of three subsets; **train**, **development** and **evaluation**. A summary of their composition is provided in the Table 5. The statistics indicate that the dataset advanced with 177 different replay sessions and 61 diverse replay configurations. The number of the replay sessions and replay configurations used in the evaluation subset is much larger than the total training and development division [20]. And also, in the evaluation dataset 24 diverse speakers were used. The evaluation dataset is designed to be rich in replay attack variations compared to the other subsets.

Table 5: THE STATISTICS OF THE ASVSPOOF-2017 VERSION-2.0 DATASET

Subset	# Spk	# Replay Sessions	# Replay Config	# Utterance	
				Bona fide	Replay
Train	10	6	3	1507	1507
Development	8	10	10	760	950
Evaluation	24	161	57	1298	12008
Total	42	177	61	3565	14465

5.1.2.3 Acoustic Environments, Replay Devices, Recording Devices, and Replay Configurations

The most important dissimilarity between the train, development, and evaluation subsets are determined by the impact of acoustic environments, replay devices, and

recording devices. The **acoustic environments** [20] define the physical space where the audio recordings were replayed and re-recorded by a specific device. The ASV spoof 2017 challenge dataset was produced with 26 different acoustic environments, and each of them are listed in Table 6. The color codes that are presented in Table (6) represent the replay attack threat levels as low (green), medium (yellow), and high (red). The *balcony* and *cantine* acoustic environments were used to simulate the noisy places which contain traffic or random background speaker voices and these types of recordings are considered as easily detectable. On the contrary, *office* environments contain fewer background noises and they are relatively harder to detect. The *anechoic* room and the *studio* recordings contain very low reverberation and background noises. The *analog wire* recordings were generated without using any physical sound extensions. However, they were produced by an electrical propagation that required a form of playback device directly connected to the ASV system [20]. These types of acoustic environments produced samples that were the most challenging to detect.

There are 26 **playback devices** and each of them are [20] listed in the Table (7). The replay samples were collected from consumer-grade devices (e.g., smartphones, laptops, and tablets). In general, the small loudspeakers have lower quality than other devices, and so they are easier (green) to detect by the SCSs. The playback devices which have larger loudspeakers (e.g., PCs with an external loudspeaker) produce higher quality sounds than the small ones and they are harder to detect (yellow) by the SCSs. The professional audio equipments (e.g., active studio monitors and studio headphones) produce the highest quality spoof samples compared to the other devices, as a result they formed the hardest (red) input samples to detect.

The total number of the **recording devices** were used in the challenge is 25

and they are listed in Table 8 [20]. The samples were collected from miniature microphones, mobile devices, and tablets that have low recording quality and these samples were referred to as easiest to detect (green) by SCSs. On the other hand, the studio-quality microphones or quality hand-held recorders have better recording technology, and they are harder to detect (red) by the SCSs. The desktop devices, headsets, or webcams produce medium (yellow) quality replay attacks.

The ASV dataset consisted of 61 different **replay configurations** (RC) [20]. Each of the replay configurations were formed with three features which were playback devices P, acoustic environments E and recording devices R. The **RC** can be defined as $\text{RC}_c = (E_i, P_j, R_k)$, where c enumerates all unique tuples (i, j, k). The i, j, and k enumerates the number of acoustic environments, playback devices and recording devices. Among the $16,900$ (e.g., $26 \times 26 \times 25$) replay configuration possibilities [20], the ASV challenge organizers reduced this number by grouping the overlapping configurations. Also, not all the device/environment combinations were available to use together. The detailed table of the **RC** is illustrated in the study [20].

Table 6: ACOUSTIC ENVIRONMENT

ID	Environment	ID	Environment
E01	Anechoic room	**E14**	Office 02
E02	Balcony 01	**E15**	Office 03
E03	Balcony 02	**E16**	Office 04
E04	Home 07	**E17**	Office 05
E05	Home 08	**E18**	Office 06
E06	Cantine	**E19**	Office 07
E07	Home 01	**E20**	Office 08
E08	Home 02	**E21**	Office 09
E09	Home 03	**E22**	Office 10
E10	Home 04	**E23**	Studio
E11	Home 05	**E24**	Analog wire 01
E12	Home 06	**E25**	Analog wire 02
E13	Office 01	**E26**	Analog wire 03

Table 7: PLAYBACK DEVICE

ID	Playback device
P01	All-in-one PC speakers
P02	Creative A60 speakers
P03	Genelec 8020C studio monitor
P04	Genelec 8020C studio monitor (2 speakers)
P05	Beyerdynamic DT 770 PRO headphones
P06	Dell laptop internal speakers
P07	Dynaudio BM5A speaker
P08	HP Laptop internal speakers
P09	VIFA M10MD-39-08 speaker
P10	ACER netbook internal speakers
P11	BQ Aquaris M5 smartphone
P12	Logitech low quality speakers
P13	Desktop PC line output
P14	Labtec LCS-1050 speakers
P15	Edirol MA-15D studio monitor
P16	Lenovo Ideatab S6000-H tablet
P17	Logitech S120 multimedia speakers
P18	MacBook pro internal speakers
P19	Altec lansing Orbit USB iML227 portable speaker
P20	Samsung GT-I9100 smartphone
P21	Samsung GT-P6200 tablet
P22	Behringer Truth B2030A studio monitor
P23	Focusrite Scarlett 2i2 audio interface line output
P24	Focusrite Scarlett 2i4 audio interface line output
P25	Genelec 6010A studio monitor
P26	AKG K242HD Headset

Table 8: THE RECORDING DEVICES USED IN THE ASV DATASET

ID	Recording device
R01	Zoom H6 handy recorder
R02	BQ Aquaris M5 smartphone
R03	Low-quality headset
R04	Nokia Lumia 635 smartphone
R05	Røde NT2 microphone
R06	Røde smartLav+ microphone
R07	Samsung Galaxy S7 smartphone
R08	Desktop PC microphone input
R09	Zoom H6 recorder with Behringer ECM8000 mic
R10	Zoom H6 recorder with MSH-6 microphone
R11	Zoom H6 recorder. with XY microphone
R12	iPhone 5c smartphone
R13	iPhone 7 plus smartphone
R14	iPhone 4 smartphones
R15	Logitech C920 webcam
R16	miniDSP UMIK-1 microphone
R17	Samsung Galaxy Trend 2 smartphone
R18	Samsung GT-I9100 smartphone
R19	Samsung GT-P6200 tablet
R20	Samsung Trend 2 smartphone
R21	AKG C3000 microphone
R22	SE electronic 2200a microphone
R23	Focusrite Scarlett 2i2 interface line input
R24	Focusrite Scarlett 2i4 interface line input
R25	Zoom HD1 handy recorder

5.2 Preprocessing of the Dataset

5.2.1 Signal to Noise Ratio of the ASV spoof Dataset

The replayed version of the RedDots dataset was generated by using replay configurations (Section 5.1.2.3) to simulate the replay attacks that occur in the wild conditions. These replay configurations include unwanted noise signals in the audio recordings, and these noise signals have an adverse impact on the learning process of deep learning models [55]. All of the train, development, and evaluation subsets were produced by different amounts of low/high dB replay attacks. To overcome the negative effects of noisy signals, the signal-to-noise ratio (SNR) was calculated for each sample and utilized as side knowledge in the decision logic. The louder sounds have higher dB values, and they are easier to detect by the human ear. A ratio smaller than 0 dB refers to more noise than the signal, and a ratio greater than 0 dB refers to more signal than noise. In general, the SNR scores, which were denoted as r, of the less clear recordings perform in between 0 dB and 15 dB. The values between 15 dB to 40 dB accept as virtuous recordings, and the intensity of the clearest recordings expect to be higher than 41 dB.

Table 9: THE SNR RESULTS OF THE ASV DATASET

Datasets	-9.36$\leq r >$15.43	15.43$\leq r >$39.94	39.94$\leq r \leq$61.42	**Mean**
Train	2453	559	2	**10.07**
Development	1527	182	1	**8.19**
Evaluation	8086	5082	134	**13.92**
Total	**12,066**	**5,823**	**137**	

As the Table 9 indicates, the evaluation dataset has less noisy speech signals compared to the train and development subsets. The 81% of the training dataset consists of the speech recordings that contain a high amount of background noise, as well as 89% of the development dataset consists of noisy samples. On the contrary, the evaluation dataset consists of 61% noisy recordings. Almost all of the best quality

recordings given with the evaluation dataset, and these samples are harder to detect by the neural nets. An educated guess made for the observations related to the r scores, which are the systems trained with a noisy dataset and tested by more quality samples.

In [56], the discussion centers on how using several recording and replay devices affect the speech signal and how it contributes to the overfitting risk. Lantian et al. (2017) used the F-ratio probing tool to analyze the effects of using several factors: the speaker identity, speech content, playback, and recording device. The study concluded that the devices were the ones that contributed to the most of the generalization problem that occurs in the ASV spoof 2017 dataset. The intensity, high/low frequency, and background noise dissimilarities of the professional (spoof left) and less quality (spoof right) devices exposed by the spectrograms that were indicated in Figure 8. The effects of devices on the audio signals can be observed clearly. The spoof samples that are denoted as easily detectable (up right), has a clear appearance compared to the harder ones (up left). The genuine samples are (below left and right) vocalized by the same person using the same sentences in different environmental conditions.

5.2.2 Data Preprocessing

The deep neural network architecture performs better with the equal-size inputs. However, each of the audio recordings in the ASV spoof 2017 dataset has a different duration. Therefore, the padding was applied to equalize their wavelengths into 64,000. If the audio recording has a longer wavelength (e.g., more than 64,000 wavelength), they were shrunk by taking the last $64,000$ wavelength of the input signals. This method has provided a better EER score performance than concatenating all the input samples into one matrix to separate them into a fixed size (e.g., 100) to

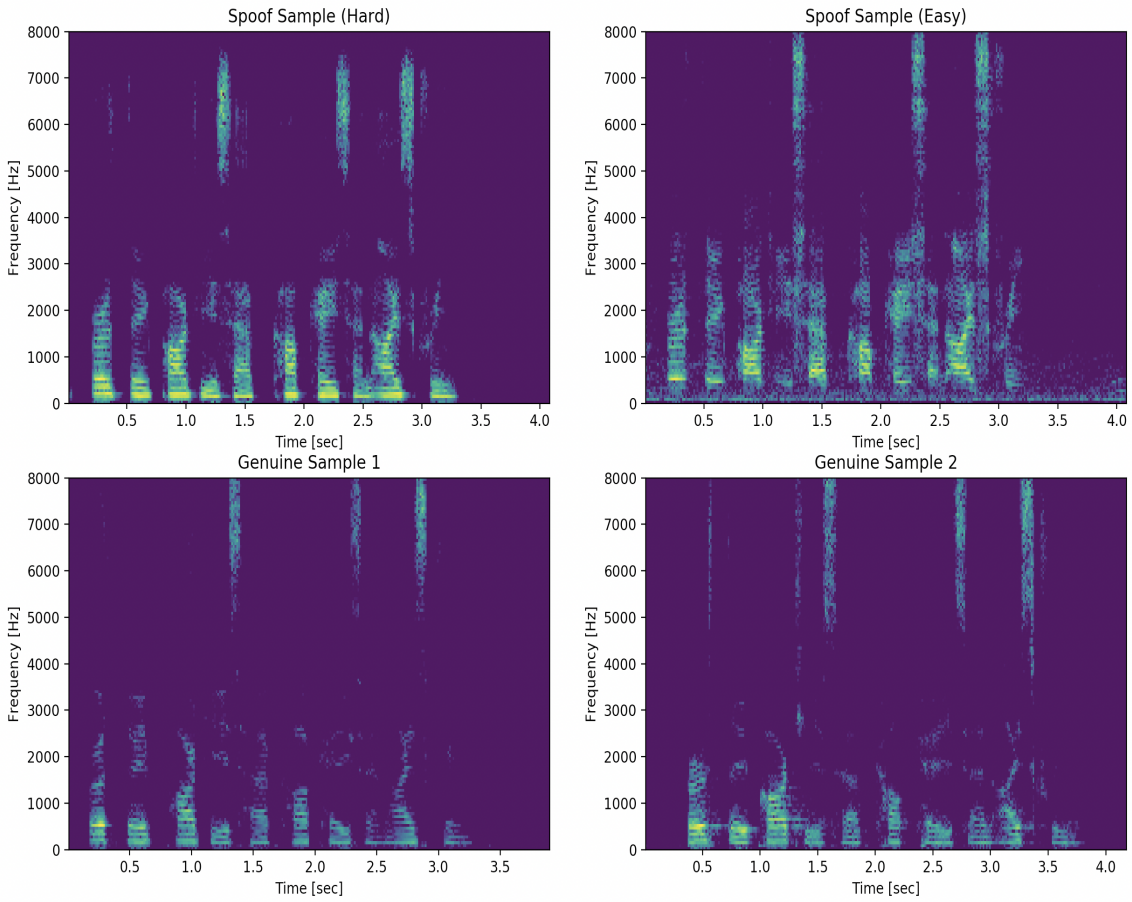

Figure 8: The spectrogram of genuine and replayed audio (spoof) of the vocalization "Birthday parties have cupcakes and ice-cream" from the same speaker. The hard detectable spoof sample (top left) produced by "Edirol MA-15D studio monitor" replay device, " iPhone 7 plus smartphone" recording device and "Anechoic room" environment. The easy detectable spoof sample (top right) produced by "Samsung GT-P6200 tablet" replay device, "Zoom HD1 handy recorder" recording device and "Balcony 01" environment. Genuine sample 1 (below left) and genuine sample 2 (below right) taken from the same speaker. The spectogram created by the transformation of amplitude to dB.

form equal-length matrices.

After forming the fixed-size speech signals, by using the information obtained from the Section 5.2.1, the *mel-frequency cepstral coefficients* (MFCC) is used to transform the audio signals into the time T and feature F matrices. The application details are provided in Section 5.2.3.

Finally, before the $\mathbf{X}$ is given to the feed-forward system, it is normalized by the zero-mean and unit-variance normalization algorithm. The mean μ and variance σ^2 of the all samples $\mathbf{X} \in \mathbb{R}^{T \times F}$ computed with the Equations (28) and (29), where T and F referred to time and feature. N denoted to the total number of the input samples. $\mathbf{X}$ is formed by concatenating each sample into the one matrix form.

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{28}$$

$$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N} \tag{29}$$

Where, $x_i \in \mathbb{R}^F$ was denoted to the sample input vector and i enumerates the total number of the samples. The zero-mean and unit-variance normalization was applied to the input matrix $\mathbf{X}$ with the Equation (30).

$$x_i' = \frac{x_i - \mu}{\sigma^2} \tag{30}$$

Where, i was enumerated each samples in $\mathbf{X}$.

5.2.3 Mel-Frequency Cepstral Coefficients

The Mel-frequency cepstral coefficients (MFCC) [57] were introduced in the early 1980s to be used in the speech recognition systems and after that, it was frequently used for the speaker recognition systems [12]. The MFCCs are the representation of

the short-term power spectrum of the sound signals. They are computed by using the mel-filterbanks of the spectrums and the discrete cosine transform (DCT) of the log-filterbank energies. To generate a robust SCS, extracting features using the MFCC algorithm is recommended by the study [12]. The dimension of the static features are commonly chosen from the small numbers such as 13, 20, or 30 [58]. In our case, setting the MFCC feature dimension into 24 provided the best results. To integrate additional information into the feature values of the MFCC outputs, which also suggested by the study[12], the 1st (*delta* Δ) and 2nd (*double-delta* Δ^2) order time derivatives of the MFCC sequences were taken. Instead of using the 24-dimension MFCC features F, after this method the F values increased to 72. This resulted with the improvement of classification in the system.

5.2.4 Experimental Setup for the Neural Network Architectures

The MFCC algorithm was applied to all the neural network models: FNN, RNN, ResNet, and LCNN. All systems were trained using the train and development subsets. Training was implemented with the Pytorch library. The batch size is set to 32 and all systems were normalized using zero-mean and unit-variance normalization operation. All of the models were trained for 50 epochs, and the epoch that performed the best EER score is taken as a result.

5.3 Performance Metrics

5.3.1 Equal Error Rate

The *equal error rate* (EER) is a point where the false acceptance rate (FAR) and the false rejection rate (FRR) intersect. The false acceptance ratio (FAR) and the false rejection ratio (FRR) are inversely proportional to each other. The ideal system is accepted to be able to produce 0% EER while keeping a good balance between these factors. The EER is accepted as the official evaluation metric for the $ASVspoof2017$ challenge to provide an *threshold-free* score system. The input matrix $x_{ij}^k \in \mathbb{R}^{B \times k}$, and

true labels $L_{ij} \in \mathbb{R}^B$ along with the k-dimensional class given as the input parameters where, $i = \{1, ..., N\}$, $j = \{1, ..., B\}$. The N and B denoted as the total number of the samples and the batch size. The EER calculation is provided in Equation (31) [28].

$$\begin{aligned} \mathbf{P}_{far}(\theta) &= \frac{\#\{\text{spoof trials with score} > \theta\}}{\#\{\text{total spoof trials}\}}, \\ \mathbf{P}_{frr}(\theta) &= \frac{\#\{\text{human trials with score} \leq \theta\}}{\#\{\text{total genuine trials}\}} \end{aligned} \tag{31}$$

Let denote FAR and FRR as $\mathbf{P}_{far}$ and $\mathbf{P}_{frr}$ at the threshold θ, The θ_{EER} refers to the EER threshold where, $\mathbf{P}_{far}(\theta_{EER}) = \mathbf{P}_{frr}(\theta_{EER})$. The *countermeasure scores* $\mathbf{S}_{ij} \in \mathbb{R}^B$ calculated as $\mathbf{S}_{ij} = x_{ij}^{n=1} - x_{ij}^{n=2}$. The *bob.measure.eer_rocch* library is applied to compute the (EER) outputs. The (EER) calculation of the *bob.measure* library is replicated from the *Bosaris toolkit* [59], which calculates the EER score based on the convex hull (ROCCH-EER) and it is accepted as the official library to calculate EER for the $ASVspoof2017$ challenge. Finally, *eer* calculated by $\frac{\sum_i eer_i}{total}$ where, *total* is the total number of the *eer* scores.

5.3.2 Probability Distribution Function (PDF)

The *probability distribution function* (PDF) is a statistical function that is used to describe all the possible probabilities and likelihood values that a random variable possibly takes. PDF is used to analyze the estimated class probability scores of the neural networks. The logits were given to the PDF where, $x_{ij} \in \mathbb{R}^{B \times k}$, $i = \{1, ..., N\}$, and $j = \{1, ..., B\}$. B were referred to the batch size, and N were denoted to the number of samples over k-dimensional class. The dimension of the logits scores x_{ij} were reduced as $x_{ij} = x_{ij}^{n=1} - x_{ij}^{n=2}$, where $x_{ij} \in \mathbb{R}^{B \times 1}$, and then given to the $h(x_{ij})$ to convert them into the sigmoid (Equation 32) outputs to extinguish negative logits scores.

$$h(x_{ij}) = h\left(\frac{1}{1+e^{-x_{ij}}}\right) \tag{32}$$

In this work, the sigmoid outputs were separated into true classified and misclassified samples, and these samples were given to the PDFs to analyze the uncertainty of the models. The direction of the estimated class probabilities towards their true labels were accepted as a measurement for the uncertainty of the models.

5.3.3 ROC Curve and AUC

The *Receiver Operating Characteristics* (ROC) Curves determines the trade-off between the true-positive *tpr* and the false-positive ratios fpr to measure the prediction performance classifiers against various thresholds. The Area Under the ROC Curve (AUC) determines the quality of the estimated class probabilities of the predictive models without considering the applied threshold. The ROC and AUC operations are used to compare the uncertainty performances of the neural networks by using their estimated class probabilities. The logits matrix $x_{ij} \in \mathbb{R}^{B\times k}$ over is given as the input over the k-dimensional class where, $i = \{1, ..., N\}$, $j = \{1, ..., B\}$. The $miss_{cls}$, and $true_{cls}$ parameters denoted as $m_{ij}^k \in \mathbb{R}^{B\times k}$, and $t_{ij}^k \in \mathbb{R}^{B\times k}$. The Equation (33) refers to the calculation of the m_{ij}^k, and t_{ij}^k.

$$\begin{aligned} true_{cls} &= \begin{cases} x_{ij}, & \text{if } x_{ij} = L_{ij} \\ \text{None}, & \text{otherwise} \end{cases} \\ miss_{cls} &= \begin{cases} x_{ij}, & \text{if } x_{ij} \neq L_{ij} \\ \text{None}, & \text{otherwise} \end{cases} \end{aligned} \tag{33}$$

The roc curve computed by the Equation (34).

$$fpr, tpr = \mathbf{R}(\mathbf{X},\ \mathbf{L}) \tag{34}$$

Where, the $\mathbf{X}_{ij}^{k} \in \mathbb{R}^{B \times k}$ is formed by concatenating the m_{ij}^{k}, and t_{ij}^{k} parameters. The $\mathbf{L}_{ij}^{k} \in \mathbb{R}^{B \times k}$, which is denoted to the true labels of the x_{ij}^{n}, and it generated by the concatenation of 0 and 1 repeated as the same size with m_{ij} and t_{ij}. The fpr, and tpr refers to false positive ratio, and true positive ratio parameters. The area under the curve is computed by using the parameters fpr, and tpr. $\mathbf{R}(.)$, and AUC methods are calculated using the *sklearn.metrics* library.

5.3.4 Signal to Noise Ratio (SNR)

The signal-to-noise ratio (SNR) is the ratio between the desired signal power (e.g., the information) and the undesired signal power (e.g., background noise), and it expresses in decibels (dB). Accordingly, the SNR algorithm was applied to measure the noise in each of the audio recordings which comes with the *.wav* format. In general, the prerecorded audio samples have the possibility to contain noisy signals at the beginning and end of the recordings. Correspondingly, the noisy signals were discovered at the first and last 30ms of each audio recording, and these signals were used in the SNR calculation as a noise signal. The averages of the first p_{beg} and last p_{end} noise signals are calculated in the Equation (35).

$$p = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{n} | \text{ fft}(w \odot x_j) |^2 \tag{35}$$

Where, $x_j \in \mathbb{R}^{L}$ and j were enumerated the each frame which was the result of the fft(.) applied on a window taken from the input signal $X \in \mathbb{R}$. The sample rate of each speech signal was 16,000 and the *window size* L was set to 10ms. The x_j made element-wise multiplication with the *hamming window* denoted as w. The window applied on x_j by skipping every 5ms. To transform time signal into frequency, the *Fast Fourier Transform* method was used and it was denoted as fft(.) and i was enumerated the number of the each frames' noise energy. After calculating the p_{beg} and p_{end}, to obtain the average noise m_p the Equation (36) is used.

$$m_p = \frac{p_{beg} + p_{end}}{\text{noise\_frame}} \tag{36}$$

Where, the noise_frame was denoted to the total number of the noise frames. The remaining of the input signal X was given to the Equation (37) as x_j to compute speech signal $s \in \mathbb{R}$.

$$s = \sum_{j=1}^{n} \mid \text{fft}(w \odot x_j) \mid^2 \tag{37}$$

Thereafter, to compute SNR score in Equation (38), the noise m_p subtracted from the noisy speech signal s for getting the speech signal without noise.

$$SNR = \frac{1}{N} \sum_{i=1}^{N} 10 * \log 10 \left(\frac{|s - m_p|}{m_p} \right) \tag{38}$$

Where, i was enumerated the number of the audio recordings.

5.4 Training Process

5.4.1 Environment

In this work, the Ubuntu operating system was used with version 16.04. To speed up the training process, the NVIDIA GeForce GTX 1080 was used as the Graphics Processing Unit (GPU) along with the NVIDIA driver with version 440.33.01. Due to a compatibility issue, the CUDA Toolkit with version 10.2 was applied with the proper CuDNN library. Each neural network was generated using the pytorch library with version-1.4 and the python-3.5 environment.

5.4.2 Architecture Selection

5.4.2.1 Fully Connected Network Architecture

To develop the FNN model, at first, the architecture was generated by using only 3 fully-connected layers. Then, based on the research [60] conducted by Huy (2016), which supports the positive effects of using 1-dimensional max-pooling to the noisy

audio signals, a 1-dimensional max-pooling layer was applied as a feature extractor. Also, the studies [61, 62], which had the best two EER results in the ASV spoof 2017 challenge, utilized max-pooling in their base models to improve their performances. After the 1-dimensional max-pooling operation, the EER score of the FNN was decreased. To experiment only with the fully-connected layers, any convolutional layer before the 1-dimensional max-pooling operation was not included. In general, the highly complex neural network models do not work well with the ASV spoof 2017 dataset and since the FNN network has the highest number of the parameters, 1-dimensional max-pooling layer was used to decrease the complexity of the network. In order to prevent overfitting, the 70% of the outputs of fully-connected layers were eliminated by using a dropout operation [46]. Compared to the sigmoid, the ReLU was performed much better as an activation function. The FNN was failed to perform well with a high number of the neuron sizes. When the neuron sizes were increased to 2000, 3000 or more, the performance of the network was decreased also with the EER score. Therefore, the highest node size was determined as 1024 and for the other layers in the system the dimension was reduced by half of it. The node size of the last hidden layer was set to 128 instead of 256 due to a small decrease in the EER score. The FNN model was able to outperform the EER score of the base model of the challenge.

5.4.2.2 Recurrent Neural Network Architecture

The RNN architecture is similar to the FNN model except for a few differences. Another approach they have used to improve the robustness of the SCSs was to employ a Bi-LSTM layer. In [63], the GMM and Bi-LSTM models were used to generate a robust SCS and they have accomplished the 5th best EER performance of the ASV challenge 2017. Also, in [64], they have utilized the Bi-LSTM layer to generate a SCS for the ASV spoof 2017 challenge. According to the study [65], which was conducted

by using the TIMIT [1] a speech dataset stated that the Bi-LSTM based neural network model was able to outperform both of the GMM and the DNN neural networks. Another study [66] applied Bi-LSTM based model to improve the performance of their base system by using the same dataset.

Accordingly, the 1-dimensional max-pooling layer was replaced with a bidirectional long short-term memory (Bi-LSTM) layer to be used as a feature extractor. That directly increased the classification performance of the system by decreasing the EER score. The study [67] suggests to apply a dropout operation to the LSTM layers by only implementing them to the non-recurrent connections. Therefore, in our system, the dropout was not applied to the outputs of the each Bi-LSTM layer to prevent its memorization ability to corrupt the information carried by units. The number of hidden layers that applied to the Bi-LSTM layer selected according to its effect on the performance of the EER score. According to the findings, the 8-layers presented the best classification performance.

Another approach was done by applying the batch normalization [49] operation to the RNN system. The study [49] states that the batch normalization decreases the internal covariate shift of the input samples and provides higher learning performances. Also, the same study suggests not to use the batch normalization with the dropout operation, in case it decreases the performance of the batch normalization on input samples. As another study [68] stated, combining both operations reduces the error of the network and improves classification performance for some applications. However, the study suggests to apply the dropout operation after the batch normalization layer by using a small probability value. Another study [69] stated that

[1]TIMIT is a speech dataset that has recordings taken from 680 speakers which consist of 8 American and English speakers. The speakers were reading ten various sentences in a noise-free environment.

the dropout reduces the effects of overfitting in speech and object recognition tasks. According to the related studies, the batch normalization layer was applied to each fully connected layer before the activation function. Also, the dropout probability was decreased from 70% to 50%. The default value of the epsilon was changed to 0.001 to attain a better performance from the batch normalization. To decrease the complexity of the network the highest node size of the fully-connected layer was set to 512. The Rest of the neuron sizes of the network were determined by decreasing the 512 by half. The final form of the RNN system succeeded to outperform the FNN model.

5.4.2.3 ResNet Architecture

Another preferred neural network architecture by the ASV challenge participants was the ResNet model. In [70], the best model was a fusion system composed of the MFCC-ResNet and the CQCC-ResNet networks. With this model, they achieved the 3rd best EER score in the challenge. Another study [64] stated that the ResNet reveals a promising potential for an end-to-end SCS. Therefore, the MFCC-ResNet model which was proposed in [52] was applied as the ResNet architecture. The model was developed to be used in the ASV spoof 2019 challenge, which studies the three major forms of the spoofing attacks: speech synthesis, voice conversion and replay attack. As stated in [52], the ASV Spoof 2019 dataset has more realistic replay attack samples compared to version-2017.

In this study the MFCC was used as a feature extractor. Therefore, the obtained model from the study [52] was the MFCC-ResNet architecture for this work. First, the retrieved model was applied with the same architecture that was used in the related study. However, because of the number of the ResNet block layers (e.g., high complexity), it has performed worse than the RNN system. Therefore, the number

of the ResNet blocks were reduced from nine to three. Then, the Leaky Relu was replaced with the ReLU activation function. Some of the max-pooling operations were causing an unnecessary dimension reduction for the input matrix, and this was causing an critical information loss during the backpropagation process. At first, the last two max-pooling operations were canceled and this has caused a small improvement in the EER scores. And then, the stride values of the two max-pooling operations were reduced from 3 to 1 instead of canceling them. That presented a better classification improvement than canceling the max-pooling layers. In the studies [52, 61], the probability of the dropout values were determined as 70%. Therefore, for this architecture, the probability of the dropout operation was set to 70%.

The ResNet block was kept similar to the original. The channel sizes of the convolutional layers were kept small. The maximum channel size was determined as 32 for not to increase the complexity of the network so much. When the channel size was set into 64, no significant performance improvements occurred in the SCS, and when the channel size increased even more, the EER score began to increase. The final form of the ResNet architecture succeeded to outperform the FNN and RNN networks.

5.4.2.4 Light Convolutional Neural Network Architecture

Xiang et al. (2017) has developed a system called Light CNN to produce outstanding classification performances for the face recognition systems. The network contains the Max-Feature-Map (MFM) activation function, which is a special case of the max out operation. Xiang stated that the MFM operation provided a neural network model to perform a better *generalization* ability against noisy labels, and as the study claimed, it has outperformed the performance of the ReLU activation function [53]. Since the

ASV spoof 2017 challenge dataset was designed to measure the generalization performance of the neural network systems, the Light CNN model is used in this work to outperform the base ASV system.

The LCNN architecture was replicated from the studied [53]. In the study, they have released three different versions of the LCNN architectures. The Light CNN-9 model was employed due to its complexity, instead of using the Light CNN-4 and Light CNN-29. The system has outperformed the EER score of the ResNet model. Also in the study [61], they have utilized the same architecture to outperform the base system of the ASV spoof 2017 challenge. They have succeeded in obtaining the best EER result from the challenge with their fusion system. Their fusion model was composed of LCNN, SVM, and CNN+RNN neural nets. In the CNN+RNN architecture, they have used a bidirectional gated recurrent unit layer (Bi-GRU) [71] before the fully-connected layers. Accordingly, in the LCNN system, the Bi-GRU layer was implemented before the fully-connected layers. However, this method decreased the evaluation performance of the model by increasing the EER score. When the channel sizes were increased in the convolutional layers, similar to the previous neural networks, it did not provide any successful classification performances. Therefore, the channel sizes of the convolutional layers were set between 8 to 32, which was also similar to the study [61]. The LCNN model successfully outperformed all the other neural network models used for this work according to the EER score performance.

5.4.3 Hyperparameter Arrangement

In this work, the learning rate, batch size and Adam optimizers' epsilon parameter were tuned. The learning rate was selected as 0.001. Increasing or decreasing the learning rate more than 0.001 did not provide any noticeable improvement on the SCS. As the study [72] suggested, the *epsilon* parameter of the Adam optimizer was

set to 0.1. This method has improved the classification performance of the SCS more than using the default value. The optimum batch size for this work was set to 32. Training the predictive models with 16, 64 and 128 batch sizes gradually decreased the robustness of the SCS. Similar findings about the batch sizes were also stated in [72].

5.5 Results and Discussion

5.5.1 Evaluation of the Equal-Error-Rate Scores of Models

The first experiment was conducted by using the FNN architecture with only 3 fully-connected layers and the EER scores obtained for the development and evaluation datasets was 20% and 26%. The SCS was underfit after training it by using only 3 fully-connected layers. To prevent this problem, the 1-dimensional batch normalization was applied for each of the fully-connected layers. Using the batch normalization alone did not prevent the underfitting problem, and the EER score of the evaluation dataset increased to 28%. Instead of using batch normalization, the dropout layer was applied for each of the fully-connected layers with a probability 70%. And finally, the 1-dimensional max-pooling layer was employed to use as a feature extractor against the noisy input samples. Note that no convolutional layers were used before the max-pooling layer. Because, the FNN model was generated to experiment only by using the fully-connected layers. The dropout and max-pooling layers improved the classification performance of the system that was trained with the noisy dataset by decreasing the EER score. As shown Figure (10), the dropout and max-pooling layers improved the classification performance of the SCS by decreasing the EER score of development and evaluation subsets to 19.20%, and 24%.

As a next step, the architecture was updated by replacing the 1-dimensional max-pooling layer with a bidirectional LSTM layer, which was formed with 4-hidden-layers

and 8-hidden-sizes. This modification caused the SCS system to overfit by increasing the validation accuracy more than the training accuracy. As referred in section 4.2.2, to prevent the overfitting problem, the 1-dimensional batch normalization was employed for each of the fully-connected layers, and the probability of the dropout layer was reduced from 70% to 50%. These changes improved the classification performance of the model considerably.

The replay spoofing attacks (RA) in the each subsets of the ASV spoof 2017 dataset was prepared by using various replay/recording devices, acoustic environments and replay configurations to cause a generalization problem [20]. Especially using various devices (e.g., studio recording device, telephone, laptop) to produce the replay spoofing attacks have reduced the generalizability of the SCSs [56]. Since RAs were produced by using different devices, each input sample may have a different distribution. The batch normalization is reducing the internal covariate shift between the input samples (caused by different distributions) [49] and this might be the reason why the robustness of the SCS has improved after employing the batch normalization.

Using the batch normalization with dropout caused EER scores of the development and evaluation datasets to decrease to 1.87% and 20.49%. Another notable performance improvement occurred when the epsilon[2] parameter of the batch normalization increased to 0.001.

The same system was tested by using the LSTM layer without the bidirectional form. However, this method did not provide any significant changes in the EER score. As illustrated in Figure (10), the final form of the RNN architecture has outperformed the EER score of the FNN architecture.

[2]it adds to the denominator for the numerical stability

In the following, the ResNet architecture was tested as a spoofing countermeasure system. At first, the system was containing 9-ResNet blocks, then it was decreased to 3-Resnet block layers to reduce the complexity of the system. Then the stride of the 2nd and 3rd max-pool layers were dropped from 3 to 1 to preserve the dimension of the input matrix. The system was formed by a 2-dimensional batch normalization and reducing the probability of the dropout layer from 70% to 50% did not affect the EER score of the system significantly. Reducing the complexity of the system and the stride parameter has improved the classification performance of the system by decreasing the EER score into 19% on the evaluation dataset. However, the EER score of the development dataset was increased to 2.05%.

In addition, the convolutional layers of the ResNet blocks were tested with a higher filter size (e.g., 8×32 for block1, 32×64 for block2 and 64×64 for block3) and the EER score did not improved. However, decreasing the filter sizes (e.g., 4×8 for block1, 8×16 for block2 and 16×8 for block3) has dropped the EER scores of the development and evaluation subsets noticeably.

As demonstrated in Table (10), the final form of the ResNet system has outperformed the FNN and RNN architectures with the 0.42% EER score for the development dataset and 18.04% EER score for the evaluation dataset.

Finally, the LCNN architecture was tested. As shown in Table (10), the system has performed 4.96% EER score for the development dataset and 17.22% EER score for the evaluation dataset by providing the best EER score. Therefore, the LCNN_{CE} model was accepted as the conventional cross entropy loss metrics and softmax function SCS of the ASV spoofing 2017 challenge. Even though the number of

the convolutional layers were changed to decrease or increase the complexity of the architecture, the EER did not improve. Also, similar to the work in [61], the bidirectional gated recurrent unit (Bi-GRU) was employed for the LCNN system before the final fully connected layer. However, after the modification, the EER score of the system was increased considerably. Similar to the other systems (FNN, RNN, ResNet), smaller filter sizes have performed better with the noisy samples.

The EER score of the LCNN system, which was obtained in this study, was different from the work in [61] (7.37%). Since they have eliminated some of the noisy samples from the training subset to prevent the overfitting problem, the training subsets of both studies were different from each other.

The MFM activation function was able to discriminate the informative features from the noisy ones and it has increased the robustness of the SCS by eliminating the low-activation neurons (noise).

As indicated in Table (10), the overall best score has been obtained with the LCNN_{EDL} model, which was formed by the EDL loss function (*detail in* Section 2.4). To decide the best, multiple EER scores were obtained from the LCNN_{CE} and LCNN_{EDL} systems, and the lowest score was accepted as the best. According to the EER scores that are listed in Table (10), the Bayesian approach has improved the classification performance of the LCNN_{CE} by decreasing the EER score of the both subsets.

During run-time, the Graphical Processing Unit (GPU) processes the deep neural network parameters simultaneously. Therefore, to achieve an accurate analysis for

Table 10: THE EER SCORES OF THE NEURAL NETWORKS MODELS

Architectures	Development	Evaluation
CQCC-GMM (Base System)	11.0%	28.3%
MFCC-FNN	19.20%	24.10%
MFCC-RNN	1.87%	20.49%
MFCC-ResNet	0.42%	18.04%
MFCC-LCNN$_{CE}$	4.96%	17.22%
MFCC-LCNN$_{EDL}$	0.37%	**16.79%**

the complexity of the SCSs, the run-times of each system during the evaluation process (only evaluation dataset was used) are shown in Table (11). Additionally, the run-time of the models were also measured for the Central Processing Unit (CPU) to compare with the GPU.

As shown in Table (11), the MFCC-RNN model has the longest run-time compared to the other SCSs. The MFCC-RNN model is containing an Bi-LSTM layer, which is a recurrent operation, and this might be the reason why the run-time of the system has increased more than other models. On the contrary, the fastest system has appeared to be the MFCC-FNN model. Even though using only full-connected layers has increased the number of the parameters of the model, the model can still process faster than the other systems.

Table 11: THE LIST OF THE RUN-TIMES OF EACH NEURAL NETWORK MODEL

Systems	# Parameters	Time (sec)	
		CPU	GPU
MFCC-FNN	4.9M	3.299	1.327
MFCC-RNN	1.2M	80.836	9.570
MFCC-ResNet	44.1K	19.482	2.425
MFCC-LCNN$_{CE}$	563.3K	23.785	2.087
MFCC-LCNN$_{EDL}$	563.3K	22.849	3.398

5.5.2 Bayesian Approach for the Confidence Measurement

The selected SCS for the Bayesian approach was determined according to the EER scores that are illustrated in Table (10). As the table indicates, the best EER score was produced by the LCNN system. Therefore, the LCNN model was redesigned using the Bayesian approach to generate LCNN$_{EDL}$ model and the LCNN$_{EDL}$ was investigated to outperform the uncertainty measures of the conventional cross entropy loss metrics and softmax function spoofing countermeasure system (e.g., LCNN$_{CE}$). The LCNN$_{EDL}$ and LCNN$_{CE}$ networks are very similar to each other except their loss functions and classification layers. As mentioned in the section 2.4, the Bayesian approach was applied to the LCNN$_{EDL}$ system by replacing its cross entropy loss function with the EDL loss and its softmax layer with an activation layer (e.g., $\exp^{x_i}$, where x_i refers to logits over i-dimension). To compare the uncertainty of the LCNN$_{EDL}$ and LCNN$_{CE}$ systems, the PDF and ROC curve metrics were employed.

To illustrate the distribution of the predicted class probabilities of the systems, the sigmoid outputs of the networks were used and they were computed with $x_i = h(x_i)$, where h(.) was denoted to the sigmoid function and x_i was denoted to the logits over the i-dimension. In each figure, the misclassified samples P_{miscls} (left) and true classified samples P_{true} (right) of the neural networks were visualized separately to be discussed as individual. As shown in Figures (9) and (10), the estimated class probabilities of each model were expected to approach to their true label values (e.g., 0, 1).

When the figures were compared with each other for the P_{true} values, it is apparent that the estimated class probabilities of the LCNN$_{EDL}$ model between the intervals $\{0.5 \leq p_1, ..., p_n \leq 1.0\}$ (e.g., true label 1.0) and $\{0 \leq p_1, ..., p_n \leq 0.5\}$ (e.g., true label 0) were approached to their true labels more than the probabilities of the LCNN$_{CE}$ system.

Another considerable change was observed between the P_{miscls} distributions of the systems. As discovered in Figure (10), the P_{miscls} samples of LCNN$_{EDL}$ model were appeared more separated compared to the P_{miscls} probabilities of the LCNN$_{CE}$. Because these were misclassified samples, if they were to converge on their correct labels, they would appear more towards the central section of the PDF (e.g., $\{0.40 \leq p_1, ..., p_n \leq 0.60\}$). Therefore, the Bayesian network might be increasing the predicted probabilities of the misclassified samples when the system believes the mistakenly classified samples were correct.

According to the SNR scores shown in Table (9), the ASV dataset contains highly challenging replay spoofing attacks (e.g., high (dB)), which were almost indistinguishable from authentic recordings [28], and the network might be producing high probabilistic opinions for these samples when it misclassifies them.

As the PDFs indicated, the Bayesian model has decreased the uncertainty of the true classified samples by approaching their probabilities P_{true} to their true labels. However, because of the effect acquired from the Bayesian network for the P_{miscls} scores, it was hard to claim that the Bayesian network has decreased uncertainty of the LCNN$_{CE}$ model. Therefore, the ROC curve was used to compare the uncertainty of the systems to come up with a final decision.

To illustrate the ROC curve, the logits L were separated to their misclassified L_{miscls} and true classified L_{true} samples. And also the L_{miscls} and L_{true} scores were used as class types instead of using spoof and genuine. According to Figure (11), the LCNN$_{EDL}$ system was able to decrease the uncertainty of the conventional cross entropy metrics and softmax function SCS (e.g., $LCNN_{EDL}$) considerably by increasing the AUC value 14%.

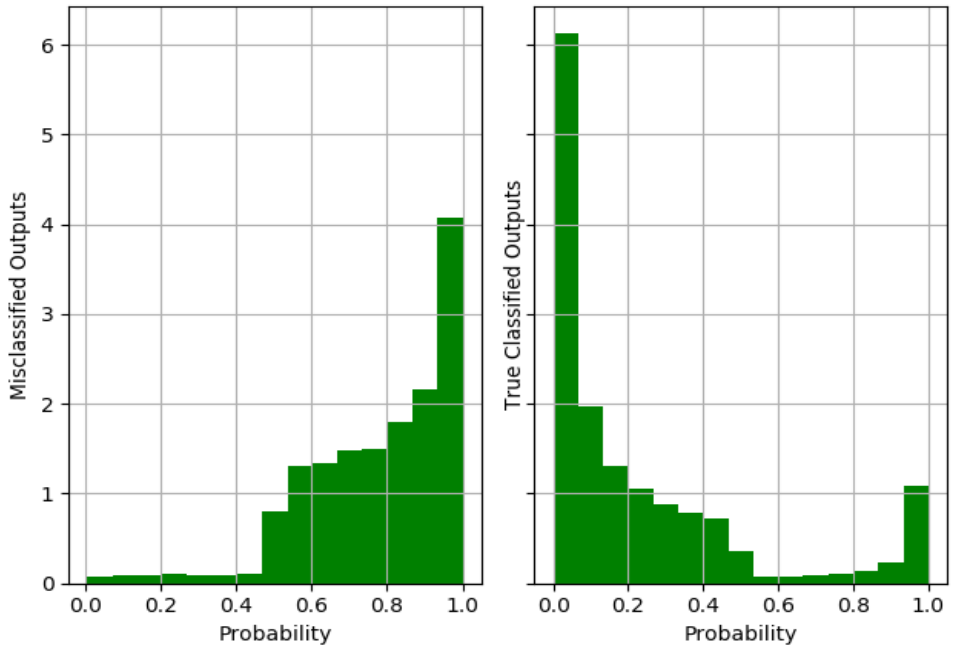

Figure 9: The Probability Distribution Function of the $LCNN_{CE}$ Outputs

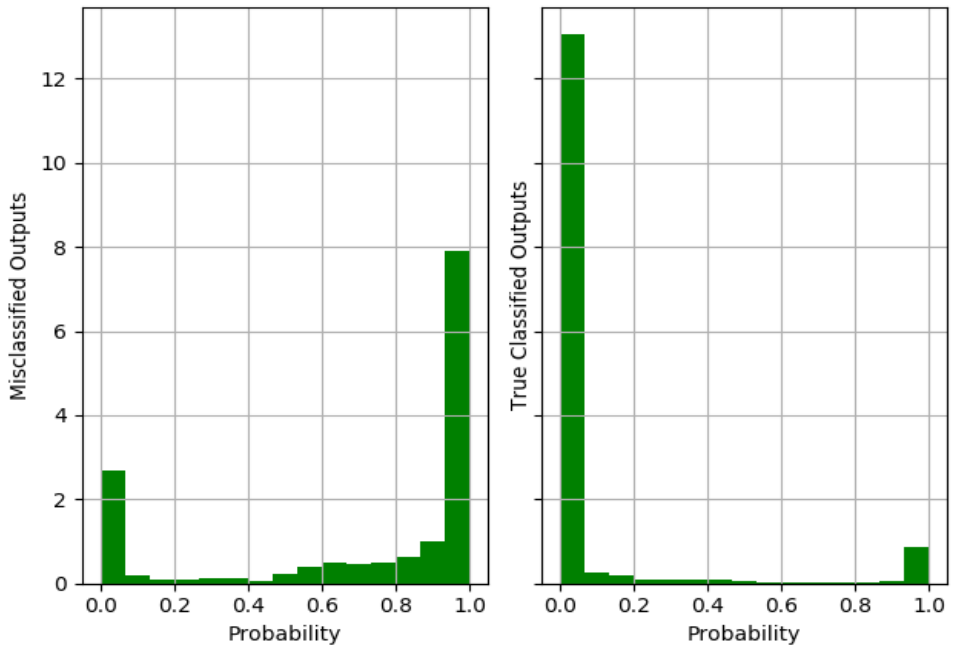

Figure 10: The Probability Distribution Function of the $LCNN_{EDL}$ Outputs

As demonstrated in the ROC curve, the LCNN$_{CE}$ system has performed poorly with a 66% AUC value. On the other hand, the Bayesian network was able to separate the true classified (e.g., true-positive and true-negative) and misclassified (e.g., false-positive and false-negative) samples better than the LCNN$_{CE}$. In this way, the Bayesian network provided the LCNN$_{CE}$ system to be more certain about its decisions by increasing its ability to discriminate its true (e.g., L_{true}) and false (e.g., L_{miscls}) choices about the samples that were obtained from the ASV spoof 2017 challenge dataset.

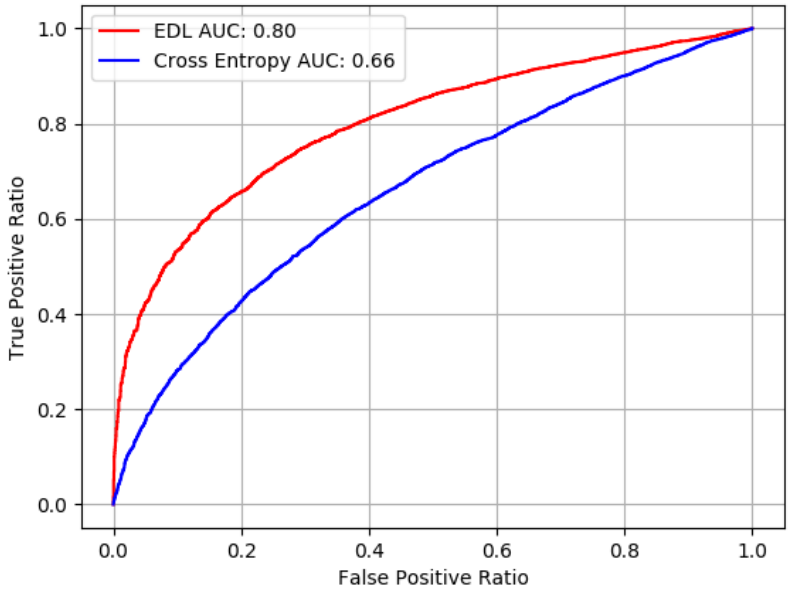

Figure 11: The ROC Curve of the Networks

CHAPTER VI

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this work, the Bayesian approach (e.g., LCNN_{EDL}) was applied to decrease the uncertainty of the conventional cross entropy metrics and softmax function SCS (e.g., LCNN_{CE}). These two systems were compared by using the PDF, AUC value and ROC curve metrics. According to the PDFs of the systems, the LCNN_{EDL} decreased the uncertainty of the true classified (e.g., true positive and true negative) samples compared to the LCNN_{CE} network. However, the Bayesian approach also caused the estimated class probabilities of the false negative and false positive samples to approach their wrongly decided class labels more. Therefore, the decrease in the uncertainty of the system could not be explained by the PDF results and accordingly the ROC curve was employed to compare the both systems to come up with the final decision. According to the AUC and ROC curve results, the Bayesian approach was able to decrease the uncertainty of the system by increasing the AUC value 14%. Concurrently, the Bayesian approach has performed the best EER score (16.79%) among other networks and this novel approach has successfully increased the robustness of the conventional cross entropy metrics and softmax function SCS.

6.2 Future Work

In this work, the LCNN_{EDL} and LCNN_{CE} models were compared based on their best-performed (e.g., lowest EER score) epochs. Comparing these systems' uncertainty measures by taking many evidence scores from each network to evaluate their overall performances is also an important evaluation approach. In future work, the Monte

Carlo Dropout will be applied during the validation process. And also, the LCNN$_{EDL}$ and LCNN$_{CE}$ networks will be converted into probabilistic networks by getting each input samples' estimated class probabilities 100 times and the mean of them will be taken as a final score. Those scores will be used to measure the uncertainty of the systems.

CHAPTER VII

APPENDIX

The algorithm of the FNN architecture:

```
class FNN(nn.Module):
    def __init__(self):
        super(FNN, self).__init__()
        self.feature = nn.Sequential(
            nn.MaxPool1d(6, stride=2, ceil_mode=True),
        )
        self.main = nn.Sequential(
                nn.Linear(in_features=4284,  out_features=1024),
                nn.ReLU(),
                nn.Dropout(0.7),
                nn.Linear(in_features=1024, out_features=512),
                nn.ReLU(),
                nn.Dropout(0.7),
                nn.Linear(in_features=512, out_features=128),
                nn.ReLU(),
                nn.Dropout(0.7),
        )
        self.logits = nn.Linear(in_features=128, out_features=2)
```

The algorithm of the RNN architecture:

```
class RNN(nn.Module):
```

```
    def __init__(self):
        super(RNN, self).__init__()
        self.lstm = nn.LSTM(input_size=72, hidden_size=8, num_layers=4,
        batch_first=True, bidirectional=True, dropout=0)
        self.main = nn.Sequential(
                nn.Linear(in_features=2016,  out_features=512),
                nn.BatchNorm1d(512, eps=0.001, momentum=0.1),
                nn.ReLU(),
                nn.Dropout(0.5),
                nn.Linear(in_features=512, out_features=256),
                nn.BatchNorm1d(256, eps=0.001, momentum=0.1),
                nn.ReLU(),
                nn.Dropout(0.5),
                nn.Linear(in_features=256, out_features=128),
                nn.BatchNorm1d(128, eps=0.001, momentum=0.1),
                nn.ReLU(),
                nn.Dropout(0.5),
                )
        self.logits = nn.Linear(in_features=128, out_features=2)
```

The algorithm of the ResNet architecture:

```
class ResNet(nn.Module):
    def __init__(self):
        super(ResNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, kernel_size=3, stride=1, padding=1)
        self.block1 = ResNetBlock(8, 16,  True)
        self.mp = nn.MaxPool2d(2, stride=3, padding=1)
        self.block2 = ResNetBlock(16, 32,  False)
```

```
        self.mp_stride_1 = nn.MaxPool2d(2, stride=1, padding=1)
        self.block3 = ResNetBlock(32, 16, False)
        self.mp_stride_2 = nn.MaxPool2d(2, stride=1, padding=1)
        self.dropout = nn.Dropout(0.7)
        self.fc1 = nn.Linear(144, 64)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(64, 2)


class ResNetBlock(nn.Module):
    def __init__(self, in_depth, depth, first=False):
        super(ResNetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_depth, depth,
        kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(depth)
        self.relu1 = nn.ReLU()
        self.dropout = nn.Dropout(0.3)
        self.conv2 = nn.Conv2d(depth, depth,
        kernel_size=3, stride=3, padding=1)
        self.conv3 = nn.Conv2d(in_depth, depth,
        kernel_size=3, stride=3, padding=1)
        self.bn2 = nn.BatchNorm2d(depth)
        self.relu2 = nn.ReLU()
```

The algorithm of the LCNN architecture:

```
class LCNN(nn.Module):
    def __init__(self, num_classes=2):
        super(LCNN, self).__init__()
        self.features = nn.Sequential(
```

```
        mfm(1, 8, 5, 1, 2),
        nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True),
        group(8, 16, 3, 1, 1),
        nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True),
        group(16, 32, 3, 1, 1),
        nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True),
        group(32, 24, 3, 1, 1),
        group(24, 24, 3, 1, 1),
        nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True),
    )
    self.block = nn.Sequential(
        mfm(960, 256, type=0),
        nn.Dropout()
    )
    self.logits = nn.Linear(256, 2)
```

Bibliography

[1] Z. Wu, N. Evans, T. Kinnunen, J. Yamagishi, F. Alegre, and H. Li, "Spoofing and countermeasures for speaker verification: a survey," *Speech Communication*, pp. 130 – 153, 2015.

[2] J. Villalba and E. Lleida, "Detecting replay attacks from far-field recordings on speaker verification systems," pp. 274–285, 03 2011.

[3] Z. Kons and H. Aronowitz, "Voice transformation-based spoofing of text-dependent speaker verification systems," *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pp. 945–949, 01 2013.

[4] Z. Wang, G. Wei, and Q. He, "Channel pattern noise based playback attack detection algorithm for speaker recognition," vol. 4, pp. 1708–1713, 2011.

[5] J. Villalba and E. Lleida, "Preventing replay attacks on speaker verification systems," pp. 1–8, 2011.

[6] Z. Wu, S. Gao, E. S. Cling, and H. Li, "A study on replay attack and anti-spoofing for text-dependent speaker verification," pp. 1–5, 2014.

[7] T. Kinnunen, M. Sahidullah, H. Delgado, M. Todisco, N. Evans, J. Yamagishi, and K. A. Lee, "The asvspoof 2017 challenge: Assessing the limits of replay spoofing attack detection," *Interspeech 2017*, 2017.

[8] T. Kinnunen, M. Sahidullah, M. Falcone, L. Costantini, R. Hautamäki, D. Thomsen, A. Sarkar, Z.-H. Tan, H. Delgado, M. Todisco, N. Evans, V. Hautamäki, and K. A. Lee, "Reddots replayed: A new replay spoofing attack corpus for text-dependent speaker verification research," 03 2017.

[9] S. Ergunay, E. Khoury, A. Lazaridis, and S. Marcel, "On the vulnerability of speaker verification to realistic voice spoofing," pp. 1–6, 09 2015.

[10] M. Sensoy, L. Kaplan, and M. Kandemir, "Evidential deep learning to quantify classification uncertainty," pp. 3179–3189, 2018.

[11] J. P. Campbell, "Speaker recognition: a tutorial," *Proceedings of the IEEE*, vol. 85, no. 9, pp. 1437–1462, 1997.

[12] T. Kinnunen and H. Li, "An overview of text-independent speaker recognition: From features to supervectors," *Speech Communication*, vol. 52, pp. 12–40, jan 2010.

[13] N. Evans, T. Kinnunen, and J. Yamagishi, "Spoofing and countermeasures for automatic speaker verification," *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 01 2013.

[14] M. Hébert, "Text-dependent speaker recognition," pp. 743–762, 2008.

[15] R. Hautamäkia, V. Hautamäki, and T. Kinnunen, "On the limits of automatic speaker verification: Explaining degraded recognizer scores through acoustic changes resulting from voice disguise," *The Journal of the Acoustical Society of America*, 2019.

[16] M. Nachappa, A. Bojamma, C. Prasad, and N. M, "Automatic speaker verification system," *International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)*, July 2014.

[17] M. Jin and C. Yoo, "Speaker verification and identification," *Behavioral Biometrics for Human Identification: Intelligent Applications*, pp. 264–289, 01 2009.

[18] Z. Wu, J. Yamagishi, T. Kinnunen, C. Hanilçi, M. Sahidullah, A. Sizov, N. Evans, M. Todisco, and H. Delgado, "Asvspoof: The automatic speaker verification spoofing and countermeasures challenge," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 4, pp. 588–604, 2017.

[19] N. Evans, T. Kinnunen, J. Yamagishi, Z. Wu, F. Alegre, and P. De Leon, "Speaker recognition anti-spoofing," 08 2014.

[20] H. Delgado, M. Todisco, M. Sahidullah, N. Evans, T. Kinnunen, K. A. Lee, and J. Yamagishi, "Asvspoof 2017 version 2.0: meta-data analysis and baseline enhancements," 2017.

[21] N. Evans, F. Alegre, Z. Wu, and T. Kinnunen, "Anti-spoofing, voice conversion," pp. 1–10, 01 2014.

[22] Y. Stylianou, "Voice transformation: A survey," 2009.

[23] P. Perrot, G. Aversano, R. Blouet, M. Charbit, and G. Chollet, "Voice forgery using alisp : Indexation in a client memory," *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, vol. 1, pp. 17–20, 02 2005.

[24] V. Hautamäki, R. G. Hautamäki, T. Kinnunen, and A.-M. Laukkanen, "Comparison of human listeners and speaker verification systems using voice mimicry data," 2014.

[25] A. J. Hunt and A. W. Black, "Unit selection in a concatenative speech synthesis system using a large speech database," *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, vol. 1, pp. 373–376 vol. 1, 1996.

[26] K. Tokuday and H. Zen, "Directly modeling speech waveforms by neural networks for statistical parametric speech synthesis," pp. 4215–4219, 04 2015.

[27] T. Masuko, T. Hitotsumatsu, K. Tokuda, and T. Kobayashi, "On the security of hmm-based speaker verification systems against imposture using synthetic speech," *Sixth European Conference on Speech Communication and Technology (EUROSPEECH'99)*, pp. 1223–1226, 1999.

[28] T. Kinnunen, N. Evans, J. Yamagishi, K. A. Lee, M. Sahidullah, M. Todisco, and H. Delgado, "Automatic speaker verification spoofing and countermeasures challenge evaluation plan," 2018.

[29] K. A. Lee, B. Ma, and H. Li, "Speaker verification makes its debut in smartphone," *IEEE Signal Processing Society Speech and Language Technical Committee Newsletter*, 2013.

[30] W. Shang and M. Stevenson, "Score normalization in playback attack detection," pp. 1678–1681, 2010.

[31] A. Alexander, F. Botti, D. Dessimoz, and A. Drygajlo, "The effect of mismatched recording conditions on human and automatic speaker recognition in forensic applications," *Forensic science international*, vol. 146 Suppl, pp. S95–9, 01 2005.

[32] J. González-Rodríguez, D. Garcia-Romero, M. Garcia-Gomar, D. Ramos, and J. Ortega-Garcia, "Robust likelihood ratio estimation in bayesian forensic speaker recognition," 2003.

[33] T. Niemi-Laitinen, J. Saastamoinen, and T. Kinnunen, "Applying mfcc-based automatic speaker recognition to gsm and forensic data," 01 2005.

[34] T. Thiruvaran, E. Ambikairajah, and J. Epps, "Fm features for automatic forensic speaker recognition," *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pp. 1497–1500, 01 2008.

[35] N. Evans, J. Yamagishi, and T. Kinnunen, "Spoofing and countermeasures for speaker verification: a need for standard corpora, protocols and metrics," *IEEE Signal Processing Society Speech and Language Technical Committee Newsletter*, 2013.

[36] J. Lindberg and M. Blomberg, "Vulnerability in speaker verification - a study of technical impostor techniques," 01 1999.

[37] J. Villalba and E. Lleida, "Speaker verification performance degradation against spoofing and tampering attacks," *FALA 2010*, 2010.

[38] F. Alegre, A. Janicki, and N. Evans, "Re-assessing the threat of replay spoofing attacks against automatic speaker verification," pp. 1–6, 2014.

[39] J. Gałka, R. Samborski, and M. Grzywacz, "Playback attack detection for text-dependent speaker verification over telephone channels," *Speech Communication*, vol. 67, p. 143, 01 2015.

[40] Z. Wu, T. Kinnunen, N. Evans, and J. Yamagishi, "Asvspoof 2015: Automatic speaker verification spoofing and countermeasure challenge evaluation plan," 2014.

[41] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[42] M. Todisco, H. Delgado, K. A. Lee, M. Sahidullah, N. Evans, T. Kinnunen, and J. Yamagishi, "Integrated presentation attack detection and automatic speaker verification:common features and gaussian back-end fusion," *Interspeech 2018*, 2018.

[43] K.-A. Lee, A. Larcher, G. Wang, P. Kenny, N. Brümmer, D. A. van Leeuwen, H. Aronowitz, M. Kockmann, C. Vaquero, B. Ma, H. Li, T. Stafylakis, M. J. Alam, A. Swart, and J. Perez, "The reddots data collection for speaker recognition.," pp. 2996–3000, 2015.

[44] Y. Boureau, J. Ponce, and Y. Lecun, "A theoretical analysis of feature pooling in visual recognition," pp. 111–118, 2010. 27th International Conference on Machine Learning, ICML 2010 ; Conference date: 21-06-2010 Through 25-06-2010.

[45] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[46] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[47] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," 05 2015.

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, pp. 1735–1780, 1997.

[49] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," vol. 37, pp. 448–456, 07–09 Jul 2015.

[50] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," pp. 770–778, 2016.

[52] M. Alzantot, Z. Wang, and M. Srivastava, "Deep residual neural networks for audio spoofing detection," 06 2019.

[53] X. Wu, R. He, Z. Sun, and T. Tan, "A light cnn for deep face representation with noisy labels," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2884–2896, 2018.

[54] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," vol. 28, pp. 1319–1327, 17–19 Jun 2013.

[55] Y. Qian, N. Chen, H. Dinkel, and Z. Wu, "Deep feature engineering for noise robust spoofing detection," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 10, pp. 1942–1955, 2017.

[56] L. Li, Y. Chen, D. Wang, and F. Zheng, "A study on replay attack and anti-spoofing for automatic speaker verification," pp. 92–96, 08 2017.

[57] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[58] C. You, J. Yang, and H. Tran, "Device feature extractor for replay spoofing detection," pp. 2933–2937, 09 2019.

[59] N. Brummer and E. Villiers, "The bosaris toolkit: Theory, algorithms and code for surviving the new dcf," *arXiv*, vol. 1304, 04 2013.

[60] H. Phan, L. Hertel, M. Maaß, and A. Mertins, "Robust audio event recognition with 1-max pooling convolutional neural networks," *arXiv:1604.06338*, 09 2016.

[61] G. Lavrentyeva, S. Novoselov, E. Malykh, A. Kozlov, K. Oleg, and V. Shchemelinin, "Audio replay attack detection with deep learning frameworks," pp. 82–86, 08 2017.

[62] P. Nagarsheth, E. Khoury, K. Patil, and M. Garland, "Replay attack detection using dnn for channel discrimination," 2017.

[63] K. N. R. K. Alluri, S. Achanta, S. Kadiri, S. Gangashetty, and A. Vuppala, "Sff anti-spoofer: Iiit-h submission for automatic speaker verification spoofing and countermeasures challenge 2017," pp. 107–111, 08 2017.

[64] W. Cai, C. Danwei, W. Liu, G. Li, and M. Li, "Countermeasures for automatic speaker verification replay spoofing attack : On data augmentation, feature representation, classification and fusion," pp. 17–21, 08 2017.

[65] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2013 - Proceedings*, pp. 273–278, 12 2013.

[66] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[67] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," 09 2014.

[68] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the disharmony between dropout and batch normalization by variance shift," pp. 2677–2685, 2019.

[69] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *ArXiv*, vol. abs/1207.0580, 2012.

[70] Z. Chen, Z. Xie, W. Zhang, and X. Xu, "Resnet and model fusion for automatic spoofing detection," 2017.

[71] J. Chung, Çaglar Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *ArXiv*, vol. abs/1412.3555, 2014.

[72] B. Chettri, S. Mishra, B. L. Sturm, and E. Benetos, "A study on convolutional neural network based end-to-end replay anti-spoofing," *ArXiv*, vol. abs/1805.09164, 2018.