POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Politecnico di Torino

Master's Degree Thesis

Adversarial Machine Learning applied to Automatic Speech Recognition systems

Supervisors	Candidate
Prof. Riccardo SISTO	Damiano SERAFINO
Eng. Michele MEVI	

Academic Year 2022/2023

Summary

Nowadays, many devices use automatic speech recognition systems, which are based on machine learning models. But it is good to know that machine learning is not completely secure from attacks because there are Adversarial Machine Learning attacks which aim to deceive machine learning models by providing adversarial inputs.

So, it is very important to understand what types of attacks are possible on these models and which defences should be applied. For this, it is necessary to analyze the various types of attacks, such as FGSM and PGD which are evasion attacks, which allow to create adversarial examples that in models without any type of defence cause a considerable decline in the performance of the model.

In the audio field, a defence considered effective by many is MP3 compression, which should be able to remove the previous adversary noise applied by creating the adversarial example. But in the model analyzed, capable of classifying numbers from 0 to 9, this defence is very ineffective. Therefore, a new defence method was created, implementing a combined model of neural networks, in which given an initial audio input, a binary classifier recognizes whether the audio is original or adversarial.

The original data is forwarded to the neural network trained to classify on original audio, while the adversarial data is passed to a neural network trained with the adversarial training, which is trained not only on original data but even on adversarial data, and for this

reason the model offers greater robustness for these types of attacks. In addition, the topic of voice authentication is also dealt with, analyzing its advantages, disadvantages and risks.

Among the risks, we find the problem of voice cloning, capable of artificially producing a voice so similar to that of a particular person as to be almost indistinguishable, increasingly widespread to create even deep fake videos.

At the end, some solutions to voice cloning are presented such as liveness detection, capable of distinguishing a real voice from an artificial one.

Acknowledgements

I would like to dedicate a few lines to those who contributed to the realization of my degree thesis.

First of all, I thank my supervisor Riccardo Sisto, always ready to give me the right information at every stage of the development of the thesis.

Special thanks go to my tutor Michele Mevi, who with his patience and availability helped me conduct the research for this thesis.

I thank all the staff of the company Blue Reply, in which I carried out an internship lasting almost 5 months and complementary to the preparation of the thesis, for the hospitality and for the skills acquired in the field.

I thank my parents and my brother, who have always been close to me, with the infinite patience that distinguishes them.

Finally, I thank all my friends who have supported me during these years of university.

Table of Contents

List of Tables

List of Figures

Acronyms

ART
Adversarial Robustness Toolbox

PGD
Projected Gradient Descent

FGSM
Fast Gradient Sign Method

TP
True Positive

FP
False Positive

TN
True Negative

FN
False Negative

ASR
Automatic Speech Recognition

MFCC
Mel-frequency cepstral coefficient

GMM

Gaussian mixture model

DCT

Discrete Cosine Transform

AI

Artificial Intelligence

TTS

Text-to-speech

SV2TTS

Speaker Verification to Multispeaker Text-To-Speech Synthesis

GE2E

Generalized end-to-end

LSTM

Long short-term memory

GRU

Gated Recurring Units

Chapter 1

Introduction

Machine learning is used in a growing range of settings to make potentially safety-critical decisions: self-driving cars, drones, robots, anomaly detection, malware classification, speech authentication and recognition of voice commands and many more.

As a result, understanding the security properties of machine learning has become a crucial issue in this area. The extent to which we can build adversarial examples affects the settings of the machine learning system that we want to use, in this case, we focus on neural networks.

In the field of speech recognition, recent work has shown that it is possible to generate audio that sounds like speech for machine learning algorithms but not for humans, using deep fake techniques for voice cloning.

This can be used to control the user's devices without their knowledge. For example, by playing a video with a hidden voice command, a smartphone can visit a malicious web page to cause a drive-by download. This can also be used for bank fraud if the authentication system is not very reliable. Or simply adversarial examples are created that can deceive the machine learning model, even causing serious damage if the systems that use them are of vital importance.

1.1 Thesis objectives

The goal of my thesis is to study Adversarial Machine learning attack techniques and defensive strategies to be applied to automatic speech recognition systems, such as audio classifiers. In addition, the risks of speech recognition systems are taken into account and it analyzes how it is possible to apply the process of voice cloning and a possible defence that allows to detect cloned voices that should be considered fake.

For this reason among the various attack strategies of adversarial machine learning I have analyzed the effectiveness of evasion-type attacks, going to present the various methods used to create examples of adversarial audios and the varieties of these attacks.

Subsequently I examined the possible defences and mitigations that can be applied to make machine learning models more robust for these types of attacks, from the simple use of the Mp3 compression technique to the complete modification of the neural network with the use of adversarial training.

It also compares which model is best at resisting these types of attacks, one based on audio files and one on audio spectrograms (so it becomes an image-based classification)

It also includes the description of the process of voice authentication and recognition to present a real case of a threat, such as voice cloning, which is considered a category of the famous deep fake, increasingly widespread nowadays.

In conclusion, some solutions are presented to detect if certain audio is original or created using Artificial Intelligence.

1.2 Organization

The thesis is organized as follows:

- In the second chapter the topic of machine learning is presented, thus seeing the most common and widespread approaches and algorithms. Subsequently, its limits are also analyzed and in particular the Adversarial Machine Learning, going to deepen the strategies and types of possible attacks.
- Chapter 3 presents the evasion attack in a theoretical way and how to practically create an adversarial example through a PGD attack via a notebook written in python language that uses functions offered by the Adversarial Robustness Toolbox library [1]. Afterwards, to explain how to defend ourselves against this type of attack, MP3 compression defence is applied and it is analyzed how effective this defence is, and if this defence can be bypassed.
- Chapter 4 presents a combined model of neural networks capable of distinguishing an adversary example from a real one. This is achieved through a binary classifier that works as an initial filter, a neural network capable of classifying correctly on real data and a neural network trained with Adversarial Training capable of adequately classifying even the adversarial examples.
- The last chapter deals with the topic of Voice Recognition and is mainly composed of three sections.
  The first deals with voice authentication, explaining what are the advantages, disadvantages and risks of this type of authentication and also an example of a real application of this type of authentication is presented through a python notebook made by me.
  In the second section we can see how to apply voice cloning, and study the components of a system capable of applying it.
  In the last section we see if and how it is possible to distinguish a real voice from a fake artificial one.

Chapter 2

(Adversarial) Machine Learning

Before introducing adversarial machine learning and its applications, it is necessary to make a brief parenthesis on Machine Learning.

2.1 Machine Learning

Machine Learning (ML) is a subset of Artificial Intelligence (AI) that is concerned with building systems that learn or improve performance based on the data they use, known as training data.

The term Artificial Intelligence refers to systems that mimic human intelligence. The terms ML and AI are often misused together and interchangeably [2]. An important distinction is that while everything about ML falls under Artificial Intelligence, AI doesn't just include machine learning.
Nowadays Machine learning has many applications, among the most famous we have:

- Computer vision: this artificial intelligence technology allows computers to derive meaningful information from digital images, video and other visual inputs and then take appropriate action [3]. It uses CNN (Convolutional Neural Networks), and it is applied for radiological imaging in the healthcare industry (e.g. to detect tumours...), social media photo tagging, and autonomous vehicles in the automotive industry.

- Fraud detection: banks and other financial institutions can use ML to detect suspicious transactions. Supervised learning can train a model using known information about fraudulent transactions. Anomaly detection can identify transactions that appear atypical and need further investigation [4].
- Recommendation system: it provides suggestions for items most relevant to a specific user[5]. Recommendation systems provide suggestions for the elements most relevant to a specific user. For example, what product to buy, what music to listen to, or what online news to read.
- Customer service: Online chat-bots on many websites are now replacing traditional customer service, automatically answering frequently asked questions (FAQs) but always referring to the customer's personal information [6].
- Speech Recognition: It is also known as automatic speech recognition (ASR), and is a feature that uses natural language processing (NLP) to translate human speech into a written format. Many mobile devices include speech recognition functionalities in their systems to perform voice search, eg. Google Assistant, Siri or improve accessibility for writing text [7]. Furthermore, voice recognition can also be used as a means of biometric authentication.

2.1.1 Approaches

Machine learning approaches are generally divided into these categories:

- **Supervised learning**: it is defined by the use of labelled datasets to train algorithms to classify data or predict results accurately. As input data is entered into the model, the model adjusts its weights until it is configured appropriately [8]. This occurs as part of the cross-validation process to avoid overfitting or underfitting.
  Some methods used in supervised learning include neural networks, linear regression, logistic regression, random forest, and support vector machine (SVM).

- **Unsupervised learning**: uses machine learning algorithms to analyze and group datasets, that have not been labelled. These algorithms discover hidden patterns or clusters of data without the need for human intervention. One of the most common methods used in unsupervised learning is cluster analysis [9].

- **Semi-supervised learning**: it is considered a mix between supervised and unsupervised learning. It combines a small amount of tagged data with a large amount of unlabeled data during training[10].

- **Reinforcement learning**: is a machine learning approach similar to supervised learning, but the algorithm is not trained using sample data. This model learns as it goes using trial and error [11]. A sequence of positive results will be strengthened to develop the best recommendation for a given problem.

2.1.2 Models

- **Artificial Neural Networks**: ANN is a model based on interconnected nodes, it is inspired by the biological network of neurons in the brain. Each connection transmits information from one neuron to another (as it does with signals from brain synapses).

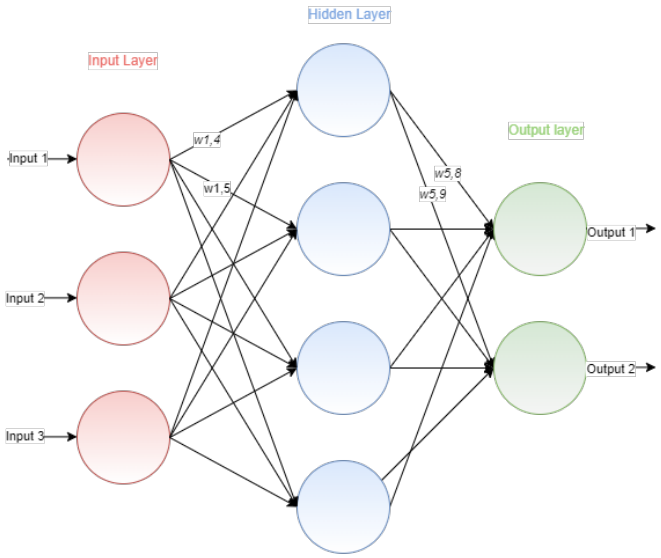


Figure 2.1: Artificial Neural Network

This network has three or more levels interconnected with each other. The first is used for input, which will send data to successive levels, called hidden levels, which adaptively modify the information received through transformations based on a weight that adapts in the learning process, the inputs get multiplied by the weight and then passed to the next layer. This weight simulates the electrical stimulation of a nerve cell, therefore the transfer of information to the nervous system [12]. This allows the units to generate a result, which is then sent as an output through the output layer.

The backpropagation process is also often used, which can modify the output results by taking into account errors. Whenever the output is tagged as an error during the supervised training phase, the information is sent backwards. Each weight is updated in proportion to how much they were responsible for the error [13].

- **Clustering**: Clustering refers to the grouping of unlabeled examples, so it is based on unsupervised learning.

  One of the most famous clustering algorithms is k-means, which determines the best k centroids and assigns each data to the nearest centroid, thus dividing a dataset into groups.

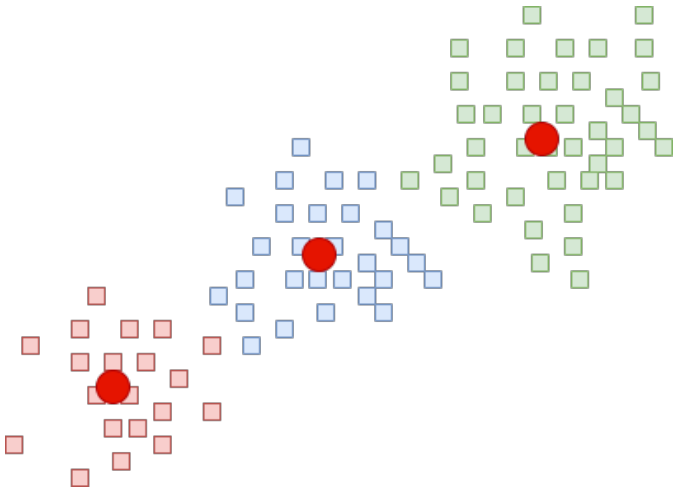


Figure 2.2: Example of clustering with k-means algorithm (k=3)

  There are different types of clustering:

  - **Centroid-based**: as seen with the k-means algorithm, it divides into groups based on proximity to centroids.
  - **Density-based**: it forms clusters based on the density of an area.
  - **Distribution-based**: it groups the data into a distribution (e.g. Gaussian). As the distance from the center of the distribution increases, the probability that a point belongs to the distribution decreases.
  - **Hierarchical**: it is used for hierarchical data, which allows you to create a tree of clusters.

- **Decision Tree**: it can be considered a predictive model to go from observations on an element (represented in branches) to conclusions about the target value of the element (represented in leaves) [14].

  It is one of the predictive modeling approaches used in statistics, data mining, and machine learning. Tree models in which the target variable can take on a discrete set of values are called classification trees; in these tree structures, the leaves represent class labels and the branches represent conditions that lead to those class labels.

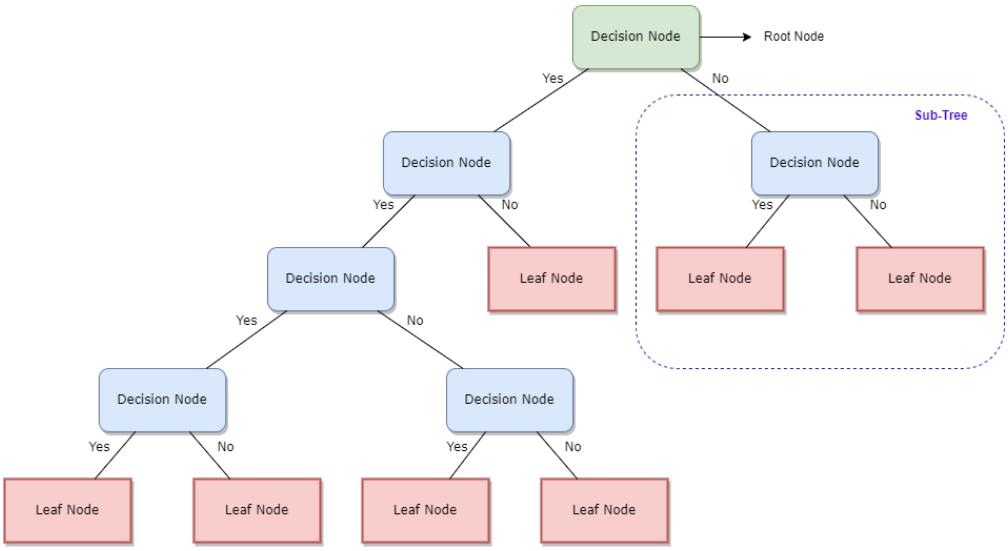


Figure 2.3: Decision Tree structure

- **Random forests**: it is a classifier that contains many decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset [15]. Instead of relying on a single decision tree, it takes the forecast from each tree based on the majority of the forecast votes and predicts the final output.

  It can be considered an example of ensemble learning, which is the process of combining multiple classifiers to solve a problem and improve the performance of the model.

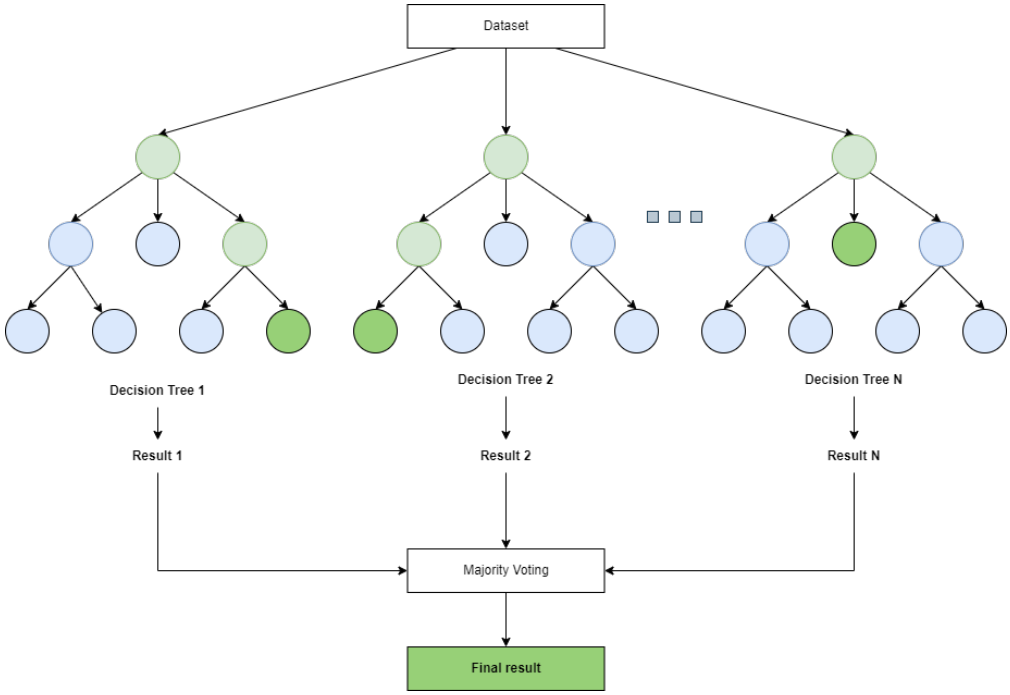


Figure 2.4: Random forests

- **Regression analysis**: it is a statistical method that is part of supervised learning techniques, to express the dependence between a dependent variable (target) and an independent one (predictor), in which there are more than one or more variables. In a nutshell, it helps us to understand how the dependent variable changes in correspondence with the independent one and the other variables are held fixed [16].
  By performing regression, you can determine the most influencing factor, the least influencing factor, and how each factor affects the other factors.

  There are several types of regression, each adapted for different scenarios.

  - **Linear Regression**: it is often used for predictive analysis. It simply shows the dependency between the independent variable x and the dependent variable y.
  - **Logistic Regression**: It is a predictive algorithm which works on the concept of probability. It uses the sigmoid function

    $$sig(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

    and thresholds levels. Values above the threshold will be assigned to 1, and values below to 0.
  - **Polynomial Regression**: it allows to represent a non-linear model using a linear model. The original features are transformed in polynomial values of given degree and at the end used in the linear model.

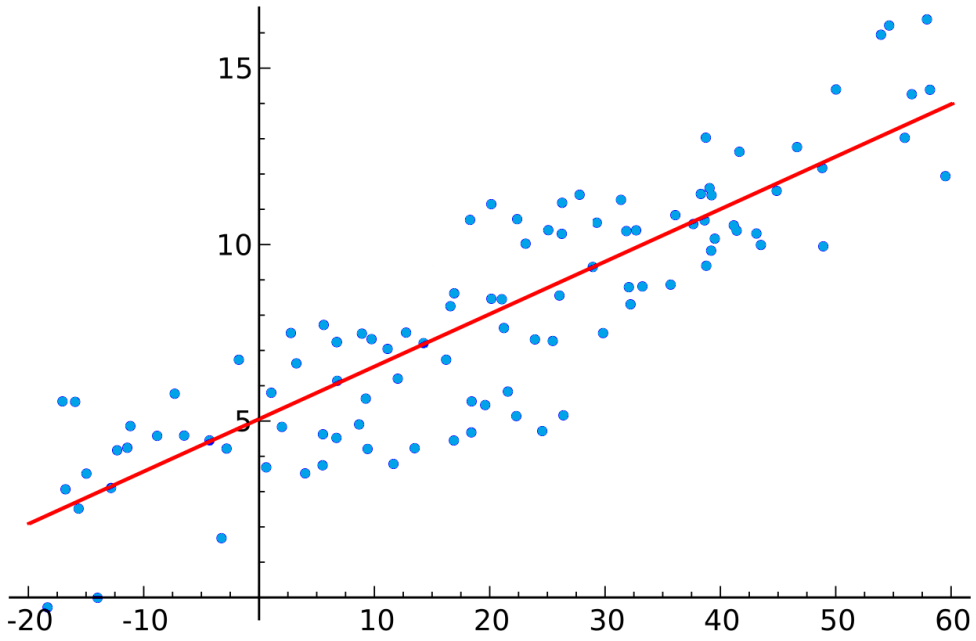


Figure 2.5: Linear regression

- **Support-vector machines**: it is one of the most common supervised algorithms used for classification and regression purposes. Its goal is to find the line separating the classes that maximize the margin between the classes themselves, identifying the margin as the minimum distance between the line and the points of the two classes. This model allows us to obtain a non-probabilistic binary classifier[17].

  The line is calculated using only a portion of the data set, ie the values of a class that are closest to the separation line called "support vectors". The support vectors represent the data of the training set which can be classified with greater difficulty.

The line separating the two classes is called "Hyperplane" and based on the number of features it can be a straight line (if there are two features, Linear SVM) or a two-dimensional plane (if there are three features, Non-linear SVM), while the "margin" represents the distance between the Hyperplane and the support vectors. If in logistic regression the threshold values are represented by the interval [0,1], in the SVM it is the values [1, -1] that represent the margin.

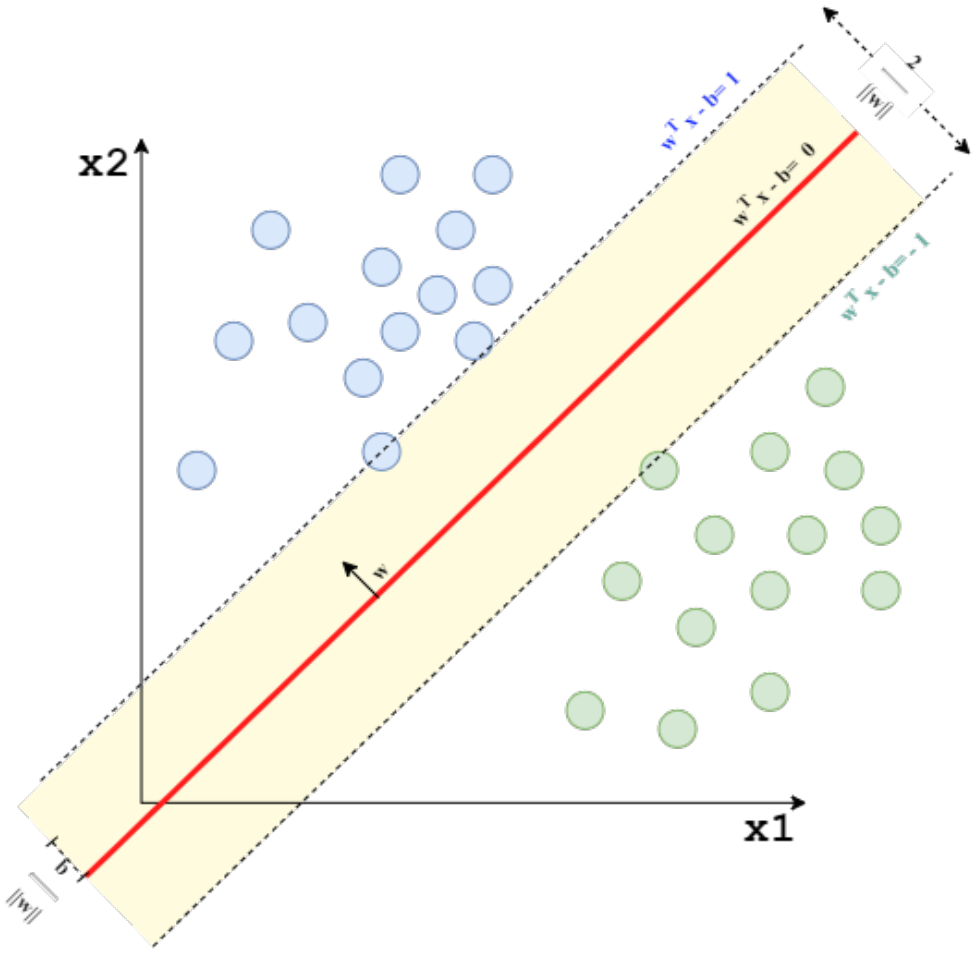


Figure 2.6: Support Vector Machine

2.1.3 Limits

Despite there are continuous updates of Machine Learning models and algorithms, it may happen that machine learning programs do not give the expected results. This can be due to several factors, often linked together:

- **Lack of (good) data**: Most ML algorithms need a huge amount of data to train the model, otherwise, with little training data, the model will not be able to provide useful results [18]. But not only is the number of training data important but also its quality. If the training data does not meet a certain standard, the performance of the algorithm will be very bad.

  For example, in the number classifier presented in the next chapter, if in the dataset we use poor quality audio (with a lot of noise), in which the number is not said correctly, the model could fail to give a correct result when we use a given of excellent audio quality, as the model is trained on data that does not conform to the quality of the test dataset.

- **Overfitting**: As defined in the Oxford Dictionary[19], Overfitting is

  > "the production of an analysis which corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably"

  So in this case the model loses the ability to generalize, as trained on too specific data, despite having excellent performance in the training phase, it will not be able to predict correctly on new data, never met during its training, that is slightly different from that of the training dataset. There are some ways to resolve this overfitting problem [20]:

  - Training with more data: By putting more training data into the model, he will not be able to learn in detail about all test samples and will be forced to generalize to get results.
  - Data augmentation: it is a less invasive strategy than the previous one, it consists in creating slightly different examples starting from a starting example (in the case of audio files a slight noise is added to the input data). This allows us to have unique (but similar) data for the model and to prevent the model from learning the specific training dataset.

    However, it is important not to overdo the changes during this phase, as it should not affect the quality of the training data, or the model will have poor performance.

  - Model simplification: You can solve by decreasing overfitting by simplifying the configuration of the ML model, to reduce its complexity. One of the techniques is the use of dropout functions in neural networks, which allows us to temporarily ignore some nodes in the training phase. [21]
  - Ensembling: as seen previously it is an ML technique that combines several separate models to have a reliable result. An example is random forests.

- **Data Bias**: Generally ML bias is a phenomenon that occurs when a certain algorithm produces results that are conditioned by prejudice due to incorrect assumptions in the learning process.

  In the case of data bias, there is a problem with the training dataset, which is not large enough or in any case not very representative to allow the system to learn correctly. For example, if the training dataset contains only photos of male programmers the algorithm will learn that all programmers are male.

  This problem leads us to consider the ethical problem related to Machine Learning. There have been quite a few problems in the past due to prejudice. As can be seen in a 2017 The New York Times article [22], in 2013 in the US due to a risk assessment algorithm used by the Wisconsin police called COMPAS a black person ended up in jail because COMPAS defined this person as at high risk of recidivism. Probably a lot of data about black people had been used in the training data to influence the algorithm.

  Another serious case was those of Google Photos which tagged black people as gorillas [23][24]. This problem lasted a good three years, from 2015 to 2018, when they decided to delete gorilla images from the training dataset, but in doing so they were no longer able to recognize them. Google preferred to limit the service rather than be accused of racism.

- **Underfitting**: it is the opposite problem of overfitting. In this case, the ML model is too generalizing and simplistic to offer decent performance. But this model has bad results not only in the test phase but also in the training one, in fact, it is characterized by a high bias and low variance.

  A possible solution is to increase the model parameters or add a lot of training data, or even increase the training time (without exaggerating or you will have the problem of overfitting).

Another limitation may be due to not being able to protect against adversarial machine learning attacks.

2.2 Adversarial Machine Learning

Adversarial Machine Learning is a subfield of machine learning that aims to study attacks on its algorithms and also find defences to these attacks. AML attacks can be divided into two main categories:

- **Black Box**: the attacker is completely unaware of what the model he wants to attack looks like, and has no access to its architecture or its parameters.
- **White Box**: the attacker has complete knowledge of the target model, its parameters, its architecture, and data used in the training phase.

There are several different adversarial attack strategies that can be performed against machine learning models, the most common are:

- **Poisoning attacks**: the main goal of the attacker is the contamination of the training data by injecting malicious samples, in such a way that the model will not learn correctly and the final classifier would misclassify the test data during the testing phase or there could be a reduction of the performance of the target model or it can lead to a backdoor attack [25].
- **Evasion attacks**: With this attack, the attacker aims to create an adversarial example using knowledge of the target model, which leads the target model to falsely predict with high confidence [26]. If an adversarial example made on one model is likely to be effective for other models, this is known as transferability. With this attack, the attacker aims to create an adversarial example using knowledge of the target model, which leads the target model to falsely predict with high confidence. If an adversarial example made on one model is likely to be effective for other models, this is known as transferability.
- **Model extraction**: the main purpose of the attack is to steal the parameters of the target ML model using a black box approach, which will compromise the secrecy of this data and can lead to model stealing [27], which allows the model to be reconstructed based on the stolen data.
- **Model inversion**: it is another attack that affects the privacy of the model, the attacker simply tries to recover the private training dataset of a supervised neural network [28]. A model inversion attack could be considered successful only if it generates realistic and different data that accurately describe each of the classes in the training dataset.

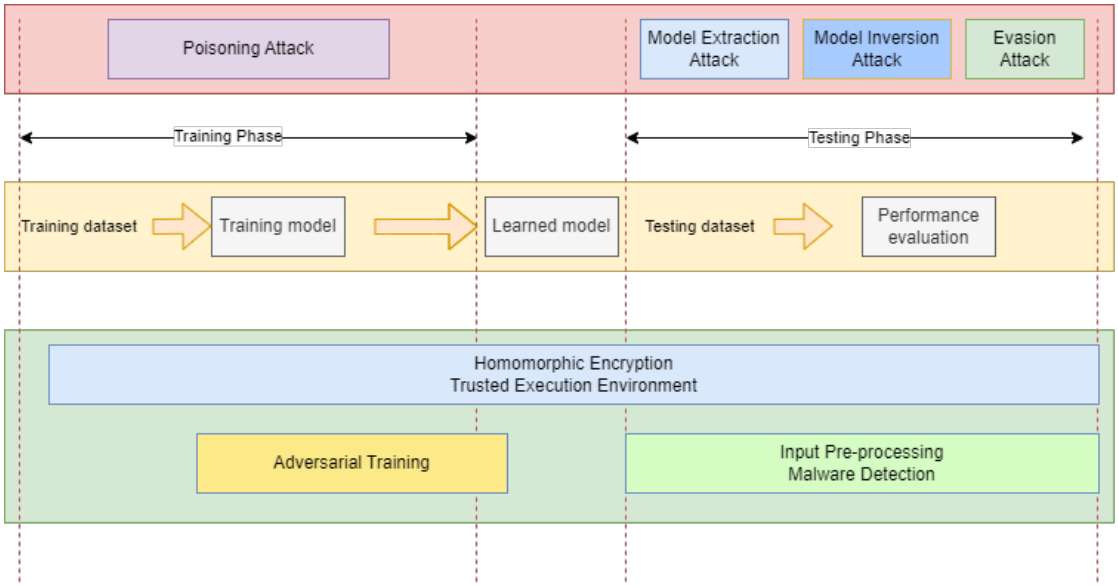


Figure 2.7: Overview AML attacks and defenses

Nowadays, many defenses against adversarial attacks are used, both for the attack in the training phase and for the test phase.

- **Homomorphic Encryption**: it is a special encryption technique that allows us to perform complex operations on encrypted data as if they are still in their original form. Its main disadvantage is the reduction of the efficiency of the model, the high computational cost of performing the encryption and the execution of the algorithm's operations [29].
- **Trusted Execution Environment**: it is a secure and independent operating environment of the CPU that provides protection to code, data and execution operations stored inside it respecting the confidentiality and integrity principles through Data and Code Integrity. One example of TTE application on ML model can be considered the privacy-preserving multi-party ML system on untrusted platform proposed by Ohrimenko [30], that based on Intel SGX [31], provides an additional level of security, preventing exploitation of side channels made through memory and network access.

- **Adversarial training**: it is a mechanism that aims to improve the robustness of a neural network by training it also with adversarial examples, to allow the model to classify correctly even in the case of evasive attacks. At the moment it is considered one of the most effective methods to defend against adversarial examples, but it has the big disadvantage of its large computational expense and that it is effective to specific adversarial attacks (FGSM, PGD).
- **Input preprocessing**: this defence mechanism has the purpose of reducing the impact of the perturbation added by the attacker by performing some operations such as randomization, denoising, and reconstruction, and this happens directly on the input data in the first level of the model, therefore it has the advantage of not modifying successive levels of the ML model[32].
- **Malware Detection**: it is deployed between the input and the first layer of the model to detect the input and determine if the input is an adversarial sample and take corresponding measures accordingly.

Chapter 3

Evasion attack on audio classifier

3.1 Introduction

For my experiments I have used IBM Adversarial Robustness Toolbox (IBM ART)[1], a Python library for Machine Learning security. ART provides tools that enable developers and researchers to defend and evaluate Machine Learning models and applications against adversarial Machine Learning attacks such as evasion, poisoning, mining and inference.

ART supports all popular machine learning frameworks (TensorFlow, Keras, PyTorch, etc.), all types of data (images, tables, audio, video, etc.) and machine learning tasks (classification, object detection, speech recognition, generation, certification, etc.).

The experiment consists of these steps:

- The configuration of the classification model able to classify correctly digits 0-9,
- The creation of adversarial examples using the PGD attack.
- The application of an Mp3 compression defence mechanism.
- The application of an adaptive whitebox attack to defeat Mp3 compression defence

In the notebook, created for this experiment, it has been used the AudioMNIST dataset, which consists of 30000 audio recordings (about 9.5 hours) of spoken

digits (from 0 to 9) in English with 50 repetitions per digit for each of the 60 different speakers. Recordings were collected in quiet offices with a RØDE NT-USB microphone as a mono channel signal at a sampling rate of 48 kHz and saved in full 16-bit format. In addition to the audio recordings, meta-information was also collected including age (from 22 to 61 years), gender (48 males and 12 females), accent and origin of all speakers[33].

3.2 Classification based on raw waveforms

First of all, I have imported all the libraries needed in this notebook and set the parameters (output size, original sampling rate, downsampled sampling rate, audio data path and the audio model path that contains the pre-trained model file). After that, I have defined the classes useful for configuring the dataset and preprocessing the audio file, which are used subsequently in the CNN.

3.2.1 Convolution Neural Networks

CNN (Convolutional Neural Networks) are a class of Deep Neural Network models (with the term "deep" we refer to the high number of levels present in the network), often used for image classification.

A CNN is made up of three main parts: input layers, hidden layers and output layer.

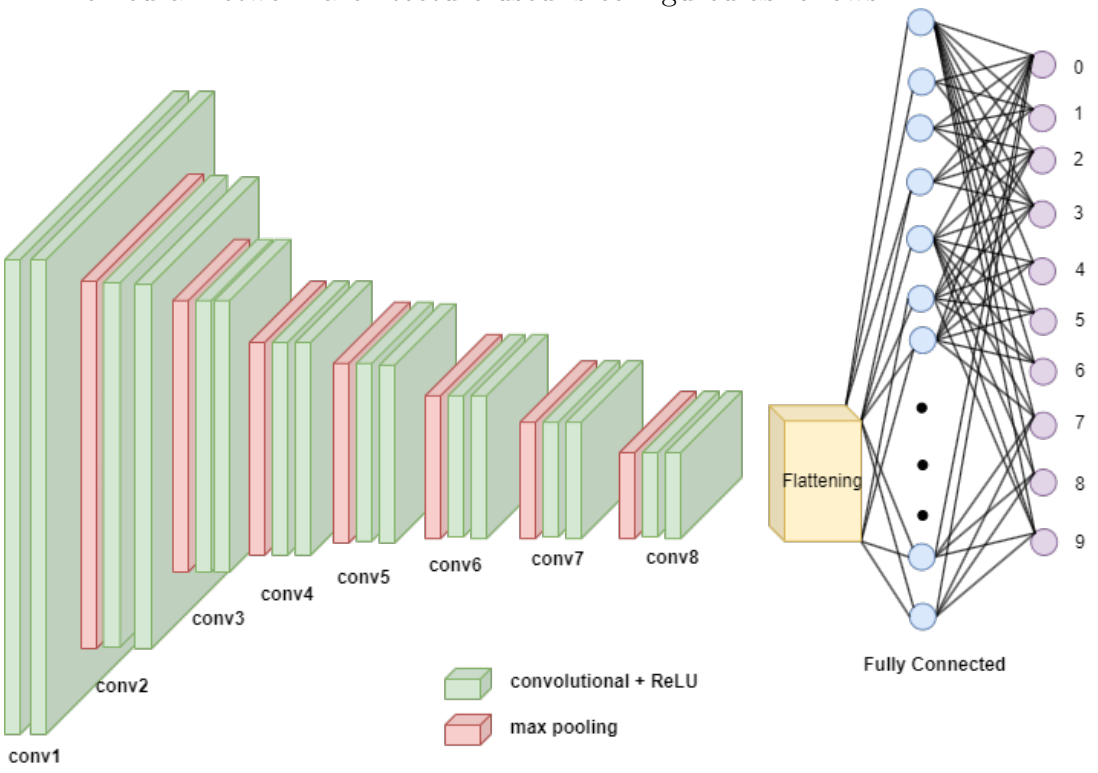
We can call hidden layers also feature extraction layers because is here that there is the local processing tries to extract the information taken from the input layer. Usually these layers consist of three kinds of layers repeated: convolutional layers, pooling layers and ReLu layers; and at the end, there is the fully-connected layer.

- Convolutional layer: it is the main component of the CNN.
  Here a filter also called Kernel is applied. It multiplies (scalar product) the portion of the image by the kernel and consequently, we have a reduction of the input image. An important parameter for this layer is the stride, which defines the amount of motion of the kernel overlay.
  Furthermore we can also set the padding parameter, which allows us to control the size of the output by adding zeros along the edges to the tensor given as input.

- ReLu layer: here it is applied the activation function $f(x) = max(0, x)$, called Rectified Linear Unit function. It removes all the negative values and set them to zero.

- Pooling layer: it is often between two convolutional layers and its purpose is to reduce the size of the images while keeping the important information extracted during the convolutional process. It combines the outputs of the previous layer to send it as input to the next layer. In this case, I have used the Max variant of the pooling, which considers the maximum value of the feature map.
- Fully connected layer: it is the last step that performs the final classification. But before this step, it has been executed the flattening process that moves the value of the feature map in a vector.
- The application of an adaptive whitebox attack to defeat Mp3 compression defence.

The neural network architecture used is configured as follows:




```
# RawAudioCNN model class
class RawAudioCNN(nn.Module):
    def __init__(self):
        super().__init__()
        # 1 x 8000
        self.conv1 = nn.Sequential(
```

```
            nn.Conv1d(1, 100, kernel_size=3, stride=1, padding=2),
            nn.BatchNorm1d(100),
            nn.ReLU(),
            nn.MaxPool1d(3, stride=2))
        # 32 x 4000
        self.conv2 = nn.Sequential(
            nn.Conv1d(100, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 64 x 2000
        self.conv3 = nn.Sequential(
            nn.Conv1d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 128 x 1000
        self.conv4 = nn.Sequential(
            nn.Conv1d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 128 x 500
        self.conv5 = nn.Sequential(
            nn.Conv1d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 128 x 250
        self.conv6 = nn.Sequential(
            nn.Conv1d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 128 x 125
        self.conv7 = nn.Sequential(
            nn.Conv1d(128, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 64 x 62
        self.conv8 = nn.Sequential(
            nn.Conv1d(64, 32, kernel_size=3, stride=1, padding=0),
            nn.BatchNorm1d(32),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))

        # 32 x 30
        self.fc = nn.Linear(32 * 30, 10)
```

```
56
57      def forward(self, x):
58          x = self.conv1(x)
59          x = self.conv2(x)
60          x = self.conv3(x)
61          x = self.conv4(x)
62          x = self.conv5(x)
63          x = self.conv6(x)
64          x = self.conv7(x)
65          x = self.conv8(x)
66          x = x.view(x.shape[0], 32 * 30)
67          x = self.fc(x)
68          return x
```

For clarity, this model will be referred to as AudioNet, even if it is an adaptation of the original Audionet model [33]. The network was trained with stochastic gradient descent (SGD) with a batch size of 100 and a constant momentum of 0.9 for 50,000 epochs and an initial learning rate of 0.0001 which was reduced every 10,000 steps by a factor of 0.5.

After that I have retrained the model for other 10 epochs starting from the previous pre-trained model checkpoint, to improve the performance of the model, also because the dataset of the digits has probably changed in the last few years and I had to adapt the model to the new audio files.

At the end to see the performance of this model I have created the table 3.1 using the function classification_report of the sklearn.metrics library, considering 1000 testing data.

Original prediction:				
class	precision	recall	f1-score	support
0	0.92	0.99	0.95	100
1	1.00	0.91	0.95	100
2	0.82	1.00	0.90	100
3	1.00	0.47	0.64	100
4	0.92	1.00	0.96	100
5	1.00	0.97	0.98	100
6	0.75	0.88	0.81	100
7	0.83	1.00	0.91	100
8	0.98	0.98	0.98	100
9	1.00	0.88	0.94	100
accuracy			0.91	1000
macro avg	0.92	0.91	0.90	1000
weighted avg	0.92	0.91	0.90	1000

Table 3.1: Original prediction

3.3 Adversarial Audio Examples

Adversarial attacks in the audio domain can be classified into two main categories: non-targeted and targeted attacks.

In targeted attacks, the attacker's goal is to add a small perturbation δ to an audio signal that causes the victim's ASR to transcribe the audio into a certain target phrase y_{adv}.

$$y(x + \delta) = y_{adv} \tag{3.1}$$

In non-targeted attacks, the goal is simply to cause a significant error in the transcription of the audio signal so that the original transcript cannot be correctly predicted.

$$y(x + \delta)! = y(x) \tag{3.2}$$

An example of a targeted attack on automatic speech recognition was made by Carlini and Wagner[34].
Given any audio waveform, they can produce another that is over 99.9% similar,

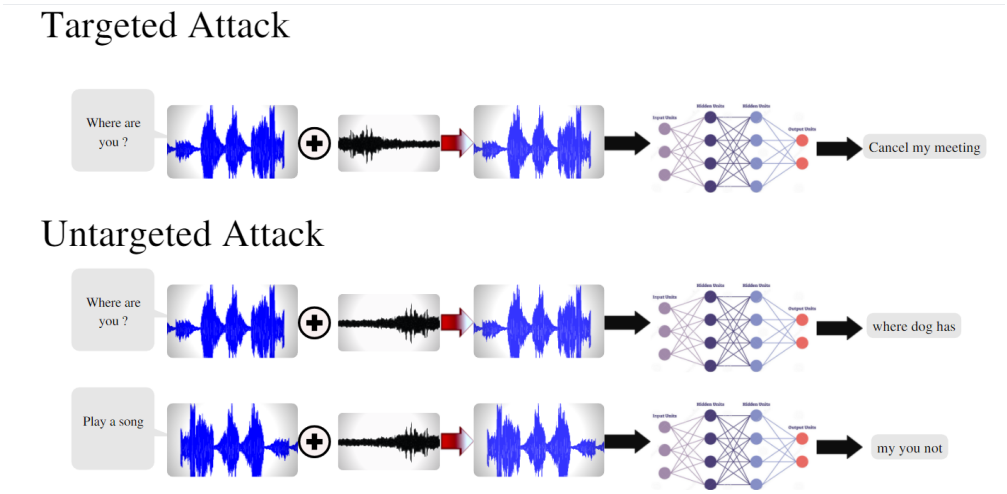


Figure 3.1: Targeted and non-targeted attacks

but he transcribes it as any sentence we choose. They summarily do this, they take an audio waveform x, and target transcription y, the task is to construct another audio waveform x'= x + δ so that x and x' sound similar, but so that C(x')=y. The attack is considered successful only if the output of the network matches exactly the target phrase.
They apply their white-box iterative optimization-based attack to Mozilla's implementation DeepSpeech end-to-end, and show it has a 100% success rate.
In this experiment I performed a non-targeted attack.

3.3.1 Adversarial audio generation

After loading the model and the dataset, we are ready to use the features offered by the ART library. Now we see an Adversarial audio example. We will first load a sample, which here has label 1. The classification model correctly classifies it. Then, we will use the ART library and perform a Projected Gradient Descent attack. The attack will modify the spoken audio and will be incorrectly classified as 9. However, there is almost no hearable difference between the original audio file and the adversarial audio file, there is only a slight background noise.

```
# wrap model in a ART classifier
classifier_art = PyTorchClassifier(
    model=model,
    loss=torch.nn.CrossEntropyLoss(),
    optimizer=None,
    input_shape=[1, DOWNSAMPLED_SAMPLING_RATE],
```

```
    nb_classes=10,
    clip_values=(-2**15, 2**15 - 1)
)
```

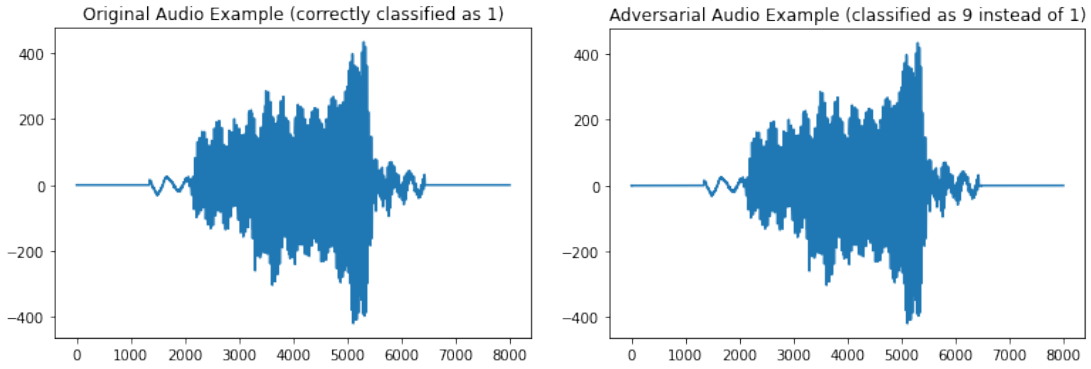
```
# load a test sample
sample = audiomnist_test[3559]

waveform = sample['input']
label = sample['digit']

# craft adversarial example with PGD
epsilon = .005
pgd = ProjectedGradientDescent(classifier_art, eps=epsilon)
adv_waveform = pgd.generate(
    x=torch.unsqueeze(waveform, 0).numpy()
)

# evaluate the classifier on the adversarial example
with torch.no_grad():
    _, pred = torch.max(model(torch.unsqueeze(waveform, 0)), 1)
    _, pred_adv = torch.max(model(torch.from_numpy(adv_waveform)), 1)

# print results
print(f"Original prediction (ground truth):\t{pred.tolist()[0]} ({label})")
print(f"Adversarial prediction:\t\t\t{pred_adv.tolist()[0]}")
```



To perform an adversarial attack, the model parameters θ are considered as fixed and optimized over the input space.
In this experiment, an adversarial example is a perturbed version of the input waveform x.

$$x' = x + \delta \tag{3.3}$$

where δ is small enough such that the reconstructed speech signal of the perturbed version x' is indiscernible from the original signal x by humans, but causes the network to make an inaccurate prediction. Searching for an appropriate δ can be formulated as solving the following optimization problem:

$$\max_{\delta \in \Delta} Loss(\theta, x + \delta, y_x) \tag{3.4}$$

where Loss$(\cdot)$ is the Loss function, y_x is the label of x and Δ is a set of allowed perturbations that formalizes the manipulative power of the attacker.[35]

3.3.2 Fast Gradient Sign Method

Basically, this attack, shorted as FGSM method, calculates the gradients of a loss function (e.g. mean square error or categorical cross-entropy) concerning the input audio and then uses the sign of the gradients to create a new audio (e.g. the adversarial audio) which maximizes the loss.

It consists of taking a single step along the direction of the gradients:

$$\delta = e \cdot sign(\nabla_x Loss(\theta, x, y_x)) \tag{3.5}$$

where the sign$(\cdot)$ function simply takes the sign of the gradient and e is a small value we multiply the signed gradients by to guarantee the perturbations are small enough that the human ear cannot notice them but large enough that they fool the neural network. Therefore, given an utterance x, the adversarial spectral feature can be simply computed as

$$x' = x + e \cdot sign(\nabla_x Loss(\theta, x, y_x)). \tag{3.6}$$

Although FGSM benefits from being the simplest adversarial attack method, it is often relatively inefficient in solving the maximization problem (3.4).

3.3.3 Projected Gradient Descent Method

Unlike FGSM, which is a single-pass method, the PGD method is iterative. Starting with the original input x_0 = x, the input is updated iteratively as follows:

$$x_{k+1} = clip(x_k + \alpha \cdot sign(\nabla_{x_k} Loss(\theta, x_k, y_{x_k}))) \tag{3.7}$$

for k=0,...,K-1.
where α is the step size, K the number of iterations and the clip$(\cdot)$ function applies element-wise clipping such that:

$$||x_k - x||_\infty \leq e, e \geq 0 \in \mathbb{R} \tag{3.8}$$

x_k represents the final perturbed value. Intuitively, the PGD method can be thought of as iteratively applying small-step FGSM, but pushing the perturbed input to stay within the permitted set Δ at every step.
The PGD method allows for more effective attacks but is naturally more computationally expensive than FGSM.

PGD performance is still limited by the ability to stick to the local optimum of the loss function. To mitigate this problem, random restarts are incorporated into the PGD method [36]. The PGD method with random restarts will be executed multiple times. The starting position of the adversarial example is randomly chosen within the collection of permissible perturbations Δ and the PGD method will be performed a particular number of times in one run. The last adversarial example is the one that conducts to the maximum loss.

As I did previously with the original inputs, I created a table to look at the metrics of the model used on adversarial inputs. As we can see in the table 3.2 the performance of the model has dropped considerably. For many classes, it has not been able to classify correctly even once. So the PGD attack on this model can be considered a success.

Adversarial prediction:				
class	precision	recall	f1-score	support
0	0.00	0.00	0.00	100
1	0.00	0.00	0.00	100
2	0.00	0.00	0.00	100
3	0.00	0.00	0.00	100
4	0.62	0.40	0.48	100
5	0.21	0.95	0.35	100
6	0.06	0.27	0.10	100
7	0.27	0.17	0.21	100
8	0.00	0.00	0.00	100
9	1.00	0.01	0.02	100
accuracy			0.18	1000
macro avg	0.22	0.18	0.12	1000
weighted avg	0.22	0.18	0.12	1000

Table 3.2: Adversarial prediction

3.4 MP3 compression defense

Afterwards, we'll apply Mp3 compression, which is a simple input preprocessing defence, namely Mp3 compression. Ideally, we want this defence to result in correct predictions when applied both to the original and the adversarial audio waveforms.

MP3 is an audio compression algorithm that uses a lossy perceptual audio coding scheme based on a psychoacoustic model to discard audio information below the hearing threshold and thereby reduce the file size.
Schönherr[37] recently hypothesized that MP3 could be a solid countermeasure to Audio Adversarial Examples attacks, as it could remove exactly those inaudible ranges in the audio where adversarial noise lies.

```
# initialize Mp3Compression defense
mp3_compression = Mp3CompressionPyTorch(sample_rate=
    DOWNSAMPLED_SAMPLING_RATE, channels_first=True)

# apply defense to original input
waveform_mp3 = mp3_compression(torch.unsqueeze(waveform, 0).numpy())
    [0]


# apply defense to adversarial sample
adv_waveform_mp3 = mp3_compression(pgd.generate(
    x=torch.unsqueeze(waveform, 0).numpy()))[0]

# evaluate the classifier
with torch.no_grad():
    _, pred_mp3 = torch.max(model(torch.Tensor(waveform_mp3)), 1)
    _, pred_adv_mp3 = torch.max(model(torch.Tensor(adv_waveform_mp3))
    , 1)
```

In my experiment this defence as you can see from the tables 3.3 and 3.4, even if it has still good performance on original input, it was not so effective on adversarial examples, as the accuracy of the model that used mp3 compression on adversarial examples only went from 18% to 26%.

Original prediction with MP3 compression:				
class	precision	recall	f1-score	support
0	0.85	1.00	0.92	100
1	1.00	0.85	0.92	100
2	0.96	0.97	0.97	100
3	1.00	0.44	0.61	100
4	0.83	1.00	0.91	100
5	0.92	1.00	0.96	100
6	0.68	1.00	0.81	100
7	0.77	1.00	0.87	100
8	0.97	0.71	0.82	100
9	1.00	0.75	0.86	100
accuracy			0.87	1000
macro avg	0.90	0.87	0.86	1000
weighted avg	0.90	0.87	0.86	1000

Table 3.3: Original prediction with MP3 compression

Adversarial prediction with MP3 compression:				
class	precision	recall	f1-score	support
0	1.00	0.05	0.10	100
1	0.00	0.00	0.00	100
2	0.00	0.00	0.00	100
3	0.00	0.00	0.00	100
4	0.58	0.80	0.67	100
5	0.14	1.00	0.25	100
6	0.25	0.24	0.24	100
7	0.73	0.73	0.56	100
8	0.00	0.00	0.00	100
9	0.00	0.00	0.00	100
accuracy			0.26	1000
macro avg	0.27	0.26	0.18	1000
weighted avg	0.27	0.26	0.18	1000

Table 3.4: Adversarial prediction with MP3 compression

3.5 Adaptive whitebox attack to defeat MP3 compression defense

The last step of this experiment is applying the adaptive whitebox attack that is able to bypass in some cases the Mp3 compression defense.

I set up a new Pytorch classifier by setting MP3 compression as preprocessing defense. Subsequently to create the new adversarial audio example I used the previous PGD algorithm with a number of iterations equal to 40 and a step size attack at each iteration of 0.0000125.

Adversarial prediction with adaptive classifier:				
class	precision	recall	f1-score	support
0	0.11	0.10	0.11	100
1	1.00	0.15	0.26	100
2	0.00	0.00	0.00	100
3	1.00	0.01	0.02	100
4	0.19	0.91	0.32	100
5	0.25	0.39	0.31	100
6	0.01	0.01	0.01	100
7	0.23	0.19	0.21	100
8	1.00	0.09	0.17	100
9	0.95	0.18	0.30	100
accuracy			0.20	1000
macro avg	0.47	0.20	0.17	1000
weighted avg	0.47	0.20	0.17	1000

Table 3.5: Adversarial prediction with adaptive classifier:

3.6 Conclusion

So far I have described how we can apply the ART library to audio data. By providing a pretrained PyTorch model and loading it via ART's PyTorchClassifier we can easily plug in several off-the-shelf attacks like Projected Gradient Descent.

Furthermore, I have demonstrated how to apply the Mp3 compression defence and demonstrated how to circumvent it with an adaptive whitebox attack.

In the following, we investigate the performance of the model in the various prediction phases both on clean inputs and on adversaries to analyze how effective these attacks are and whether the Mp3 compression defence was useful or not.

3.6.1 Statistics

One of the key concepts in classification performance is the confusion matrix, which is a tabular view of model predictions against ground truth labels.

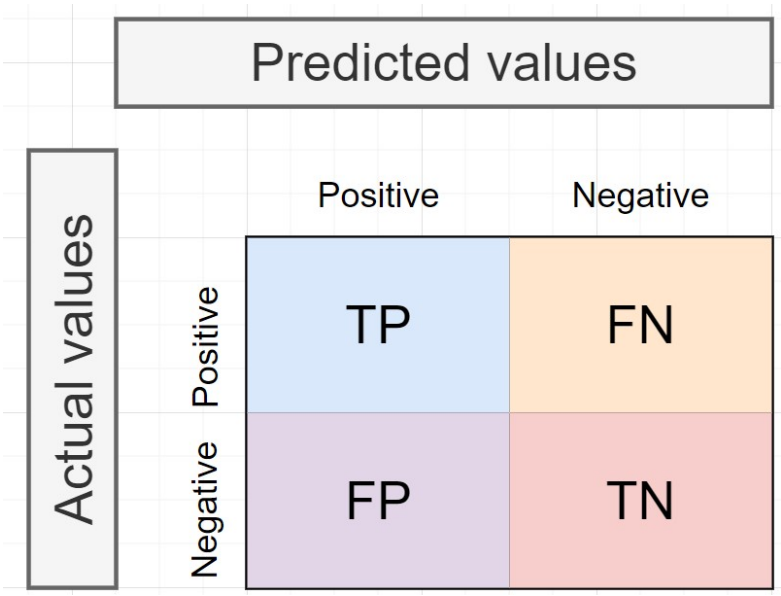


Figure 3.2: Confusion Matrix

- TP: it means True Positive and it represents the number of predictions for a target class where the actual class and the class predicted are the same and refer to the target class.
- TN: it means True Negative and it represents the number of predictions for a

target class where the actual class is not the class considered as well as the predicted class.

- FP: it means False Positive and it represents the number of predictions for a target class where the actual class is not the target class but it is predicted as the target class.
- FN: it means False Negative and it represents the number of predictions for a target class where the actual class is the target class but it is predicted as a different class.

To better understand the graph it is useful to know the various definitions of the metrics.

- Accuracy: it is the most intuitive performance metric and it is the proportion of correct predictions (both true positives and true negatives) among the total number of predictions.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.9}$$

- Precision: it is the fraction of relevant instances among the retrieved instances

$$Precision = \frac{TP}{TP + FP} \tag{3.10}$$

- Recall: it is also called sensitivity or true positive rate, it indicates the ratio of positive instances correctly identified by the machine learning system.

$$Recall = \frac{TP}{TP + FN} \tag{3.11}$$

- F1-Score: it is for definition the harmonic mean of precision and recall. Compared to a conventional average, the harmonic one gives greater weight to the values. This causes a classifier to get a high F1 score only when accuracy and recovery are both high.

$$F1 - Score = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision} \tag{3.12}$$

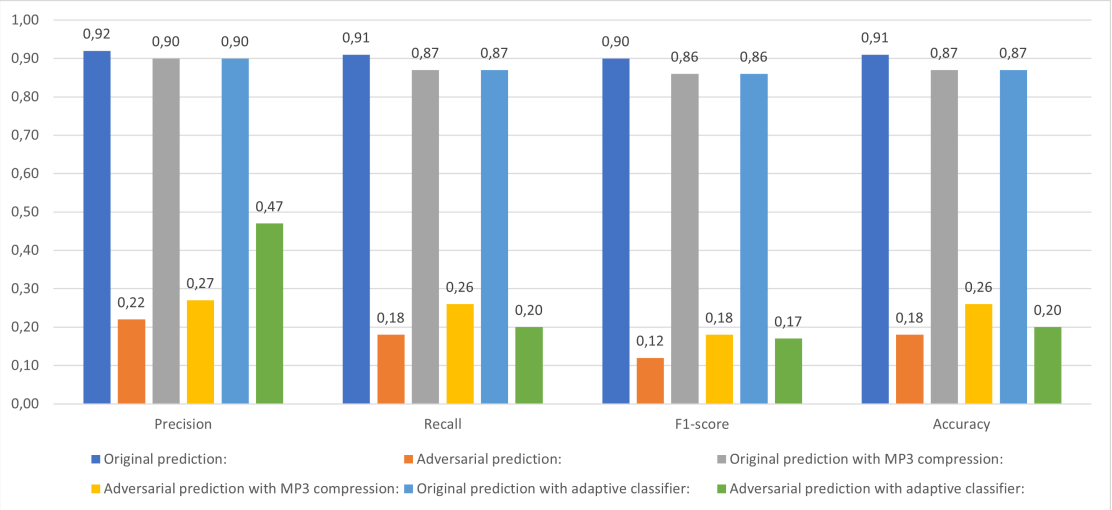


Figure 3.3: Statistics

As we can see in the graph 3.3, both the original model and the one with MP3 compression are very accurate on the original data with an accuracy of around 90%. The most striking thing is the great effectiveness of the PGD attack, which causes a considerable decrease in the accuracy of the model, in fact, it goes from 91% to 18%.

Another important aspect to consider is that Mp3 compression, although it has always been considered an effective form of defence in many studies, in this model it is not able to significantly improve the accuracy of the model on adversarial examples, in fact, the accuracy only increases from 18% to 26%.

not only the mp3 defence is not effective, but it can also be circumvented with the adaptive white box attack, which allows obtaining a very low accuracy, passing from 26% of the adversarial data preprocessed with mp3 compression to 20%, a value very close to 18% accuracy obtained on the opponent's data without any kind of defence.

Chapter 4

Neural network model robust against audio evasion attacks

4.1 Introduction

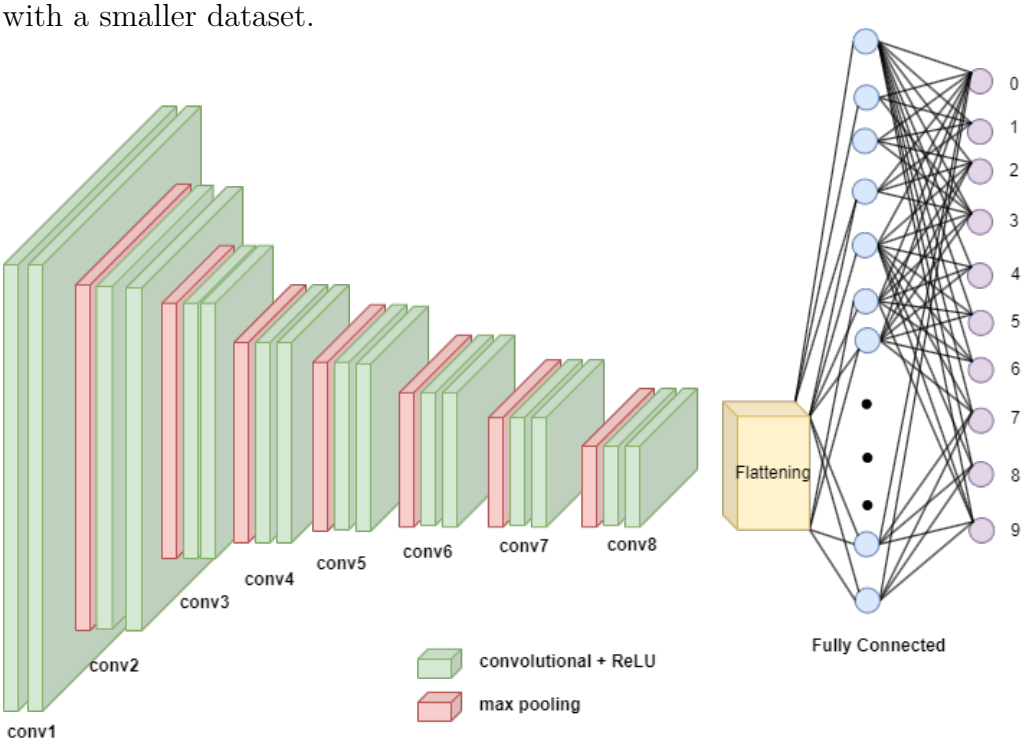
As we saw in the previous chapter, mp3 compression is not a very effective defence against evasion attacks. For this reason, I have studied a new type of defence, designing a neural network of neural networks, capable of distinguishing if audio is an adversarial example, and behaving accordingly, giving this audio input to the network trained with the adversarial training if classified as an adversarial, or to the original network if classified as clean.

Furthermore, in this chapter two models trained to classify numbers from 0 to 9 are compared, the first model (called Audionet) is based on the audio waveforms while the second model (called Spectrum for convenience) first converts the audio into an image by processing its spectrogram and only subsequently is being trained.
I used the new Spectrum model to also have a comparison between the two types of classification, one based directly on audio, and the other on images, and see which of the models performs better.

4.2 Training

I retrained the "Audionet" model, previously mentioned in chapter 2, used to present attacks with the ART library and trained a new "Spectrum" model, based on the classification of spectrograms generated from audio files.

- Audionet : this model uses exactly the same old neural network but retrained with a smaller dataset.



- Spectrum. Compared to "Audionet" this model does not work as a classifier of audio but of images, since from the audio dataset it generates the respective spectrograms that are saved as images and on which the model is trained.

The "Spectrum" model uses a different neural network that consists of 6 weight layers that are organized in series as follows:

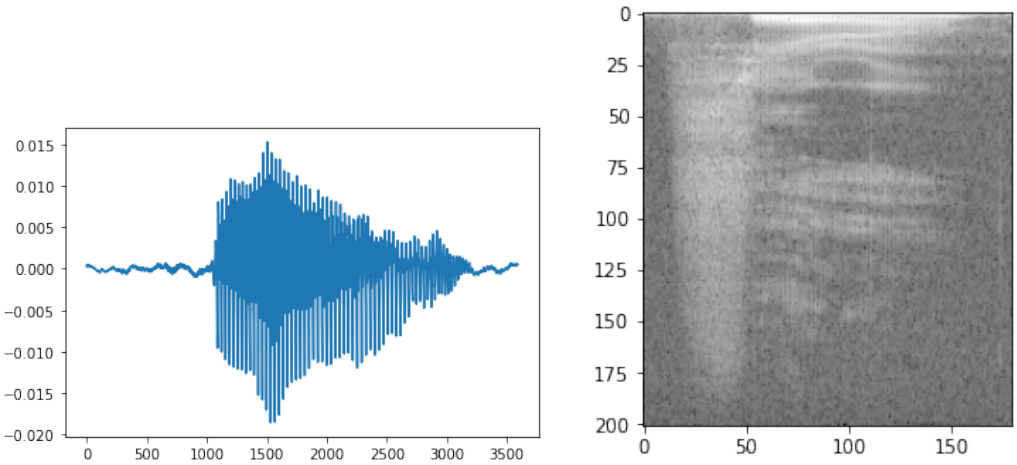
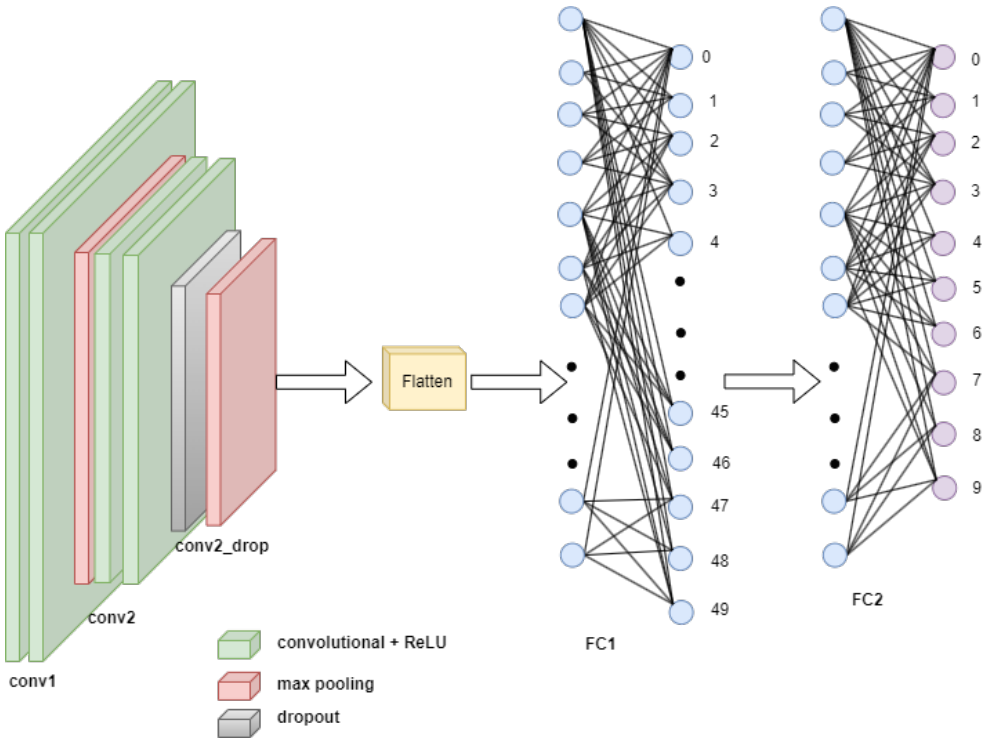


Figure 4.1: Audio spectrogram



As you can see, the architectures of the two models are very different.

CNNet has only 3 convolutional levels, one of which performs the Dropout function (a regularization technique to reduce overfitting and consists in ignoring some nodes during training),and at the end there are two fully connected

layers (the first with 50s output and the last with 10)

```
class CNNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(51136, 50)
        self.fc2 = nn.Linear(50, 10)


    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)),
    2))
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = F.relu(self.fc2(x))
        return F.log_softmax(x,dim=1)
```

The CCNet was trained with the Adam optimizer with a batch size of 15 for 20 epochs with a learning rate of 0.0001. The learning rate is a very important hyperparameter that allows us to define how much to update the parameters of the model at each batch.

After various attempts I chose this learning rate which is so low, of only 0.0001, because although it requires more epochs and causes a slowdown in the learning process, it allowed me to accurately train my model and find the optimal solution, unlike high learning rate values that although they would have speeded up the learning process and decreased the number of epochs, they would not have led to an optimal balancing of the model weights, but would have found a solution that can be considered sub-optimal.
I have used 4000 audio files in the training phase and 1000 in the testing phase.

4.3 Adversarial training

Adversarial training, that consists of training the model on adversarial examples, is one of the few defenses against adversarial attack.

Unfortunately, the high cost of generating strong adversarial examples makes standard adversarial training on large-scale problems impractical.
Adversarial training can be traced back to [38], where models were strengthened by producing adversarial examples and injecting them into the training data. The robustness achieved by the adversarial training depends on the strength of the adversarial examples used [39], in fact this defense has the disadvantage of being effective only against specific adversarial attack.

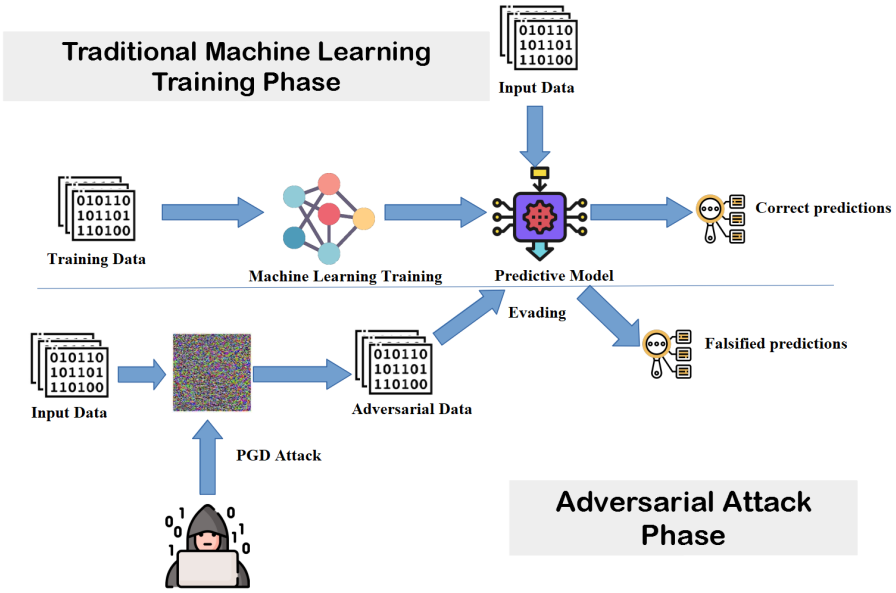


Figure 4.2: Traditional training

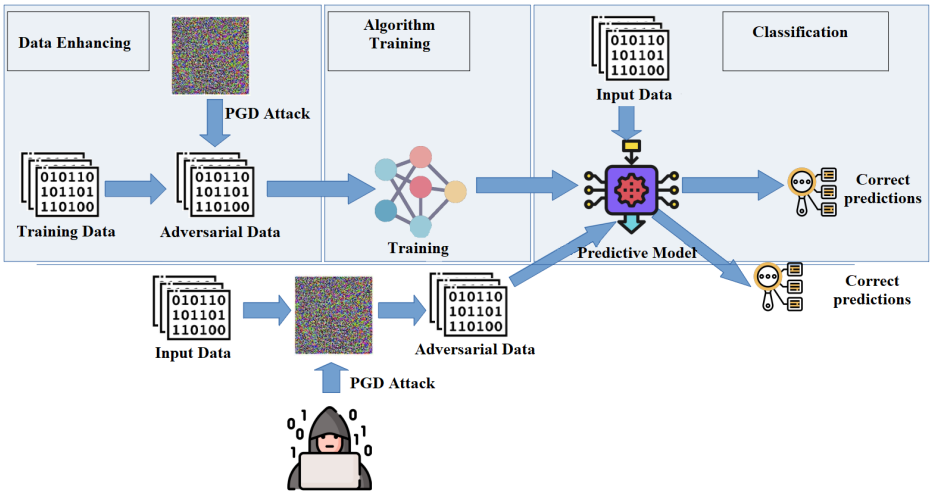


Figure 4.3: Adversarial training

Training on fast non-iterative attacks such as FGSM only results in robustness against non-iterative attacks and not against PGD attacks [40]. In this case, I have created adversarial examples with the PGD attack using the ART library.

- Audionet

  Also for the Adversarial Training I have used the previous "Audionet" model but trained with the dataset containing also adversarial examples (20% of total examples), so 4000 original inputs and 1000 adversarial inputs. The network was trained with stochastic gradient descent with a batch size of 64 and constant momentum of 0.9 for 60 epochs with a learning rate of 0.0001.

- Spectrum

  I have used the previous "CNNet" model but trained with the dataset containing also adversarial examples (20% of total examples), so 4000 original inputs and 1000 adversarial inputs. The network was trained with Adam optimizer with a batch size of 15 for 30 epochs with a learning rate of 0.0001

4.4 Binary classifier

Afterwards, I trained a binary classifier to predict whether a particular audio file is considered an adversarial example or original.

- Audionet

  Also for the binary classifier, I have used the previous "Audionet" model but with a single difference in the last layer, only two output channels instead of ten.
  The network was trained with stochastic gradient descent with a batch size of 64 and constant momentum of 0.9 for 10 epochs with a learning rate of 0.0001. I have used 4000 original audio and 1000 adversarial audio in the training phase. For the testing phase I have used 200 adversarial audio and 300 original audio and the performance are excellent, reaching the 100% of accuracy.

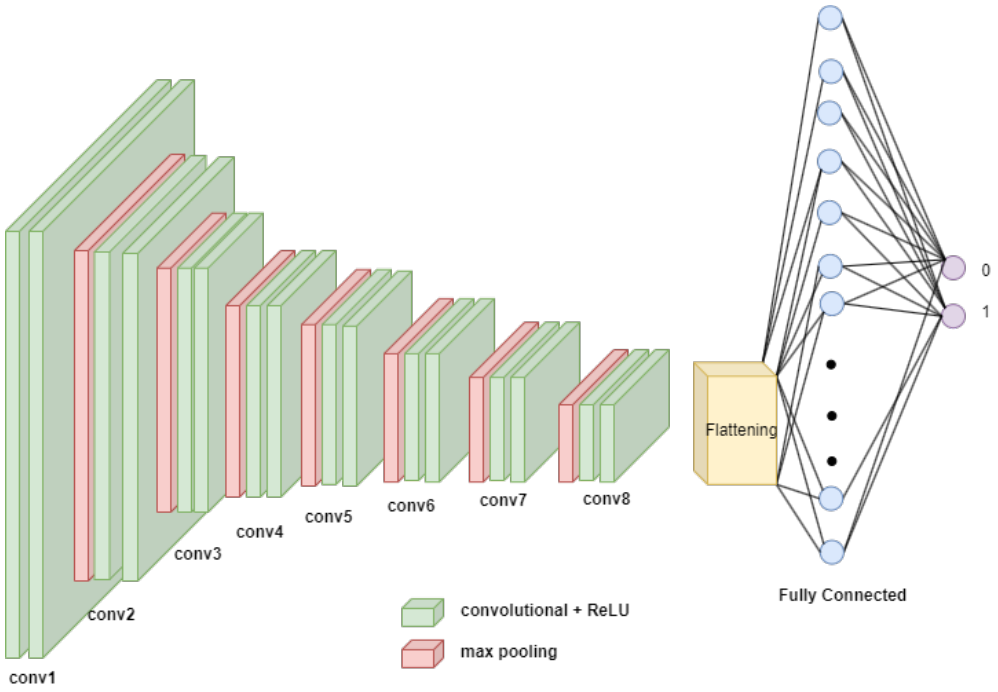


Figure 4.4: Audionet Binary Classifier

```
class RawAudioCNN(nn.Module):
    """Adaption of AudioNet (arXiv:1807.03418)."""
    def __init__(self):
        super().__init__()
        # 1 x 8000
        self.conv1 = nn.Sequential(
            nn.Conv1d(1, 100, kernel_size=3, stride=1, padding=2)
    ,
            nn.BatchNorm1d(100),
            nn.ReLU(),
            nn.MaxPool1d(3, stride=2))
        # 32 x 4000
        self.conv2 = nn.Sequential(
            nn.Conv1d(100, 64, kernel_size=3, stride=1, padding
    =1),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 64 x 2000
        self.conv3 = nn.Sequential(
            nn.Conv1d(64, 128, kernel_size=3, stride=1, padding
    =1),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 128 x 1000
        self.conv4 = nn.Sequential(
            nn.Conv1d(128, 128, kernel_size=3, stride=1, padding
    =1),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 128 x 500
        self.conv5 = nn.Sequential(
            nn.Conv1d(128, 128, kernel_size=3, stride=1, padding
    =1),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 128 x 250
        self.conv6 = nn.Sequential(
            nn.Conv1d(128, 128, kernel_size=3, stride=1, padding
    =1),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.MaxPool1d(2, stride=2))
        # 128 x 125
        self.conv7 = nn.Sequential(
```

```
43              nn.Conv1d(128, 64, kernel_size=3, stride=1, padding
    =1),
44              nn.BatchNorm1d(64),
45              nn.ReLU(),
46              nn.MaxPool1d(2, stride=2))
47          # 64 x 62
48          self.conv8 = nn.Sequential(
49              nn.Conv1d(64, 32, kernel_size=3, stride=1, padding=0)
    ,
50              nn.BatchNorm1d(32),
51              nn.ReLU(),
52              # maybe replace pool with dropout here
53              nn.MaxPool1d(2, stride=2))
54
55          # 32 x 30
56          self.fc = nn.Linear(32 * 30,2)
57
58      def forward(self, x):
59          x = self.conv1(x)
60          x = self.conv2(x)
61          x = self.conv3(x)
62          x = self.conv4(x)
63          x = self.conv5(x)
64          x = self.conv6(x)
65          x = self.conv7(x)
66          x = self.conv8(x)
67          x = x.view(x.shape[0], 32 * 30)
68          x = self.fc(x)
69          return x
```

- Spectrum

  I have used the same "CNNet" neural network with only 2 output channels.
  The network was trained with Adam optimizer with a batch size of 15 for 20 epochs with a learning rate of 0.0001.

  I have used 4000 original audio and 2500 adversarial examples for this classifier, 5200 of them used in the training phase and 1300 in the testing phase. The performance of this binary classifier are excellent, it is able to classify correctly with 100% accuracy.

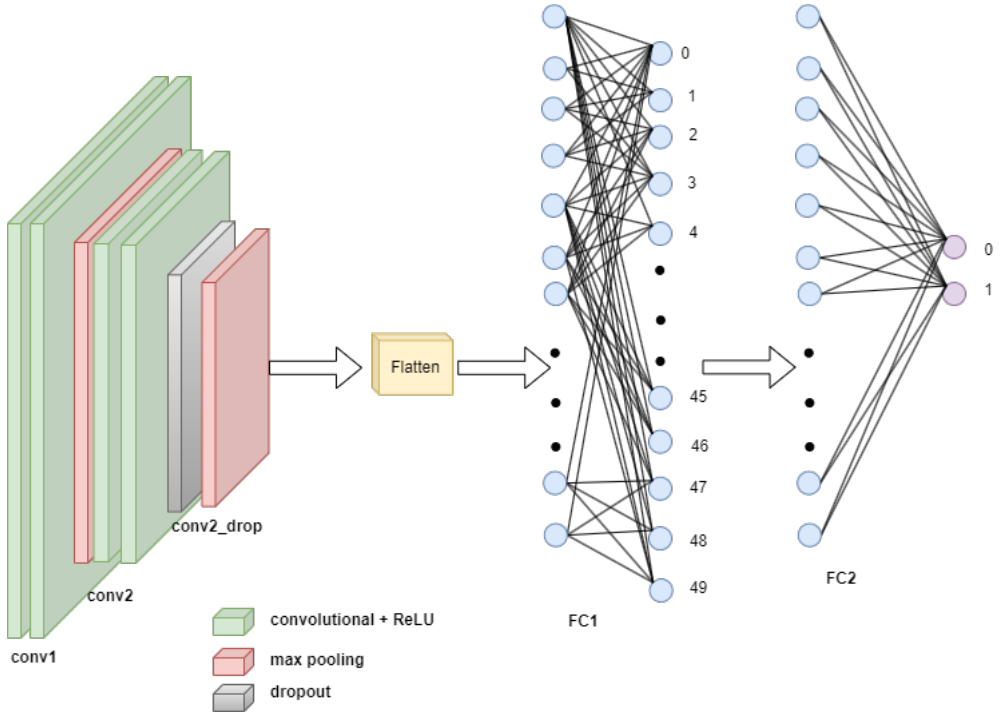


Figure 4.5: Spectrum Binary Classifier

```
class CNNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(51136, 50)
        self.fc2 = nn.Linear(50, 2)


    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)),
    2))
        #x = x.view(x.size(0), -1)
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = F.relu(self.fc2(x))
        return F.log_softmax(x,dim=1)
```

4.5 Combined model

Now, these models are used together, the binary classifier works as a filter for the inputs. If the inputs are classified as original will be sent to the original neural network that has high performance with both Spectrum and Audionet.

Instead, if it is classified as adversarial will be sent to the neural network with adversarial training that has worse performance than the original model but it is more robust to adversarial examples.

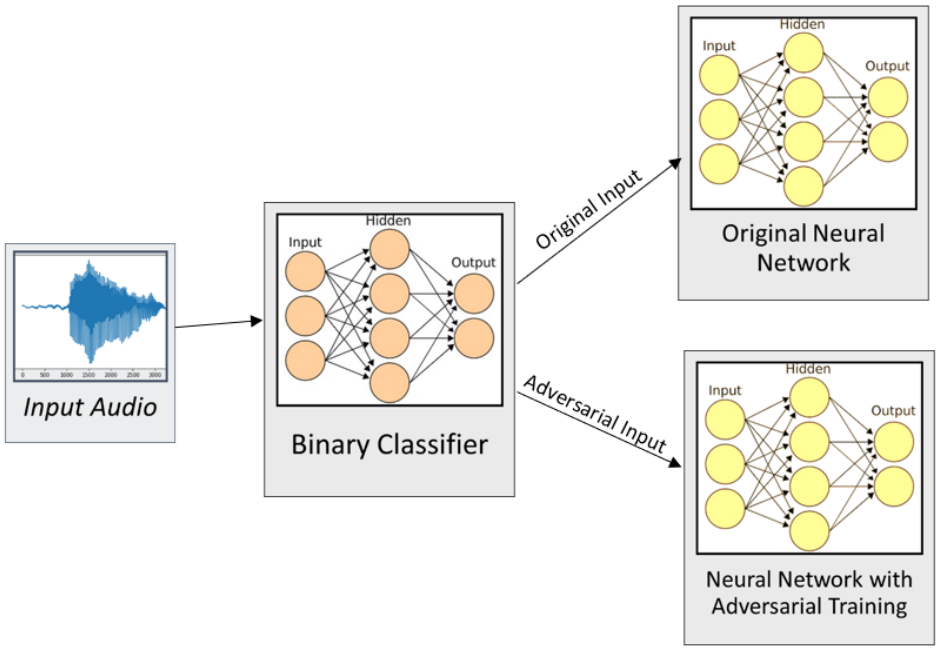


Figure 4.6: Network of neural network

4.5.1 Audionet

On average, considering the whole Audionet combined model and 2000 test data (1000 adversarial and 1000 original), the model has an accuracy of 0.82, a precision of 0.83, recall of 0.82 and f1-score 0.82

Audionet original prediction:				
class	precision	recall	f1-score	support
0	0.91	1.00	1.00	100
1	0.92	0.99	0.95	100
2	0.89	1.00	0.94	100
3	1.00	0.87	0.93	100
4	0.96	1.00	0.98	100
5	0.98	1.00	0.99	100
6	0.99	1.00	1.00	100
7	0.93	1.00	0.97	100
8	1.00	0.93	0.96	100
9	1.00	0.76	0.86	100
accuracy			0.95	1000
macro avg	0.96	0.96	0.95	1000

Table 4.1: Audionet original prediction

Audionet Adversarial prediction:				
class	precision	recall	f1-score	support
0	0.50	0.86	0.63	100
1	0.57	0.46	0.51	100
2	0.55	0.62	0.58	100
3	0.46	0.41	0.43	100
4	0.62	0.91	0.74	100
5	0.86	0.88	0.87	100
6	0.92	0.88	0.90	100
7	0.73	0.37	0.49	100
8	0.89	0.64	0.74	100
9	0.94	0.74	0.83	100
accuracy			0.68	1000
macro avg	0.70	0.68	0.67	1000

Table 4.2: Audionet adversarial prediction

4.5.2 Spectrum

On average, considering the whole Spectrum combined model and 2000 test data (1000 adversarial and 1000 original), the model has an accuracy of 0.84, a precision of 0.85, recall of 0.84 and f1-score 0.84

Spectrum original prediction:				
class	precision	recall	f1-score	support
0	0.85	0.99	0.91	100
1	0.80	0.98	0.88	100
2	0.97	0.92	0.94	100
3	0.99	0.96	0.97	100
4	0.99	1.00	1.00	100
5	0.97	1.00	0.99	99
6	1.00	1.00	1.00	100
7	0.97	0.89	0.93	100
8	1.00	1.00	1.00	100
9	0.99	0.72	0.83	103
accuracy			0.95	1002
macro avg	0.95	0.95	0.94	1002

Table 4.3: Spectrum original prediction

It can be seen that in total 1002 test samples were classified as original and 998 as adversarial. This is due to the binary classifier which did not correctly predict all the data.

Spectrum Adversarial prediction:				
class	precision	recall	f1-score	support
0	0.81	0.51	0.63	100
1	0.68	0.68	0.68	100
2	0.73	0.60	0.66	100
3	0.65	0.53	0.58	100
4	0.64	0.80	0.71	100
5	0.69	0.82	0.75	101
6	0.94	0.84	0.89	100
7	0.95	0.87	0.91	100
8	0.65	0.97	0.78	100
9	0.76	0.74	0.75	97
accuracy			0.74	998
macro avg	0.75	0.74	0.73	998

Table 4.4: Spectrum Adversarial prediction

4.6 Conclusion

In this chapter we have demonstrated how we can build a more robust model and what kind of architecture is better at predicting correctly even when there are adversarial examples.

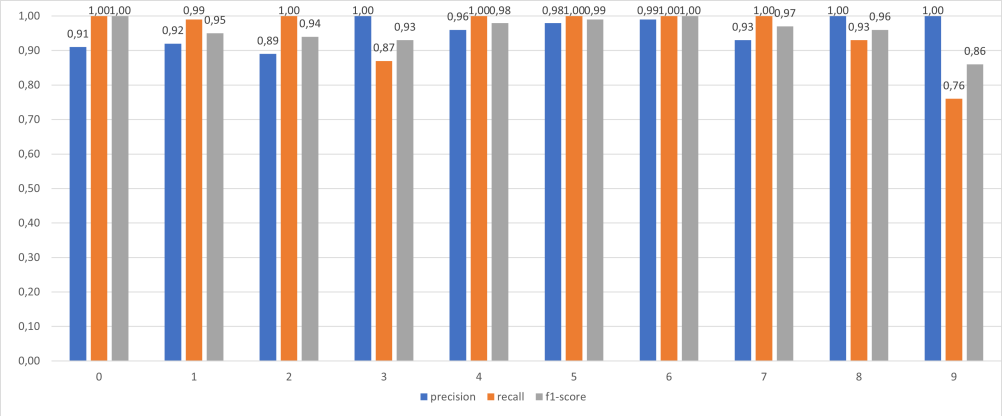


Figure 4.7: Audionet original predictions

As we can see in figure 4.7 and in figure 4.8 both models are very efficient and uniform on original audio examples, having very similar metric values across classes. Audionet model is slightly more efficient on the original inputs than on Spectrum model, considering the F1-score, there is an almost insignificant difference, of only 1%, due to worse performances on classes 0 and 9.

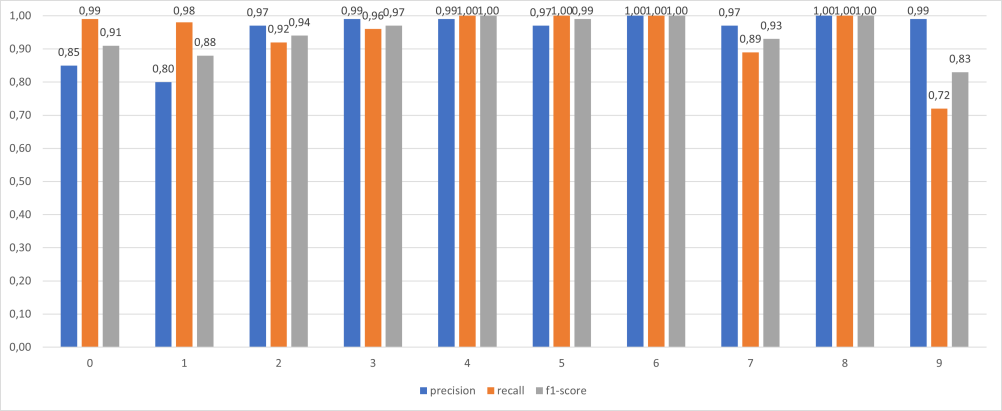


Figure 4.8: Spectrum original predictions

As we can see in the figures 4.9 and 4.10 that represent the models trained with the adversarial training, we immediately notice in both cases that the values of the

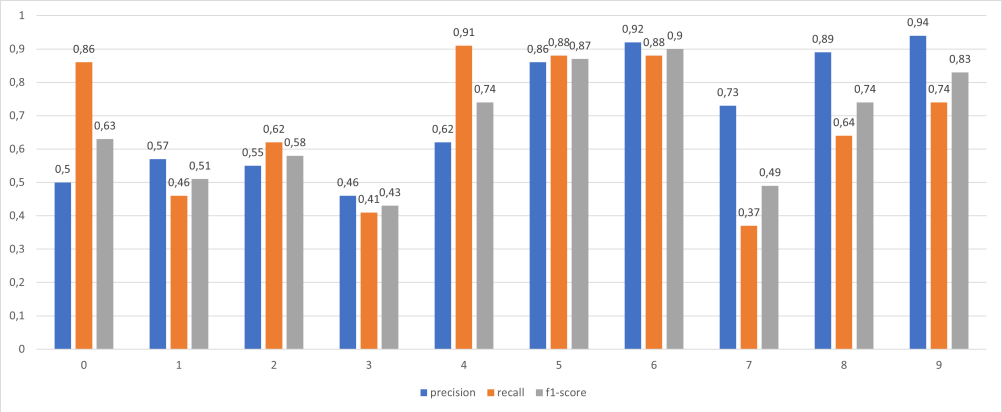


Figure 4.9: Audionet adversarial predictions

various metrics (precision, recall and F1-score) are much lower than the models working on original inputs.

In particular, we can see how the Spectrum model performs better than Audionet, in fact it has an accuracy of 74% compared to Audionet's 68%.

On the Audionet model it can be seen how the PGD attack is very effective especially on classes 1,2,3 and 7, causing a considerable decay in performance. about 30-40%, considering the F1-score metric.

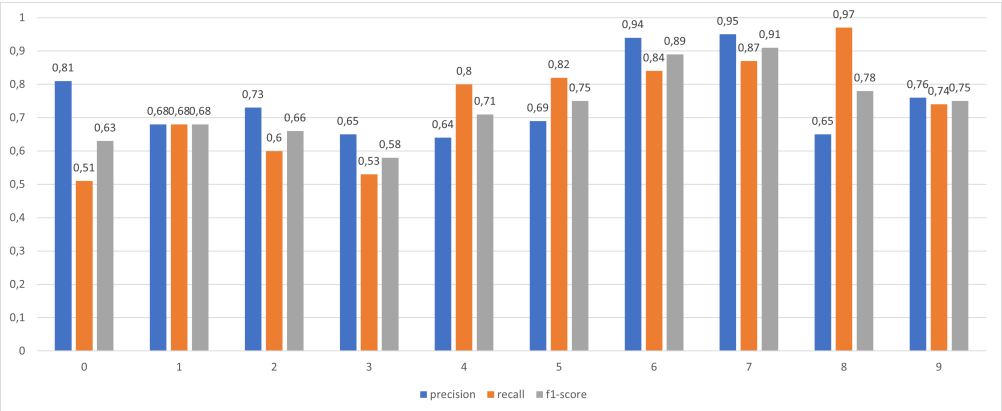


Figure 4.10: Spectrum adversarial predictions

Instead, the Spectrum model can be considered much more robust. Although the attack had a big impact on the 0,1,2 and 3 classes, it only caused a loss of 20-30% considering the F1-score.

Chapter 5

Voice Authentication and Voice Cloning

Voice authentication is a form of biometric authentication. The term Biometrics defines the use of unique physical characteristics such as facial features or fingerprints[41]. Speech recognition is a type of biometrics, and speech authentication is the use of a user's speech to perform the authentication process.

5.1 Voice Authentication

Like fingerprints, the user's voice can act as a unique indicator of a user's identity. This means that voice authentication brings many of the same benefits as other biometrics, including:

1. More difficult to falsify than other forms of authentication.
   It is possible to steal a password and copy or fake a token if security is not guaranteed. Biometric data is much more difficult to falsify, all things being equal, and much more difficult to steal through practices such as wide-ranging phishing attacks.

2. Affordable and accessible on a variety of devices.
   Biometrics is becoming incredibly common on devices such as laptops, tablets or smartphones. This greatly simplifies integrating next-level security into a productive device ecosystem for distributed teams.

3. Contactless access.
   Nowadays, this is an important property, almost unique to voice recognition (among a few other biometric data such as face scanning). Using speech

recognition, you don't have to touch anything, which, during the pandemic period, greatly helps to reduce the risks and be more protected from the virus.

4. It supports a simplified user experience.
   Getting into a system shouldn't be difficult, regardless of whether or not it is one of your employees or one of your customers or clients. With biometrics, secure authentication methods can rely only on the person's presence, rather than remembering a strong password.

Speech recognition works by dividing recordings into frequency segments and using them to create a single "fingerprint" to identify different inflexions and tones in a user's voice, which can then act as an artefact for identity verification.

5.1.1 Experiment

I created a notebook to do an example of voice authentication, reflecting a real case. Authentication systems often ask the user to say certain words to allow access to various services. For example, they can ask to say a list of numbers, for this, I have trained a model capable of distinguishing the numbers from 0 to 9 said by a particular speaker and the speaker himself.

In this notebook, I extracted features from the voice input using MFCC (Mel-frequency cepstral coefficients) followed by training and evaluation of the developed GMM (Gaussian mixture model).

For the feature extraction I have used the mfcc function of python_speech_features [42] library, with this configuration: rate=8000, length of the analysis window= 25 ms, the step between successive windows = 10 ms, number of cepstrum to return = 20 and a default value (26) Filterbanks.

```
mfcc_feat = mfcc.mfcc(audio,rate, 0.025, 0.01,20,appendEnergy = True)
```

For the training i have used the Gaussian Mixture function of sklearn.mixture library[43].

```
gmm = mixture.GaussianMixture(n_components = 16, max_iter  = 200,
    covariance_type='diag',n_init = 3)
gmm.fit(features)
```

For the testing phase i have used the log-likelihood score to decide the winner model.

```
# Read the test directory and get the list of test audio files
path_predicted_dict = dict()

for path in test_file:


    sr,audio = read(path)
    vector   = extract_features(audio,8000)

    log_likelihood = np.zeros(len(models))

    for i in range(len(models)):
        gmm    = models[i]  #checking with each model one by one
        scores = np.array(gmm.score(vector))
        log_likelihood[i] = scores.sum()

    winner = np.argmax(log_likelihood)
    print("\tDigit & Speaker detected as - ", speakers_digit[winner])
    path_predicted_dict[path] = speakers_digit[winner]
```

I have used 3600 audio for the training phase and 1400 for testing phase. The results were really good as the model was able to predict with 99.85% accuracy.

MFCC Features Extraction

The first and most relevant step in any ASR system is feature extraction, which allows us to consider only the components of the audio signal that are useful to identify the linguistic content and discard all the other things such as background noise.
It's important to understand that the sounds generated by a human being are filtered by the anatomy of the vocal tract, determining which sound comes out. If we can accurately determine the shape of the vocal tract, this should give us an accurate representation of the phoneme produced. The shape of the vocal tract shows itself in the envelope of the short-term power spectrum and the task of the MFCCs is to accurately represent this envelope.
The explanation on how to create MFCCs is based on the written tutorial by James Lyons, the creator of the library python_speech_features used in the experiment [44]. The various steps are as follows:

1. Frame the signal into short frames.
   An audio signal is constantly evolving, so for simplicity, it's assumed that on short time scales the audio signal does not change much. This is why the signal it's usually divided into frames of 20-40 ms (in my experiment I have used 25 ms). If the frame is much shorter we do not have enough samples to

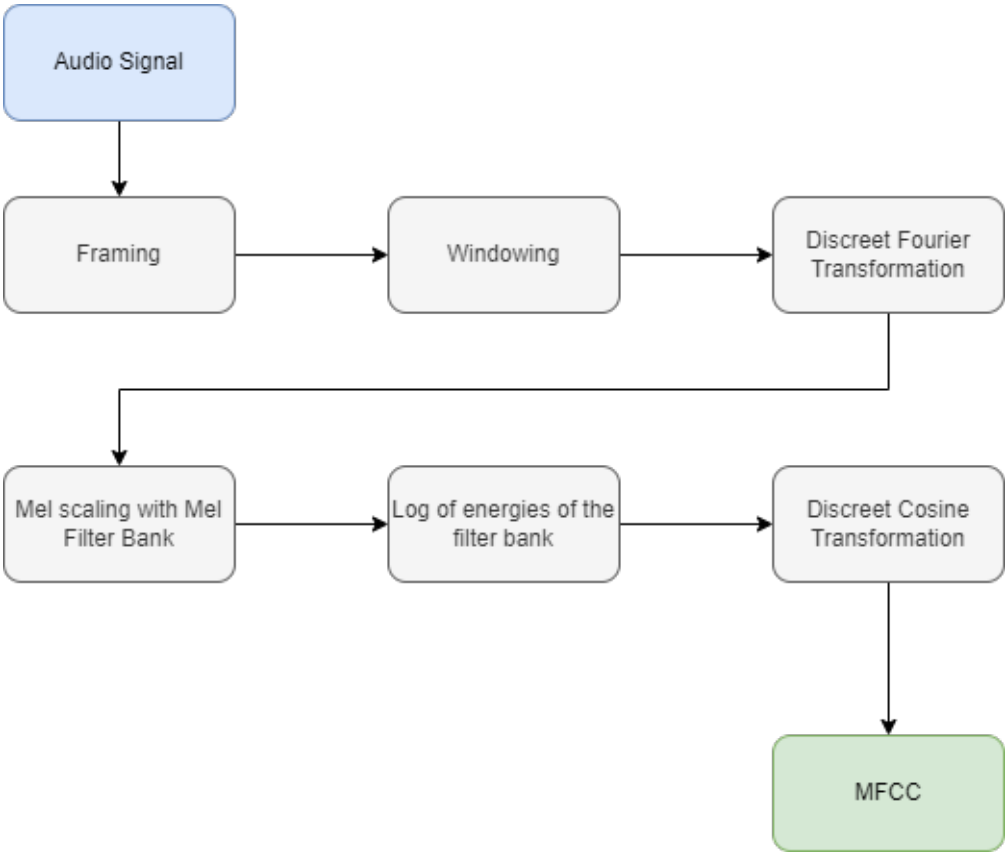


Figure 5.1: MFCC steps

obtain a reliable spectral estimate, if it is longer the signal changes too much in the whole frame. On each frame, a window is applied to thin the signal towards the frame boundaries.

2. For each windowed frame calculate the periodogram estimate of the power spectrum applying Discreet Fourier Transformation (DFT).
   This is motivated by the human cochlea (an organ in the ear) which vibrates at different points depending on the frequency of the incoming sounds. Depending on the position in the vibrating cochlea, different nerves are activated informing the brain that certain frequencies are present[45]. The periodogram estimate does a similar job of identifying which frequencies are present in the frame.

3. Apply the Mel filterbank to the power spectrum, and sum the energy in each filter.
   The spectral estimate of the periodogram still contains a lot of information that is not necessary for ASR. In detail, the cochlea can not distinguish the

difference between two closely spaced frequencies. This effect becomes stronger as the frequencies increase. For this reason, we take some groups of containers from the periodogram and summarize them to get an idea of how much energy exists in the various frequency regions. This is done by the Mel filter bank. We need to know approximately how much energy occurs at each point. The Mel scale shows exactly how to space filter banks and how wide to make them. To calculate filterbank energies, each filterbank is multiplied by the power spectrum, then add up the coefficients. After that, 26 numbers give us an indication of how much energy was present in each filterbank.

4. Logarithm of all the energies of the filter bank.
   After creating the energies of the filter bank, we take their logarithm. This is also motivated by human hearing because we don't hear volume on a linear scale. Typically, to double the perceived volume of a sound, we need to put in 8 times more energy. Logarithm allows using cepstral mean subtraction, which is a channel normalization technique [46].

5. The last step is computing the DCT (Discrete Cosine Transform) of the log filterbank energies to calculate the 26 cepstral coefficients, but for ASR are used only the lower 12.

The Mel scale relates the perceived frequency of a pure tone to its measured actual frequency. The following formula is used to convert the frequency to Mel scale :

$$M(f) = 1125\, ln\,(1 + f/700) \tag{5.1}$$

To go from Mel scale to frequency:

$$M^{-1}(m) = 700\,(exp\,(m/1125) - 1) \tag{5.2}$$

Training with Gaussian Mixture Model

After extracting features, we need to create a speaker model using some statistical model like GMM statistical model. To explain this procedure I have taken as reference the article "Gaussian Mixture Models Explained"[47] written by Oscar Contreras Carrasco and published on towardsdatascience.com. A Gaussian Mixture is a function that is comprised of several Gaussians, each identified by m $\in$ 1,. . . , M, where M is the number of clusters of our dataset. Each Gaussian m in the mixture is comprised of the following parameters:

- A mean μ that defines its centre.
- A covariance $\sum$ that defines its width.
- A mixing probability π that defines how big the Gaussian function will be.

We can see these parameters graphically: Here we can see that there are three

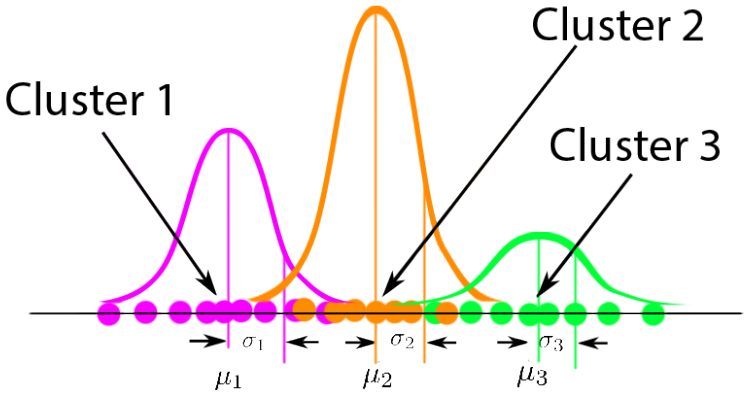


Figure 5.2: GMM parameters

Gaussian functions, therefore K = 3. Each Gaussian explains the data contained in each of the three available clusters. The mixing coefficients are themselves probabilities and must satisfy this condition:

$$\sum_{i=1}^{M} \pi_i = 1 \tag{5.3}$$

Now, to determine the optimal values for these parameters, we need to make sure that each Gaussian fits the data points belonging to each cluster, this is exactly

what maximum likelihood does.
In general, the Gaussian density function is given by:

$$b_i(\vec{x}) = N(\vec{x}|\mu, \sum) = \frac{1}{(2\pi)^{D/2}|\sum|^{1/2}} \; exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T \sum^{-1} (\vec{x}-\vec{\mu})\right) \tag{5.4}$$

Where our data points are represented by x, D is the number of dimensions of each data point. μ and $\sum$ are the mean and the covariance respectively. If we have a data set consisting of N = 1000 three-dimensional points (D = 3), then x will be a 1000 $\times$ 3 matrix. μ will be a 1 $\times$ 3 vector and $\sum$ will be a matrix 3 $\times$ 3. It may be useful to take the log of this equation, which is given by:

$$lnN(\vec{x}|\mu, \sum) = -\frac{D}{2} ln\ 2\pi \; - \frac{1}{2} ln \sum \; -\frac{1}{2}(\vec{x}-\vec{\mu})^T \sum^{-1} (\vec{x}-\vec{\mu}) \tag{5.5}$$

A Gaussian mixture density[48] is a weighted sum of M component densities given by:

$$p(\vec{x}|\lambda) = \sum_{i=1}^{M} \pi_i b_i(\vec{x}) \tag{5.6}$$

The complete density of the Gaussian mixture is parameterized by the covariance matrices, the mean vectors, and the weights of the mixtures of all the component densities. These parameters are collectively represented by the notation:

$$\lambda = \{\pi_i, \mu_i, \sum{}_i\} = 1, ..., M \tag{5.7}$$

For speaker identification, each speaker is represented by a GMM and is referred to by his model λ.

Given training speech data from a speaker's voice, the goal of speaker model training is to estimate the parameters of the GMM λ, which in some sense best matches the distribution of the training feature vectors. The most popular method for training GMM is maximum likelihood estimation.

Maximum likelihood estimation aims to find the model parameters, which maximize the likelihood of the GMM given the training data. For a sequence of T training vectors $X = (x_1.., x_T)$ the GMM likelihood can be written as:

$$P(x|\lambda) = \prod_{t=1}^{T} P(\vec{x_t}|\lambda) \tag{5.8}$$

Maximization of the quantity in 5.8 is accomplished by running the expectation-maximization algorithm. The idea is beginning with an initial model λ_0, to estimate

a new model λ_1 satisfying $p(X|\lambda_1) \geq p(X|\lambda_0)$.
The new model then becomes the initial model for the next iteration and the process is repeated until a certain convergence threshold is reached. The following formulas are used on each estimation iteration.

Mixture weights:

$$\vec{\pi_i} = \frac{1}{T}\sum_{t=1}^{T} P_i(i|\vec{x_t}, \lambda) \tag{5.9}$$

Means:

$$\vec{\mu_i} = \frac{\sum_{t=1}^{T} P_i(i|\vec{x_t}, \lambda)\vec{x_t}}{\sum_{t=1}^{T} P_i(i|\vec{x_t}, \lambda)} \tag{5.10}$$

Variances:

$$\vec{\sigma_i^2} = \frac{\sum_{t=1}^{T} P_i(i|\vec{x_t}, \lambda)\vec{x_t^2}}{\sum_{t=1}^{T} P_i(i|\vec{x_t}, \lambda)} - \vec{\mu_i}^2 \tag{5.11}$$

The posterior probability for an acoustic class is given by:

$$P(i|\vec{x_t}) = \frac{P_i b_i(\vec{x_t})}{\sum_{k=1}^{M} P_k b_k(\vec{x})} \tag{5.12}$$

For speaker identification, a group of S speakers S=(1,2,...,S) is represented by GMM's λ_1,λ_2...λ_s. The objective is to find the speaker model, which has the maximum posterior probability for a given observation sequence.

$$\hat{S} = \arg\max P(\lambda_k|X) = \arg\max \frac{P(X|\lambda_k)P(\lambda_k)}{p(X)} \tag{5.13}$$

Where the second equation is due to Bayes's rule.
Assuming equally likely speakers ($P(\lambda_k)= 1/S$) and noting that p(X) is the same for all speaker models, the classification becomes:

$$\hat{S} = \arg\max P(X|\lambda_k) \tag{5.14}$$

Finally, with logarithms the speaker identification system gives:

$$\hat{S} = \arg\max \sum_{t=1}^{T} logP(x_t|\lambda_k) \tag{5.15}$$

In which $P(x_t|\lambda_k)$ is given in equation 5.6 .

5.1.2 Disadvantages and risks

While AI and biometrics have evolved over time, that doesn't mean voices don't have weaknesses. There are several disadvantages, some unique to voice technology and others shared by other biometric data.
Voice authentication is:

1. Hackable.
   As with other types of data, the system stores voice data on a server or database. If that point isn't protected, malicious people can steal the data and use it to create matching fake records.

2. Non-replaceable.
   If the biometrics is compromised, it cannot be replaced as a password because you cannot just change your voice.

3. Not always applicable
   Speech recognition needs a quiet area. The noises and voices of the surrounding environment outside of the user's voice can interfere with authentication.

4. Not infallible.
   This also applies to speech recognition. However, industry innovation in artificial intelligence has made biometrics authentication methods practicable even for secure applications such as payment processing. In any case, it is important not to rely on a single form of identification but to always use at least 2FA (2-Factor Authentication).

Some of these limitations can lead to real security problems or limit the use of this technology.:

1. Changes to the voice can affect access.
   Users who relied on facial recognition to unlock their phones quickly learned during the pandemic (and required the use of face masks) how relying on a single method can restrict access. A noisy environment can affect authentication, but even normal incidents can include a cold, sore throat, or small changes in voice, accent, or speech.

2. Fake voice
   New technologies can manipulate voices to sound like other voices and, in some cases, can fool biometrics authentication systems. These voice cloning techniques go hand in hand with the improvement of voice authentication technologies and are becoming more and more common.

5.2 Deep Fake Voice Cloning

Voice cloning is the creation of an artificially produced voice that is capable of simulating a natural human voice. In some cases, the difference between the real and the fake voice is unnoticeable to the average person.
It takes snippets of a text recorded by a person and applies artificial intelligence (AI) to dissect speech patterns from speech samples. This gives the user the ability to create audio recordings or streams that were not spoken by the owner of the voice[49].

To produce fake voices I have used a repository on GitHub [50] that contains a neural network-based system for text-to-speech (TTS) synthesis that can be considered an implementation of Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis (SV2TTS) [51] that is able to generate speech audio in the voice of different speakers, including those unseen during training.

This system consists of three independently trained components:

- *Encoder*, that is able to generate a fixed-dimensional embedding vector from only seconds of reference speech from a target speaker;
- *Synthesizer* based on Tacotron 2 that generates a mel spectrogram from text, conditioned on the speaker embedding;
- *Vocoder* that converts the mel spectrogram generated by the synthesizer into time domain waveform samples.

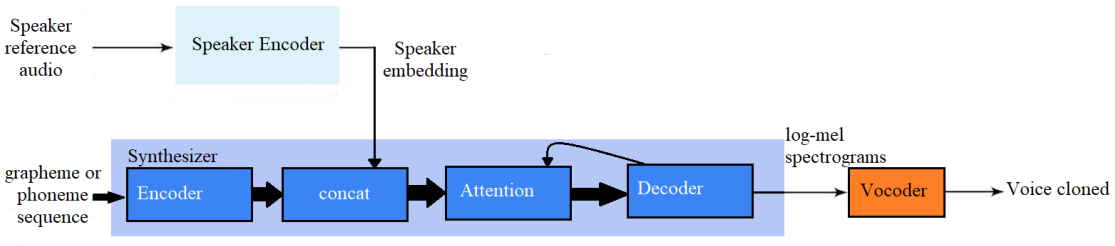


Figure 5.3: Voice cloning model architecture

Now we see each of these components in detail.

5.2.1 Encoder

The speaker encoder is used to condition the synthesis network on a reference voice signal from the desired target speaker. For a good generalization, the use of a representation that captures the characteristics of the different speakers and identifies them using only a short adaptation signal, regardless of the background noise and its phonetic content, is fundamental. These requirements are met using a discriminatory speaker model trained on a text-independent speaker verification task.

This model refers to Generalized End-to-End Loss for Speaker Verification [52], which proposed an extremely accurate and scalable neural network framework for speaker verification. A sequence of log-mel spectrogram frames computed from a speech expression of arbitrary length is mapped from this network to a d-vector, a fixed-size embedding vector.

The network is trained to optimize generalized end-to-end (GE2E) speaker verification loss, so that the embodiment of the expressions of the same speaker has a high cosine similarity, while those of the expressions of different speakers are very distant from each other in the embedding space.

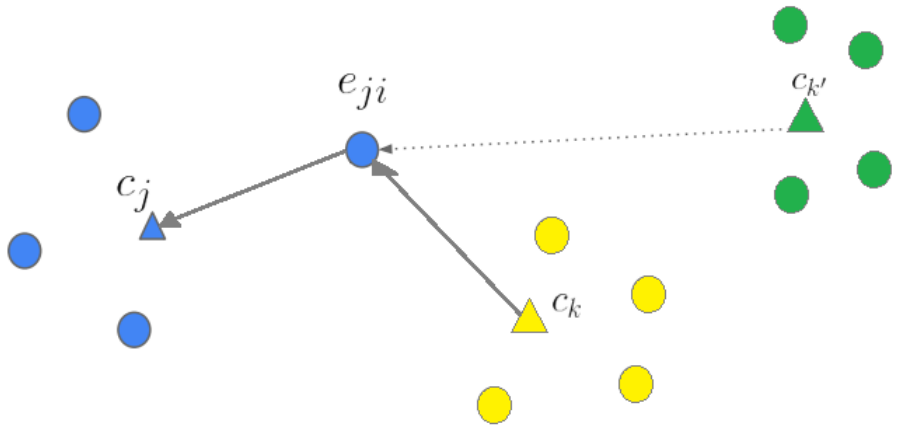


Figure 5.4: GE2E loss pushes the embedding towards the centroid of the true speaker, and away from the centroid of the most similar different speaker.

In the training phase, the model computes the embeddings $\mathbf{e}_{ij}$ $(1 \leq i \leq N, 1 \leq j \leq M)$ of M utterances of fixed duration from N speaker and for each speaker is derived a speaker embedding:

$$c_i = \frac{1}{M} \sum_{j=1}^{M} e_{ij} \tag{5.16}$$

The similarity matrix S in this application is defined as the scaled cosine similarities between each embedding vector $\mathbf{e}_{ij}$ to all centroids $\mathbf{c}_k$ $(1 \leq k \leq N)$

$$S_{ij,k} = w \cdot \cos(e_{ij}, c_k) + b = w \cdot e_{ij} \cdot ||c_k||_2 + b \tag{5.17}$$

w and b are learnable parameters.

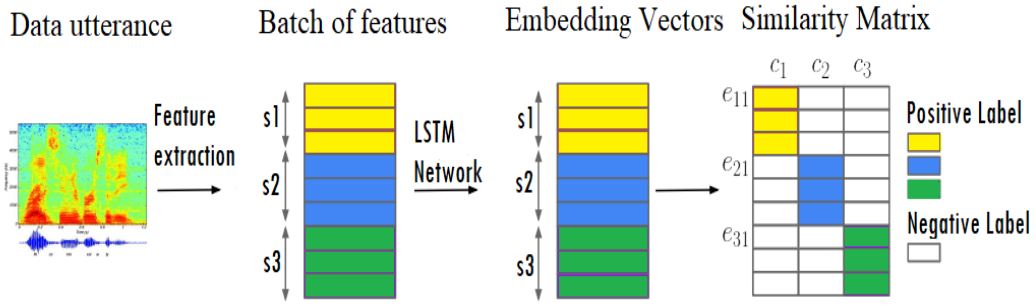


Figure 5.5: GE2E system overview[52]

A good model is expected to output high similarity values when an expression matches the speaker and lower otherwise. For this the loss is the sum of row-wise softmax losses.
The training dataset consists of speech audio examples segmented into 1.6 seconds and associated speaker identity labels and no transcripts are used.

Although the model's architecture is capable of handling variable-length inputs, it is reasonable to expect it to work better with sentences of the same duration as those seen in training. Therefore, at the time of inference, an expression is split into overlapping 1.6-second segments of 50% and the encoder forwards each segment individually. The network is run independently on each window, and the outputs are averaged and normalized to create the final embedding of the utterance.

5.2.2 Synthesizer

The synthesizer used in this repository is Tacotron 2 [53] without Wavenet using an open-source Tensorflow implementation of Tacotron 2 modified implementing the modifications added by SV2TTS.

The Tacotron model that predicts mel spectrograms from text is a recurrent sequence-to-sequence model with attention. The encoder (blue blocks in the 5.6 figure) transforms the input text into a hidden feature representation of fixed size. The autoregressive decoder (orange blocks) uses this characteristic representation to produce one spectrogram frame at a time.

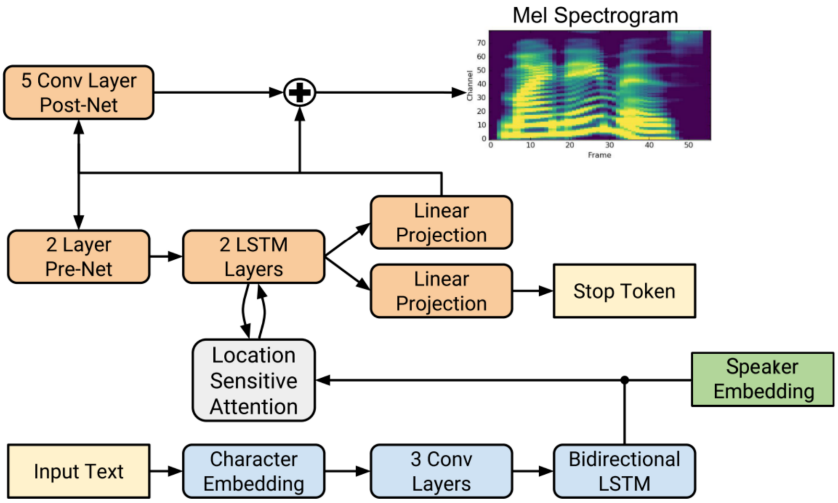


Figure 5.6: Tacotron model [53]

The individual characters of the text sequence are first embedded as vectors. Convolutional levels follow, in order to increase the width of a single frame of the encoder. These frames passed through a bidirectional LSTM (Long short-term memory) are used to produce the encoder output frames. This is where SV2TTS makes an architectural change: a speaker embedding is chained to each frame emitted by the Tacotron encoder.

The attention mechanism takes care of the encoder output frames for generating the decoder input frames. Each decoder input frame is concatenated with the previous decoder frame's output passed through a pre-net containing 2 fully connected layers, making the model autoregressive.

The pre-net output and attention context vector are concatenated and passed through a stack of two uni-directional LSTM layers before being projected onto a single frame of the mel spectrogram. Another projection of the same vector onto a scalar allows the network to predict for itself that it should stop generating frames by emitting a value above a certain threshold. The concatenation of the LSTM output and the attention context vector is projected through a linear transformation to predict the target spectrogram frame.

Finally, the predicted mel spectrogram is passed through a 5-layer convolutional post-net which predicts a residual to add to the prediction to improve the overall reconstruction.

There are also a few cleaning procedures: forcing all characters to ASCII, normalizing white spaces, replacing abbreviations and numbers by their complete textual form and making all characters lowercase.

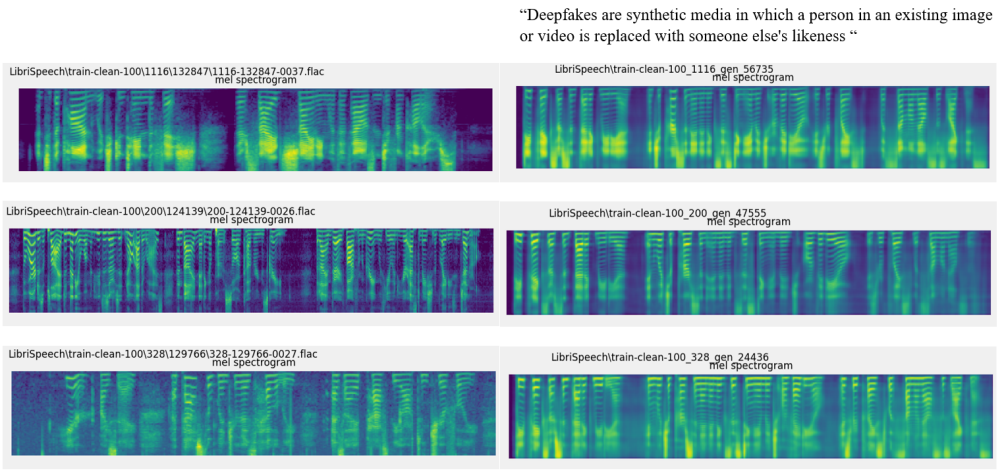


Figure 5.7: Synthesis of a sentence in different voices

On the left, we can see the reference expressions used to generate the speaker embedding of three different speakers (I have taken random sentences from the Librispeech dataset), and on the right the corresponding outputs of the synthesizer. As we can see, the generated spectrograms are similar to each other but differ from each other due to the phonetic characteristics of the speaker.

5.2.3 Vocoder

The vocoder model that was used is an open-source PyTorch implementation based on WaveRNN [54] but has some different design choices made by the user GitHub fatchord (`https://github.com/fatchord/WaveRNN`). The author of this repository used to create fake cloned voice affirms that according to the source code and the diagram of "fatchord" the architecture is as follows.

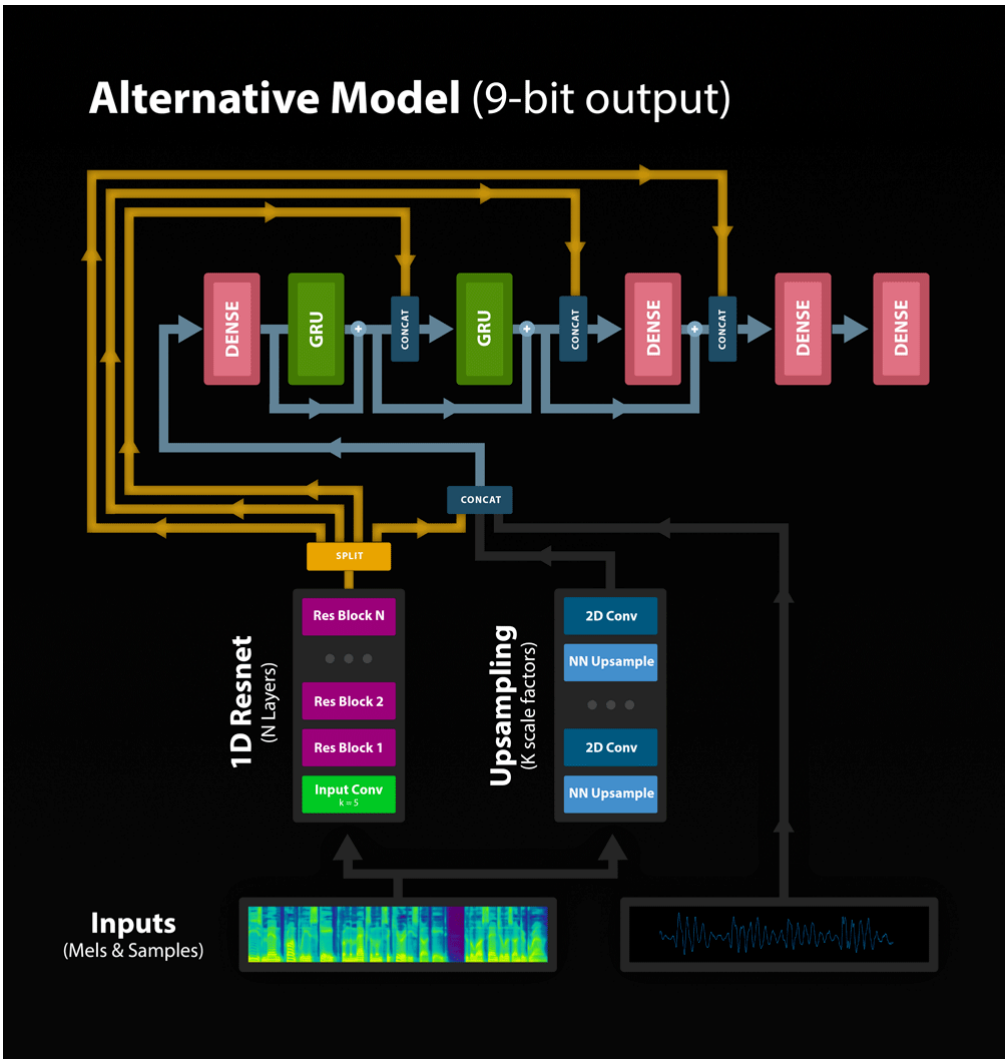


Figure 5.8: Alternative WaveRNN architecture. Image taken from `https://github.com/fatchord/WaveRNN`

The model uses the spectrogram as an input to generate characteristics that will affect the layers when transforming the spectrogram mel into a waveform.

The conditioning vector after is divided equally in four directions along the channel size, the first portion is concatenated with the upsampled spectrogram and waveform segment of the previous step.
The final vector undergoes several jump link transformations: the first two GRU (Gated Recurrent Unit) layers at the end a dense layer. In each step, the vector is chained with the intermediate waveforms.

Finally, after the last two dense layers, the cloned voice has been created and it is potentially indistinguishable from the real person.

5.3 Fake voice detection

"It's like Photoshop for voice" said Zohaib Ahmed, CEO of Resemble AI, about his company's voice cloning technology.

As for the photos bad Photoshop jobs are easily detected. But if deepfake audio is good quality, many times it can seem real, and voice clones are getting better every day as well as deep-learning systems are getting smarter and making more authentic voices.

The longer an audio clip is, the more likely you are to notice that something is wrong. For the shorter clips, however, you may not notice that it's synthetic, especially if you have no reason to doubt its legitimacy.

Sound quality is also a very important component, the clearer it is, the easier it is to notice the signs of an audio deepfake. If someone is speaking directly into a studio-quality microphone, you will be able to hear closely. But a poor quality phone call recording or conversation captured on a portable device in a noisy parking lot will be much more difficult to evaluate[55].

Even though humans have trouble separating the real from the fake, computers don't have the same limitations. Fortunately, there are already voice verification tools. Pindrop has one that pits deep learning systems against each other. Use both to find out if an audio sample belongs to the person it should be.
However, it also examines whether a human can also make all the sounds in the sample (for example, two vocal sounds have a minimum possible distance from each other. This is because it is not physically possible to tell them faster due to the speed with which the muscles of the mouth and vocal cords can reconfigure).Thanks to its anti-fraud solution, Pindrop claims to have reduced fraudulent calls to its customers by 80% and avoided $1.2 billion in losses since 2012.

Another tool is Resemble AI which is also directly dealing with the detection problem with Resemblyzer, an open-source deep learning tool available on GitHub [56]. It can detect fake voices and perform speaker verification.

I have also created a simple notebook, that works as a binary classifier that can detect if a voice is fake or real by analyzing the mfcc spectrogram of the audio using the CNNnet model trained for 10 epochs with 4000 real audio and 1250 fake audio (created with the previous voice cloning system).

This experiment consists of two main phases: the creation of the fake voice and the detection. To create the fake (cloned) voice I have used the previous system. The most relevant parts of the code of this notebook are the following:

```
#EXAMPLE: CREATE 5 CLONED VOICE AUDIO
# First, we load the wav using the function that the speaker encoder provides. This is
# important: there is preprocessing that must be applied.

# The following two methods are equivalent:
# - Directly load from the filepath:
for j in range(5):
    f=audio_files[random.randint(1,len(audio_files))]
    print(f)
    print(f"j:{j}")
    preprocessed_wav = encoder.preprocess_wav(f)
    # - If the wav is already loaded:
    original_wav, sampling_rate = librosa.load(str(f))
    preprocessed_wav = encoder.preprocess_wav(original_wav, sampling_rate)
    print("Loaded file succesfully")

# Then we derive the embedding. There are many functions and parameters that the
# speaker encoder interfaces. These are mostly for in-depth research. You will typically
# only use this function (with its default parameters):
    embed = encoder.embed_utterance(preprocessed_wav)
    print("Created the embedding")
    ## Generating the spectrogram
#text = input("Write a sentence (+-20 words) to be synthesized:\n")
    for n in range(1):
        gen = DocumentGenerator()
        text=gen.sentence()    ##USING RANDOM GENERATOR WE CAN OBTAIN RANDOM SENTENCES (TAKEN FROM BOOKS) AN
        print(text)            ## AND USE THIS TEXT AS INPUT FOR THE SYNTESIZER
        text_file.append(text)


       # The synthesizer works in batch, so you need to put your data in a list or numpy array
```

```
32          texts = [text]
33          embeds = [embed]
34          specs = synthesizer.synthesize_spectrograms(texts, embeds)
35          spec = specs[0]
36          print("Created the mel spectrogram")
37
38
39          ## Generating the waveform
40          print("Synthesizing the waveform:")
41
42
43
44          # Synthesizing the waveform is fairly straightforward.
    Remember that the longer the
45          # spectrogram, the more time-efficient the vocoder.
46          generated_wav = vocoder.infer_waveform(spec)
47
48
49          ## Post-generation
50          # There's a bug with sounddevice that makes the audio cut one
     second earlier, so we
51          # pad it.
52          generated_wav = np.pad(generated_wav, (0, synthesizer.
    sample_rate), mode="constant")
53
54          # Trim excess silences to compensate for gaps in spectrograms
     (issue #53)
55          generated_wav = encoder.preprocess_wav(generated_wav)
56
57          # Play the audio (non-blocking)
58
59
60          # Save it on the disk
61          speaker_audio=f.split("\\")[1].split(".")[0].split("-")[0]
62          filename = f"voice_cloned_test/voice_cloned_%02d_{
    speaker_audio}.wav" % num_generated
63          print(generated_wav.dtype)
64          sf.write(filename, generated_wav.astype(np.float32),
    synthesizer.sample_rate)
65          num_generated += 1
66          print("\nSaved output as %s\n\n" % filename)
67
68
69          with open("voice_cloned_test/Output.txt", "w") as output_file
    :  ##WRITE TEXT TO A FILE TO COMPARE WITH THE CREATED AUDIO
70              i=0
71              for t in text_file:
72                  print(f"{i}: {t}", file=output_file)
73                  i=i+1
```

Regarding the detection phase, I have created a Binary Classifier that uses the Spectrum model (CNNet) of chapter 4, with the only difference being that it uses mfcc spectrogram instead of the ones used previously. This choice is due to the fact that the model performs better with this type of spectrogram.

I have trained the model for 10 epochs using 1200 cloned voices and 4000 original voices I have used 500 audio files for the test phase, 100 created artificially (labelled fake) and 400 originals. The results, as we can see in the table 5.1, are quite good having achieved an accuracy of 93%.

Fake voice detection results				
class	precision	recall	f1-score	support
Fake	0.77	0.94	0.85	100
Original	0.98	0.93	0.96	400
accuracy			0.93	500
macro avg	0.88	0.94	0.90	500
weighted avg	0.94	0.93	0.93	500

Table 5.1: Fake voice detection

Chapter 6

Conclusion

In this work, we have seen the possible types of attacks such as FGSM and PGD that can be applied to speech recognition systems, significantly decreasing system performance.

Furthermore, we have noticed that MP3 Compression is not the best defence and does not guarantee complete protection from these evasion attacks.
For this problem, a combined model was subsequently presented that was capable of detecting the attack and reacting correctly to it, through Adversarial Training, causing only a small loss in terms of performance accuracy.

In the end, the topic of voice recognition was addressed, its application as an authentication method, and above all the great problem of voice cloning. Finally, some tools for detecting an artificially constructed voice are also presented.
Automatic speech recognition technologies are being improved day by day and thanks to advances in this field, intelligent voice control devices are becoming more and more popular.

Today, smart speakers like Google Home or Amazon Echo are already part of our daily life.

Recent studies show that adversarial examples (AE) can pose a serious threat to an automatic speech recognition system (ASR) "white box" when his machine learning model with all its parameters is exposed to the adversary.
Less clear is how realistic such a threat would be to commercial devices, such as Google Home, Cortana, Echo, etc., whose models are not publicly available.

Leveraging the learning model behind the black-box ASR system is arduous and gruelling, due to the presence of complicated preprocessing and feature extraction

processes even before AEs can affect the model.

However, the extensive use of voice for critical system auditing also leads to security issues, the implications of which are not yet fully understood.
Specifically, the voice is an open channel and therefore the commands received by the ASR devices could come from any source. In recent years, researchers have shown that unauthorized voice commands can be injected into wireless signals [57], in the form of noise [58] or even inaudible ultrasound [59], to stealthily gain control of voice controllable systems (VCS) devices.

Recently, adversarial examples (AEs) have been used to exploit ASR systems. For example, Carlini has successfully attacked DeepSpeech (the open source model of Mozilla) using AEs, Yuan with his CommanderSong [60] that automatically generates adversarial examples embedded into songs to attack Kaldi Aspire Chain Model over-the-air.
This demonstrates that real-world ASR systems are vulnerable in a white box model, that is their internal parameters are exposed to the outside.

And even if efficient black box attacks are not officially known, it does not mean that black box models are safe as the world of adversarial attacks is constantly evolving and as the Devil's Whisper[61] attack demonstrates, even commercial black box ASR systems are vulnerable.

Bibliography

[1] *Adversarial Robustness Toolbox*. https://github.com/Trusted-AI/adversarial-robustness-toolbox (cit. on pp. 3, 19).

[2] *What Is The Difference Between Artificial Intelligence And Machine Learning?* https://www.forbes.com/sites/bernardmarr/2016/12/06/what-is-the-difference-between-artificial-intelligence-and-machine-learning/?sh=631866112742 (cit. on p. 4).

[3] *What is computer vision?* https://www.ibm.com/topics/computer-vision (cit. on p. 4).

[4] *Fraud detection*. https://www.altexsoft.com/whitepapers/fraud-detection-how-machine-learning-systems-help-reveal-scams-in-fintech-healthcare-and-ecommerce/ (cit. on p. 5).

[5] Francesco Ricci, Lior Rokach, and Bracha Shapira. «Recommender Systems: Techniques, Applications, and Challenges». In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, and Bracha Shapira. New York, NY: Springer US, 2022, pp. 1–35. ISBN: 978-1-0716-2197-4. DOI: 10.1007/978-1-0716-2197-4_1. URL: https://doi.org/10.1007/978-1-0716-2197-4_1 (cit. on p. 5).

[6] *Chatbot*. https://www.zendesk.com/service/messaging/chatbot/ (cit. on p. 5).

[7] *IBM Speech Recognition*. https://www.ibm.com/cloud/learn/speech-recognition (cit. on p. 5).

[8] *Supervised learning*. https://www.ibm.com/cloud/learn/supervised-learning (cit. on p. 6).

[9] *Unupervised learning*. https://www.guru99.com/unsupervised-machine-learning.html (cit. on p. 6).

[10] *semi-supervised-learning*. https://www.altexsoft.com/blog/semi-supervised-learning/ (cit. on p. 6).

[11] *reinforcement-learning*. https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292 (cit. on p. 6).

[12] Steven Walczak and Narciso Cerpa. «Artificial Neural Networks». In: *Encyclopedia of Physical Science and Technology (Third Edition)*. Ed. by Robert A. Meyers. Third Edition. New York: Academic Press, 2003, pp. 631–645. ISBN: 978-0-12-227410-7. DOI: https://doi.org/10.1016/B0-12-227410-5/00837-1. URL: https://www.sciencedirect.com/science/article/pii/B0122274105008371 (cit. on p. 7).

[13] *backpropagation*. https://www.javatpoint.com/pytorch-backpropagation-process-in-deep-neural-network (cit. on p. 7).

[14] *decision-tree*. https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html (cit. on p. 9).

[15] *random-forest*. https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/ (cit. on p. 10).

[16] *regression*. https://www.nvidia.com/en-us/glossary/data-science/linear-regression-logistic-regression/ (cit. on p. 11).

[17] *support-vector-machine*. https://www.analyticsvidhya.com/blog/2021/06/support-vector-machine-better-understanding/ (cit. on p. 12).

[18] *limitations$_l$ack$_g$ood$_d$ata*. https://towardsdatascience.com/the-limitations-of-machine-learning-a00e0c3040c6 (cit. on p. 14).

[19] *Overfitting*. https://web.archive.org/web/20171107014257/https://en.oxforddictionaries.com/definition/overfitting (cit. on p. 14).

[20] *Overfitting solutions*. https://corporatefinanceinstitute.com/resources/knowledge/other/overfitting/ (cit. on p. 14).

[21] Hyun-il Lim. «A Study on Dropout Techniques to Reduce Overfitting in Deep Neural Networks». In: *Advanced Multimedia and Ubiquitous Engineering*. Ed. by James J. Park, Vincenzo Loia, Yi Pan, and Yunsick Sung. Singapore: Springer Singapore, 2021, pp. 133–139. ISBN: 978-981-15-9309-3 (cit. on p. 15).

[22] *algorithm-compas-sentencing-bias*. https://www.nytimes.com/2017/10/26/opinion/algorithm-compas-sentencing-bias.html?smid=url-share (cit. on p. 15).

[23] *Google Photos unethical*. https://www.theverge.com/2018/1/12/16882408/google-racist-gorillas-photo-recognition-algorithm-ai (cit. on p. 15).

[24] *Google Photos unethical - BBC*. https://www.bbc.com/news/technology-33347866 (cit. on p. 15).

[25] Antonio Emanuele Cinà, Kathrin Grosse, Ambra Demontis, Battista Biggio, Fabio Roli, and Marcello Pelillo. *Machine Learning Security against Data Poisoning: Are We There Yet?* 2022. DOI: 10.48550/ARXIV.2204.05986. URL: https://arxiv.org/abs/2204.05986 (cit. on p. 16).

[26] *evasion-attacks*. https://towardsdatascience.com/evasion-attacks-on-machine-learning-or-adversarial-examples-12f2283e06a1 (cit. on p. 16).

[27] Yi Shi, Yalin Sagduyu, and Alexander Grushin. «How to steal a machine learning classifier with deep learning». In: Apr. 2017, pp. 1–5. DOI: 10.1109/THS.2017.7943475 (cit. on p. 16).

[28] Kuan-Chieh Wang, Yan Fu, Ke Li, Ashish Khisti, Richard Zemel, and Alireza Makhzani. *Variational Model Inversion Attacks*. 2022. DOI: 10.48550/ARXIV.2201.10787. URL: https://arxiv.org/abs/2201.10787 (cit. on p. 16).

[29] Nikolay Kucherov, Maxim Deryabin, and Mikhail Babenko. «Homomorphic Encryption Methods Review». In: Jan. 2020, pp. 370–373. DOI: 10.1109/EIConRus49466.2020.9039110 (cit. on p. 17).

[30] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. «Oblivious Multi-Party Machine Learning on Trusted Processors». In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 619–636. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko (cit. on p. 17).

[31] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. «Innovative Instructions and Software Model for Isolated Execution». In: *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. Tel-Aviv, Israel: Association for Computing Machinery, 2013. ISBN: 9781450321181. DOI: 10.1145/2487726.2488368. URL: https://doi.org/10.1145/2487726.2488368 (cit. on p. 17).

[32] Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. «Adversarial Attacks and Defenses in Deep Learning». In: *Engineering* 6 (Jan. 2020). DOI: 10.1016/j.eng.2019.12.012 (cit. on p. 18).

[33] Sören Becker, Marcel Ackermann, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. *Interpreting and Explaining Deep Neural Networks for Classification of Audio Signals*. 2019. arXiv: 1807.03418 [cs.SD] (cit. on pp. 20, 23).

[34] Nicholas Carlini and David Wagner. *Audio Adversarial Examples: Targeted Attacks on Speech-to-Text*. 2018. arXiv: 1801.01944 [cs.LG] (cit. on p. 24).

[35] Songxiang Liu, Haibin Wu, Hung-yi Lee, and Helen Meng. *Adversarial Attacks on Spoofing Countermeasures of automatic speaker verification*. 2019. arXiv: 1910.08716 [eess.AS] (cit. on p. 27).

[36] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: 1706.06083 [stat.ML] (cit. on p. 28).

[37] Lea Schönherr, Katharina Kohls, Steffen Zeiler, Thorsten Holz, and Dorothea Kolossa. *Adversarial Attacks Against Automatic Speech Recognition Systems via Psychoacoustic Hiding*. 2018. arXiv: 1808.05665 [cs.CR] (cit. on p. 29).

[38] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML] (cit. on p. 39).

[39] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. *Adversarial Training for Free!* 2019. DOI: 10.48550/ARXIV.1904.12843. URL: https://arxiv.org/abs/1904.12843 (cit. on p. 39).

[40] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. *Adversarial Training for Free!* 2019. arXiv: 1904.12843 [cs.LG] (cit. on p. 40).

[41] *voice-authentications*. https://www.1kosmos.com/biometric-authentication/voice-authentication/ (cit. on p. 51).

[42] *python-speech-features*. https://python-speech-features.readthedocs.io/en/latest (cit. on p. 52).

[43] *sklearn.mixture.GaussianMixture*. https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html (cit. on p. 52).

[44] *Mel Frequency Cepstral Coefficient (MFCC) tutorial*. http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/ (cit. on p. 53).

[45] Xuedong Huang, Alex Acero, Hsiao-Wuen Hon, and Raj Reddy. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. 1st. USA: Prentice Hall PTR, 2001. ISBN: 0130226165 (cit. on p. 54).

[46] P Rajeswari Raghavendra M. *Determination Of Disfluencies Associated In Stuttered Speech Using MFCC Feature Extraction*. USA, 2016. URL: http://www.ijedr.org/papers/IJEDR1602364.pdf (cit. on p. 55).

[47] *gaussian-mixture-models-explained*. https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95 (cit. on p. 56).

[48] D.A. Reynolds and R.C. Rose. «Robust text-independent speaker identification using Gaussian mixture speaker models». In: *IEEE Transactions on Speech and Audio Processing* 3.1 (1995), pp. 72–83. DOI: 10.1109/89.365379 (cit. on p. 57).

[49] *The Rise Of Voice Cloning And DeepFakes In The Disinformation Wars*. https://www.forbes.com/sites/jenniferhicks/2021/09/21/the-rise-of-voice-cloning-and-deep-fakes-in-the-disinformation-wars/?sh=787cd2f438e1 (cit. on p. 60).

[50] CorentinJ. *Real TIme Voice Cloning*. https://github.com/CorentinJ/Real-Time-Voice-Cloning. 2020 (cit. on p. 60).

[51] Ye Jia et al. *Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis*. 2019. arXiv: 1806.04558 [cs.CL] (cit. on p. 60).

[52] Li Wan, Quan Wang, Alan Papir, and Ignacio Lopez Moreno. *Generalized End-to-End Loss for Speaker Verification*. 2020. arXiv: 1710.10467 [eess.AS] (cit. on pp. 61, 62).

[53] Jonathan Shen et al. *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions*. 2018. arXiv: 1712.05884 [cs.CL] (cit. on p. 63).

[54] Nal Kalchbrenner et al. *Efficient Neural Audio Synthesis*. 2018. arXiv: 1802.08435 [cs.SD] (cit. on p. 65).

[55] *Audio Deepfakes: Can Anyone Tell If They're Fake?* https://www.howtogeek.com/682865/audio-deepfakes-can-anyone-tell-if-they-are-fake/ (cit. on p. 67).

[56] *Resemblyzer*. https://github.com/resemble-ai/Resemblyzer (cit. on p. 67).

[57] Chaouki Kasmi and Jose Lopes Esteves. «IEMI Threats for Information Security: Remote Command Injection on Modern Smartphones». In: *IEEE Transactions on Electromagnetic Compatibility* 57.6 (2015), pp. 1752–1755. DOI: 10.1109/TEMC.2015.2463089 (cit. on p. 72).

[58] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. «Hidden Voice Commands». In: *USENIX Security Symposium (USENIX)*. Aug. 2016 (cit. on p. 72).

[59] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. «DolphinAttack». In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Oct. 2017). DOI: 10.1145/3133956.3134052. URL: http://dx.doi.org/10.1145/3133956.3134052 (cit. on p. 72).

[60] Xuejing Yuan et al. *CommanderSong: A Systematic Approach for Practical Adversarial Voice Recognition*. 2018. arXiv: 1801.08535 [cs.CR] (cit. on p. 72).

[61] Yuxuan Chen, Xuejing Yuan, Jiangshan Zhang, Yue Zhao, Shengzhi Zhang, Kai Chen, and XiaoFeng Wang. «Devil's Whisper: A General Approach for Physical Adversarial Attacks against Commercial Black-box Speech Recognition Devices». In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2667–2684. ISBN: 978-1-939133-17-5. URL: `https://www.usenix.org/conference/usenixsecurity20/presentation/chen-yuxuan` (cit. on p. 72).