UNIVERSITÀ
DI PAVIA

UNIVERSITY OF PAVIA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL, COMPUTER AND BIOMEDICAL ENGINEERING

MASTER'S DEGREE IN COMPUTER ENGINEERING

MASTER THESIS

Automated Feature Mapping for Audio DeepFake Detection

Mappatura Automatica delle Features per il Rilevamento di DeepFake Audio

Candidate: Andrea Alberti

Supervisor: Prof. Claudio Cusano

A.Y. 2023/2024

"A chi mi ha mostrato che con determinazione e cuore,
ogni traguardo è possibile."

Abstract

The proliferation of audio deepfake technology poses significant challenges to cybersecurity, privacy, and trust. Generated using advanced artificial intelligence techniques, audio deepfakes can imitate human speech with remarkable accuracy, leading to potential abuses such as identity theft, fraud, and disinformation. Detecting such synthetic audio has become critically important as these technologies continue to evolve.

This thesis explores audio deepfake detection by proposing novel Machine Learning (ML) and Deep Learning (DL) approaches, with a primary focus on improving feature extraction and automating the feature mapping process. A comprehensive analysis of various audio features, such as Mel-spectrograms, MFCC (Mel-Frequency Cepstral Coefficients), and CQT (Constant-Q Transform), among others, is conducted, enhancing their effectiveness through the use of CNNs. Additionally, a fully end-to-end approach (*DeepSpectraNetE2E*) is proposed, allowing the model to autonomously learn time-frequency representations directly from raw audio, thus automating the entire feature extraction process.

Alongside *DeepSpectraNetE2E*, three other models (*DeepSpectraNet*, *DeepSpectraNetFlex*, and *DeepSpectraNetLite*) are introduced, all of which surpass existing models in the literature. Experimental results demonstrate that these deep learning models significantly outperform traditional machine learning models in terms of accuracy and generalization. The results also highlight the effectiveness of combining multiple audio features and using CNN-based feature mapping strategies to enhance frequency-related information within the signal.

This work contributes to audio deepfake detection by proposing four new models that improve accuracy and generalize better in the detection of sophisticated synthetic audio by providing novel mapping strategies along with automated feature extraction.

Sommario

La proliferazione della tecnologia dei deepfake audio pone sfide significative alla sicurezza informatica, alla privacy e alla fiducia. Generati utilizzando tecniche avanzate di intelligenza artificiale, i deepfake audio possono imitare il parlato umano con notevole accuratezza, portando a potenziali abusi come furto d'identità, frodi e disinformazione. La rilevazione di tali audio sintetici è diventata di importanza critica poiché queste tecnologie continuano a evolversi.

Questa tesi esplora la rilevazione dei deepfake audio proponendo nuovi approcci di Machine Learning (ML) e Deep Learning (DL), con un focus principale sul miglioramento dell'estrazione delle features e sull'automazione del mappaggio delle stesse viene condotta un'analisi completa su diverse features audio, come spettrogrammi di Mel, MFCC (Mel-Frequency Cepstral Coefficients) e CQT (Constant-Q Transform), tra le altre, aumentandone l'efficacia attraverso l'uso di CNN. Inoltre, viene proposto un approccio completamente end-to-end (*DeepSpectraNetE2E*), che permette al modello di apprendere autonomamente le rappresentazioni tempo-frequenza direttamente dall'audio grezzo, automatizzando così l'intero processo di estrazione delle features.

Oltre a *DeepSpectraNetE2E*, vengono introdotti altri tre modelli (*DeepSpectraNet*, *DeepSpectraNetFlex*, e *DeepSpectraNetLite*), che superano i modelli esistenti in letteratura. I risultati sperimentali dimostrano come questi modelli di deep learning superino significativamente i modelli tradizionali di machine learning in termini di accuratezza e generalizzazione. I risultati, inoltre, evidenziano l'efficacia della combinazione di più features audio e dell'uso di strategie di mappaggio delle features basate su reti neurali convoluzionali (CNN) che aumentino le informazioni relative alle frequenze del segnale.

In conclusione, questo lavoro contribuisce al campo della rilevazione dei deepfake audio proponendo quattro nuovi modelli che, grazie a strategie di mappaggio innovative e all'automazione dell'estrazione delle features, migliorano l'accuratezza e la generalizzazione nella rilevazione di sofisticati audio sintetici.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The sweeping advances in the frontiers of AI and machine learning have resulted in the evolution of sophisticated technologies able to create highly realistic synthetic content. Deepfakes are one such important technological development that enables the creation of convincingly faked media in the forms of images, videos, and audio. While deepfake technology does have much potential for applications in entertainment, education, and healthcare, it is also emerging as a technology that presents significant challenges, more so in areas related to cybersecurity, fraud, and disinformation.

This thesis focuses specifically on audio deepfakes, with synthesized voices facilitated by high-end AI techniques. The misapplication of deepfakes has already caused a few fraud, identity theft, and disinformation cases, raising critical concerns across certain sectors. Audio deepfake detection thus becomes one of the important domains of study to mitigate the potential negative impacts of this technology.

The objective of this research is to explore and enhance the detection of audio deepfakes through the use of both traditional machine learning and deep learning approaches. Focusing on feature enhancement and automated mapping, this work propose novel models that aim to improve detection accuracy and generalization.

1.1 Context

In the digital era, the ability to manipulate media has evolved from simple touch-ups to sophisticated alterations that challenge our perception of reality. Among these technologies, DeepFakes represent a significant leap forward in digital content manipulation. This section delves into the origins

and evolution of DeepFakes, setting the stage for a comprehensive discussion on their implications and the technological arms race they have sparked.

1.1.1 What are DeepFakes?

Deepfakes represent a newer development in the realm of manipulation of digital media using sophisticated technologies of artificial intelligence to either fabricate or tamper with video, audio, and images in the way they appear to create a false impression of certain real situations. Deep learning algorithms such as Generative Adversarial Networks have been utilized to synthetically create human images and sounds indistinguishable from natural recordings.

The name 'DeepFake' is just an abbreviation for what this is based on, namely, 'deep learning' and 'fake', since it is based upon deep neural networks to create or manipulate fake media content. These changes are such that they can fool detection with traditional means, either human or automated, and hence are powerful in misinformation, entertainment, or evil.

While this is promising considering the line of changing reality convincingly, potential consequences for privacy, security, and integrity of information have turned this technology into one of the hot debates, with urgent calls for effective detection and regulation strategies. This research delves deeper into the aspects of technical understanding and, more importantly, ethical and social implications of the emergence of DeepFakes.

Technical Foundations

Nowadays, the creation of DeepFakes mainly involves training a model using two neural networks that work against each other in what's known as a Generative Adversarial Network (GAN). One network, the generator, creates images or sequences, while the other, the discriminator, evaluates them. Over time, the generator learns to produce more accurate fakes, trying to outsmart the discriminator until the discriminator can no longer distinguish real from fake.

This technology stems from research that has grown rapidly since the introduction of GANs in 2014 by Ian Goodfellow and his colleagues [4]. Early DeepFakes were rudimentary and easily detectable, but recent advancements have led to hyper-realistic results. These improvements are

powered by the increasing availability of computational resources and large datasets of facial images and audio recordings, which allow the algorithms to learn and mimic finer human details.

1.1.2 History of DeepFakes

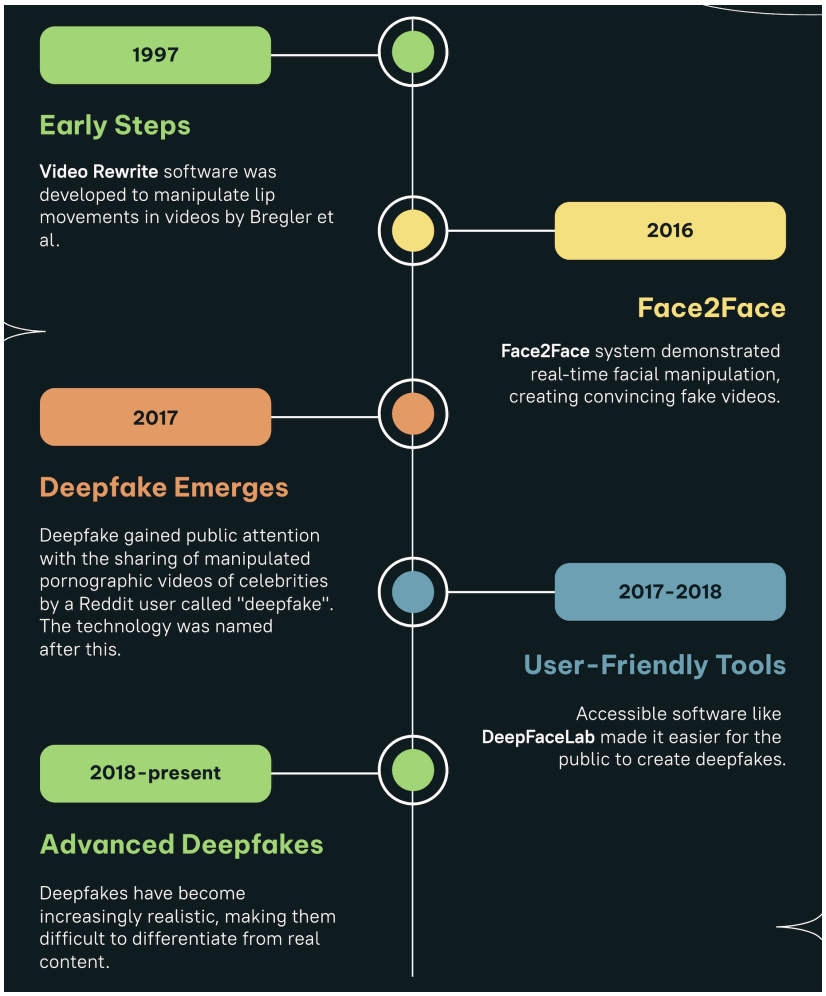


Figure 1.1: Timeline of DeepFake Technology

The concept of altering video and audio content dates back to the late 20th century. In 1997, Bregler et al. developed 'Video Rewrite' [5], a program that could modify the lip movements in video footage to match a different audio track, laying the groundwork for more advanced techniques in facial reanimation and synchronization. The wider academic community began to take notice of DeepFakes in 2016 when Thies et al. presented 'Face2Face' [6], a real-time facial reenactment system that could manipulate a video of one person to match the facial expressions of another. This groundbreaking work demonstrated the potential of AI to create convincing

fake videos. One year later, deepfake was brought to the public's attention when a Reddit user named 'deepfake' [7] shared explicit videos with the faces of celebrities swapped into pornographic content. This marked a turning point, bringing the potential and risks of this technology to the forefront of public awareness. Soon after, tools that enabled literally anybody to generate DeepFakes in a painless manner were created. A good example is the open-source 'DeepFaceLab' [8] where face swapping in a target video is enabled an open-source software that allows users to swap faces in videos with minimal effort. During the last few years, the quality of DeepFakes has reached an unprecedented level of realism, making them nearly indistinguishable from genuine media to the human eye and ear. Advances in deep learning and generative adversarial networks (GANs) have enabled the creation of synthetic content that mimics intricate facial expressions, voice intonations, and other subtle nuances with astonishing accuracy. As a result, even trained professionals often struggle to identify DeepFakes without assistance. This has led to a pressing need for sophisticated detection technologies which leverage AI and machine learning to analyze inconsistencies and artifacts that are imperceptible to humans. Tools like these are crucial for maintaining the integrity of information and preventing the malicious use of DeepFake technology. This work aims to explore the detection of DeepFakes in audio content, focusing on synthetic voice generated by state-of-the-art text-to-speech models.

1.2 Research Rationale

The decision to focus this thesis on audio deepfake detection comes from the critical need to address the challenges posed by the misuse of synthetic audio technologies. As discussed in the following section about the applications of audio deepfakes, the technology offers both positive potential and significant risks. The rapid advancement and accessibility of deepfake technology amplify its capacity for harm, particularly in the realms of misinformation, fraud, and cybersecurity threats.

The proliferation of deepfake tools has simplified the creation and distribution of fake audio content by malicious actors, posing threats to individuals, organizations, and societal stability. The impact of these threats is not merely hypothetical but has been demonstrated through various incidents that have had real-world consequences. Thus, developing effective detection

methods is paramount to maintaining trust in audio communications and safeguarding digital interactions.

This thesis aims to contribute to the growing field of digital forensics by focusing on audio deepfakes. By enhancing our understanding of audio deepfake generation and detection, we can develop more robust systems to identify and mitigate these threats, ensuring that the benefits of synthetic audio technologies can be realized without the accompanying risks overshadowing them.

1.2.1 DeepFakes Applications

In this section, we explore the applications of audio deepfakes, highlighting their potential benefits and risks across various sectors.

Positive Applications

- **Healthcare**: Audio deepfakes are used to create synthetic voices for patients with *speech impairments* [9]. This technology can help head and neck cancer survivors communicate more effectively, significantly improving their quality of life. Synthetic voices can also *anonymize patient data*, facilitating safer and broader sharing of medical information for research purposes while maintaining patient confidentiality [10].

- **Entertainment and Media**: The entertainment industry utilizes audio deepfakes to recreate voices for dubbing in different languages, enabling a wider reach for movies and TV shows. It also helps in preserving the voices of deceased actors for continuity in long-running series or films. Moreover, audio deepfakes can enhance the experience of audiobooks by using the author's synthetic voice for narration [11].

- **Education**: For educational purposes, audio deepfakes can produce personalized learning experiences, such as creating custom voice assistants that can interact with students in their native languages or accents. This technology can also help in producing accurate and engaging audio content, broadening access to knowledge and learning resources.

Negative Applications

- **Fraud and Identity Theft**: One of the most concerning uses of audio deepfakes is in fraud and identity theft. Fraudsters can use cloned voices

to impersonate individuals in phone scams. For instance, a notable case involved criminals using AI to mimic a CEO's voice, successfully tricking a subordinate into transferring a significant amount of money [12].

- **Disinformation and Fake News**: Audio deepfakes can be employed to spread disinformation by creating fake speeches or statements from public figures, thereby influencing public opinion and creating political unrest. A relevant example was a deepfake created in 2022, featuring Ukrainian President Volodymyr Zelensky instructing Ukrainian forces to surrender to Russian troops, which was shared on social media platforms [13]. This shows how these fabricated audio clips can be used in spear-phishing attacks or to propagate false information on social media platforms, undermining trust in legitimate news sources [7].

- **Cybersecurity Threats**: The use of audio deepfakes poses significant risks to cybersecurity. Cybercriminals can exploit this technology for espionage, blackmail, and other malicious activities. Deepfake audio can manipulate voice-operated systems, potentially gaining unauthorized access to secure systems or sensitive information [12].

In summary, while audio deepfakes offer innovative applications across various sectors, they also pose substantial risks. Positive implementations in healthcare, entertainment, and education showcase the potential for beneficial impacts. However, the intrinsic risks associated with their misuse, such as loss of trust and significant economic damages, underscore the urgent need for vigilance and proactive measures. This thesis explores the complexities of audio deepfakes, aiming to contribute to a deeper understanding and development of strategies to harness their potential while mitigating the risks.

1.3 Research Questions

The increasing sophistication of audio deepfakes presents both opportunities and challenges across various domains, from entertainment and healthcare to cybersecurity and disinformation. Given the critical importance of detecting audio deepfakes accurately, this thesis addresses several key research questions aimed at improving detection methodologies and exploring the underlying challenges.

- **RQ1: What are the preliminary factors that most influence a correct deepfake detection?**
  The aim of this question is to identify the key factors that contribute to the accurate detection of audio deepfakes. Examples of such factors are the length of the audio clip and the window and hop sizes for spectral features extraction.

- **RQ2: What are the most important audio features for accurately identifying deepfakes?**
  This question is focused to determine which audio features, such as MFCC, CQT, Mel-spectrogram, and others, contribute the most to differentiating between authentic and synthetic audio. Through an empirical analysis of feature importance in both machine learning and deep learning models, the research aims to identify the strengths and limitations of various audio features in detecting deepfakes.

- **RQ3: How effective are traditional machine learning models in detecting audio deepfakes?**
  With this research question the goal is to explore the capabilities of traditional machine learning techniques in the detection of synthetic audio. Traditional models, such as Random Forest and Support Vector Machines (SVM), will be evaluated based on their ability to distinguish between real and fake audio samples. The research will also investigate how these models can be optimized to improve detection accuracy and robustness.

- **RQ4: Can deep learning models represent a leap forward in audio deepfake detection?**
  With the rise of end-to-end deep learning models, this question investigates whether deep learning approaches provide superior performance compared to traditional machine learning methods. By leveraging neural networks that extract high-level features automatically, this research examines whether these models can generalize better to unseen deepfake generation techniques.

- **RQ5: How does feature combination affect the performance of models in detecting audio deepfakes?**
  Different audio features capture distinct aspects of the audio signal, such as frequency, time, and amplitude. This question explores the impact of combining multiple audio features on the performance of deepfake detec-

tion models. By combining diverse features like MFCC, CQT, and Mel-spectrograms, the research aims to assess the models' ability changes in detecting synthetic audio.

- **RQ6: How can the features be improved to enhance the detection of audio deepfakes?**
  This question delves into feature engineering techniques that can enhance the quality and informativeness of audio features for deepfake detection. By implementing Deep Learning based automated feature extraction and mapping strategies, the goal is to improve the performance of the models in distinguishing between real and synthetic audio.

- **RQ7: How can model explainability techniques be applied to understand the decision-making process of deep learning models?**
  Given the complexity of deep learning models, understanding how these models make decisions is crucial. This question explores the use of techniques like Grad-CAM and Mapper Analysis to interpret the model's predictions, identifying key areas in the audio data that influence the detection of deepfakes. The insights gained from these explainability methods will help in improving model transparency and trustworthiness.

Through these research questions, the thesis aims to push the boundaries of current detection methods and provide a comprehensive understanding of the strengths and weaknesses of various approaches.

1.4 Thesis Outline

This section provides a roadmap of the thesis, outlining the structure and sequence of chapters and the key topics covered in each.

- **Chapter 1: Introduction**: Establishes the foundational context, motivations, and research questions related to audio deepfakes. It covers the essence of what deepfakes are, their historical context, and the rationale behind focusing on audio deepfake detection.

- **Chapter 2: Background**: Explores deepfake typologies, detection techniques, and the theoretical frameworks utilized in deepfake analysis. This

section details the technologies and methods underpinning deepfake detection, including traditional and machine learning-based approaches, alongside discussions on audio features and models.

- **Chapter 3: Methods and Experiments**: Describes the methodologies used in the research, from dataset selection and feature engineering to the detailed implementation of machine learning and deep learning models.

- **Chapter 4: Results and Explainability**: It starts by outlining the experimental setup and evaluation metrics used to assess the efficacy of proposed solutions. The main focus is on the experimental outcomes from the experiments about feature engineering and proposed models performance against state-of-the-art models. It also discusses the explainability of the models using techniques like Grad-CAM, Stage Analysis and Worst Errors Investigation.

- **Chapter 5: Conclusions**: Summarizes the research findings by answering to to the research questions. It also discusses the implications, and proposes future directions for the field of audio deepfake detection.

- **Appendices**: Include supplementary results and extended analyses that support and enhance the understanding of the research findings.

Chapter 2

Background

This chapter lays the foundational knowledge required to understand the various aspects of deepfake technology, particularly focusing on audio deepfakes. It explores the different types of deepfakes, the methods employed in their detection, and the critical audio features utilized in distinguishing genuine from fabricated content. Additionally, it delves into the machine learning models that are at the forefront of detecting and analyzing deepfakes. Each section is designed to progressively build an understanding of the complex landscape of deepfake technology, addressing both its innovative uses and the challenges it presents in digital media integrity.

2.1 Audio DeepFakes Typologies

Audio deepfake technology has evolved into various forms, each employing distinct methods to create or manipulate digital content. This section categorizes DeepFakes into four primary typologies: Text-to-Speech (TTS), Voice Conversion (VC), Partially Fake, and Replay Attack. Each typology represents a unique approach to generate deceptive media with specific techniques and implications. Therefore, each typology requires a tailored detection strategy to identify and mitigate the risks associated with deepfake technology. In this work, we focus on the synthesis-based deepfakes.

2.1.1 Text-to-Speech (TTS)

Text-To-Speech (TTS) is a technology that converts written text into spoken words. TTS systems have seen significant improvements in recent years, especially with the advent of deep learning models, which have enabled

TTS systems to produce highly natural, expressive and emotionally rich speech, narrowing the gap between synthetic and human speech.

Technological Evolution of TTS Systems

Text-to-speech systems have evolved significantly over the past few decades. Initially, TTS systems relied on concatenative synthesis techniques [14], where pre-recorded speech units (phonemes) were pieced together to form full sentences. These early systems were often rigid and lacked natural variability in speech, making them sound robotic and unnatural.

With the rise of statistical models like Hidden Markov Models (HMMs), TTS systems moved towards parametric speech synthesis, which allowed for more flexibility by generating speech through the manipulation of parameters like pitch, duration, and intensity. While this improved over concatenative approaches, the synthesized speech still struggled to reach the naturalness of human speech due to the simplified representation of speech parameters [15].

The most transformative shift occurred with the advent of deep learning-based approaches. Models like WaveNet [16] (developed by Google DeepMind) marked a breakthrough in speech synthesis by utilizing neural networks to directly model raw audio waveforms. WaveNet and similar architectures, leverage large datasets and computational resources to model intricate details of human speech, overcoming limitations of previous methods.

Currently, modern TTS systems are largely built on end-to-end deep learning architectures, with attention mechanisms enabling the alignment between text and speech and autoregressive models or parallel synthesis approaches improving the efficiency and quality of speech generation. These systems can learn from massive datasets of human speech, capturing nuanced patterns in pitch, intonation, and speech rhythm.

TTS System Architecture

Text-to-speech systems typically consist of three cooperating core modules, each responsible for a specific aspect of the text-to-speech conversion process [17].

- **Text Processing Module**: This is the first stage in a TTS system, where the input text is processed and converted into a form that can be

understood by the speech synthesis system. In traditional TTS systems, this module would often convert text into *phonemes* or *linguistic features*, which represent the smallest units of sound in speech. The text processing stage also includes linguistic analysis, such as *part-of-speech tagging*, *intonation modeling*, and *punctuation interpretation* to generate a rich linguistic representation of the input text. In modern TTS systems, deep learning models handle these linguistic features more implicitly through learned embeddings.

- **Acoustic Model**: After the text has been processed, the *acoustic model* compute the characteristics of the speech, such as *pitch*, *duration*, and *intonation*. In neural TTS systems, the acoustic model is often a deep neural network, like in *Tacotron* or *FastSpeech*, which takes the linguistic features from the text processing module and predicts the corresponding acoustic features. These latter represent how the speech should sound, leveraging tools like *mel-spectrograms* (representations of sound intensity over time) and other spectral details of the speech signal. This step aims to ensure the speech sounds natural and expressive, with proper pauses, stress, and emotion.

- **Vocoder**: The vocoder is the final stage of the TTS system and is responsible for generating the final speech waveform from the predicted acoustic and linguistic features. While older vocoders were capable of processing only acoustic features (e.g., mel-spectrogram), modern systems, such as *WaveNet* and *WaveGlow*, can take as input both linguistic (phonemes) and acoustic features, leveraging the flexibility offered by neural networks. Vocoders are essential for achieving human-like naturalness in speech synthesis, capturing nuances in sound such as background noise, breath, and fine-grained temporal variations.

Over time, Text-to-Speech (TTS) systems have evolved from traditional modular architectures to more integrated, end-to-end models. [17] In **traditional systems**, each component—text analysis, acoustic model, and vocoder—is trained separately. While this modular approach provides flexibility and control over individual parts, it also increases complexity in the training process.

Partially **end-to-end models**, such as *Tacotron 2* and *Deep Voice 3*, simplify this process by merging the text analysis and acoustic modeling stages into a single module, generating mel-spectrograms directly from text

or phonemes. The final waveform is then produced by a neural vocoder like WaveNet, maintaining some modularity but reducing complexity compared to traditional methods.

Fully end-to-end models like *FastSpeech 2s* and *ClariNet* represent the latest advancements in TTS. These systems integrate all components into a single model that converts input text directly into speech waveforms.

2.1.2 Voice Conversion (VC)

Voice Conversion (VC) refers to the process of transforming the voice of a source speaker into a target speaker's voice while preserving the original linguistic content. VC systems can be used for several purposes:

- **Voice Correction**: This finds application for rehabilitation purposes, such as helping individuals with speech disorders or injuries to communicate more effectively [18].
- **Voice Impersonation**: This focuses on altering the voice to mimic another person, which can be used for entertainment, dubbing, or even malicious purposes like fraud [18].
- **Voice Emotion Change**: This technique modifies the emotional tone or mood of the speaker's voice without altering the content or identity. It can be used to change how a message is perceived, potentially distorting the intent behind the speech. Changing the emotional content of a message can indeed alter its semantic meaning [19].
- **Scene Fake**: While not properly focused on voice, scene fake alters the acoustic environment or background of the audio, making it seem like the recording was made in a different location. While the speech itself remains unchanged, the context of the audio is manipulated to create a false impression of the setting [19].

2.1.3 Partially Fake Audios

Partially fake audios refer to manipulated recordings where only certain portions of the audio are altered or artificially generated, while the rest remains authentic. This manipulation can involve adding or modifying words [20], phrases, or specific parts of a conversation to alter its meaning without generating an entirely fake recording.

Detecting partially fake audio is particularly challenging because most of the content is genuine, and only small portions are synthetic. This makes it harder for detection algorithms to identify manipulations without advanced analysis of speech patterns and subtle inconsistencies.

2.1.4 Replay Attack

Replay attacks, though not AI-generated, represent a prevalent form of audio spoofing, where pre-recorded legitimate audio is used to deceive systems. These attacks exploit the inability of speech recognition or speaker verification systems to distinguish between live audio and pre-recorded voices.

Attack Procedure

Replay attacks typically involve two methods [18]:

- **Simple Playback**: The attacker records legitimate audio and then plays it back to a system, convincing it that the voice is live.
- **Cut and Paste**: In this case, attackers take short segments of recorded audio and stitch them together to create longer fake message or conversation and reproducing specific content or responses.

The various types of audio deepfakes discussed in this section highlight the diverse ways in which speech and sound can be manipulated using modern technology. From the complex transformation of voice characteristics through TTS and voice conversion systems to the more subtle alterations in emotion or environmental context, deepfake audio has a wide range of applications and implications.

2.2 DeepFake Detection Methods

The detection of audio deepfakes has gained significant attention, with various methods being proposed to tackle this challenge. These techniques can be broadly divided into three categories: signal processing-based methods, traditional machine learning approaches, and deep learning techniques. While signal processing methods focus on analyzing specific audio signal characteristics and often don't rely on AI techniques, the main focus of this thesis lies in machine learning and deep learning methods. Therefore, the

non-AI approaches will be briefly mentioned, and the core discussion will revolve around the more advanced AI-driven detection methods.

2.2.1 Signal Processing Methods

Signal processing-based approaches are among the earlier techniques employed for detecting audio deepfakes. These methods rely on analyzing distinct characteristics of the audio signal, such as spectral patterns, pitch variations, or phase inconsistencies. By examining these features, signal processing methods aim to detect anomalies that may reveal synthesized or manipulated audio content [21].

2.2.2 Traditional Machine Learning Methods

Traditional machine learning (ML) methods focus on extracting handcrafted features from the audio data and feeding them into classical classifiers to detect anomalies associated with deepfakes. These approaches heavily rely on feature engineering, where specific signal characteristics, such as *MFCCs (Mel-frequency cepstral coefficients)*, *zero-crossing rate*, and *Chroma features*, are used to represent the audio signal.

Given the extracted features, classical machine learning algorithms such as *Support Vector Machines (SVMs)*, *Random Forests (RFs)*, and *k-Nearest Neighbors (k-NN)* are applied to classify the audio samples as either genuine or fake. These models are trained on labeled data, learning to distinguish between real and synthetic audio based on predefined characteristics.

A key strength of traditional ML methods lies in their interpretability and ability to handle smaller datasets, which makes them well-suited for environments with limited data availability. However, these methods are highly dependent on the quality and relevance of the chosen features, making feature selection a critical step in the process. Furthermore, in the most complex deepfake scenarios, where advanced synthesis models are used, traditional ML methods may struggle to capture the subtle manipulations introduced, thereby requiring more complex models and features.

Literature

Borrelli and colleagues [22] designed a model using Support Vector Machines (SVM) alongside Random Forest (RF) to identify synthetic voices, utilizing a novel audio feature referred to as Short-Term Long-Term (STLT). Their

models were trained on data from the 2019 Automatic Speaker Verification (ASV) Spoof Challenge [23]. The results revealed that the SVM model outperformed RF, achieving 71% better performance.

Kochare et al. [24] tested different ML models, including SVM, Random Forest, and k-Nearest Neighbors (k-NN), to detect synthetic audio on the Fake or Real (FoR) dataset. The results showed that the SVM model outperformed the other models with a validation accuracy of 85% and a test accuracy of 67% using RMSE, MFCC, Chroma, and other spectral features.

Liu et al. [25] compared the robustness of SVM with a Convolutional Neural Network (CNN) to detect fake stereo audio from real ones. The comparison revealed that CNN is more robust than SVM when tested across datasets, demonstrating the higher potential of CNN in detecting deepfake audio.

2.2.3 Deep Learning Methods

Deep learning (DL) methods have become the dominant approach for detecting audio deepfakes, offering greater flexibility and accuracy compared to traditional machine learning methods. These approaches can be categorized into partially automated and fully end-to-end methods.

Partially Automated Approaches

Partially automated approaches also referred to as hybrid methods, combine handcrafted feature extraction with deep learning models. These methods partially solve the problems of traditional ML methods by enhancing feature extraction with deep learning models. In these systems, features such as *MFCCs*, *Chroma*, or *spectrograms* are first extracted using traditional signal processing techniques. These extracted features are then fed into deep learning models, typically *Convolutional Neural Networks (CNNs)* or *Recurrent Neural Networks (RNNs)*, to extract more complex patterns and relationships from the data, finally crafting **neural features** that are fed into a classifier for the final decision.

The key benefit of partially automated approaches is that they leverage the strengths of both manual feature extraction and deep learning, offering greater control over the features used while still benefiting from the deep learning model's capacity to learn complex patterns.

However, a drawback of these approaches is that they still rely on the

quality of manually extracted features, which may differ in effectiveness depending on the audio manipulation techniques used.

Literature

E.R. Bartusiak at al. [26] proposed a CNN model to detect deepfake audio using only spectrogram as feature. Despite the high accuracy of 85.99% the authors found that the model was not robust to generalize to new unseen audio.

Whang et al. [27] proposed a new solution to extract features for synthetic audio detection using a deep learning model. The authors extracted layer-wise neuron activation patterns from a DL-based Speech Recognition System. Exploring two different approaches, called TKAN and ACN, the authors found that the TKAN approach outperformed the ACN approach in detecting synthetic audio, achieving a detection rate of 98.1% across synthetic and voice conversion audio datasets.

Fully End-to-End Approaches

In fully end-to-end deep learning methods, the entire process, from raw audio input to classification, is handled by a deep neural network. These models learn to extract features autonomously from the audio data, eliminating eliminating the need for hand crafted feature engineering. This makes end-to-end models more flexible and capable of capturing fine details and patterns within the audio that traditional methods might overlook. Furthermore, end-to-end models can adapt to different types of audio deepfakes without requiring extensive modifications to the feature extraction process.

One limitation, however, is that fully end-to-end models require large amounts of labeled data for training, are computationally intensive and may be more challenging to interpret as they act as black-box models.

2.3 Audio Features

This section delves into the key audio features used in deepfake detection, providing their characteristics and interpretability.

Mel-frequency Cepstral Coefficients (MFCC)

Mel-frequency Cepstral Coefficients (MFCCs) are derived from the power spectrum of an audio signal and represent the short-term power spectrum of sound, specifically adapted to mimic the human auditory system's perception of frequency. The MFCCs are extracted by applying a set of mel-scaled filters to the signal, followed by a discrete cosine transform (DCT) to capture the important frequency components.

Interpretation MFCCs offer a semantic decomposition of audio in terms of *tonal and timbral properties*:

- *Low MFCCs (MFCC 1-2)* provide broad spectral energy and information about the *general shape* of the sound. MFCC 1 is often related to the loudness of the signal, while MFCC 2 can indicate the balance between low and high frequencies.
- *Mid MFCCs (MFCC 3-6)* focus on the *formant structure*, providing more information about *vowel sounds and timbre* or the timbral content of audio.
- *High MFCCs (MFCC 7 and above)* capture *high-frequency formants, harmonics, and articulation details*, representing the finer spectral components of the audio signal.

One point to consider is that the exact boundaries between "low," "mid," and "high" MFCCs can vary depending on the specific application and the total number of MFCCs being used.

Use in Audio Deepfake Detection MFCCs are highly effective for detecting anomalies in both broad spectral content and subtle speech details. *Low MFCCs* can help identify general discrepancies in pitch and loudness between real and fake audio. For example, if a deepfake voice consistently lacks the natural variations in overall energy or has an unusual balance of low and high frequencies that's atypical for human speech, these anomalies would likely be reflected in the *low MFCC range (MFCC 1-2)*.

Mid MFCCs are useful for detecting inconsistencies in the formant structure and timbral qualities of speech. For instance, if a synthetic voice produces vowel sounds with formant frequencies or transitions that don't match natural human speech patterns or if it exhibits unnatural timbral qualities,

these discrepancies would likely be evident in the *mid MFCC range (MFCC 3-6)*.

High MFCCs capture finer artifacts that might arise from synthetic voice generation techniques. For example, if a deepfake voice lacks natural transitions in consonant production or exhibits unnatural harmonics, these discrepancies would likely appear in the *high MFCC range (MFCC 7 and above)*.

Constant-Q Transform (CQT)

The Constant-Q Transform (CQT) is a time-frequency representation that provides a logarithmic frequency resolution. Thanks to its dynamic window length, the CQT provides higher frequency resolution in low-frequency regions and higher time resolution in high-frequency regions.

Interpretation CQT provides a clear breakdown of the audio in terms of *slow, broad sounds* versus *sharp, fast events*:

- *Low CQT values (lower frequency bands)* represent *lower frequencies* and capture *slower audio events*, like long, the low end of a speaker's voice.
- *High CQT values (upper frequency bands)* capture *higher frequencies* and are associated with *sharper, faster events*, such as consonant bursts in speech.

The *spacing between wave-like structures* in a CQT plot corresponds to rhythm and timing, offering insight into the *pacing* of speech or music.

Use in Audio Deepfake Detection CQT is useful for detecting rhythm and frequency-related artifacts in deepfake audio. *Low CQT values* can help identify unnatural modulations in speech or music's low-frequency components, such as a lack of natural variation in pitch. Meanwhile, *high CQT values* capture the more intricate, high-frequency details that may reveal synthesis imperfections, such as unnatural timing of fast speech events.

Mel-Spectrogram

A Mel-Spectrogram represents the frequency content of a signal over time, but mapped to the *mel scale*, which mirrors the human ear's perception of pitch. It is widely used in speech and audio processing.

Interpretation The Mel-Spectrogram offers insight into:

- *Low-frequency bands:* Capture bass-heavy sounds or deep voice tones.
- *High-frequency bands:* Represent sharper sounds like consonants or high-pitched notes.

The *intensity of the spectrogram* at different frequencies and time points indicates the *energy distribution* of the audio signal, with brighter regions corresponding to higher energy levels.

Others

In addition to the Mel-Spectrogram, MFCC, and Chroma STFT, several other audio features play a role in deepfake detection by capturing various aspects of the audio signal.

Chroma STFT Chroma STFT (Short-Time Fourier Transform) captures the energy distribution across 12 different pitch classes (chroma), representing musical properties like harmony or chords. This feature is particularly useful for music processing but can also reveal differences in tonal quality in speech, which may be altered or artificially manipulated in deepfake audio.

Root Mean Square (RMS) RMS measures the energy or loudness of the signal over time. It reflects the amplitude of the waveform, providing insight into how the intensity of the audio changes. Variations in RMS can be used to detect inconsistencies in loudness patterns often introduced by synthetic audio generation techniques.

Spectral Rolloff This feature represents the frequency below which a certain percentage (typically 85-90%) of the total spectral energy is concentrated. It helps distinguish between harmonic and non-harmonic content, which can be useful in identifying whether audio has been artificially generated, as deepfake audio may exhibit unnatural rolloff characteristics.

Zero Crossing Rate (ZCR) ZCR measures the rate at which the signal changes sign (crosses the zero axis). It is particularly useful for distinguishing between voiced and unvoiced segments in speech. High ZCR values typically indicate noise or sharp transients, and this feature can reveal unnatural signal fluctuations in synthetic audio.

Spectral Centroid This feature indicates where the center of mass of the spectrum is located, effectively representing the "brightness" of the sound. In human speech, this correlates with the timbre and articulation. Deepfake audio may produce anomalies in the spectral centroid due to imperfect synthesis of the sound's timbral characteristics.

Spectral Bandwidth The spectral bandwidth quantifies the range of frequencies present in the audio signal. It helps in analyzing the spread of frequencies in the signal and can highlight issues with how synthetic audio handles harmonic structures and the overall sound quality. A mismatch in the expected bandwidth can signal potential deepfake audio artifacts.

It is important to highlight that features such as *MFCC*, *CQT*, *Mel-Spectrogram*, and *Chroma STFT* provide multiple values per audio frame, resulting in a rich representation of frequency content over time. In contrast, other features like *RMS*, *Spectral Rolloff*, *Zero Crossing Rate*, *Spectral Centroid*, and *Spectral Bandwidth* produce a single value per frame. To visualize these features in image-based models, preprocessing steps are often required, as discussed in Section 3.2.2.

2.4 Machine Learning Models

In this section, we explore various models employed in our research, categorized into traditional machine learning models and deep learning models.

2.4.1 Traditional Models

The traditional models used in this project are Logistic Regression (LR), Support Vector Machines (SVM), Random Forest (RF), eXtreme Gradient Boosting (XGBoost) and CatBoost.

Logistic Regression (LR)

Logistic Regression (LR) is a fundamental statistical method used for binary classification tasks in machine learning. It models the probability that a given input $\mathbf{x}$ belongs to a particular category. LR is favored for its simplicity and interpretability, making it a common choice for problems

where understanding the influence of predictor variables is as important as prediction accuracy.

Conceptual Understanding At its core, Logistic Regression estimates the probability $P(Y = 1|\mathbf{x})$ that the dependent variable Y equals 1 (the positive class), given the independent variables $\mathbf{x}$. Unlike linear regression, which models the output as a linear combination of inputs, LR applies the logistic function to ensure the output probabilities lie between 0 and 1.

Mathematical Formulation The logistic function, also known as the sigmoid function, is defined as:

$$P(Y = 1|\mathbf{x}) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

where

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

Here, β_0 is the intercept, β_i are the coefficients for the independent variables x_i, and n is the number of predictors. The coefficients are estimated using maximum likelihood estimation (MLE), aiming to find the parameter values that maximize the likelihood of observing the given data.

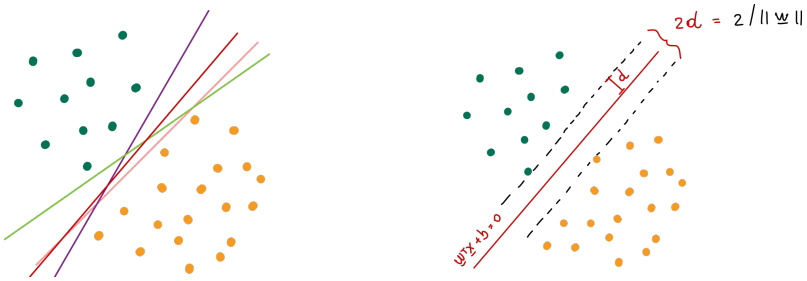
Assumptions Underlying Logistic Regression For LR to produce reliable results, several assumptions need to be satisfied:

- **Linearity in the Log-Odds**: The log-odds of the outcome are a linear combination of the independent variables.
- **Independent and Identically Distributed Data**: The training data is assumed to be independently drawn from the distribution.
- **Lack of Multicollinearity**: Independent variables are not highly correlated with each other.

Support Vector Machines (SVM)

Support Vector Machines (SVM) are powerful supervised learning models used for classification and regression tasks in machine learning. They are particularly effective in high-dimensional spaces and are known for their

ability to handle datasets where the number of features exceeds the number of samples. Indeed, SVMs aim to find the optimal hyperplane that maximally separates data points of different classes. This basically corresponds to regularization.



(a). Candidate Hyperplanes (b). Optimal Hyperplane

Figure 2.1: Support Vector Machines Hyperplane Selection

Conceptual Understanding The fundamental idea behind SVM is to find a hyperplane in an n-dimensional space (n being the number of features) that distinctly classifies the data points. The optimal hyperplane is the one that has the maximum margin, which is the largest distance between data points of both classes. Data points closest to the hyperplane are called *support vectors*, and they are critical in defining the position and orientation of the hyperplane.

Mathematical Formulation Given a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$, the SVM optimization problem can be formulated as:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

subject to:

$$\begin{cases} (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, & \text{if } y_i = +1 \\ (\mathbf{w}^\top \mathbf{x}_i + b) \leq -1, & \text{if } y_i = -1 \end{cases} \quad \forall i = 1, \ldots, m$$

Here, $\mathbf{w}$ is the weight vector perpendicular to the hyperplane, b is the bias term, and $\|\mathbf{w}\|$ denotes the Euclidean norm of $\mathbf{w}$. This formulation seeks to maximize the margin $1/\|\mathbf{w}\|$ while ensuring all data points are correctly classified.

To handle training sets that are not linearly separable instead of rejecting solutions that do not satisfy all the constraints, we penalize them by adding an extra cost to the objective function.

Consider the decision function $z_i = \mathbf{w}^\top \mathbf{x}_i + b$:

- If $y_i = +1$ and $z_i < 1$, the sample is within the margin or misclassified; it is penalized by an amount $1 - z_i$.
- If $y_i = -1$ and $z_i > -1$, the sample is within the margin or misclassified; it is penalized by an amount $1 + z_i$.

This penalization scheme corresponds to the use of the *hinge loss* function, defined as:

$$h(y_i, z_i) = \max\{1 - y_i z_i,\ 0\}$$

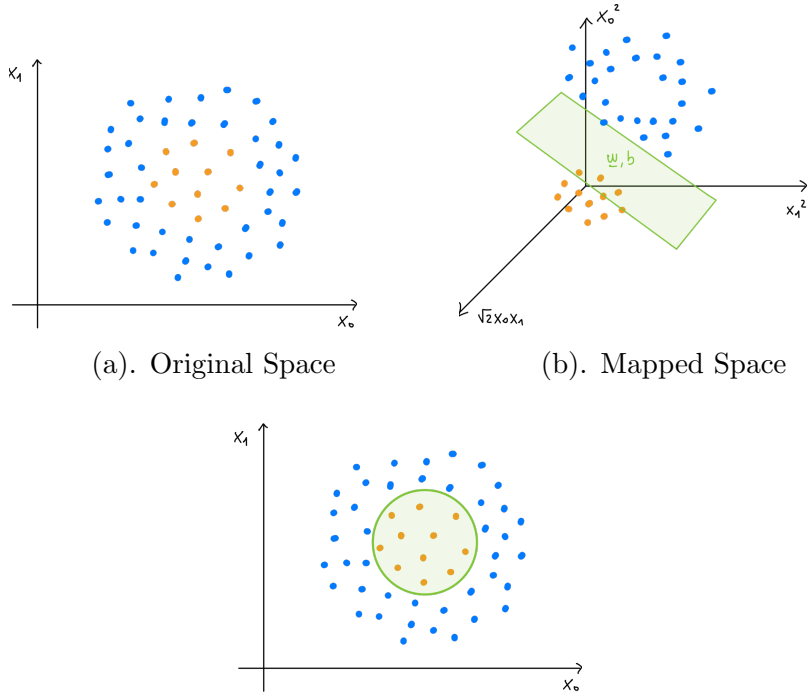
The soft-margin SVM optimization problem becomes:

$$\min_{\mathbf{w},b} \quad \frac{1}{m}\sum_{i=1}^{m} h(y_i, z_i) + \lambda \frac{\|\mathbf{w}\|^2}{2}$$

where λ is the regularization parameter that controls the trade-off between maximizing the margin ($\frac{\|\mathbf{w}\|^2}{2}$) and minimizing the classification error ($\frac{1}{m}\sum_{i=1}^{m} h(y_i, z_i)$).

The Kernel Trick When data is not linearly separable in the original feature space, SVM uses the *kernel trick* to implicitly map input features into a higher-dimensional space where a linear separator might exist. The kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ computes the inner product of the images of the data points in the feature space without explicitly performing the transformation:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$



(a). Original Space (b). Mapped Space

(c). Decision Boundary

Figure 2.2: Illustration of the Kernel Trick in SVM

In other words, the kernel function, working in the original feature space, returns a scalar whose value is the same as the inner product of the projection of the data points in the higher-dimensional space. Common kernel functions include:

- **Linear Kernel**: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$
- **Polynomial Kernel**: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + r)^d$
- **Radial Basis Function (RBF) Kernel**: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2)$
- **Sigmoid Kernel**: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{x}_i^\top \mathbf{x}_j + r)$

The kernel trick allows SVMs to build non-linear classifiers by leveraging these kernel functions to capture complex relationships in the data.

Assumptions Underlying Support Vector Machines For SVMs to perform effectively, several assumptions are considered:

- **Linear Separability**: Assumes that the classes are linearly separable in the original feature space or can be mapped to a linearly separable space using the kernel trick.

- **Independent and Identically Distributed Data**: The training data is assumed to be independently drawn from the distribution.
- **Appropriate Kernel Selection**: Assumes that a suitable kernel function can capture the underlying data patterns.
- **Support Vector Representativeness**: The support vectors are representative of the entire dataset and are critical for defining the decision boundary.

Random Forest

Random Forest is an ensemble learning method used for classification and regression tasks. It operates by constructing a multitude of decision trees during training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random Forests are known for their ability to handle large datasets with higher dimensionality and for reducing overfitting by averaging multiple decision trees.

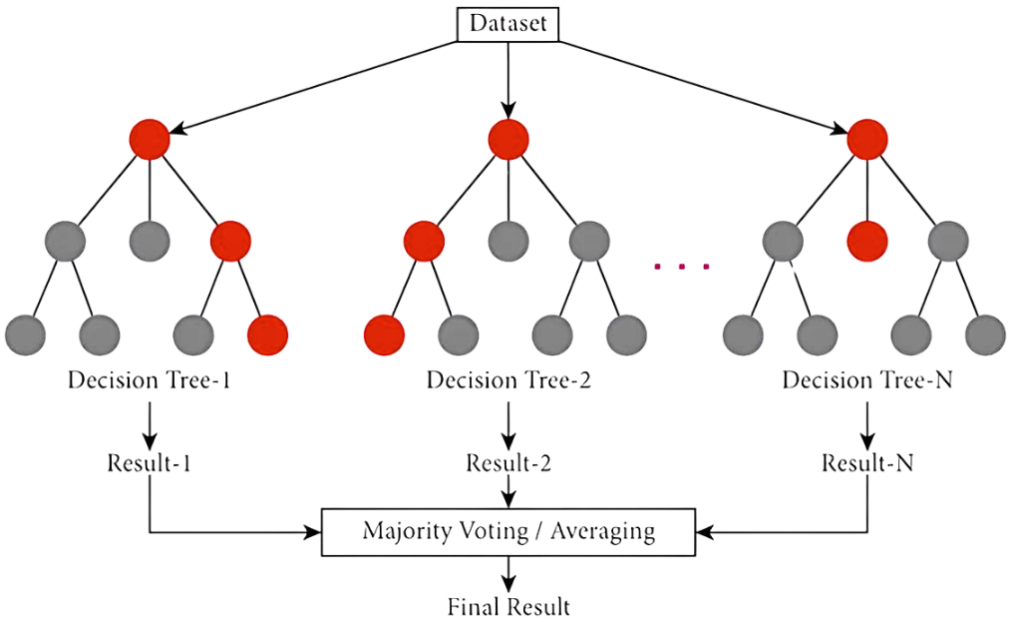


Figure 2.3: Random Forest Visualization

Conceptual Understanding The fundamental idea behind Random Forest is to combine the predictions of several base estimators built with a certain degree of randomness to improve the generalizability of the model. Each tree in the forest is built from a bootstrap sample of the data, and at

each node, a random subset of features is selected for splitting. This randomness helps in creating a diverse set of trees, which, when aggregated, produce a more robust and accurate model.

Mathematical Formulation Given a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, Random Forest constructs M decision trees $\{T_m\}_{m=1}^{M}$. The prediction $\hat{y}$ for a new input $\mathbf{x}$ is given by:

For classification (majority voting):

$$\hat{y} = \text{mode}\{T_m(\mathbf{x}) \mid m = 1, 2, \ldots, M\}$$

For regression (average prediction):

$$\hat{y} = \frac{1}{M} \sum_{m=1}^{M} T_m(\mathbf{x})$$

Each tree T_m at each split considers a random subset of k features from the total d features. The parameter k is typically set to $\sqrt{d}$ for classification and $d/3$ for regression tasks.

Assumptions Underlying Random Forest Random Forest makes just one assumption:

- **Sufficient Diversity Among Trees**: The individual trees are sufficiently diverse due to random feature selection and bootstrapping.

XGBoost

XGBoost (eXtreme Gradient Boosting) is a scalable and efficient implementation of gradient boosting machines, designed for speed and performance. It was developed by Chen and Guestrin [28] and has become a popular choice due to its speed, accuracy, and flexibility.

Conceptual Understanding XGBoost builds an ensemble of decision trees sequentially, where each new tree aims to correct the errors of the previous trees. It employs the gradient boosting framework, optimizing a differentiable loss function by adding weak learners (decision trees) in a stage-wise fashion.

Mathematical Formulation Given a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, the model predicts the output $\hat{y}_i$ by summing the predictions of K regression trees:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}$$

where $\mathcal{F}$ is the space of regression trees. The objective function to be minimized is:

$$\mathcal{L} = \sum_{i=1}^{N} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where $l(y_i, \hat{y}_i)$ is a differentiable convex loss function (e.g., squared error for regression), and $\Omega(f_k)$ is a regularization term penalizing the complexity of the model, typically defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|\omega\|^2$$

Here:

- T is the number of leaves in the tree.
- ω represents the leaf weights.
- γ and λ are regularization parameters controlling the trade-off between model complexity and training loss.

By adding new trees that predict the residuals (errors) of the previous ensemble, XGBoost minimizes the objective function using second-order Taylor approximation, which improves optimization speed and accuracy.

Assumptions Underlying XGBoost XGBoost operates under several assumptions:

- **Additive Modeling**: Assumes that the underlying relationship can be modeled as an additive combination of decision trees.
- **Independence of Residuals**: Assumes that the errors corrected by each subsequent tree are independent.

- **Sufficient Data**: Requires enough data to accurately capture the underlying patterns without overfitting.

CatBoost

CatBoost (Categorical Boosting) is a gradient boosting algorithm developed by Yandex [29] [30], designed to handle categorical features effectively. CatBoost outperforms many existing algorithms by reducing overfitting and providing state-of-the-art results with minimal hyperparameter tuning.

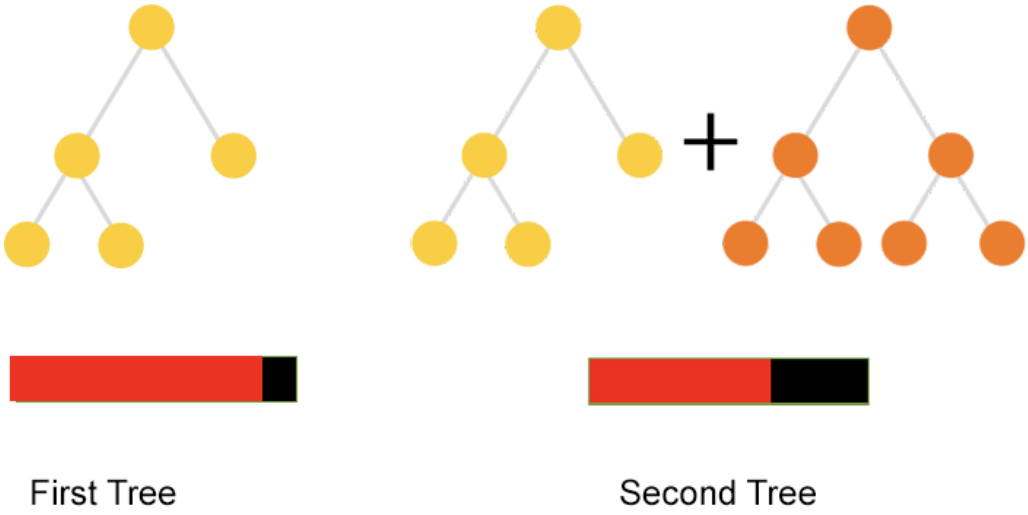


Figure 2.4: CatBoost Algorithm - Sourced from [1]

Conceptual Understanding CatBoost is based on the gradient boosting framework but introduces innovative techniques to handle categorical features and reduce overfitting:

- **Ordered Target Statistics**: For each categorical feature, CatBoost computes target statistics (e.g., mean target value) in an *ordered* manner to prevent target leakage. This means that for each data point, the statistics are calculated only using data preceding it in a given permutation. This allows CatBoost to handle categorical features efficiently without the need for extensive preprocessing like one-hot encoding.
- **Ordered Boosting**: Instead of using the same dataset for both learning the model and computing the residuals (gradients), CatBoost uses permutations of the dataset to create training sets that avoid using the

current data point when computing its own residuals. This technique reduces overfitting and improves model generalization.

Mathematical Formulation CatBoost builds an ensemble of decision trees in a stage-wise fashion, similar to other gradient boosting methods. At each iteration t, it aims to minimize a loss function L by adding a new tree $f_t(\mathbf{x})$ to the ensemble:

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \gamma_t f_t(\mathbf{x})$$

Here, $F_t(\mathbf{x})$ is the ensemble model at iteration t, and γ_t is the learning rate. The main innovation lies in how CatBoost calculates the gradients and handles categorical features, as discussed above.

Assumptions Underlying CatBoost CatBoost makes several assumptions for effective performance:

- **Dependence Structure**: Assumes that the data has a certain dependence structure that can be captured by the ordered boosting and target statistics methods.
- **Sufficient Data**: Requires a sufficient amount of data to accurately compute ordered target statistics without introducing significant noise.
- **Independent and Identically Distributed Data**: The training data is assumed to be independently drawn from the distribution.

2.4.2 Deep Learning Models

The deep learning models used in this project are Convolutional Neural Networks (CNNs) and Multilayer Perceptrons (MLPs).

Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) is a class of feedforward artificial neural networks (ANN) that consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one.

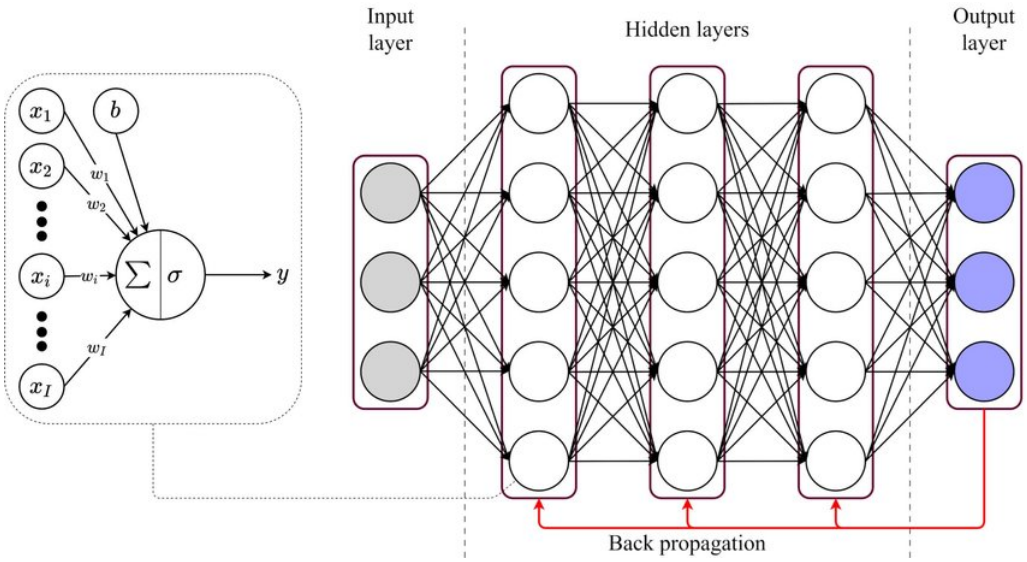


Figure 2.5: Multilayer Perceptron Architecture - Sourced from [2]

Conceptual Understanding An MLP comprises an input layer, one or more hidden layers, and an output layer. Each layer is made up of nodes (neurons) that take as input the weighted sum of the outputs from the previous layer, apply an activation function, and pass the result to the next layer. In this way, the network is capable of approximating any continuous function given sufficient data and appropriate network architecture.

Mathematical Formulation Given an input vector $\mathbf{x} \in \mathbb{R}^n$, the MLP computes an output $\hat{\mathbf{y}}$ through a series of transformations:

1. **Forward Propagation**:

   For each layer $l = 1, 2, \ldots, L$:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = \phi^{(l)}(\mathbf{z}^{(l)})$$

   Where:

   - $\mathbf{W}^{(l)}$ is the weight matrix connecting layer $(l-1)$ to layer l.
   - $\mathbf{b}^{(l)}$ is the bias vector for layer l.
   - $\mathbf{a}^{(l)}$ is the activation of layer l.
   - $\phi^{(l)}$ is the activation function for layer l (e.g., sigmoid, ReLU, tanh).

- For the input layer, $\mathbf{a}^{(0)} = \mathbf{x}$.

The output of the network is $\hat{\mathbf{y}} = \mathbf{a}^{(L)}$.

2. **Loss Function**:

   A loss function $L(\mathbf{y}, \hat{\mathbf{y}})$ measures the discrepancy between the true outputs $\mathbf{y}$ and the predicted outputs $\hat{\mathbf{y}}$. Common loss functions include mean squared error (MSE) for regression and cross-entropy loss for classification.

3. **Backpropagation and Weight Updates**:

   The network's weights and biases are updated using gradient descent optimization algorithms to minimize the loss function. The gradients are computed using backpropagation:

$$\text{For each parameter } \theta \in \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\} :$$

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

   Where η is the learning rate.

Assumptions Underlying MLP For MLPs to perform effectively, several assumptions are made:

- **Sufficient Data**: Assumes access to a large and representative dataset for training to capture the underlying patterns.
- **Appropriate Network Architecture**: The architecture (number of layers, number of neurons per layer, activation functions) must be suitable for the complexity of the task.
- **Independent and Identically Distributed Data**: The training data is assumed to be independently drawn from the distribution.
- **Stationarity**: Assumes that the underlying data distribution does not change over time.

Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs), firstly introduced by Fukushima et al. [31] are a class of deep learning models specifically designed to process data with a grid-like topology, such as images. CNNs are characterized by their use of convolutional layers that apply filters to local regions of the input, capturing spatial hierarchies and patterns and producing feature maps. Pooling layers are used to reduce the dimensionality of the feature maps, and fully connected layers interpret the extracted features for classification or regression tasks.

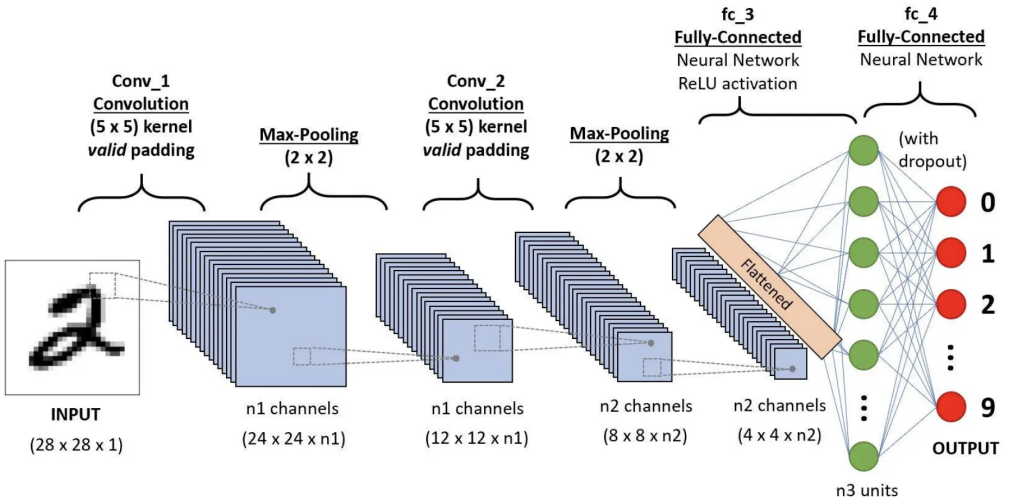


Figure 2.6: Convolutional Neural Network Architecture - Sourced from [3]

Conceptual Understanding The main mathematical operation in CNNs is the **convolution**, which involves sliding a small matrix of weights, known as a filter or kernel, over the input data. At each position, the filter performs element-wise multiplication between its weights and the corresponding input values, summing the results to produce a single output value. This process' output is a feature map that highlights the presence of certain features in the input.

During training, the CNN learns how to adjust the weights of these filters in a way that they become sensitive to specific patterns important for the task at hand, such as edges, textures, or shapes in an image. Because the same filter is applied across all regions of the input (a concept called *weight sharing*), the network can detect these features regardless of their position in the image.

Once trained, the CNN has filters that activate strongly when their

learned features are present in the input. This means the network effectively highlights the regions of the input that are most relevant for classification. By stacking multiple convolutional layers, the CNN can learn a hierarchy of features, with earlier layers detecting simple patterns and deeper layers combining them into more complex representations like objects or faces.

In essence, CNNs are capable of automatically learning to detect important features in the input data, adjusting filter weights during training to focus on meaningful patterns and building hierarchical representations that enable accurate classification or recognition tasks.

Mathematical Formulation CNNs follow a series of operations to process input data and learn the underlying patterns:

1. **Convolution Operation**:

   For an input image (or feature map) $\mathbf{X} \in \mathbb{R}^{H \times W \times C_{\text{in}}}$, where H is the height, W is the width, and C_{in} is the number of input channels, the convolution operation with a filter (kernel) $\mathbf{K} \in \mathbb{R}^{k_H \times k_W \times C_{\text{in}}}$ produces an output feature map $\mathbf{Y} \in \mathbb{R}^{H' \times W' \times C_{\text{out}}}$:

$$Y_{i,j}^{(c)} = \sum_{m=1}^{k_H} \sum_{n=1}^{k_W} \sum_{c'=1}^{C_{\text{in}}} K_{m,n}^{(c,c')} X_{i+m-1,j+n-1}^{(c')}$$

   where:

   - $Y_{i,j}^{(c)}$ is the output at position (i, j) for output channel c.
   - $K_{m,n}^{(c,c')}$ is the filter weight at position (m, n) connecting input channel c' to output channel c.
   - The output dimensions H' and W' depend on the input size, filter size, padding, and stride.

2. **Activation Function**:

   After convolution, an activation function ϕ is applied element-wise:

$$\mathbf{A} = \phi(\mathbf{Y})$$

   Common activation functions include:

- **ReLU (Rectified Linear Unit)**: $\phi(z) = \max(0, z)$
- **Sigmoid**: $\phi(z) = \frac{1}{1+e^{-z}}$
- **Tanh**: $\phi(z) = \tanh(z)$

3. **Pooling Operation**:

   Pooling layers reduce the spatial dimensions by applying a pooling function over non-overlapping or overlapping regions:

$$P_{i,j}^{(c)} = \text{pool}\left(A_{r,s}^{(c)} \mid r \in R_i, s \in R_j\right)$$

   where pool could be:

   - **Max Pooling**: Takes the maximum value within the region.
   - **Average Pooling**: Computes the average value within the region.

4. **Fully Connected Layers**:

   The output of the convolutional and pooling layers is flattened into a vector and passed through one or more fully connected layers (Section 2.4.2).

5. **Loss Function and Optimization**:

   The network's parameters are learned by minimizing a loss function $L(\mathbf{y}, \hat{\mathbf{y}})$ using optimization algorithms like stochastic gradient descent (SGD) or Adam. Backpropagation is used to compute gradients with respect to the parameters.

Assumptions Underlying CNN CNNs are based on several key assumptions:

- **Local Connectivity**: Assumes that local pixels are more related than distant pixels, allowing the use of small filters to capture local patterns.
- **Shared Weights (Stationarity)**: The same filter (set of weights) is applied across the entire input, assuming that the features are translationally invariant.
- **Sufficient Data**: Requires large amounts of labeled data to effectively learn complex patterns.

Chapter 3

Methods and Experiments

This chapter details the methods and experiments conducted in the audio deepfake detection research. The chapter is divided into three main sections: Dataset, Feature Engineering, and Models.

3.1 Dataset

A deep understanding of the dataset is fundamental as it forms the foundation of the machine learning models developed for this research. The section is divided into two main parts: the selection of the dataset and the preprocessing steps applied to it. The selection part elaborates on the importance of choosing the right dataset, highlights the strengths and limitations of the chosen dataset, and briefly summarizes other datasets used in similar research. The preprocessing part describes the transformations performed on the data, which are essential for ensuring the quality and consistency of the dataset.

3.1.1 Dataset Selection

Selecting the appropriate dataset is critical for developing robust audio deepfake detection models. Four main factors contribute to the effectiveness of a dataset in this domain:

- Diverse Speaker Voices: A comprehensive dataset should include a wide range of speaker voices to ensure the model can generalize across different vocal characteristics. This diversity includes variations in accent, tone, pitch, and speaking style.

- High-Quality TTS Algorithms: it is essential to include samples generated by state-of-the-art Text-to-Speech (TTS) algorithms. This ensures that the model is trained to detect the latest synthetic speech technologies, which are often more sophisticated and harder to identify. Including diverse TTS algorithms is fundamental to training a robust model that can detect a wide range of deepfake techniques.

- Number of Utterances: The dataset should contain a sufficient number of utterances to train a deep learning model effectively. A larger dataset allows the model to learn more complex patterns and generalize to unseen data.

- Gender Balance: To avoid gender bias and ensure the model performs equally well on male and female voices, the dataset should have a balanced representation of genders.

The Fake or Real (FoR) dataset proposed by Reimao et al. [32] aligns well with the outlined criteria. It includes 140 different real speakers and 33 synthetic voices generated by seven state-of-the-art commercial and open-source TTS systems. The dataset comprises over 84,000 fake utterances and 111,000 real utterances, both of which are also available in a gender-balanced version. A notable feature of the FoR dataset is its test set, which contains audio samples from a TTS algorithm not present in the training set, thus enabling the evaluation of the model's generalization capabilities.

Two limitations of the dataset are the presence of only TTS-generated fake utterances and the lack of noise and environmental variations. However, the latter can be addressed by augmenting the dataset with additional samples containing background noise and other environmental factors. The former is not a problem for this research, as the focus is on detecting TTS-generated deepfake audio.

Furthermore, the dataset includes several versions. The original dataset, "for-original," contains the raw audio files. "For-norm" is the normalized version mainly used in this project, with preprocessing steps detailed in the next section. The "for-2seconds" version truncates and balances the data, resulting in 17,870 utterances. Additionally, to simulate real-world attacks, the re-recorded version, "for-rerec," mimics synthetic speech generated on one device and recorded on another.

Several other datasets are widely used in the field of audio deepfake detection:

- ASVspoof Datasets: The ASVspoof challenge series has produced several datasets focused on protecting automatic speaker verification systems from spoofing attacks [33] and [23]. The ASVspoof 2021 dataset also includes audio deepfake samples, which consider data compression effects [34].

- ADD Datasets: The ADD (Audio Deepfake Detection) challenge series has released two datasets, ADD 2022 [35] and ADD 2023 [36], aiming at covering many real-life and challenging scenarios not covered by existing datasets. Specifically, the ADD 2023 dataset focused on localizing the manipulated intervals in a partially fake utterance and pinpointing the generation source for any fake audio [19].

- In-the-Wild Dataset: This dataset was proposed by Muller et al. [37] to test the generalization capabilities of audio deepfake detection models. The authors collected deepfake audio recordings of celebrities and politicians and provided them with their real counterparts.

3.1.2 Dataset Preprocessing

The transformations applied can significantly impact the model's performance by enhancing the quality and consistency of the data. The "for-norm" version of the FoR dataset underwent several preprocessing steps detailed in the paper by Reimao et al. [32] and briefly summarized below:

- Format Conversion: The audio files were converted to the WAV format, preferred for machine learning applications.

- Volume Normalization: The audio files were normalized to 0dBFS to avoid volume to be a distinguishing factor.

- Resampling: The audio files were resampled to 16 kHz to ensure uniformity across the dataset.

- Channel Selection: The synthetic audio was mono and the real audio was stereo, therefore the synthetic audio was converted to mono to avoid the model learning to distinguish between the two.

- Silence Removal: Silence segments at the beginning and end of each audio file were removed.

- Gender and Class Balancing: The two classes were balanced in terms of gender and number of samples, resulting into a reduction of the dataset size to 69,400 utterances.

3.2 Feature Engineering

Feature engineering is fundamental in developing machine learning models. In audio deepfake detection, the aim is to convert raw audio data into a format that mathematical models can effectively process. This section outlines the feature engineering process, starting with the preliminary decisions regarding the extraction hyperparameters. Subsequently, the technical choices for each feature type are documented, including the number of features extracted and any mathematical transformations applied.

In this research, both traditional machine learning models and CNN deep learning models are used, requiring different input data formats: 1D for traditional models and 2D/3D for CNNs. Each feature type is analyzed in two versions: one where features are used "raw" as input to classifiers, and another where features are processed through a CNN, and the resulting neural features are used as input. Due to these differences, both the extraction hyperparameters selection and the practical extraction details must be discussed separately for the two versions. Therefore, in the following sections, there will be a distinction between the raw features approach and the neural features approach.

3.2.1 Feature Extraction Hyperparameters

Extracting high-quality features from audio data involves several considerations, such as the extraction interval, window length, and hop length. The complexity arises from the interdependence of these choices; optimal settings for one feature type may not be suitable for another. Additionally, different models may respond differently to these configurations, making the search space for possible solutions enormous. Given the impracticality of exploring all combinations, a greedy approach was adopted to efficiently identify the best settings, trying to minimize compromises at the same time. To achieve this, an experimental approach was used to select intervals, feature types, and hop and window lengths based on the model's performance. The approach, illustrated by algorithm 1 consists of three phases:

Algorithm 1: Feature Engineering Process

Data: Audio dataset, list of classifiers C, list of feature types F, list of extraction intervals I, list of hop-window length combinations H

Result: Optimal feature extraction settings

Phase 1: Evaluate All CFI Combinations

foreach *classifier* $c \in C$ **do**
  foreach *feature type* $f \in F$ **do**
    foreach *interval* $i \in I$ **do**
      Evaluate classifier c with feature type f and interval i;
      Store performance metrics on train, test, and val set;
    end
  end
end

Phase 2: Select Best Feature Types and Intervals

Select the top-performing combinations of feature types F_{best}, intervals I_{best} based on performance metrics and two classifiers C_{best};

Phase 3: Investigate Hop and Window Length Effect

foreach *classsifier* $c \in C_{best}$ **do**
  foreach *feature type* $f \in F_{best}$ **do**
    foreach *interval* $i \in I_{best}$ **do**
      foreach *hop-window combination* $h \in H$ **do**
        Evaluate classifier c with feature type f, interval i, and hop-window combination h;
        Store performance metrics on train, test and val set;
      end
    end
  end
end

Select the optimal settings based on final performance metrics;

1. Evaluate All Possible CFI Combinations: This phase involves testing all combinations of classifiers (C), feature types (F), and extraction intervals (I).

2. Select the Best Feature Types and Intervals: The best performing combinations of feature types and intervals are chosen for further investigation.

3. Investigate the Effect of Hop and Window Length: The impact of different hop and window lengths is analyzed on the previously selected best combinations.

The subsequent paragraphs detail the hyperparameters considered for the feature engineering experiments.

Raw Features Approach

The traditional ML experiments considered the following hyperparameters:

- 4 classifiers: Random Forest, Support Vector Machine, CatBoost and Logistic Regression.
- 2 hop lengths (wl/4, wl/2) and 5 window lengths (256, 512, 1024, 2048, 4096), resulting in 10 hop-window combinations.
- 9 feature types (mel_spec, mfcc, cqt, chroma_stft, spectral_centroid, spectral_bandwidth, spectral_rolloff, zero_crossing_rate, rms).
- 4 extraction intervals (0.5s, 1s, 1.5s, 2s).

Neural Features Approach

The DL experiments were conducted slightly differently according to the CNN model. In the case of MobileNetV3 the setup was similar to the ML models, with fewer feature types and intervals due to the model's complexity:

- 4 classifiers: Random Forest, Support Vector Machine, CatBoost and XGBoost.
- 2 hop lengths (wl/4, wl/2) and 5 window lengths (256, 512, 1024, 2048, 4096), resulting in ten hop-window combinations.
- 6 feature types (mfcc, cqt, chroma_stft, rms, spectral_rolloff, zero_crossing_rate).
- 4 extraction intervals (1s, 2s).

For VGG16, only the Random Forest classifier was used, as it outperformed others in prior experiments. Various VGG16 layers were tested as feature extractors: specifically the last convolutional layer, the penultimate, and the antepenultimate, with pooling incorporated in VGG16. Consequently, the hyperparameters considered for the VGG16 experiments were:

- 1 classifiers: Random Forest.
- 3 layers of VGG16 (layer 43, layer 33 and layer 23).
- 2 hop lengths (wl/4, wl/2) and 5 window lengths (256, 512, 1024, 2048, 4096), resulting in ten hop-window combinations.

Type	Model	Classifiers	Layers	Hparams
CNN	MNETv3	RF, SVM, CatBoost, XGBoost	last	table 3.2
	VGG16	RF	43, 33, 23	table 3.2
ML	-	RF	-	table 3.2
	-	SVM	-	table 3.2
	-	CatBoost	-	table 3.2
	-	LR	-	table 3.2

Table 3.1: Feature Engineering Hyperparameters - Part 1

Type	Features	Intervals (s)	Win Lengths (WL)	Hop Lengths
CNN	MFCC, CQT, Chroma, RMS, SR, ZCR	1, 2	256, 512, 1024, 2048, 4096	WL/4, WL/2
ML	Mel, MFCC, CQT, Chroma, SC, SB, SR, ZCR, RMS	0.5, 1, 1.5, 2	256, 512, 1024, 2048, 4096	WL/4, WL/2

Table 3.2: Feature Engineering Hyperparameters - Part 2

- 6 feature types (mfcc, cqt, chroma_stft, rms, spectral_rolloff, zero_crossing_rate).
- 4 extraction intervals (1s, 2s).

A summary of the hyperparameters considered for the feature engineering experiments is provided in tables 3.1 and 3.2.

3.2.2 Extraction Details

The extraction process differs based not only on whether the features are used in their raw form or processed through a CNN, but also on the feature type within the same model category. Specifically, it is possible to differentiate between features that produce multiple values per time frame (e.g., MFCC, CQT, Chroma) and those that produce a single value per frame (e.g., RMS, ZCR).

Raw Features Approach

Raw features that generate multiple values per time frame include Mel Spectrogram, MFCC, CQT, and Chroma. For these features, the number of extracted values per time frame is user-specified. The extraction interval,

window length, and hop length determine the number of time frames. For example, to extract 20 MFCC features from a 2-second interval with a window length of 1024, a hop length of 256, and a sampling rate of 16 kHz, we would have:

$2s \times 16kHz = 32000$ samples 32000 samples/256 hop length = 125 frames

These frames are made up of 1024 samples each. Each frame is transformed into 20 MFCC features, resulting in a 20×125 matrix for the 2-second interval. To convert this into a 1D input for classifiers, the matrix is averaged along the time axis, yielding a 20-dimensional vector. This process is similarly applied to other feature types. While this method loses time information, it preserves frequency information.

Features that produce a single value per frame include Spectral Centroid, Spectral Bandwidth, Spectral Rolloff, Zero Crossing Rate, and RMS. Averaging these features over time would result in a single value per interval. To avoid this, multiple values were retained for each interval by not averaging along the time axis. The number of features may be controlled by adjusting the window length and hop length for a given interval; the larger the hop length, the fewer the extracted features. This approach retains time information but provides less frequency information.

Notably, the feature engineering experiments on raw features were based on a subset of the data. This latter was made up of 18000 training audio files, 3600 validation audio, and the same amount of test audio. The subset was reduced to 3000, 750, and 750 audio files for the hop and window length experiments. In all cases, the data points are evenly distributed across fake and real classes, using undersampling and oversampling techniques to address possible class imbalances introduced by the different durations of fake and real audio files.

The `scikit-learn` library was used for the models, with the default parameters for each classifier while the `librosa` and `torchaudio` libraries were used for the feature extraction.

Neural Features Approach

In the neural features approach, the feature representation was transformed from a 1D vector to a 2D image, where the x-axis represented time frames, and the y-axis displayed feature values. This transformation balanced computational complexity and information preservation by extending the features to an image size of 112×112 pixels.

For features that produced a single value per frame, the 2D extension was achieved by stacking these values along the y-axis to reach 112 values. For features with multiple values per frame, the process was more complex. MFCC could directly generate 112 values, and Chroma was automatically extended to 112 using the Librosa function. However, CQT, which was limited to 84 values, was extended to 112 by padding with zeros.

Further processing was required to fit the input shape needed by Convolutional Neural Networks (CNNs), which is $3 \times 224 \times 224$. The PyTorch [38] `transforms.Resize` function was used to resize the image to 224×224 pixels through interpolation. To extend to 3 channels, two approaches were considered: stacking the 2D image three times along the channel axis, or creating a CNN 3D Mapper that returned three RGB values for each pixel. The latter approach, which yielded optimal results, is detailed in Section 3.4.2.

Additionally, pixel values were transformed to the standard image range $[0, 255]$. Since some features produced negative values, a min-max scaling was applied, followed by multiplication by 255 and conversion to integers. The scaling parameters were calculated on the training set and applied to other sets to prevent data leakage. To enhance robustness against outliers, the minimum and maximum values were computed using the 5th and 95th percentiles as empirically tested optimal values.

```
    # Compute the 5th and 95th percentiles
    Q1 = np.percentile(train_data, 5)
    Q3 = np.percentile(train_data, 95)
    IQR = Q3 - Q1

    # Compute the min and max values
    data_min = Q1 - 1.5 * IQR
    data_max = Q3 + 1.5 * IQR
```

Listing 3.1: IQR based Min-Max Scaling for images

The last step was to prepare the data for the CNN pre-trained models. All the necessary transformations were applied with the ready-to-use PyTorch transform functions for the specific CNN model, `VGG16_Weights.IMAGENET1K_V1.transforms` [1] for VGG16 and `MobileNet_V3_Small_Weights.IMAGENET1K_V1.transforms` [2] for MobileNetV3.

[1] `https://pytorch.org/vision/main/models/generated/torchvision.models.vgg16.html`

[2] `https://pytorch.org/vision/main/models/generated/torchvision.models.mobilenet_v3_small.html`

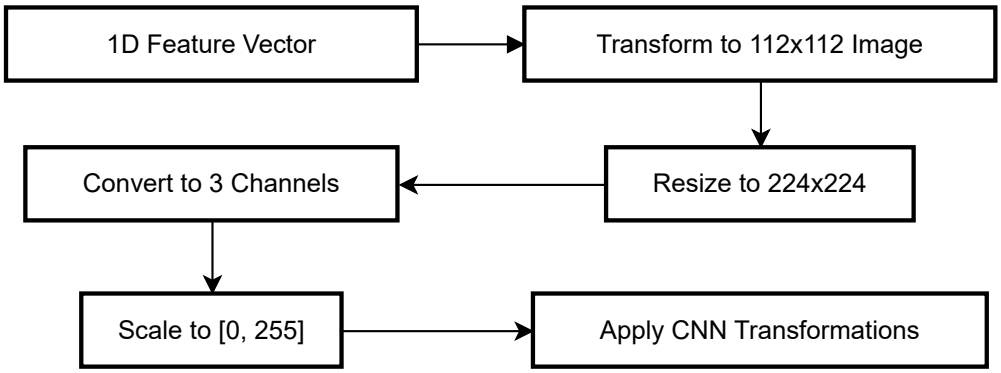


Figure 3.1: Neural Features Extraction Process

The entire process is summarized in Figure 3.1

Due to the increased computational complexity of deep learning models, experiments with neural features used a smaller data subset consisting of 3000 training audio files, 750 validation audio files, and 750 test audio files. Data points were evenly distributed across fake and real classes using PyTorch's `torch.utils.data.WeightedRandomSampler`.

The `torchvision` library was used for `vgg16_bn` and `MobileNetV3` pretrained CNN models, while `scikit-learn`, `librosa` and `torchaudio` libraries were used respectively for the models and feature extraction.

3.3 ML Models

In this section are discussed the traditional machine learning models used in the audio deepfake detection experiments, starting from the base model choices and moving to the methods used for improving the model's performance.

3.3.1 Base Models

The initial step in developing a detection model involved establishing baseline models. Four classifiers were selected for this purpose: Random Forest, Support Vector Machine with an RBF kernel, CatBoost, and Logistic Regression. These classifiers were chosen based on their proven effectiveness and versatility across various domains. The baseline performance for each model was defined by identifying the best result achieved across all feature types and intervals without additional tuning. Additionally, two versions of each model were considered: one optimized for data learning and the

other for generalization. These choices produced models with strong initial performance, setting a solid foundation for further enhancement.

3.3.2 Features Reduction (RFE)

A possibility to improve the model's performance is to reduce the number of features used for training, trying to get rid of irrelevant or redundant features. Principal Component Analysis (PCA) is a common method for this purpose, and it demonstrated effectiveness in research by Iqbal et al. [39]. However, the primary concern with PCA is its potential to cause difficulties when used with Random Forests (RF). RF models make branch decisions based on specific, individual features, whereas PCA generates new features as combinations of all original features. This can result in unpredictable behavior and diminished performance when PCA is applied prior to RF, as the interpretability and decision-making process of RF are disrupted. Therefore, RFE, which retains the original feature set, was selected as a more suitable method for this task.

Two main experiments were conducted with different feature sets. The first experiment considered 20 features per type, while the second expanded this to 40 features per type (only Chroma STFT was set to 12). For both experiments, models were evaluated under two scenarios: the first involved retaining all features types without any selection, the second, instead, involved applying RFE to retain only the top 20, 30, 40, 80, or 100 features using the `scikit-learn` RFE implementation as a selector.

The tested models included Random Forest, CatBoost, and XGBoost. Random Forest and CatBoost were selected because they had performed best in previous experiments. Additionally, XGBoost was included based on its proven effectiveness as highlighted in the article by Iqbal et al. [39].

These experiments were performed in two modalities. Initially, the Mel spectrogram was included as a feature type. However, the results were suboptimal. An analysis of the features retained by RFE indicated that the selected features were predominantly MFCC and Mel spectrogram, which are known to provide correlated information. This redundancy likely hindered performance. Consequently, in the second modality, the Mel spectrogram was removed to assess whether the remaining features would offer better generalization. This approach aimed to reduce redundancy and encourage the selection of a more diverse set of features.

A summary of the RFE process is provided in Algorithm 2.

Algorithm 2: Feature Selection Process Using RFE

Data: Feature types F_t (e.g., MFCC, CQT), Feature sets F_s, Models M (e.g., RF, CatBoost, XGBoost), RFE thresholds R (e.g., 20, 30, 40), Normalization techniques N (e.g., RobustScaler)

Result: Optimized feature set for model training

Phase 1: Feature Combination Evaluation

foreach *feature type* $f_t \in F_t$ **do**
    Generate feature set F_s by retaining 20 and 40 features for each f_t;
    foreach *normalization technique* $n \in N$ **do**
        Apply $n \rightarrow F_s$;
        Evaluate F_s by concatenating all retained features;
        Train models M using F_s;
        Store performance metrics P_{mn};
    end
end

Phase 2: Recursive Feature Elimination (RFE)

foreach *model* $m \in M$ **do**
    foreach *RFE threshold* $r \in R$ **do**
        Apply RFE to F_s to retain top r features based on model importance;
        Train model m on reduced feature set F_{rfe};
        Store performance metrics P_{rfe};
    end
end

Phase 3: Exclusion of Mel Spectrogram Features

Exclude Mel Spectrogram features from F_t;
Generate new feature set $F_{s'}$ by repeating Phase 1 without Mel Spec;
foreach *model* $m \in M$ **do**
    foreach *RFE threshold* $r \in R$ **do**
        Apply RFE to $F_{s'}$;
        Train model m on F_{rfe};
        Store performance metrics $P_{rfe'}$;
    end
end

Phase 4: Analysis and Final Selection

Compare P_{mn}, P_{rfe}, and $P_{rfe'}$ across all models;
Identify optimal feature set and normalization method based on performance metrics;
Finalize feature set F_{opt} and scaling method for final model training;

Normalization

Throughout the above-listed experiments, it became evident that the features had vastly different ranges, necessitating normalization. Several normalization techniques were tested, including `StandardScaler`, `MinMaxScaler`, `RobustScaler`, and `MaxAbsScaler` from the scikit-learn preprocessing module [3]. Among these, the RobustScaler consistently yielded the best results, as it was particularly effective in handling the presence of outliers in the data, which may have skewed the results of other scalers. Consequently, the RobustScaler was selected as the normalization technique for the final models.

3.4 DL Models

In this section are discussed the methodologies and choices behind the definition of the deep learning models, from the base models to the final models, illustrating the reasons behind the made choices.

3.4.1 Transfer Learning

Transfer learning has emerged as a powerful technique in machine learning, particularly in scenarios where large datasets are unavailable or when leveraging pre-trained models can significantly enhance performance. In this project, transfer learning is used to extract features and fine-tune models for audio deepfake detection. Two architectures, MobileNetV3 and VGG16, were utilized for this purpose.

Feature Extraction

Using the pre-trained models as feature extractors is the first way to transfer their knowledge to the new task. It consists of processing the audio data through the extractor module of the pre-trained model and using the output as input to classifiers.

MobileNetV3 was employed as a feature extractor for the audio data, as detailed in Section 3.2.1. This model, known for its efficiency and compact architecture, processed different feature types to learn the most relevant aspects for the task at hand. The output from the last layer of MobileNetV3 (after pooling), a 1D array consisting of 576 values, was then

[3] `https://scikit-learn.org/stable/api/sklearn.preprocessing.html`

fed into traditional classifiers such as Random Forest (RF), Support Vector Machine (SVM), CatBoost, and XGBoost. Each classifier was trained using default parameters, leveraging the rich features extracted by MobileNetV3 to enhance classification performance.

VGG16 was used to assess feature extraction at different layers of the network, focusing on layers 23, 33, and 43 as identified in Listing 3.2. By examining features extracted at various stages of the VGG16 architecture, the aim was to determine which layer provided the most effective representation for the task. To reduce complexity, only RF was used as the classifier in these experiments.

```
    # Load the pre-trained VGG16_bn model
    vgg16_bn = models.vgg16_bn(weights='DEFAULT')

    # Initialize a list to hold the indices of the layers
    conv_layer_indices = []

    # Iterate over the model's features to find Max Pooling
   layers
    for idx, layer in enumerate(vgg16_bn.features):
        if isinstance(layer, nn.MaxPool2d):
            conv_layer_indices.append(idx)

    # Print the indices of the convolutional layers
    print("Indices:", conv_layer_indices)

    # Output: Indices: [6, 13, 23, 33, 43]
```

Listing 3.2: VGG16 Layers Selection

The feature extraction process choices for both models is summarized in Table 3.3.

Choice	VGG16	MobileNetV3
Extraction Method	Extracted from Layers 23, 33, 43	Extracted from the final layer
Classifiers	Random Forest	Random Forest, SVM, CatBoost, XGBoost
Feature Vector Size	Varied depending layer	576-dimensional vector

Table 3.3: Comparison of Transfer Learning (Feature Extraction) Choices for VGG16 and MobileNetV3

Fine Tuning

Fine Tuning is the second way to transfer knowledge from pre-trained models. It consists of training the entire model or a subset of it on the new task. This process was done in two steps: first, an MLP was trained and attached to the frozen pre-trained convolutional layers of the original model. The whole model or a subset of it was then trained. This two-step approach ensured that the pre-trained models' learned knowledge was not lost during fine-tuning.

For **MobileNetV3**, four different MLP architectures were tested to identify the best-performing model, as detailed in Table 3.4. Each classifier was trained for 50 epochs, stopping the training if the validation loss did not improve for five consecutive epochs.

Version	Architecture
MLP1	576 (input) $\rightarrow$ 400 $\rightarrow$ 2 (output)
MLP2	576 (input) $\rightarrow$ 256 $\rightarrow$ 2 (output)
MLP3	576 (input) $\rightarrow$ 128 $\rightarrow$ 2 (output)
MLP4	576 (input) $\rightarrow$ 328 $\rightarrow$ 128 $\rightarrow$ 2 (output)

Table 3.4: MLP Architectures for MobileNetV3

Once the best-performing MLP was selected, the entire MobileNetV3 model was fine-tuned by lowering the learning rate by a factor of ten (to 0.0001) and training for 80 epochs, with early stopping applied if no improvement was seen for eight epochs. The Adam optimizer and CrossEntropyLoss were used throughout the training process.

For **VGG16**, given its more complex architecture, a single MLP was trained for each feature type. The MLP architecture used is detailed in Listing 3.3.

```
1    class Classifier(nn.Module):
2        def __init__(self, vgg16):
3            super(Classifier, self).__init__()
4
5            # Use existing layers from VGG16
6            self.linear1 =
   vgg16.classifier[0].requires_grad_(False)
7            self.relu1 = vgg16.classifier[1]
8            self.dropout1 = vgg16.classifier[2]
9            self.relu2 = vgg16.classifier[4]
```

```
            self.dropout2 = vgg16.classifier[5]

            # Define new layers
            self.linear2 = nn.Linear(4096, 2048, bias=True)
            self.linear3 = nn.Linear(2048, 2, bias=True)

        def forward(self, x):
            x = self.linear1(x)
            x = self.relu1(x)
            x = self.dropout1(x)
            x = self.linear2(x)
            x = self.relu2(x)
            x = self.dropout2(x)
            x = self.linear3(x)
            return x
```

Listing 3.3: MLP for VGG16

By freezing the first layer of the VGG16 classifier, a huge reduction in parameters $(102,764,544)$ was achieved while still maintaining optimal performance. The fine-tuning process involved freezing the first two convolutional layers of the feature extractor and the first layer of the classifier, with the remaining layers being trained for 40 epochs using a learning rate of 0.0001, with early stopping after seven epochs if no improvement was noted. Adam was used as the optimizer, with CrossEntropyLoss as the loss function.

The fine-tuning process for both models is summarized in Table 3.5.

Choice	VGG16	MobileNetV3
MLPs Tried	1 MLP (Listing 3.3)	4 MLPs (Table 3.4)
MLP Training Epochs and Patience	40 epochs, patience 7	50 epochs, patience 5
MLP Learning Rate	0.001	0.001
Frozen Layers	Classifier: only first, Extractor: up to 27	None
Fine-Tuning Epochs and Patience	40 epochs, patience 7	80 epochs, patience 8
Learning Rate (LR) for Fine-Tuning	0.0001	0.0001
Optimizer	Adam	Adam
Loss Function	CrossEntropyLoss	CrossEntropyLoss

Table 3.5: Comparison of Transfer Learning (Fine-Tuning) Choices for VGG16 and MobileNetV3

3.4.2 3-Channels Mapping

Transfer learning is not the only strategy to enhance model performance. In fact, particularly when fine-tuning, it can be both computationally expensive and time-consuming. To address this challenge while improving the VGG16 model's performance, a novel approach was introduced to increase its flexibility. This approach involved converting the extracted 2D feature images into 3D representations by mapping each pixel to an RGB value. This additional processing step, applied before feeding the data into the extractor, allowed the model to capture more complex patterns and interactions within the features. The process is illustrated in Figure 3.2.

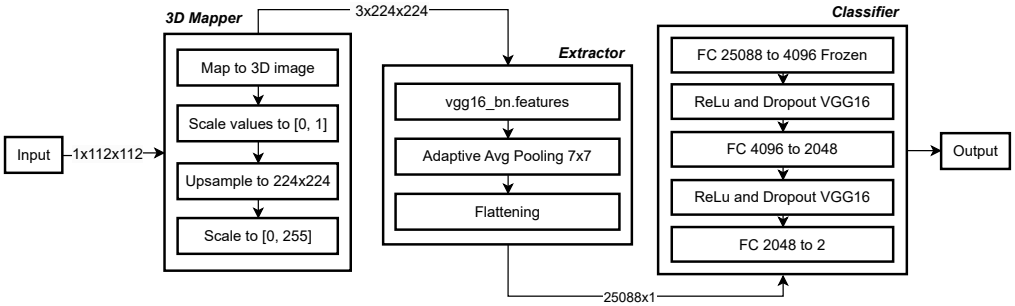


Figure 3.2: 3D Mapper Implementation Schema

The 3D Mapper was implemented using two distinct methods:

- Convolutional Neural Network (CNN)
- Look-Up Table (LUT)

The CNN-based approach, as shown in Listing 3.4, integrated a convolutional module within the model, responsible for converting the 2D feature image into a 3D image suitable for the classifier. Several configurations were explored, varying in the number of convolutional layers, kernel sizes, and scaling methods for the output, as detailed in Table 3.6. The scaling methods included: no scaling (*64-128-3-none*), applying a sigmoid function to scale the values between 0 and 1, and learning the scaling values with a subsequent sigmoid to ensure output within the $[0, 1]$ range (*64-128-3-learnable*). In all cases, the final output was rescaled to the standard image range of $[0, 255]$.

```
1    class Mapper3D(nn.Module):
2        def __init__(self):
3            super(Mapper3D, self).__init__()
```

```
            self.conv1 = nn.Conv2d(in_channels=1,
    out_channels=3, kernel_size=1)
            self.bn1 = nn.BatchNorm2d(3)
            self.upsample = nn.Upsample(scale_factor=2,
    mode='bilinear', align_corners=True)

        def forward(self, x): # Input: (B, 1, 112, 112)
            x = F.relu(self.bn1(self.conv1(x))) # (B, 3,
    112, 112)
            x = torch.sigmoid(x) # scale to [0, 1]
            x = self.upsample(x) # (B, 3, 224, 224)
            x = x * 255.0 # scale to [0, 255]
            return x
```

Listing 3.4: 3D Mapper CNN Implementation Example

The LUT-based approach, depicted in Figure 3.3, offers a simpler yet more computationally efficient alternative. This method directly maps the 2D image to a 3D image using a lookup table, requiring less training time but offering less flexibility compared to the CNN approach. The key variables in this method include the number of quantization levels, the α parameter, and the initialization method (random or linear). These parameters were adjusted to evaluate their impact on model performance.

Each version of the 3D Mapper was integrated into the VGG16 model with the feature extractor, and the initial layer of the classifier kept frozen (Listing 3.3). Only the 3D Mapper and the final layers of the MLP, were left trainable. The model was trained over ten epochs, using MFCC and CQT features, as they largely outperformed other feature types in prior experiments. The Adam optimizer and CrossEntropyLoss were used, with a learning rate of 0.001, and early stopping was triggered if no improvement was observed after five consecutive epochs.

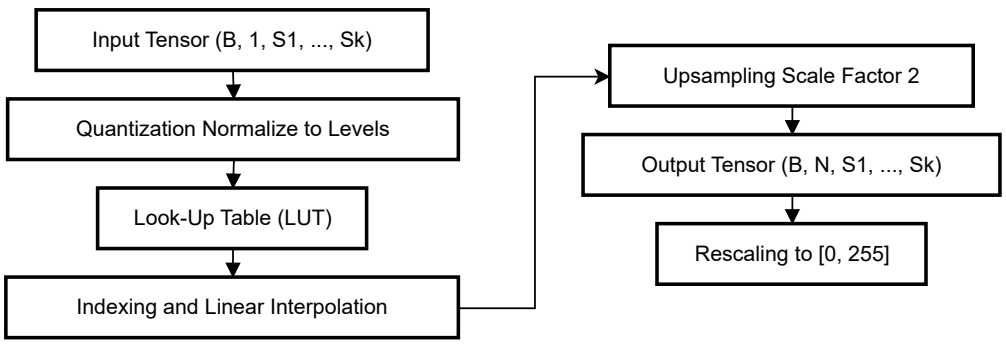


Figure 3.3: 3D Mapper Look-Up Table Schema

Version Name	Mapping	Architecture
64-128-3-sigmoid	CNN	Appendix Listing B.1
64-128-3-learnable	CNN	Appendix Listing B.2
64-128-3-none	CNN	Appendix Listing B.3
64x5-3x1-sigmoid	CNN	Appendix Listing B.4
3-1-sigmoid	CNN	Appendix Listing B.5
5-3-random	LUT	$Levels = 5, \alpha = 3, Init = Random$
10-3-random	LUT	$Levels = 10, \alpha = 3, Init = Random$
20-3-random	LUT	$Levels = 20, \alpha = 3, Init = Random$
40-3-random	LUT	$Levels = 40, \alpha = 3, Init = Random$
10-3-linear	LUT	$Levels = 10, \alpha = 3, Init = Linear$
10-1.5-linear	LUT	$Levels = 10, \alpha = 1.5, Init = Linear$

Table 3.6: 3D Mapper Architectures

3.4.3 Combining Fine-Tuning and Mapping

Both fine-tuning and mapping demonstrated significant but distinct improvements in model performance: fine-tuning was particularly effective with MFCC features, while mapping yielded better results with CQT features. To further enhance performance, a combined approach that integrates these two methodologies was explored.

In this combined approach, the four mapper models described in Table 3.6 were subjected to fine-tuning. The fine-tuning process was conducted as detailed in Section 3.4.1. Specifically, the trainable layers of the end-to-end (E2E) model included the mapper, the last two layers of the feature extractor, and the last two layers of the classifier. The model was trained for 25 epochs, with early stopping applied if no improvement was observed for five consecutive epochs. The Adam optimizer and CrossEntropyLoss were used, with a learning rate of 0.0001.

To obtain a 1D vector from the extractor's 3D output, two strategies were employed:

Flattening The first strategy focused on preserving as much information as possible by inserting a flattening layer before the classifier, as illustrated in Figure 3.2. This flattening layer reshaped the output from $(B, 512, 7, 7)$ to $(B, 25088, 1, 1)$, which was then fed into the classifier. While this approach retained all the information, it substantially increased the computational complexity, as the first fully connected (FC) layer of the classifier had to process a much larger input vector. To mitigate the impact on training time, a technique applied in this project was to freeze the entire first classifier

layer. However, this approach did not reduce the model's memory footprint, therefore a pooling-based method was explored as an alternative.

Parameters Reduction The second strategy aimed to reduce computational complexity while preserving performance by replacing the flattening layer with a pooling layer, as depicted in Figure 3.4. Two pooling techniques were tested:

- Global Average Pooling: This approach averaged the entire feature map, reducing the output from $(B, 512, 7, 7)$ to $(B, 512, 1, 1)$. While this drastically decreased the number of parameters, it discarded frequency information, which could potentially hinder model performance.
- Column Average Pooling: To retain frequency information, column average pooling was applied, where the average was taken across columns, reducing the output from $(B, 512, 7, 7)$ to $(B, 512, 7, 1)$. The output was then flattened to obtain the final 1D vector. This method struck a balance between preserving important frequency data and reducing the model's parameter count.

The parameter reduction achieved through these pooling methods is summarized in Table 3.7, which details the versions tested, the number of parameters involved, and the percentage reduction achieved.

Version	Number of Parameters		Percentage Reduction	
	Total	Trainable	Total	Trainable
Flattening	125 884 427	20 200 907	-	-
Global Avg Pooling	14 988 811	12 069 835	88.1%	40.2%
Column Avg Pooling	16 561 675	13 642 699	86.8%	32.5%

Table 3.7: Parameter Reduction through Pooling Techniques

3.4.4 Combining Features

The combination of MFCC and CQT features was explored to harness the complementary strengths of these two feature types. Indeed, while MFCC features demonstrated strong learning capabilities, especially in terms of capturing detailed nuances in the audio signals, CQT features excelled in generalization, particularly in handling diverse and unseen data.

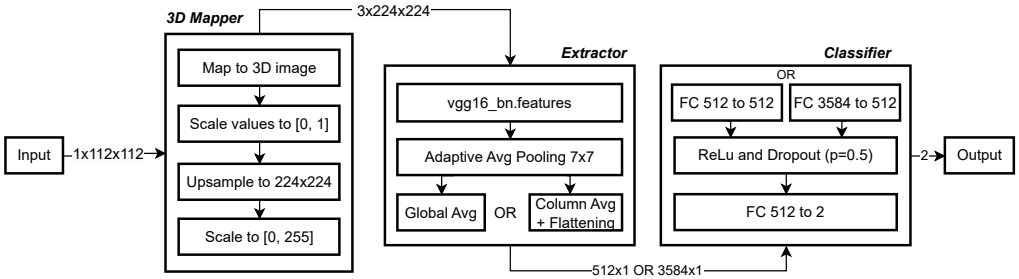


Figure 3.4: 3D Mapper Schema with Parameters Reduction

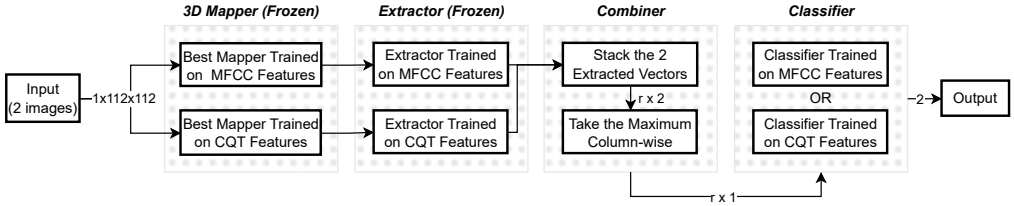


Figure 3.5: Ensemble DL Model Schema

Several methods exist for combining multiple features, such as concatenation, averaging, stacking, or employing ensemble methods. Given that we already had well-calibrated models for each feature type, the ensemble method was selected. Specifically, the ensemble model combined the outputs from the MFCC and CQT models by taking the maximum value of their respective extractor outputs, effectively integrating the strengths of both features at the model level.

The ensemble methodology is illustrated in Figure 3.5, where the outputs of the two extractors (one trained on MFCC and the other on CQT) are combined by taking the maximum value column-wise. This approach preserves the individual strengths of each feature set while creating a more robust combined model.

Following the same strategies outlined in Section 3.4.3, two techniques were employed to convert the 3D output of the extractor into a 1D vector: Flattening and Pooling for parameters reduction. For each scenario, the best mapping version for each feature type was selected (i.e. models leveraging both 3D Mapping and Fine-Tuning), and three model types were tested:

1. Model A: Utilized the best-performing models for each feature type (mapper and extractor), with the classifier trained solely on MFCC features.

2. Model B: Similar to Model A, but with a classifier trained solely on CQT features.

3. Model C: This model was trained on the newly combined features, starting from the weights of the previous models, aiming to further refine the ensemble's performance.

It is worth noting that Model A and Model B were not retrained on the combined features. Specifically for Model A the classifier of the best MFCC based model from previous experiments was used, with its learned weights. The same was done for Model B, using the best CQT based model. Model C was trained for 35 epochs, with an early stopping patience of ten epochs, using the Adam optimizer and CrossEntropyLoss, with a learning rate of 0.0001.

3.4.5 Final Model

Based on the previous sections, the best performing model was selected for the final evaluation. Given that earlier evaluations were conducted on small subsets of the data (i.e., test and validation sets), the final models were trained and evaluated on the full dataset to provide a more reliable estimate of their performance.

Three versions of the final model were proposed, each following the architecture depicted in Figure 3.5 but differing in their training strategies:

- **DeepSpectraNet**: In this version, the submodels handling MFCC and CQT features were trained separately. The final model was formed by combining these submodels with the `torch.max` operation on the extractor's output, and only the final classifier was trained end-to-end (E2E), with the first fully connected (FC) layer and the rest of the model frozen.

- **DeepSpectraNetLite**: This model is similar to DeepSpectraNet but utilizes global average pooling instead of flattening to reduce the number of parameters. This approach was aimed at creating a lighter version of the model with reduced computational complexity.

- **DeepSpectraNetFlex**: Starting from the weights of the flattening-based submodels trained separately on a data subset, this version was trained E2E on the full dataset. The only frozen layers were the first two convolutional layers of the submodels extractors and the first fully connected layer of the classifier.

3.4.6 Fully E2E Version

One of the limitations of the previous models, is their reliance on pre-determined audio features such as Mel-spectrograms, MFCCs, and CQT. This makes those models highly task-specific, as different audio-related problems may require different types of features. To address this issue and craft a more generalizable solution, was developed a fully end-to-end (E2E) model that can autonomously extract relevant audio features directly from raw waveform data. This allows the model to be used across a broader range of audio data without the need for hand crafted feature engineering. To implement this end-to-end approach, a new preprocessing module was embedded into the *DeepSpectraNetFlex* model. This module is designed to take raw audio as input and produce two different 112x112 images, which are then processed by the existing model (as shown in Figure 3.5). The new module leverages the outcomes gained from previous experiments, thus it focuses on capturing frequency-based representations of the audio signal and consists of three main blocks (Figure 3.6):

Signal Preprocessing Block This block is responsible for converting the raw audio waveform into a time-frequency representation. It achieves this using a Short-Time Fourier Transform (STFT) with a Hann window. Two different STFT operations are performed with varying window and hop lengths to generate two complementary images. One image focuses on shorter time frames, capturing fine temporal details, while the other emphasizes longer time frames to capture broader frequency information.

Convolutional Block The time-frequency representations generated in the previous block are further processed by a convolutional block. This block enhances the images by selectively focusing on the most important regions of the spectrogram. Each image is passed through three convolutional layers, with 32, 64, and 128 filters, respectively. Each convolutional layer is followed by batch normalization, a ReLU activation, and a Channel and Spatial Attention Module (CSAM).

The CSAM enables the model to learn which features are most critical by applying both channel and spatial attention. Channel attention learns a weighting vector that scales each filter's contribution, while spatial attention applies a similar weighting mechanism across the pixels of the image. These weights are computed using the mean and maximum values of the

activations and are learned during end-to-end training.

Shape Adaptation Block The shape adaptation block ensures that the output images from the convolutional block match the dimensions expected by the downstream modules in the original *DeepSpectraNetFlex* model. This allows the two newly generated images to be seamlessly integrated into the pre-existing architecture.

Training DeepSpectraNetE2E The fully end-to-end model, named *DeepSpectraNetE2E*, was trained for 30 epochs with an early stopping criterion based on validation loss, using a patience of seven epochs. The Adam optimizer was employed, with a learning rate of 0.0001 and a weight decay of 0.00001. The same frozen layers as *DeepSpectraNetFlex* were applied, including the first two convolutional layers of the submodels' extractors and the first fully connected layer of the classifier. CrossEntropyLoss was used as the loss function to optimize classification performance.

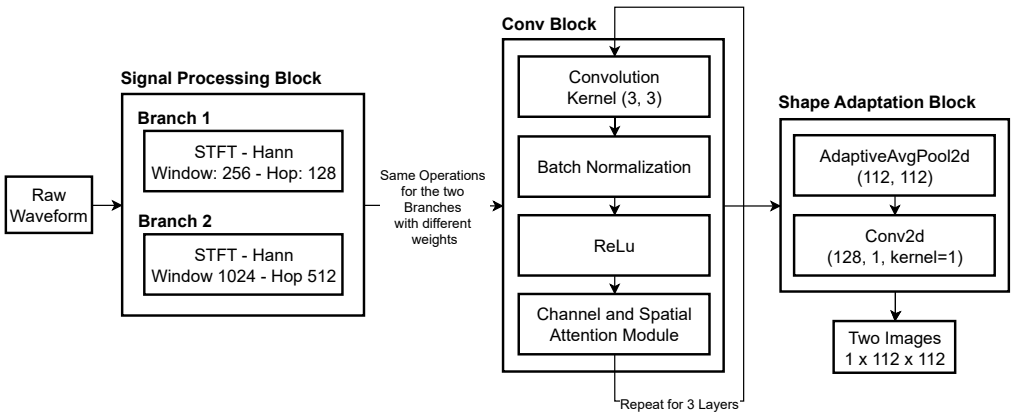


Figure 3.6: Fully E2E Model Additional Preprocessing Module

Chapter 4

Results and Explainability

4.1 Experimental Setup

In this section, we outline the experimental setup used for training and evaluating the models. Two primary environments were employed: a local machine for initial testing and development, and the Lightning AI Cloud [1] for large-scale, final model training and optimization.

4.1.1 Local Machine

The local machine used for the experiments is a Apple MacBook Pro (M1 Pro) [2]. It features an Apple M1 Pro chip, with an 8-core CPU that includes 6 performance cores and 2 efficiency cores. The machine is equipped with a 14-core GPU and a 16-core Neural Engine, providing up to 200GB/s memory bandwidth and 16GB unified memory.
The programming language used for the experiments is Python 3.11.5 [40]. To manage the environment and dependencies, a virtual environment running Python 3.11.5 was used. The primary libraries employed include:

- Machine Learning Models: Scikit-learn
- Metrics: Scikit-learn Metrics, pyeer
- Statistics: Scipy
- Neural Networks: PyTorch
- Audio Processing: Torchaudio, Librosa
- Data Handling: Pandas, Numpy
- Visualization: Matplotlib, Seaborn

[1] https://lightning.ai
[2] https://support.apple.com/en-md/111902

- Pretrained Models: torchvision models, including `vgg16_bn(weights='IMAGENET1K_V1')`, `mobilenet_v3_small(weights='IMAGENET1K_V1')`

4.1.2 Lightning AI Cloud

For more demanding training, the Lightning AI cloud was utilized, leveraging the power of NVIDIA L4 GPUs. The virtual environment on the cloud machine also ran Python 3.11.5 with similar library setups as the local machine, ensuring consistent execution of models across both platforms.

4.2 Evaluation Metrics

During the experiments, the primary metrics used to evaluate the performance of the models were Balanced Accuracy and F1 Score, with a greater emphasis on balanced accuracy in situations where the choice between the two metrics was not obvious. These metrics were selected because they are well-suited for handling imbalanced data, which was a critical aspect of this project. For the final evaluation, several additional metrics were considered to provide a comprehensive assessment of the models' performance.

Two main categories of metrics are discussed in this section, with a focus on binary classification problems:

- **Threshold-dependent metrics**: These metrics are calculated at a specific threshold setting and include Accuracy, Balanced Accuracy, and F1 Score. They depend directly on the classification threshold used to determine the final output class.
- **Threshold-independent metrics**: These metrics, including PR AUC, ROC AUC, and EER, evaluate the model's performance across a range of threshold settings. They provide a measure of the model's ability to discriminate between the classes irrespective of the specific decision threshold. This category of metrics is especially useful for evaluating the performance of binary classifiers at various levels of decision-making rigor.

The threshold-dependent metrics can be derived from the confusion matrix. In a binary classification problem, the confusion matrix consists of four values:

- **True Positive (TP)**: The model correctly predicts the positive class.

- **True Negative (TN)**: The model correctly predicts the negative class.
- **False Positive (FP)**: The model incorrectly predicts the positive class when the true class is negative (also known as a "Type I error").
- **False Negative (FN)**: The model incorrectly predicts the negative class when the true class is positive (also known as a "Type II error").

Scenario	P	N	TP	FP	TN	FN	Acc.	Bal. Acc.	F1 Score
A	900	100	890	50	50	10	94.0 %	74.5 %	96.5 %
B	100	900	50	10	890	50	94.0 %	74.5 %	64.1 %

Table 4.1: Different Class Imbalance Scenarios for Metrics Comparison

4.2.1 Accuracy

Accuracy is used in classification tasks to measure the proportion of correct predictions made by the model.

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

This metric represents the fraction of correctly classified instances (both true positives and true negatives) over the total number of instances in the dataset. In other words, it quantifies how often the model is correct, regardless of the class.

While it offers a general overview of model performance, its reliability reduces in imbalanced datasets, where one class largely outnumbers the other. In such cases, a model could achieve high accuracy by predominantly predicting the majority class while neglecting the minority class entirely. An example is provided in Table 4.1. In scenario A, the model is equivalent to a random classifier in the negative class, but it achieves a high accuracy of 94% due to the large number of positive samples correctly classified. The same holds for scenario B, where the model is equivalent to a random classifier in the positive class. Therefore, accuracy must be evaluated alongside more class-sensitive metrics to ensure a fair assessment of model performance across both classes.

4.2.2 Balanced Accuracy

Balanced accuracy addresses the limitations of standard accuracy, ensuring that both classes contribute equally to the evaluation.

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

Balanced accuracy is the average of the true positive rate (sensitivity) and true negative rate (specificity). This allows the metric to capture the model's performance across both classes without being biased towards the majority class. Each class's contribution to the final result is equal, even if the dataset is skewed.

Looking at the scenarios in Table 4.1, the balanced accuracy for both scenarios is 74.5%, which is lower than the accuracy metric. This shows that the balanced accuracy provides a more accurate representation of the model's performance in imbalanced datasets.

4.2.3 F1 Score

The F1 Score quantifies test accuracy by harmonizing precision and recall. Precision is defined as the ratio of correctly predicted positive observations to the total predicted positives, while recall represents the ratio of correctly predicted positive observations to all observations in the actual class.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{where Precision} = \frac{TP}{TP+FP} \quad \text{and Recall} = \frac{TP}{TP+FN}$$

The F1 Score, being the harmonic mean of precision and recall, offers a more balanced measure of a model's accuracy, particularly useful in scenarios with imbalanced datasets. This metric is sensitive to the performance of the positive class, highlighting its utility in fields where missing positive cases (such as fraud or disease) carries a higher risk than missing negative cases. Therefore, its usage must be carefully considered based on the problem at hand.

For instance, in Table 4.1, Scenario A shows an F1 Score of 96.5%, indicating a very strong classifier, despite a weak performance on the negative

class. Conversely Scenario B has an F1 Score of 64.1%, suggesting a weaker classifier, despite the positive class performance is the same as the negative class in Scenario A, where the F1 Score suggested a strong classifier. This reflects the F1 Score's sensitivity to the positive class, which is instead balanced in the Balanced Accuracy metric, with this latter returning the same value for both scenarios.

In summary, the F1 Score is indeed a valuable metric in scenarios where false negatives are more critical than false positives, however if it is not the case, balanced accuracy should be considered for a more uniform assessment across classes, particularly when equal importance is assigned to both positive and negative classifications. For this reason the balanced accuracy was chosen as the primary metric for model evaluation in this project.

4.2.4 PR AUC

The Precision-Recall Area Under the Curve (PR AUC) reflects the relationship between precision and recall for different probability thresholds. Thereby it provides a comprehensive evaluation of the model's performance.

The PR AUC is derived from the Precision-Recall curve, which plots the precision (y-axis) against the recall (x-axis) for every possible cutoff. The area under this curve represents the model's ability to correctly classify the positive class across different threshold settings. A higher area indicates better performance, with a perfect score of 1 indicating that the classifier can achieve high precision without sacrificing recall.

PR AUC provides a more informative measure in situations where positive class predictions are more critical, while it may be misleading in scenarios where both classes are equally important as it struggles to capture the negative class's performance. Thereby, as seen for F1 Score, it finds practical applications in fraud detection or disease screening, where missing a positive case could be detrimental.

4.2.5 ROC AUC

The Receiver Operating Characteristic Area Under the Curve (ROC AUC) evaluates a model's discriminative ability at various threshold settings by plotting the true positive rate (TPR) against the false positive rate (FPR).

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP + FN} = \text{Recall}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{TN + FP}$$

The ROC curve plots TPR against FPR, illustrating the trade-offs between true positive identifications and the false positive rate at which they occur. An AUC of 1.0 signifies a perfect model that discriminates perfectly between the classes, while an AUC of 0.5 indicates a model with no discriminative ability, equivalent to random guessing.

Conversely from PR AUC, which focuses on the positive class, ROC AUC provides a global view of the model's performance across both classes. This makes it suitable for situations where the costs of false positives and false negatives have similar importance. ROC AUC remains robust across class imbalances, making it a reliable metric even when positive and negative classes are not equally represented.

4.2.6 Equal Error Rate (EER)

Equal Error Rate (EER) is a metric commonly used to determine the threshold value where the false positive rate (FPR) and the false negative rate (FNR) are equal. In the context of binary classification, this metric is particularly useful in scenarios where the cost of false positives is equivalent to the cost of false negatives.

The EER can be derived from the point on the ROC curve where the line $y = 1-x$ intersects the ROC curve. This point means a balance between the false acceptance rate (FAR, synonymous with FPR) and the false rejection rate (FRR, synonymous with FNR). Mathematically, the EER is the specific value where:

$$\text{FPR} = \text{FNR} \quad \text{where} \quad \text{FNR} = 1 - \text{TPR}$$

EER is particularly useful in security systems like biometric verification, where it's crucial to balance between denying access to valid users and granting access to imposters.

By focusing on the point where FPR equals FNR, the EER provides a clear criterion for comparing different biometric systems or any classification system where decision thresholds are adjustable and equal importance is given to both types of errors.

4.3 Results

As explained in Section 3.1.1, the test set consists of synthetic data generated by a generative audio model that is not included in the training set. This allows us to use the test set results to evaluate the model's generalization capability. In contrast, the validation set results reflect the model's ability to learn from the data. Therefore, in the following section, we will consistently distinguish between the validation and test set results.

During the experiments, the primary metrics used to evaluate the performance of the models were Balanced Accuracy and F1 Score, with a greater emphasis on balanced accuracy in situations where the choice between the two metrics was not obvious. The reason behind this choice is tackled in Section 4.2, specifically in the subsection 4.2.3.

4.3.1 Traditional Machine Learning

In this section are presented the results of the approaches based on traditional machine learning models, from the feature engineering phase (encompassing the extraction interval, feature type, and hop and window length impact) to the final models.

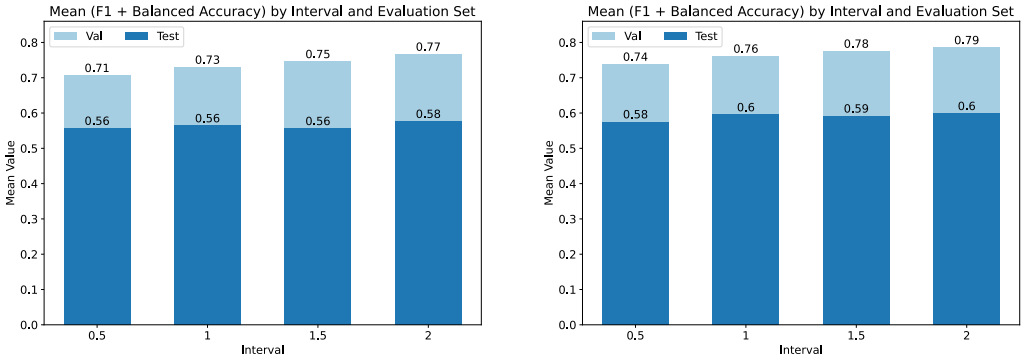
All results related to specific components of feature engineering are presented as averages. For completeness, the detailed results are available in the Appendix, specifically in Figures A.1 and A.2.

Extraction Interval

The extraction interval refers to the length of the audio segment from which features are extracted, consequently it directly influences the number of extracted data points (with longer intervals yielding fewer data points). Since in this phase only a subset of the data is used, the number of data points may have a large influence, therefore, models were tested under two scenarios:

- Equalized (Figure 4.1a): The number of data points was equalized across all intervals to enable an unbiased comparison. The number was chosen as the minimum number of data points across all intervals, 4000 in this case.

- Not Equalized (Figure 4.1b): The number of data points varied with the interval length, leading to fewer data points for shorter intervals.



(a) (b)

Figure 4.1: Avg Extraction Interval Impact Across Models: a) equalized num. samples across intervals, b) varying num. samples across intervals.

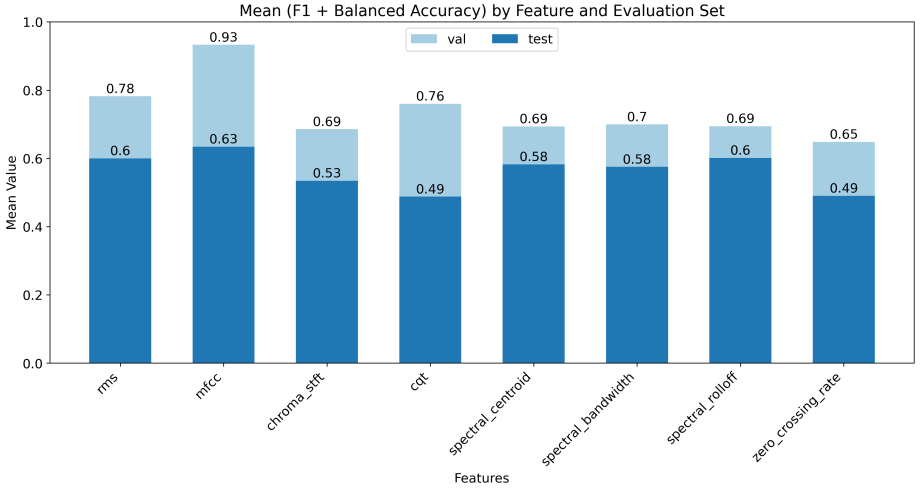
specifically 30000, 15000, 8000, and 4000 data points for intervals of 0.5, 1, 1.5, and 2 seconds respectively.

As expected, the non-equalized scenario exhibited slightly better performance. Interestingly, both scenarios demonstrated an increasing trend in performance as the extraction interval increased, particularly in the validation set. Unfortunately, it was not possible to evaluate larger intervals, as the majority of the fake test data was approximately 2 seconds long. Introducing padding to the shorter intervals would have introduced bias into the evaluation.

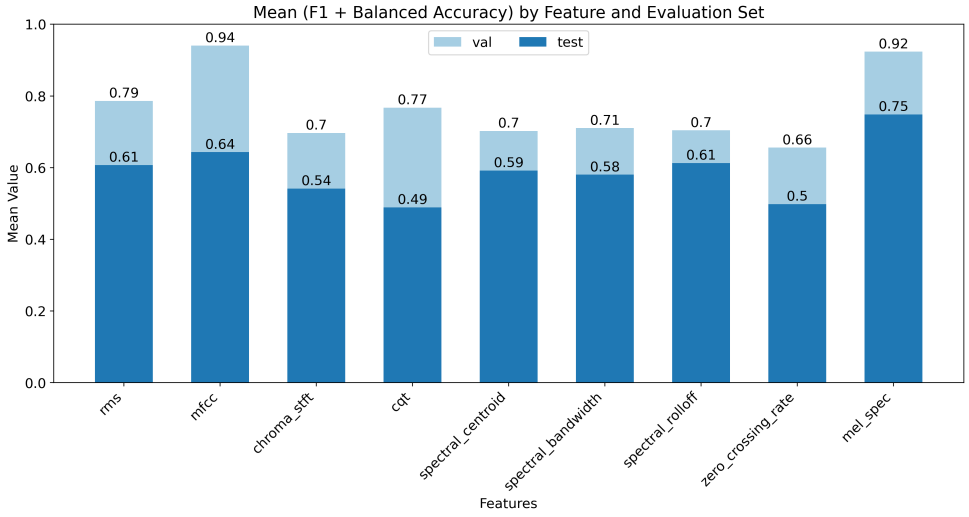
Based on these results, only two out of the four extraction intervals were retained for subsequent experiments on hop and window length: 2 seconds, as it demonstrated the best performance, and 1 second, to provide a more comprehensive analysis.

Feature Type

In Figure 4.2, the performance of various feature types is depicted, with Figure 4.2a illustrating the results when the number of samples is equalized across intervals, and Figure 4.2b showing results with a varying number of samples across intervals.



(a)



(b)

Figure 4.2: Avg Feature Type Performance Across Models: a) equalized num. samples across intervals , b) varying num. samples across intervals.

The initial observation is that the difference between the two scenarios is not substantial. The non-equalized variant exhibited only a slight improvement in the performance of all features. Consequently, we can focus on the non-equalized scenario, as the "data points augmentation" effect should be leveraged.

What clearly emerges is that the Mel spectrogram and MFCC features are the top performers, with MFCC showing a slight edge in learning the data and the Mel spectrogram significantly emerging in generalization. RMS and CQT also showed potential, although they did not perform as well as the Mel spectrogram and MFCC. The remaining features demonstrated uniformly poor performance.

For this reason and due to the computational complexity of the models, only the Mel spectrogram and MFCC features were retained for subsequent experiments on the hop and window length.

Hop and Window Length

The impact of the hop and window lengths was evaluated for the Mel spectrogram and MFCC features, considering 1-second and 2-second extraction intervals, and using Random Forest (RF) and Support Vector Machine (SVM) models as classifiers. The RF model was chosen as it demonstrated the best performance in previous experiments, while the SVM was selected for being a well-known traditional, non-ensemble model. The results are illustrated in Figures 4.3a and 4.3b.

The impact of hop and window lengths was not significant, as different models and features had varying optimal values. For the Random Forest results, it is interesting to note that MFCCs seem to benefit from shorter extraction intervals and window lengths. The reasons for this are not entirely clear, but it is possible that MFCCs, which are more focused on high frequencies, benefit from higher temporal resolution. About the hop length effect, the results showed a minimal impact, almost null in the case of SVM.

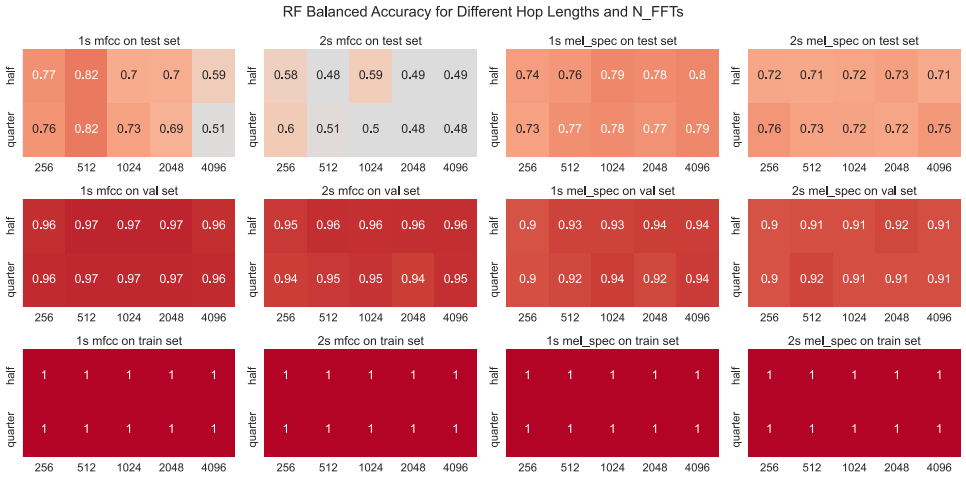
Generally, a 1-second extraction interval appeared to provide better generalization on the test set, but the SVM showed reduced performance on the validation set.

At the moment the goal is to find the best possible feature setting to learn the data, therefore the the 2-second extraction interval was retained. This choice helped to reduce model complexity and computational time. To further enhance generalization on the test set, we attempted to combine different features and reduce them using Recursive Feature Elimination (RFE), as detailed in Section 3.3.2.

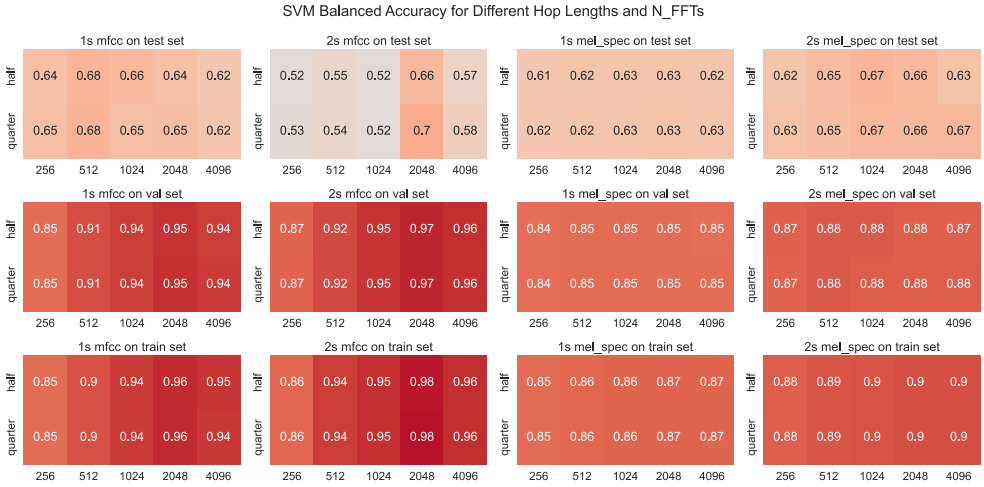
Models

In this section are presented the results of the experiments conducted to find the best traditional machine learning model. The results encompass the baseline models, the impact of Mel spectrogram features, the effectiveness of Recursive Feature Elimination (RFE), and the final model evaluation.

Baseline Models Table 4.2 presents the performance of the traditional ML baseline models. For each classifier, the best results have been ex-



(a)



(b)

Figure 4.3: Impact of hop (y-axis) and window length on: a) Random Forest performance, b) SVM performance.

tracted from Figure A.2, with separate considerations for scenarios focused on learning and generalization. The balanced accuracy metric, particularly relevant in cases of unbalanced data with a predominant positive class, highlights the Random Forest model using Mel spectrogram features and a 1.5-second extraction interval as the best for generalization. This model achieved a balanced accuracy of 77% on the test set, a notable result given the task's complexity. On the other hand, the Catboost model, also utilizing Mel spectrogram features and a 1.5-second extraction interval, excelled in learning-oriented tasks, achieving a balanced accuracy of 99% on the validation set. These two models were established as baselines for subsequent experiments.

Classifier	Feature	Interval (s)	Goal	F1 (%)		Bal_Acc (%)	
				Val	Test	Val	Test
RF	MFCC	2	Learning	99	83	98	59
	Mel Spec	1	Generalizing	98	85	96	77
SVM - RBF	MFCC	1	Learning	97	76	96	63
	Mel Spec	2	Generalizing	95	75	91	75
Catboost	Mel Spec	1.5	Learning	99	86	99	68
	MFCC	0.5	Generalizing	97	79	95	73
LR	MFCC	2	Learning	90	55	84	44
	Mel Spec	2	Generalizing	91	79	81	80

Table 4.2: Traditional ML baseline results

To increase models' performance, various feature types were combined, to leverage their different strengths. The results in Table 4.3 provide a comprehensive view of the performance differences between models trained on the full feature set and those with features reduced via Recursive Feature Elimination (RFE). The comparison between validation (Val) and test set results sheds light on the models' capabilities to learn from the training data and generalize to new, unseen data, respectively.

Impact of Mel Spectrogram Features Considered stand-alone, the Mel spectrogram features were the best for generalization, but when combined with other features, they behaved oppositely. Including Mel spectrogram features generally resulted in high validation performance across all classifiers but showed varied results in generalization. For instance, the Random Forest (RF) classifier achieved a balanced accuracy (Bal_Acc) of 77% on the test set with Mel spectrograms included, which was outperformed (85%) when they were excluded. This pattern is consistent across the XGBoost and Catboost models, where excluding Mel spectrograms led to better generalization on the test set, particularly for XGBoost and Catboost, which saw increases of 16% and 13% in F1 scores, respectively.

This suggests that while Mel spectrograms can enhance the model's ability to learn from training data, they may introduce redundancy or correlated information that hinders the model's ability to generalize to unseen data. This observation is further supported by the RFE analysis, which indicates that when Mel spectrogram features were included, the retained features were predominantly MFCC and Mel spectrograms—potentially leading to

overfitting due to the overlap in information.

Classifier	Feat. per Type	Feat after RFE	Mel Spec	F1 (%)		Bal_Acc (%)	
				Val	Test	Val	Test
RF	20	Not Applied	Included	97	78	97	77
	20	Not Applied	Excluded	94	78	95	85
	20	40	Included	98	78	97	76
	20	30	Excluded	96	92	96	92
XGBoost	20	Not Applied	Included	99	72	99	71
	40	Not Applied	Excluded	98	89	98	88
	20	40	Included	99	79	99	77
	20	40	Excluded	98	89	98	87
Catboost	20	Not Applied	Included	99	76	99	74
	40	Not Applied	Excluded	99	87	99	85
	20	80	Included	99	76	99	74
	40	80	Excluded	98	89	98	87

Table 4.3: Traditional ML Results Combining the Features and Reducing them Using RFE

Effectiveness of Recursive Feature Elimination (RFE) RFE's effectiveness in improving model performance is evident, especially when Mel spectrogram features are excluded. For instance, RF with 30 features retained through RFE and without Mel spectrogram features achieved the highest test set performance, with an F1 score and balanced accuracy both at 92%. Similarly, for XGBoost and Catboost, using RFE to select a reduced feature set without Mel spectrograms consistently improved test set performance, achieving balanced accuracy scores up to 89% and 87%, respectively.

The RFE process appears to effectively mitigate overfitting by focusing the model on the most informative and non-redundant features, thus enhancing generalization capabilities. The models' ability to retain high validation performance while improving test set performance post-RFE demonstrates that this method is well-suited for the feature selection task in this context.

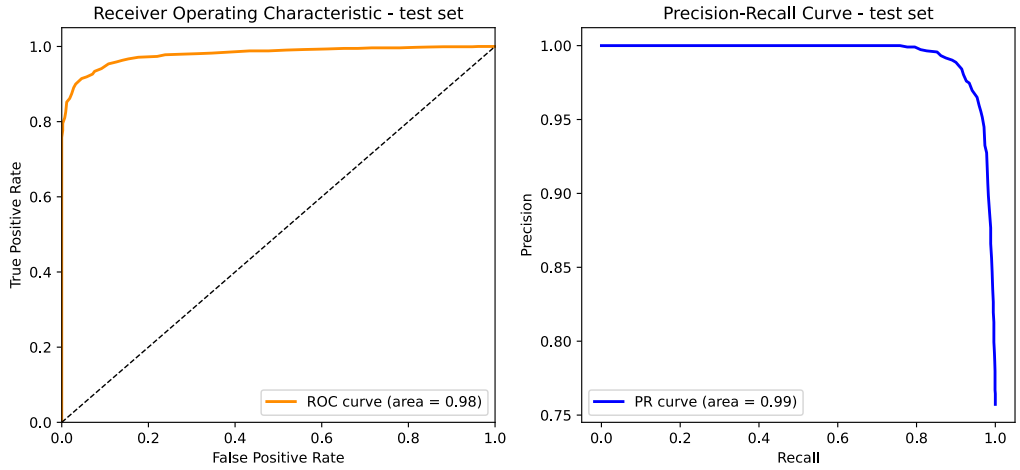


Figure 4.4: Final RF Model Curves for Test Set

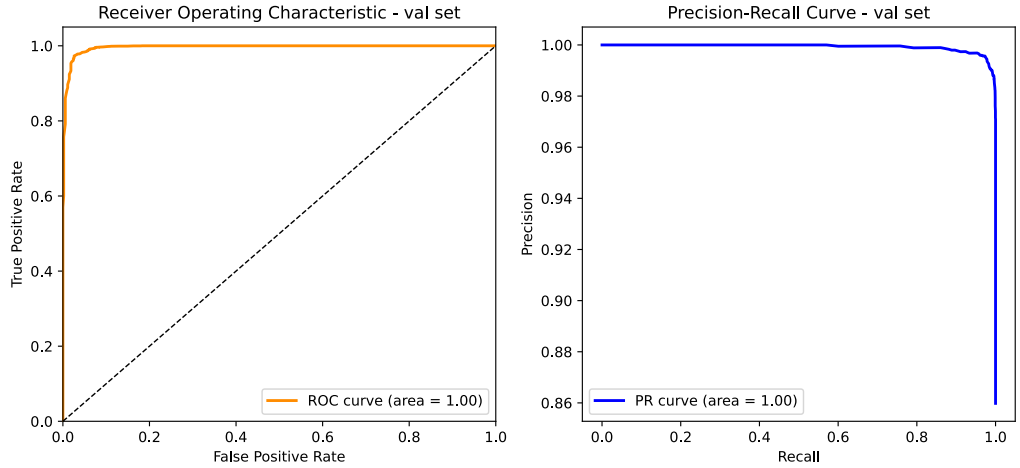


Figure 4.5: Final RF Model Curves for Validation Set

Final Model The final Random Forest (RF) model was thoroughly valuated to compare its performance against baseline models and existing state-of-the-art methods.

The ROC and PR curves (Figures 4.5 and 4.4) provide insights into the model's discriminative power and precision across different thresholds. On the validation set, the final RF model achieved an area under the ROC curve (AUC) of 1.00, and the PR AUC was also 1.00, indicating excellent learning capabilities and almost perfect precision-recall tradeoff. The test set results were similarly robust, with an ROC AUC of 0.98 and a PR AUC of 0.99, demonstrating strong generalization to unseen data, despite a slight drop in performance compared to the validation set.

The confusion matrix, presented in Figure 4.6, further highlights the

model's accuracy. The final RF model showed a strong ability to correctly classify both positive and negative classes, reflecting its balanced performance and low misclassification rates on the validation set. On the test set, most misclassifications were false negatives, indicating that the model was more likely to incorrectly classify real audio as fake.

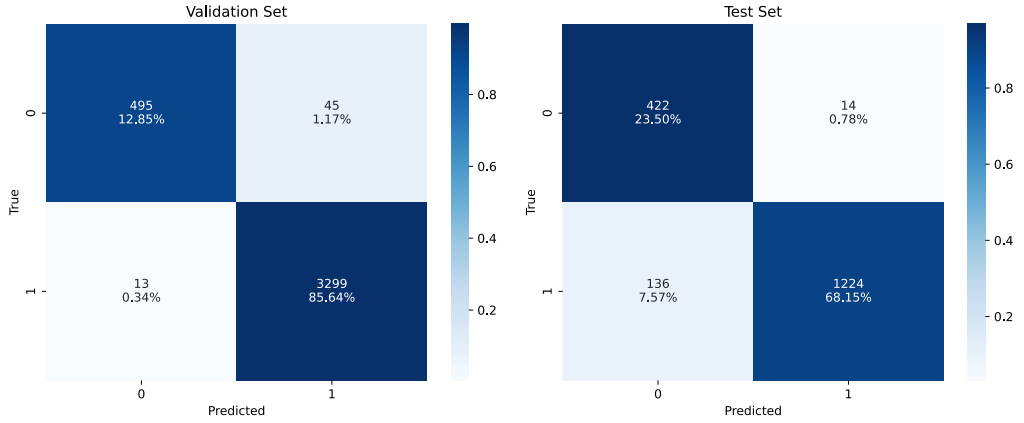


Figure 4.6: Final RF Model Confusion Matrix

Classifier	Acc		F1		Bal Acc		PR AUC		ROC AUC		EER	
	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
Existing Models												
SVM [24]	0.85	0.67	-	-	-	-	-	-	-	-	-	-
RF [24]	0.80	0.62	-	-	-	-	-	-	-	-	-	-
KNN [24]	0.75	0.62	-	-	-	-	-	-	-	-	-	-
XGBoost [24]	0.70	0.59	-	-	-	-	-	-	-	-	-	-
LGBM [24]	0.75	0.60	-	-	-	-	-	-	-	-	-	-
XGBoost [39]	-	0.93	-	-	-	-	-	-	-	-	-	-
RF [32]	0.79	0.72	-	-	-	-	-	-	-	-	-	-
DTrees [32]	0.77	0.70	-	-	-	-	-	-	-	-	-	-
Proposed Models												
RF-Table 4.2	-	-	0.98	0.85	0.96	0.77	-	-	-	-	-	-
RF-RFE	0.99	0.92	0.99	0.94	0.96	0.93	0.99	0.99	0.99	0.98	0.03	0.07

Table 4.4: Final RF Model Metrics Compared with Baseline Models and State-Of-The-Art ML

Table 4.4 contrasts the final RF model's performance with baseline models and other state-of-the-art machine learning techniques. All the metrics are calculated using the optimal binary threshold, defined as the one that maximizes the delta between true positive rate and false positive rate. For the test set the threshold is 0.58, while for the validation set is 0.42.

The proposed RF model with Recursive Feature Elimination (RFE) outperformed the baseline models significantly, particularly in generalization metrics, with a balanced accuracy of 0.93 on the test set, coupled with a 96% on the validation set and optimal EER values.

4.3.2 Deep Learning

In this section are presented the results of the approaches based on deep learning models, from the feature engineering phase to the final models.

Like for the traditional ML approach, even in this case the results concerning specific components of feature engineering are presented as averages. These averages include metrics, features, and classifiers for MobileNetV3, and metrics, features, and layers for VGG16. Detailed results are provided in the Appendix, specifically in Figures A.3 and A.4.

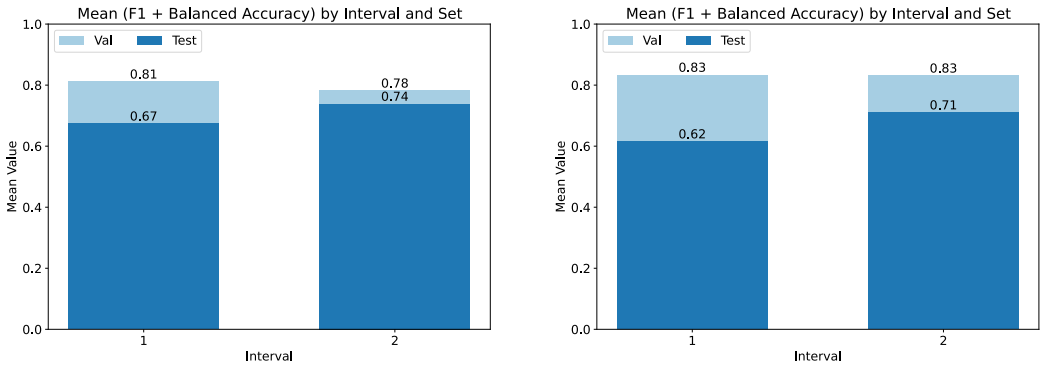
Furthermore it is important to note that due to the similarity of the information provided by the equalized and not equalized scenarios, only the latter will be adopted from this point on.

Extraction Interval

Figure 4.7 presents the results of the extraction interval's impact using the neural features-based approach. For VGG16 (Figure 4.7b), a peak validation performance was observed with a 1-second interval, which contrasts with the traditional ML approach where a longer interval was favored. However, the 2-second interval demonstrated superior generalization capabilities in this case. This trend is further supported by MobileNetV3 (Figure 4.7a), where the 2-second interval not only exhibited better performance on the test set but also matched the 1-second performance on the validation set. Based on these findings, the 2-second interval was selected as the optimal choice for feature extraction in the final models.

Feature Type

Comparing the performance of feature types in the "raw features" approach (Figure 4.2) with the neural features-based approach (Figure 4.8) reveals three key insights. First, the neural features-based approach generally enhances feature performance, improving both data learning and generalization to unseen audio generative algorithms. However, this improvement

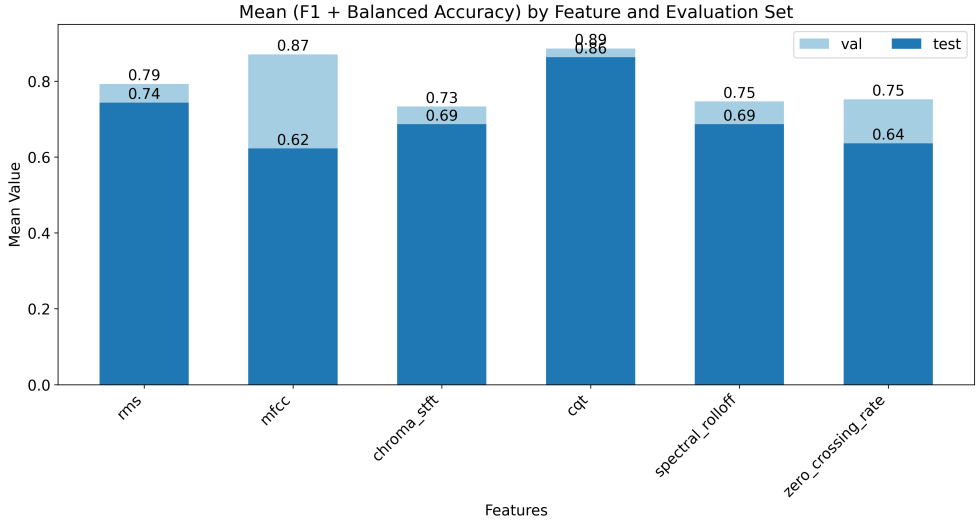


(a) (b)

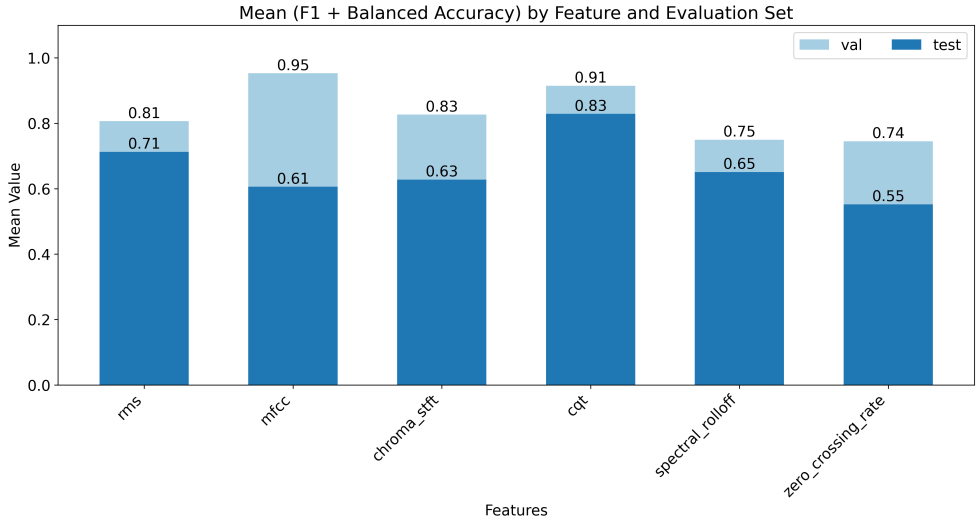
Figure 4.7: Extraction Interval Impact with Varying Num. Samples Across Intervals. Features Extracted Using: a) VGG16 with batch normalization, b) MobileNetV3.

does not extend to MFCC features. This limitation may stem from the distribution of information within the MFCC image representation. As shown in the top left image of Figure 4.9, potentially valuable information is concentrated in a small section at the top of the image, which the model might not always capture effectively. To address this, introducing a 3D-Mapping CNN, as detailed in Section 3.4.2, could provide the model with greater flexibility in learning a more effective data representation. Despite this, MFCC remains the best feature type for data learning.

The third point is that CQT features exhibit promising generalization capabilities, suggesting that a combination of MFCC and CQT features might offer the optimal balance between high data learning performance and strong generalization. The raw Mel Spectrogram was excluded from this comparison due to its inferior generalization compared to CQT and weaker data learning relative to MFCC, as seen in Figure 4.10.



(a)



(b)

Figure 4.8: Feature Type Performance with Varying Num. Samples Across Intervals. Features Extracted Using: a) VGG16 with batch normalization, b) MobileNetV3.

Hop and Window Length

To manage the computational demands of deep learning models, the impact of hop and window lengths was evaluated exclusively using MobileNetV3, which is less complex than other models. Instead of testing numerous combinations, the following were selected based on the findings of Küçükbay et al. [41]: (512, 256), (1024, 512), (2048, 512), and (4096, 1024). Their research demonstrated that a hop length of 50% of the window length generally yields good results, with a 25% hop length becoming more effective

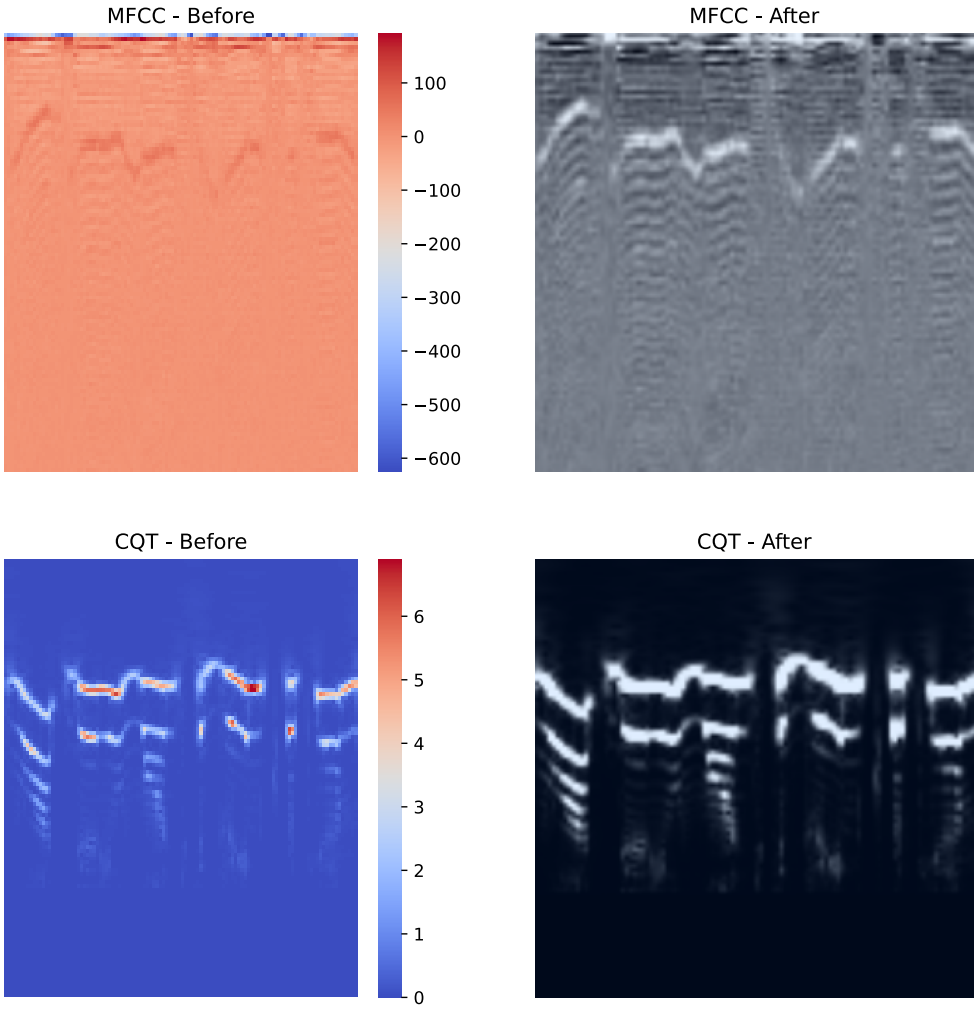


Figure 4.9: MFCC and CQT Features Before and After Preprocessing.

as the window length increases. It is important to note that CQT automatically adapt the window length to the frequency as illustrated in Section 2.3, therefore only the hop length was fixed.

Figure 4.10 shows that shorter window lengths tend to produce better results on both the validation and test sets, particularly for MFCC and CQT features. This suggests that higher temporal resolution, achieved with shorter window lengths, enhances performance by providing more temporal information. Consequently, final models utilized window and hop lengths close to (512, 256).

In conclusion, hop and window lengths significantly influence the performance of both ML and DL models in audio-related tasks, and their values should be carefully optimized for each specific application.

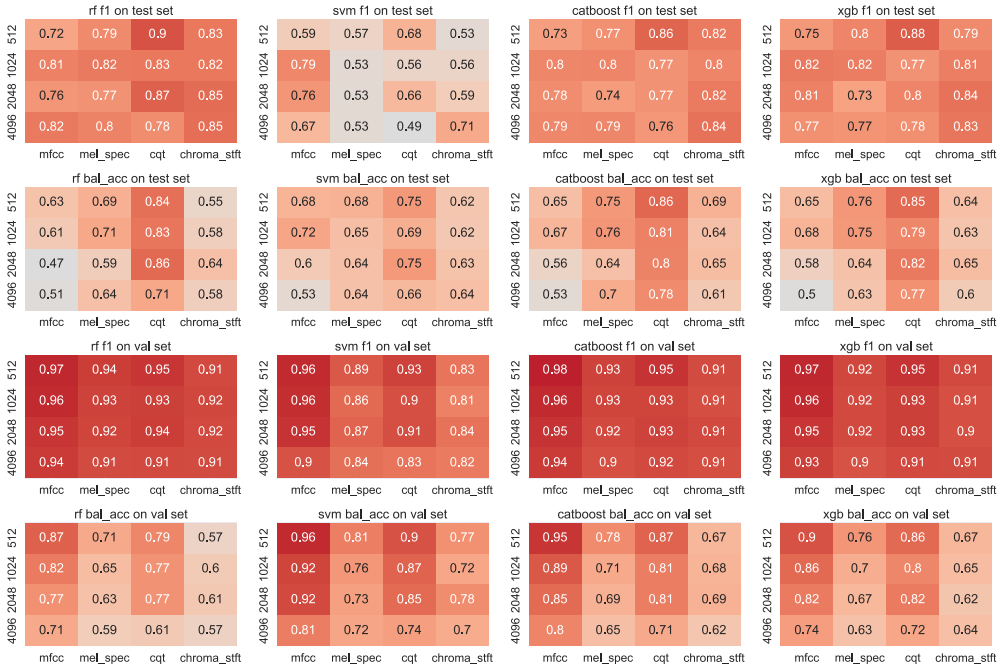


Figure 4.10: Impact of Hop and Window Length (Reported on y-axis) on MobileNetV3 Performance.

Models

In this section, we discuss the outcomes of the deep learning models from the transfer learning phase through to the final models, focusing on both learning and generalization capabilities.

Transfer Learning The transfer learning phase aimed to leverage the pre-trained VGG16 and MobileNetV3 models in two ways: by extracting features from different layers and by fine-tuning the models.

VGG16 Layer Impact From the analysis depicted in Figure 4.11, it is evident that layer 43 of the VGG16 model demonstrates the most consistent learning performance across the validation and test sets. However, as detailed in Figure A.4, the peak generalization performance, achieving approximately 93% accuracy, was observed in layer 23 when using CQT features. Interestingly, the best learning performance was achieved by layer 33. These peaks in performance might be attributed to specific characteristics of the dataset or the nature of the features being extracted, which could favor certain layers over others under particular circumstances.

Given that the primary objective in this phase is to establish a stable and balanced base model for further refinement (e.g., through 3D-Mapping

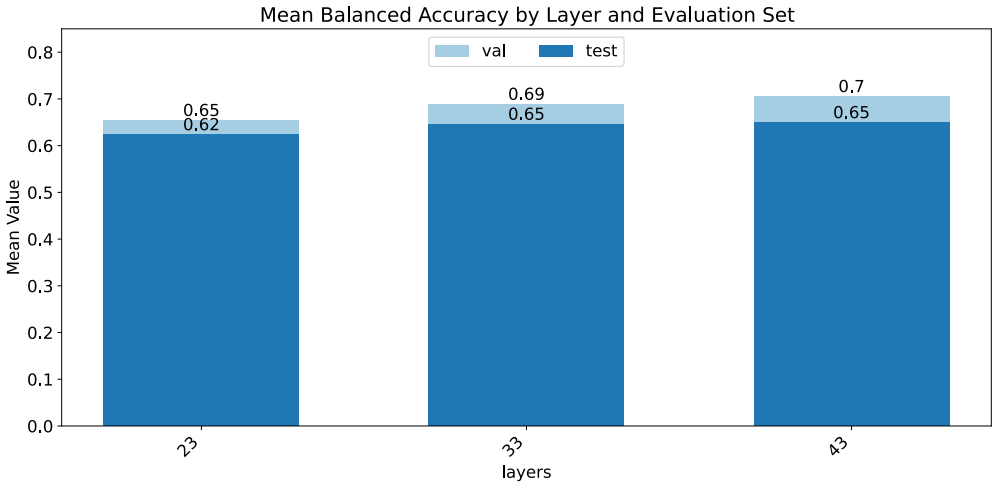


Figure 4.11: VGG16 Layers Impact on Performance.

Model	Classifier	Feature	Interval (s)	Goal	F1 (%)		Bal_Acc (%)	
					Val	Test	Val	Test
MNETv3	SVM	MFCC	2	Learning	98	60	96	70
	CatBoost	CQT	2	Generalizing	96	91	88	90
VGG16	RF	MFCC	2	Learning	97	80	83	68
	RF	CQT	2	Generalizing	96	91	79	85

Table 4.5: DL Transfer Learning (Feature Extraction) Results

or feature combination), layer 43 was selected as the optimal compromise between learning efficacy and generalization capability.

Feature Extraction Table 4.5 compares the performance of VGG16 and MobileNetV3 during the feature extraction phase. Although VGG16 was only paired with the Random Forest classifier, and MobileNetV3 was tested with various classifiers (including SVM, CatBoost, and XGBoost), MobileNetV3 demonstrated superior performance overall, especially when paired with CatBoost for generalization tasks.

Fine Tuning As discussed in Section 3.4.1, four different MLP classifiers were tested for each feature type within MobileNetV3. The best-performing classifier was selected for each feature: MLP3 for MFCC and RMS, MLP4 for Mel Spectrogram, and MLP1 for CQT. However, due to the significantly lower performance of the RMS and Mel Spectrogram features, they are excluded from the subsequent analysis.

Table 4.6 presents the results of fine-tuning the VGG16 and MobileNetV3

Model	Classifier	Feature	Epoch	Goal	Bal Acc (%)		
					Train	Val	Test
Feature Extractor							
MNETv3	SVM	MFCC	-	Learning	98	96	70
	CatBoost	CQT	-	Generalizing	100	88	90
Fine Tuning							
VGG16	RF	CQT	6	-	99.97	98.11	85.21
	RF	MFCC	7	-	99.68	98.95	89.41
MNETv3	RF	CQT	10	-	99.81	92.90	77.04
	RF	MFCC	11	-	99.42	97.18	70.63

Table 4.6: DL Transfer Learning (Fine Tuning) Results

models. Notably, when fine-tuned with a Random Forest classifier, the VGG16 model achieved higher balanced accuracy on the test set compared to MobileNetV3. This indicates that while MobileNetV3 serves as a strong baseline, VGG16 may deliver superior results with additional optimization.

Quantitatively, the fine-tuned VGG16 model showed a considerable enhancement in performance, particularly when leveraging MFCC features. This improvement highlighted the rich information contained within the MFCC features, which VGG16 effectively learned during the fine-tuning process. On the test set, the Balanced Accuracy (Bal Acc) surged by 21.41% compared to the feature extraction phase, reaching a remarkable 89.41%. This outcome, coupled with an impressive 98.95% Bal Acc on the validation set, underscored the effectiveness of the fine-tuning process in significantly boosting the model's overall performance. These results strongly position VGG16 as a leading candidate for the final model selection.

3-Channels Mapping The full results of the 3D mapping approach can be found in the Appendix (Table B.1), with a summary of the best results presented in Table 4.7.

The table highlights the top-performing configurations for each of the two mapping methods (i.e., CNN and LUT) across two feature types (i.e., MFCC and CQT), along with the epochs at which these results were achieved.

When comparing these results with those obtained through transfer learning (both feature extraction and fine-tuning), the following observations can be made:

Version Name	Mapping	Features	Epoch	Bal Acc (%)		
				Train	Val	Test
64-128-3-learnable	CNN	MFCC	10	97.4	97.5	65.2
64x5-3x1-sigmoid	CNN	CQT	8	94.5	90.7	90.1
10-1.5-linear	LUT	MFCC	10	98.2	96.9	72.1
10-3-linear	LUT	CQT	7	97.8	94.8	92.1

Table 4.7: Best Results of Different 3D Mapper Architectures with VGG16

- Model Flexibility and Training Accuracy: The increased model flexibility provided by the 3D mapping approach is reflected in the training accuracy, which is noticeably lower compared to the fine-tuning case. The reduction in training accuracy suggests that the model has access to more complex patterns, potentially avoiding overfitting and capturing more generalizable features.

- MFCC Performance: With the introduction of the mapping, the performance of MFCC features showed a significant improvement—4.1% on the test set and 13.9% on the validation set—compared to using the VGG16 model as a feature extractor (Table 4.5). It's important to note that different classifiers were used: an MLP in the 3D mapping approach and a Random Forest during the transfer learning phase. However, compared to fine-tuning results (Table 4.6), MFCC's performance with 3D mapping was less impressive.

- CQT Performance: The results for CQT features reveal a different trend. The improvement over the feature extraction method was substantial, but more notably, the generalization capability saw a significant boost. The test accuracy increased from 85.21% in fine-tuning to 92.1% with 3D mapping, demonstrating the potential of CQT features in this context.

In summary, the 3D mapping approach enhanced the overall model performance, particularly for CQT features, which exhibited exceptional generalization capabilities. Conversely, fine-tuning was more beneficial for MFCC features, significantly boosting performance compared to the feature extraction phase.

Given these promising results, the next section explores the combination of these two approaches to further enhance the model's performance.

Combining Fine Tuning and Mapping Table 4.8 presents the results of integrating fine-tuning with mapping techniques in the VGG16 model. This section explores the impact of this combination on model performance, particularly in terms of learning capabilities and generalization.

The results clearly show that adding fine-tuning to the 3D mapping approach led to significant improvements in the model's ability to learn. While this improvement in learning might have been expected, a notable finding is that generalization also improved in most cases.

Previously, the best-performing model in terms of learning was a fine-tuned model without any mapping, using CQT features. This model achieved a balanced accuracy of 98.95% on the validation set (Table 4.6). However, its performance dropped on the test set, with a balanced accuracy of 89.41%. On the other hand, the best model for generalization was one using 3D mapping with CQT features, which reached a balanced accuracy of 92.1% on the test set, while its validation performance was slightly lower at 94.8% (Table 4.7). These results highlighted a trade-off between learning and generalization, depending on the model used.

However, when fine-tuning was combined with mapping, the model's performance improved significantly on both the validation and test sets. The best model, using CQT features, surpassed the previous best CQT-based model by 4.1% on the test set, achieving a balanced accuracy of 96.2%. Similarly, the validation set performance increased by 3.2%, reaching a balanced accuracy of 98.0%. This improvement was also observed with MFCC features, where the combined approach resulted in a balanced accuracy of 91.9% on the test set and 98.0% on the validation set.

This indicates that combining fine-tuning with mapping is an effective strategy for enhancing the model's performance, not only in learning but also in generalizing to new unseen audio deepfake generation algorithm.

Impact of Parameter Reduction Methods The combination of fine-tuning and mapping significantly enhanced the model's performance, particularly in terms of generalization, as demonstrated in the previous analysis. The two top-performing models from this approach, highlighted in blue in Table 4.8, were selected to assess the impact of parameter reduction methods on model performance.

Table 4.9 presents the results of this analysis, comparing the performance of the two best models models (originally using flattening) with

Model Name	Fine Tuning	Mapping	Features	Epoch	Bal Acc (%)		
					Train	Val	Test
3-1-sigmoid	No	CNN	MFCC	6	96.8	92.6	76.1
3-1-sigmoid	Yes	CNN	MFCC	4	98.3	99.9	70.2
64x5-3x1-sigmoid	No	CNN	CQT	8	94.5	90.7	90.1
64x5-3x1-sigmoid	Yes	CNN	CQT	6	98.3	98.0	96.2
10-1.5-linear	No	LUT	MFCC	10	98.2	96.9	72.1
10-1.5-linear	Yes	LUT	MFCC	8	99.7	98.0	91.9
10-3-linear	No	LUT	CQT	7	97.8	94.8	92.1
10-3-linear	Yes	LUT	CQT	14	99.4	98.1	90.0

Table 4.8: Results of Combining Fine Tuning and Mapping with Different Architectures (VGG16)

alternative parameter reduction methods: **global average pooling** and **column-wise average pooling**. Surprisingly, despite the significant reduction in parameters achieved through these methods, the models' performance remained relatively stable. In some instances, performance even improved. For example, the model using CQT features with global average pooling achieved a balanced accuracy of 97.3% on the test set, representing a 1.1% improvement over the flattening-based model. Another notable improvement was observed in the model using MFCC features with column-wise average pooling, which reached a balanced accuracy of 99.0% on the validation set, surpassing the flattening-based model by 1.0%.

It's important to note that in the flattening-based method, the first fully connected layer was not trained to reduce complexity. This suggests that further performance gains could potentially be achieved by training this layer.

Among the parameter reduction methods, global average pooling proved to be more effective in reducing the number of parameters while maintaining performance. It performed comparably to column average pooling on MFCC features and outperformed it on CQT features. Consequently, only the global average pooling method was selected for the subsequent section, where both feature types were combined using an ensemble approach.

This analysis indicates that strategic parameter reduction methods, particularly global average pooling, can maintain or even enhance model performance while significantly reducing complexity, which is crucial for deploying efficient models in real-world applications.

Model Name	Fine Tuning	Features	Epoch	Version	Bal Acc (%)		
					Train	Val	Test
64x5-3x1-sigmoid	Yes	CQT	6	Flattening	98.3	98.0	96.2
64x5-3x1-sigmoid	Yes	CQT	8	Global	98.5	97.5	97.3
64x5-3x1-sigmoid	Yes	CQT	8	Column	97.9	98.5	94.0
10-1.5-linear	Yes	MFCC	8	Flattening	99.7	98.0	91.9
10-1.5-linear	Yes	MFCC	14	Global	99.8	98.8	90.2
10-1.5-linear	Yes	MFCC	8	Column	99.8	99.0	90.2

Table 4.9: Results of Fine Tuning and Mapping with Different Parameters Reduction Methods (VGG16)

Combining Features The results of the feature combination approach are presented in Table 4.10. This analysis investigates the impact of combining CQT and MFCC features on the model's performance and compares the performance of pre-trained classifiers with those trained specifically on the combined feature set.

Overall, the results demonstrate that combining features can lead to substantial improvements in model performance, also when utilizing pre-trained classifiers. The highest balanced accuracy on the test set was achieved by the model using the *Trained-classifier-weights* with flattening, reaching 98.70%. This performance is slightly superior to the *CQT-classifier-weights* with flattening, which achieved a test set accuracy of 98.68%. Both of these models also performed exceptionally well on the validation set, with balanced accuracies of 100% and 99.76%, respectively.

Interestingly, while the Global Average Pooling method performed well with individual features, it did not perform as effectively with the combined features. Specifically, the *Trained-classifier-weights* model using Global Average Pooling saw a drop in performance, achieving only 94.08% on the test set, compared to 98.70% with flattening. This suggests that, while pooling methods can reduce complexity and maintain performance with individual features, they may not be as effective when features are combined, possibly due to a loss of feature-specific information during the pooling process.

Another noteworthy observation is the *MFCC-classifier-weights* performance with Global Average Pooling, which achieved a test accuracy of 98.51%. This indicates that the pre-trained MFCC classifier can effectively generalize to the combined feature set, particularly when utilizing pooling techniques. However, despite this strong performance, it still falls short of the top-performing *Trained-classifier-weights* model with flattening, high-

Model Name	Version	Epoch	Bal Acc (%)		
			Train	Val	Test
CQT-classifier-weights	Flattening	6	99.30	99.76	98.68
CQT-classifier-weights	Global Avg Pooling	4	99.06	99.29	98.34
MFCC-classifier-weights	Flattening	8	99.74	99.84	97.20
MFCC-classifier-weights	Global Avg Pooling	6	99.15	99.29	98.51
Trained-classifier-weights	Flattening	10	99.42	99.73	98.70
Trained-classifier-weights	Global Avg Pooling	8	99.19	99.03	94.08

Table 4.10: Results of Multi-Feature Approach (Evaluated on a Subset of the Full Evaluation Data)

lighting the benefit of training the classifier directly on the combined features.

In conclusion, the combination of features, can enhance model performance, with the best results observed using the *Trained-classifier-weights* model with flattening. This latter was the unique model whose test learning curve showed a clear learning trend, meaning the model is actually learning how to generalize. The other models showed a wiggly test learning curve, meaning the generalization is aleatory. Pooling methods may not always preserve the full benefit of combining feature sets, as seen in the slight performance drop with Global Average Pooling in the combined feature context. However, considering the other two models, the pooling methods results were impressive, particularly with the MFCC features where the Global Average Pooling method achieved a test accuracy of 98.51%, surpassing the flattened model by 1.31% and placing itself only 0.19% below the top-performing model.

Final Model Based on the previous sections, the best model identified was the one using trained classifier weights with a flattening layer. Consequently, as explained in Section 3.4.5, the model was proposed in three variants: *DeepSpectraNet*, *DeepSpectraNetLite*, and *DeepSpectraNetFlex*, trained and evaluated on the full dataset. A fourth model, *DeepSpectraNetE2E*, distinguished itself by taking as input the raw audio data, but its performance is tackled in the next section. The results, as shown in Table 4.11, indicate that training on the full dataset did not drastically surpass the results obtained on smaller subsets, suggesting that the model is efficient at learning from a relatively small amount of data (less than 6%

Classifier	Acc		F1		Bal Acc		PR AUC		ROC AUC		EER	
	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
Existing Models												
TCN [24]	98.00	92.00	-	-	-	-	-	-	-	-	-	-
STN [24]	89.00	80.00	-	-	-	-	-	-	-	-	-	-
CNN Scatter [42]	-	88.98	-	-	-	-	-	-	-	-	-	-
DeepSonar [27]	99.98	-	99.98	-	-	-	-	-	99.98	-	0.02	-
VGG16 [32]	97.64	89.25	-	-	-	-	-	-	-	-	-	-
VGG19 [32]	97.58	90.72	-	-	-	-	-	-	-	-	-	-
MobileNet [32]	96.89	92.00	-	-	-	-	-	-	-	-	-	-
CNN-BiLSTM [43]	97.82	-	-	-	-	-	-	-	-	-	0.03	-
MFAAN [44]	-	94.47	-	-	-	-	-	-	-	-	-	0.79
VGG16 [45]	94.00	93.00	-	-	-	-	-	-	-	-	-	-
Proposed Models												
DeepSpectraNet	99.77	98.27	99.87	98.85	99.81	98.37	100	99.96	99.87	99.88	0.19	1.59
DeepSpectraNetLite	99.43	97.40	99.67	98.27	99.27	97.62	99.98	99.93	99.92	99.78	0.59	2.38
DeepSpectraNetFlex	99.89	97.58	99.94	98.41	99.91	96.16	100	99.82	99.99	99.55	0.07	3.23
DeepSpectraNetE2E	98.43	94.43	98.98	96.26	98.69	94.20	99.92	99.44	99.81	98.27	1.43	5.82

Table 4.11: Final DL model Metrics Compared with Baseline Models and SOTA DL models (All the Values Are in %. Evaluation Done on The Full Dataset)

of the full dataset).

Fully E2E Version The key challenge for the *DeepSpectraNetE2E* model was achieving strong generalization. While it easily reached high accuracy on the validation set, test performance was notably weaker, often falling below 70%. This gap was due to the model's extreme flexibility, which allowed it to fit the training data very well (with 99.99% balanced accuracy on validation) but hampered its ability to generalize.

Several architectures were explored, including fully DNN approaches with 1D convolutions, but test results remained suboptimal. The introduction of the Channel and Spatial Attention Module (CSAM) improved the model's learning process by directing attention to relevant features, enhancing generalization. The final *DeepSpectraNetE2E* achieved a balanced accuracy of 98.69% on validation and 94.20% on the test set, with an EER of 1.43%.

Analysis of Results The *DeepSpectraNetFlex* model, demonstrated its higher flexibility with high results on the validation set, however this this came at the expense of generalization, as reflected by a lower test set balanced accuracy of 96.16%. A possible solution consists in removing some training data, following a dropout inspired strategy, to prevent overfitting. With this approach, the model reached a balanced accuracy of 97.92% on the

test set and 99.84% on the validation set at batch 540 (with a total of 17,280 samples). *DeepSpectraNet*, on the other hand, displayed a strong generalization ability, achieving a balanced accuracy of 98.37% on the test set, while maintaining robust performance on the validation set (99.81%). *DeepSpectraNetLite*, though designed for reduced complexity, still performed well, albeit slightly behind DeepSpectraNet, achieving a balanced accuracy of 97.62% on the test set. *DeepSpectraNetE2E* is the worst performer, however considering the absence of the hand crafting process, the results are promising.

Comparison with State-of-the-Art Models The proposed models, particularly *DeepSpectraNet* and *DeepSpectraNetFlex*, demonstrate strong performance when compared to state-of-the-art models in the literature, though it's essential to carefully consider the differences in evaluation methodologies across studies. In this project the reference metric is the balanced accuracy for the reasons explained in Section 4.2.3. However in case of balanced data, as in all the other studies, it is equivalent to the accuracy and the comparison can be made directly.

TCN and *STN*, as reported by Khochare et al., were evaluated separately on validation and test sets. *TCN* achieved a balanced accuracy of 98% on the validation set and 92% on the test set. Similarly, *STN* recorded 89% on the validation set and 80% on the test set. When compared to these results, *DeepSpectraNet* outperformed *TCN* and *STN* both on the validation and test sets, by 1.81% and 6.37%, respectively.

DeepSonar presents another point of comparison, but their evaluation strategies introduce complications. These models have merged all the sets in a single one, thus the training set contains the unseen TTS algorithm from the test set. As a consequence the models were only tested for their learning capabilities, thus the results can only be compared to the validation set of the proposed models. *DeepSonar*, for instance, reports a validation accuracy of 99.98% while *DeepSpectraNetFlex*, when evaluated under similar conditions (validation set), achieved a balanced accuracy of 99.91%, only 0.07% below *DeepSonar*. Unfortunately a comparison with the test set is not possible. *DeepSonar* surpasses *DeepSpectraNetFlex* in the EER metric, with 0.02% against 0.07%, both can be considered excellent results.

VGG16 and *VGG19*, proposed by Reimao and Tzerpos, were evaluated separately on the validation and test sets. *VGG16* achieved a balanced

accuracy of 97.64% on the validation set and 89.25% on the test set, while *VGG19* reached 97.58% on validation and 90.72% on test. *DeepSpectraNet* surpasses both in terms of generalization, with a test balanced accuracy of 98.37%, and outperforms them in learning with a validation balanced accuracy of 99.81%. These results are noteworthy, as VGG16 is an important piece of the proposed model and this demonstrates the effectiveness of the mapping and features combination strategies.

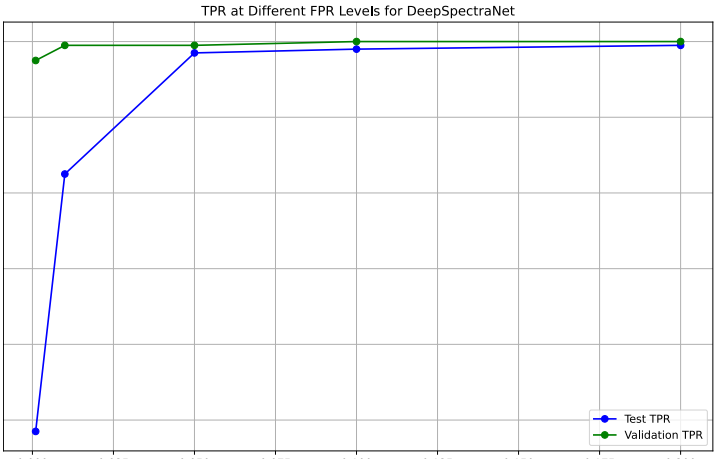
MobileNet, another model evaluated by Reimao and Tzerpos, achieved a balanced accuracy of 96.89% on the validation set and 92.00% on the test set. *DeepSpectraNet* exceeds these metrics on both validation and test sets, further solidifying its status as a more effective model for both learning and generalization.

Finally, *CNN-BiLSTM* reports a balanced accuracy of 97.82% but adopt the same methodology as *DeepSonar*. Following the same reasoning we can only compare the validation set results, where *DeepSpectraNet* outperforms *CNN-BiLSTM* by 1.99%.

In summary, *DeepSpectraNet* and its variants, with a mention to *DeepSpectraNetFlex* for the learning capabilities and to *DeepSpectraNetLite* for the optimal complexity-performance trade-off, demonstrate competitive or superior performance compared to state-of-the-art models. The rigorous evaluation on separate validation and test sets, combined with the model's ability to maintain high performance under different conditions, highlights the robustness and efficacy of the proposed approach.

Performance at Different FPR Levels To provide further insights into the model's robustness, the ROC curve of *DeepSpectraNet* was magnified to evaluate the true positive rate (TPR) at different false positive rate (FPR) levels. Figure 4.12 demonstrates that the model maintains a high true positive rate (TPR) across various FPR thresholds, with a slight drop at the lowest FPR level (0.001) more pronounced in the test set.

Complexity Analysis Table 4.12 provides a detailed overview of the computational complexity associated with training the final models. The total pure training time is split into two parts: the base submodels (focused separately on CQT and MFCC features) and the end-to-end (E2E) phase. It is important to note that the training times reported here exclude any overhead, such as data loading or preprocessing, which can vary depending



FPR	Test TPR (%)	Validation TPR (%)
0.2	99.9	100.0
0.1	99.8	100.0
0.05	99.7	99.9
0.01	96.5	99.9
0.001	89.7	99.5

Figure 4.12: ROC Curve Detail of DeepSpectraNet at Different FPR Levels

on the computational environment. For context, the models were trained using the Lightning.ai platform [3] on a 1 GPU L4 hardware setup.

In terms of training times, *DeepSpectraNet* required 7,130 seconds for the base submodels and an additional 3,449 seconds for the E2E phase, with the best performance achieved at the 10th epoch. The model includes a total of 140,607,593 parameters, of which 8,394,754 were trained during the E2E phase. Despite the substantial number of parameters, the model's training time reflects a balanced trade-off between complexity and performance, demonstrating the efficiency of its architecture.

DeepSpectraNetLite was designed with a primary focus on reducing the model's memory usage and producing a lightweight, yet high-performing architecture. While its training time might not seem drastically reduced compared to other models, due to the unchanged VGG16 core, which is inherently computationally demanding, the true advantage lies in the significant reduction in the number of parameters. The model reduced the total number of parameter by almost 80% compared to *DeepSpectraNet*, yet providing high performance.

DeepSpectraNetFlex required 7,130 seconds for base training and 1,002 seconds for the E2E phase. Although it involved the highest number of trainable parameters during E2E training (32,005,097), the model achieved its best performance after just one epoch. This rapid convergence sug-

[3] `https://lightning.ai`

Model	Best Epoch	Training Time (s)		Number of Parameters	
		Base	E2E	Total	E2E Trained
DeepSpectraNet	10	7130	3449	140 607 593	8 394 754
DeepSpectraNetLite	2	10 780	674	29 711 977	1026
DeepSpectraNetFlex	1	7130	1002	140 607 593	32 005 097
DeepSpectraNetE2E	4	8132	2956	140 800 055	32 197 559

Table 4.12: Final Models Training Time and Number of Parameters

gests that while *DeepSpectraNetFlex* is highly flexible and capable of fast learning, it is also prone to overfitting, as evidenced by its relatively lower performance on the test set compared to its validation results.

DeepSpectraNetE2E relies on *DeepSpectraNetFlex* as a base model thus the base training time is 8,132 seconds. The E2E phase required 2,956 seconds, with the best performance achieved at the 4th epoch. Its total time is comparable to the other models, despite its higher complexity. However it is important to note that the number of training samples was slightly smaller than the other models, due to the way the audio is preprocessed.

In summary, the *DeepSpectraNetLite* model presents an attractive option for scenarios where computational resources are limited, offering competitive performance with significantly reduced parameter count. Meanwhile, *DeepSpectraNet* and *DeepSpectraNetFlex* provide robust alternatives for more resource-intensive applications, with the latter offering rapid convergence at the potential cost of overfitting. *DeepSpectraNetE2E* is a promising model, as it doesn't require any hand-crafted feature extraction, but its generalization capabilities need further improvements. The choice of model should thus be aligned with the specific goals and constraints of the application, balancing the need for computational efficiency and ease against the desired level of model performance.

4.3.3 Overall Comparison

In this section there is a brief comparison between the traditional machine learning and deep learning approaches.

Features Engineering

In both approaches, a 2-second extraction interval was identified as the optimal choice, delivering the best performance on validation data and superior generalization on test data, with the exception of the VGG16 model, where

a 1-second interval slightly outperformed. Regarding feature type selection, MFCC consistently emerged as the most effective for data learning in both approaches. However, the neural representation of CQT features demonstrated remarkable learning and generalization capabilities. Overall, the neural features-based approach improved feature performance.

Concerning window length, traditional ML approaches did not reveal a definitive optimal value, as it varied depending on the model and feature type. In contrast, the deep learning approach favored shorter window lengths, particularly for MFCC and CQT features. The impact of hop length was assessed only in the ML approach and the difference between a quarter and a half of the window length was found to be negligible.

Models

The comparison between the traditional Machine Learning (ML) approach and the Deep Learning (DL) models in Table 4.13 clearly shows the superior performance of the DL models.

The ML-based RF-RFE model delivered a solid test set balanced accuracy of 93.39%, with a slightly higher 95.64% on the validation set. These results are strong, particularly for a model that is generally less complex and more interpretable than deep learning models. However, the DL models, especially *DeepSpectraNet*, significantly outperformed the RF-RFE model across all metrics. *DeepSpectraNet* achieved a balanced accuracy of 98.37% on the test set and 99.81% on the validation set. Its exceptional performance is also reflected in its nearly perfect ROC AUC of 99.87% on the test set. *DeepSpectraNetLite* offered a good balance between performance and efficiency. Despite a reduction in parameters, it still achieved a test set balanced accuracy of 97.62%, slightly lower than *DeepSpectraNet*, but with the added benefit of reduced model complexity. *DeepSpectraNetFlex*, while highly adaptable, showed a tendency towards overfitting, achieving a high balanced accuracy on the validation set (99.91%) but a lower score on the test set (96.16%). *DeepSpectraNetE2E* demonstrated the potential of an end-to-end approach, as it offers both a good performance and the possibility to avoid the hand-crafted feature extraction process, which is a time-consuming and case specific task.

In summary, DL models, particularly *DeepSpectraNet*, demonstrated clear advantages over traditional ML approaches, excelling in both learning and generalization.

Classifier	Acc		F1		Bal Acc		PR AUC		ROC AUC		EER	
	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
Machine Learning												
RF-RFE	98.49	91.65	99.13	94.23	95.64	93.39	99.92	99.35	99.59	97.94	2.65	7.20
Deep Learning												
DeepSpectraNet	99.77	98.27	99.87	98.85	99.81	98.37	100	99.96	99.87	99.88	0.19	1.59
DeepSpectraNetLite	99.43	97.40	99.67	98.27	99.27	97.62	99.98	99.93	99.92	99.78	0.59	2.38
DeepSpectraNetFlex	99.89	97.58	99.94	98.41	99.91	96.16	100	99.82	99.99	99.55	0.07	3.23
DeepSpectraNetE2E	98.43	94.43	98.98	96.26	98.69	94.20	99.92	99.44	99.81	98.27	1.43	5.82

Table 4.13: Comparison of The Proposed Models Based on Traditional ML and DL Approaches (All The Values Are in %. Evaluation Done on The Full Dataset)

4.4 Explainability

In this section, the explainability of the final models is analyzed to provide insights into their decision-making processes. For the traditional ML approach, the Random Forest (RF) model is examined, focusing on feature importance and feature behavior. For the deep learning approach, the four models, *DeepSpectraNet*, *DeepSpectraNetLite*, *DeepSpectraNetFlex*, and *DeepSpectraNetE2E*, are analyzed through confusion matrices, Mapper analysis, with the addition of Grad-CAM for the former three models.

4.4.1 Traditional Machine Learning

The analysis of the final Random Forest (RF) model's behavior and decision-making process is critical for understanding how the model distinguishes between real and fake data. The explainability analysis, focusing on feature importance and feature behavior, provides insights into the model's reasoning.

Most Important Features

The feature importance analysis (Figure 4.13) reveals that the model primarily relies on MFCC (Mel-Frequency Cepstral Coefficients) features, which account for 61% of the overall importance. This dominance of MFCC features confirms their suitability to audio classification tasks, as demonstrated by other studies [46, 45]. RMS (Root Mean Square) features contribute 21%, while CQT (Constant-Q Transform) features contribute 18%. This distribution indicates that while MFCC is the most critical, the model also con-

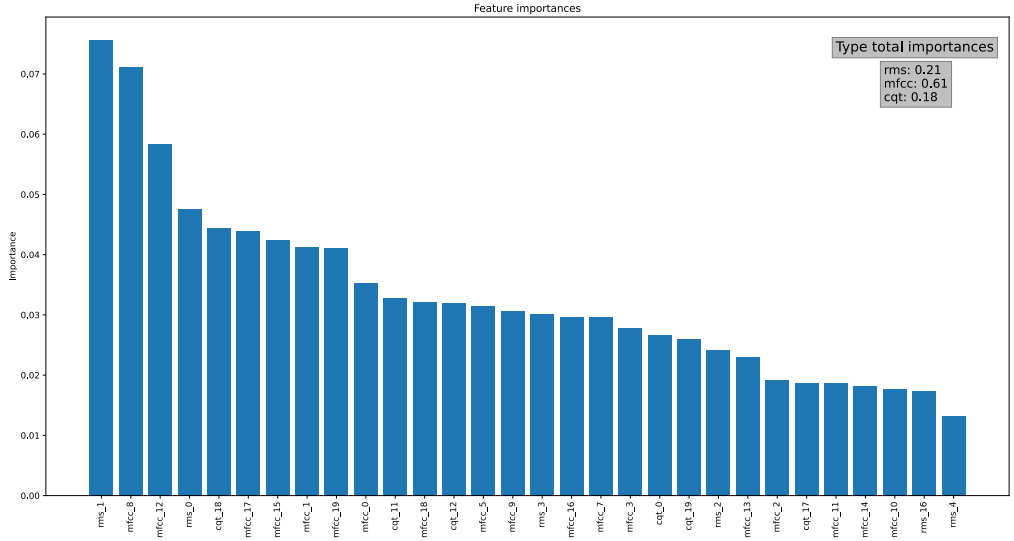


Figure 4.13: Final RF model feature importance according to RFE

siders a combination of amplitude-related and frequency-related features to make predictions. This multi-faceted approach likely enhances the model's robustness, especially when handling complex audio patterns.

In Figure 4.14, are presented the mean and variance across the data of the most important features for each class. To ensure a fair comparison, the features were normalized using the Max-Abs method, using the same max value for both classes. The plot shows clear differences in the mean values of these features between the real and fake classes. For instance, certain MFCC coefficients show distinct mean value separations between classes, which likely helps the model in making accurate predictions. The RMS features, which are related to the energy of the signal, also show variability, contributing to the model's ability to detect subtle differences in the audio data. These findings show that the model's decision-making is not based on a single dominant feature but rather a combination of features that together capture the complex characteristics of the data.

For sake of completeness, Figure 4.15, shows the top 3 MFCC values over time compared to the waveform and mel spectrogram.

The RF model's explainability analysis demonstrates that the model leverages a combination of MFCC, RMS, and CQT features to distinguish between real and fake audio data. The fact that the 18 out of 20 MFCCs were retained through RFE, further supports the importance of these features in the model's decision-making process.

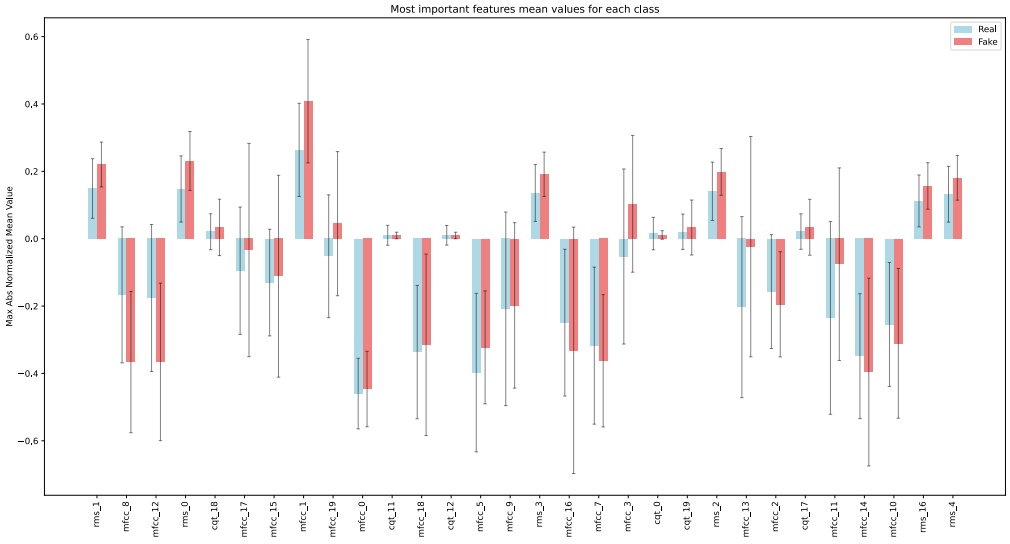


Figure 4.14: Final RF Model Most Important Features Mean and Variance

4.4.2 Deep Learning

This section examines the decision-making process and explainability of the best DL models. We begin by analyzing the confusion matrix to evaluate how well the model distinguishes between real and fake data. Next, we review the model's most significant errors to identify areas for improvement. The Mapper analysis will then provide insight into the learned feature space through the 3DMapper. Finally, the Grad-CAM analysis will visually highlight the key areas of the input data that influence the model's predictions, offering a clear view of its focus during decision-making.

In the following sections the main focus is on the results of the DeepSpectraNet model, with a brief comparison with the other two models. All the images and charts for these latter are made available in the Appendix.

Curves and Confusion Matrices

The objective of this section is to evaluate the performance of the *DeepSpectraNet* model using ROC and PR curves, which measure the model's ability to distinguish between real and fake audio data. Additionally, confusion matrices are analyzed to give a more detailed look into classification accuracy and errors.

Both the ROC and PR curves for *DeepSpectraNet* (Figure 4.16) indicate near-perfect performance on the validation set and only a slight degradation on the test set, where the PR AUC reached 99.96% and the ROC AUC 99.87%. These scores clearly demonstrate that the model is highly effective

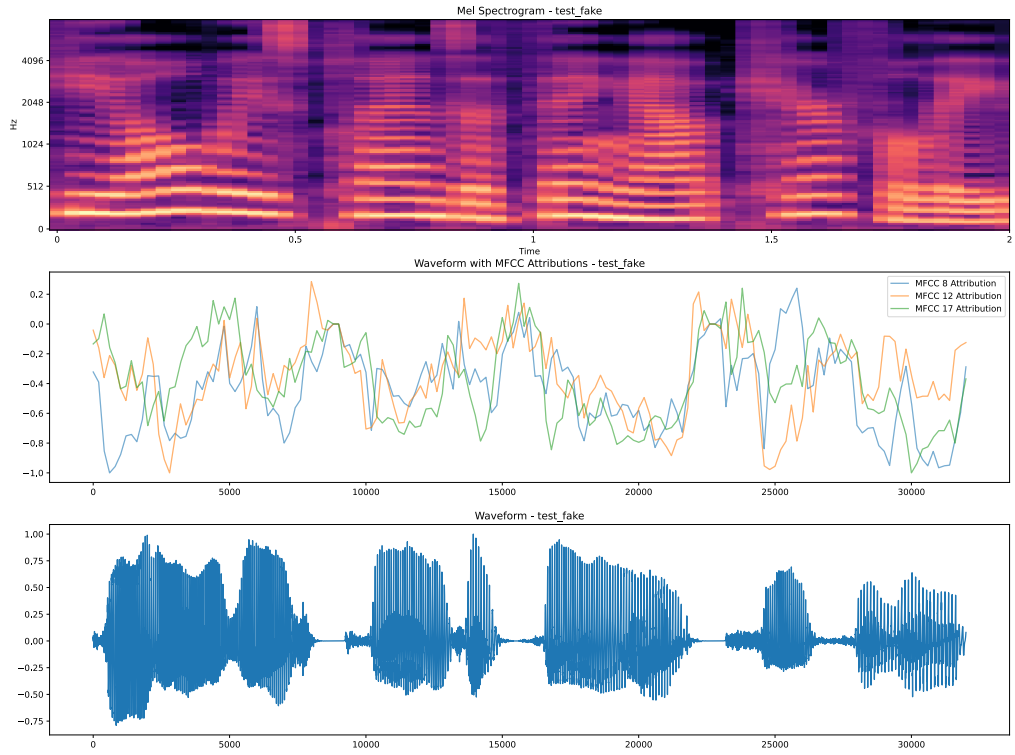


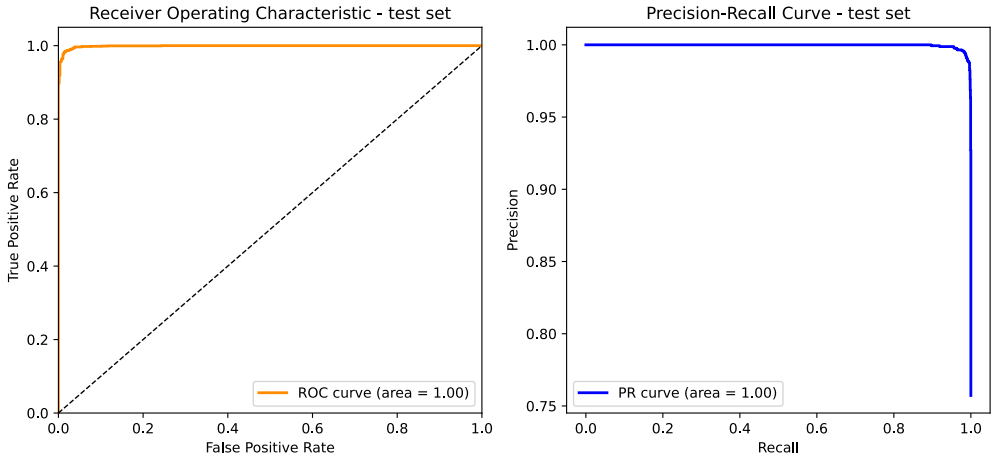
Figure 4.15: Final RF Model MFCC Behavior on Waveform and Spectrogram

in discriminating between the two classes, with strong generalization across both the validation and test datasets. A more granular view of the ROC curve at different false positive rates (FPR) is presented in Figure 4.12.

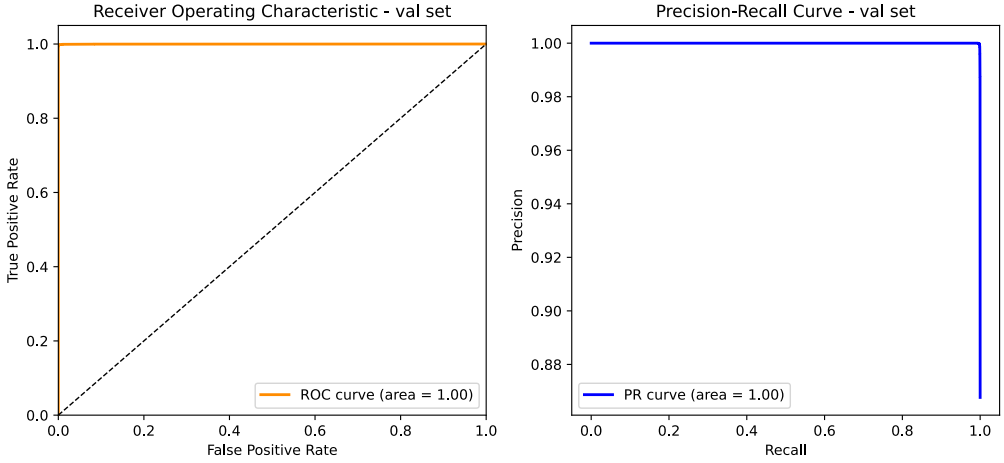
The confusion matrix (Figure 4.17) complements these metrics by offering a closer look at classification performance. On the validation set, *DeepSpectraNet* shows similar accuracy between both classes, though the fake class has a slightly lower rate of false negatives (0.13%) compared to the real class (0.24%). The test set results align with this trend, albeit with slightly higher error rates: the fake class has 1.42% false positives, while the percentage of false negatives is 1.82%.

Similar patterns are evident in the confusion matrices of the *DeepSpectraNetLite* (Figure C.2), *DeepSpectraNetFlex* (Figure C.4) and *DeepSpectraNetE2E* (Figure C.6) models. The only significant deviation occurs in *DeepSpectraNetFlex*, which exhibits a higher proportion of false positives on the test set.

In conclusion the ROC and PR curves demonstrate that *DeepSpectraNet* performs remarkably well, especially on the validation set. Confusion matrices reveal that although the model slightly struggles more with real data



(a)



(b)

Figure 4.16: ROC and PR curves of DeepSpectraNet on: a) Test Set, b) Validation Set.

on the test set, it maintains a strong ability to classify both real and fake audio data accurately.

Worst Errors

Understanding the model's behavior further requires analyzing its worst errors (samples that the model misclassified with the highest confidence). The worst errors of the *DeepSpectraNet* model (Figure 4.18) revealed that the top 15 errors on the validation set were all false negative, and the same on the test set with 13 out of 15. This suggests that the model struggles more with recognizing real audio data, sometimes confidently misclassifying it as fake. This behavior highlights the subtle distinctions between real and fake audio data.

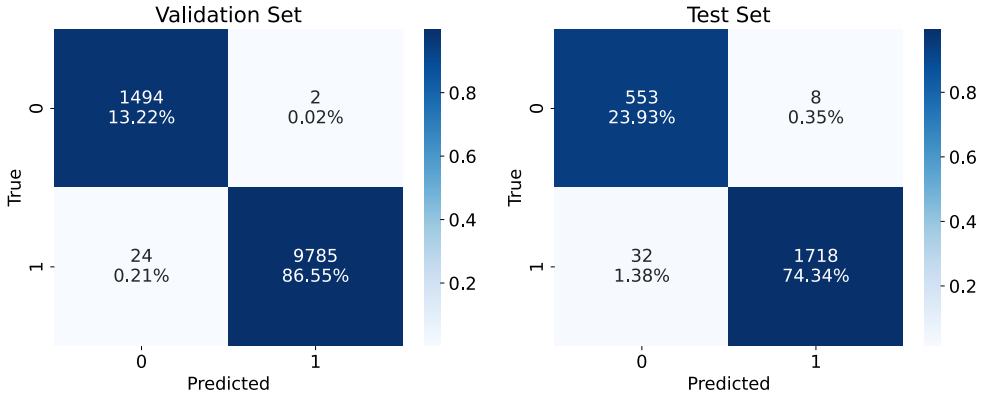


Figure 4.17: Confusion Matrix of DeepSpectraNet

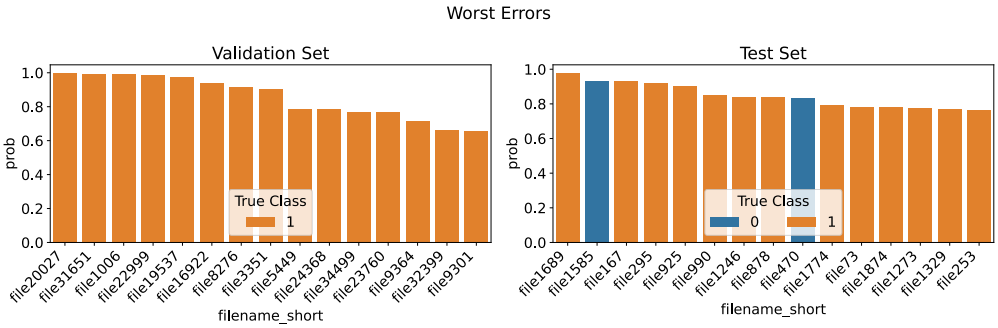


Figure 4.18: Worst Errors of DeepSpectraNet

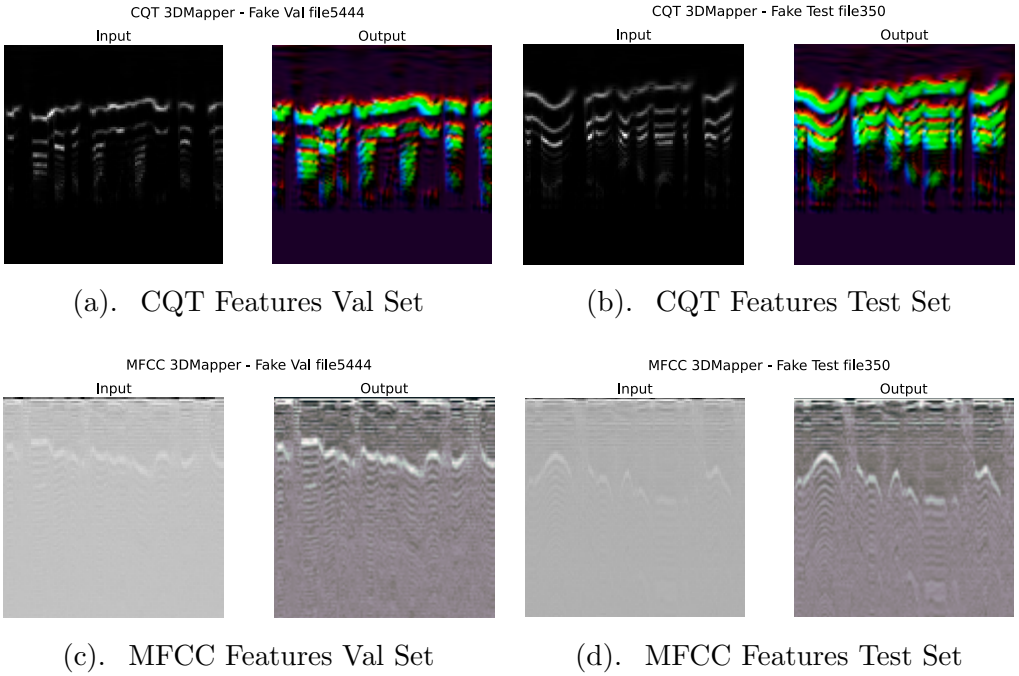
Interestingly, a different trend is observed in the worst errors of the *DeepSpectraNetLite* (Figure C.7) and *DeepSpectraNetFlex* (Figure C.8) models. Specifically, in *DeepSpectraNetFlex*, 12 out of the 15 worst errors on the test set were false positives, indicating a stronger tendency to misclassify fake audio data as real.

This behavior variability should be considered when selecting the appropriate model for specific applications. For example, in content moderation tasks where preventing the spread of AI-generated, potentially harmful content is critical, a model like *DeepSpectraNet* that is less prone to misclassify fake data as real would be a more suitable choice.

In summary *DeepSpectraNet* tends to misclassify real data as fake with high confidence, especially on the validation set, while *DeepSpectraNetFlex* shows a higher rate of false positives, particularly on the test set. DeepSpectraNetE2E confirms the DeepSpectraNet behavior. This difference in behavior can guide model selection based on the specific needs of the task at hand, such as content moderation.

Mapper Analysis

This section examines how the Mapper enhances different areas of the input data, providing insights into the model's feature extraction process, independently for CQT and MFCC images.



(a). CQT Features Val Set (b). CQT Features Test Set

(c). MFCC Features Val Set (d). MFCC Features Test Set

Figure 4.19: DeepSpectraNet Mapper Output for a Random Fake Audio Sample on Different Evaluation Sets. (Mapper Input on the left, Mapper Learned Output on the right).

CQT Image: Figure 4.19a shows the Mapper output for a randomly selected fake audio sample from the validation set using the Constant-Q Transform (CQT). Since the test set produced similar results (Figure 4.19b), we focus on the validation set for the analysis.

In the output, the input's visible wave-like elements are enhanced, with each color channel emphasizing different parts of the input. The red channel is concentrated on the upper portions of the white wave-like structures, the green channel highlights the lower parts, and the blue channel emphasizes vertical structures. Remarkably, the model reveals new structures both above and below the visible area that were not present in the original image. This suggests the model is capable of extracting and enhancing hidden features, leading to more informed decisions.

Interestingly, the behavior of other models differs. The *DeepSpectraNet-Flex* model (Figure C.14) presents a more detailed learned image, indicating

its greater flexibility. In contrast, the *DeepSpectraNetLite* model (Figure C.10) displays a different color channel distribution. In this case, the red channel dominates and captures vertical structures, the green channel covers the entire wave-like elements, while the blue channel is concentrated in the middle of those elements, surrounded by the green channel.

MFCC Image: For the Mel-Frequency Cepstral Coefficient (MFCC) images, the learned features are less detailed due to the lower flexibility of the Lookup Table (LUT) mapping method, as opposed to the CNN-based models. However, the Mapper output still enhances the input image, revealing new details not initially visible (Figures 4.19c and 4.19d).

When comparing the MFCC-based results across other models (Figures in Appendix: C.12, C.16, C.17, and C.13), no significant differences were observed, indicating a consistent behavior among the models for this feature type.

In summary, the results showed the efficacy of the Mapper in enhancing the input data, feeding the model with more detailed and informative representations. The CNN based Mapper used for the CQT images was able to extract more detailed features, while the LUT based Mapper used for the MFCC images was less flexible but still able to enhance the input data.

Fully E2E Features

Unlike previous models, the fully end-to-end model doesn't rely on pre-extracted features such as CQT or MFCC. Instead, it utilizes raw audio data to compute a time-frequency representation through its Signal Preprocessing Block, which is further enhanced by the Convolutional Block.

This section explores how the model processes and transforms these features. Figures 4.20 and 4.21 depict the learned features at three key stages: the initial spectrogram (input), the output of the Convolutional Block (enhanced features), and the final representation after the 3D Mapper (mapped features).

The most significant transformation occurs in the Convolutional Block, where high-frequency content is enhanced, revealing new details not present in the original spectrogram. The Mapper then refines these features, with each RGB channel focusing on distinct areas, further sharpening the representation for better classification performance.

In summary, the analysis suggest the high frequencies are crucial for the

model's decision-making process, as it tends to focus on these areas during feature learning.

Grad-CAM

Grad-CAM (Gradient-weighted Class Activation Mapping) is a visualization technique that highlights the most important areas in the input data influencing the model's predictions. The goal of this analysis is to understand how feature combinations affect the decision-making process and whether the "fakeness" of an audio sample is contained in smaller, more subtle details.

For a correct understanding of the results, it is important to note that the features images have a reverted y axis, with the lowest values at the top and the highest at the bottom. For example, in the CQT images, the lowest frequencies are at the top and the highest at the bottom.

What is the Effect of Feature Combination? To investigate the effect of feature combination, two Grad-CAM analyses were conducted: one by separating the model into the two submodels (one for CQT and one for MFCC), and another by using the combined model with combined features. Comparing the Grad-CAM outputs from these approaches reveals which features are crucial for the model's decision-making process and provides insight into whether the features are redundant or complementary.

The analysis was performed on a highly confident true fake sample. In the validation set (Figure 4.22), notable differences emerged between the separated and combined Grad-CAM outputs. The combined model exhibited a broader focus, expanding its attention to higher frequencies and time frames in the CQT map, while also concentrating on lower MFCC coefficients. This demonstrates that the model relies on both broad audio characteristics (captured by lower MFCCs) and finer details. The fact that both activation maps changed with the combined model suggests that the features are complementary and jointly contribute to the decision-making process.

In the test set (Figure 4.23), the model showed a stronger reliance on CQT features as their activation map remained similar across combined and separated approaches, with changes in the MFCC activation map. This indicates that CQT features play a dominant role in the test set predictions. However, MFCC features still provide valuable supplementary information,

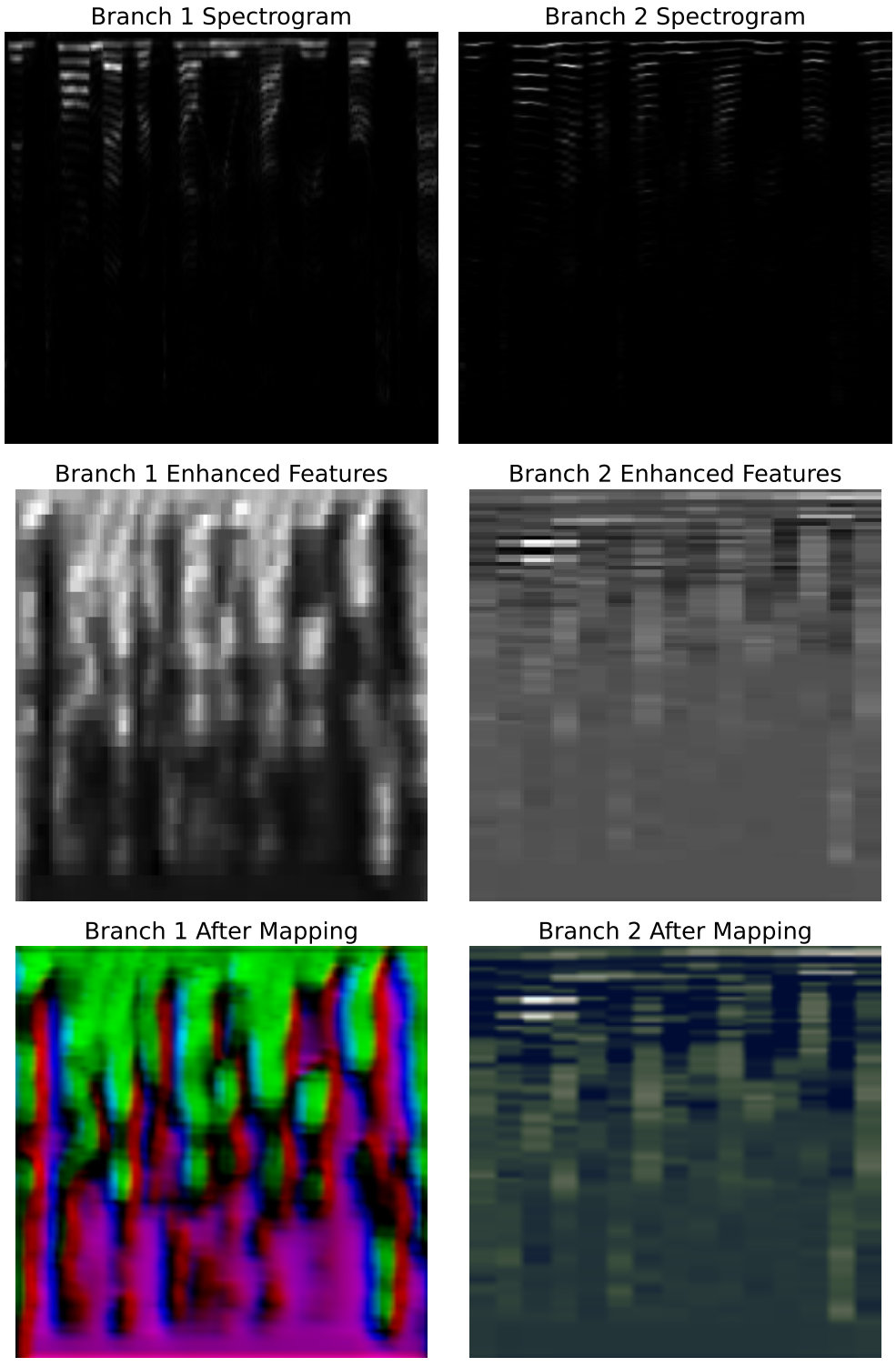


Figure 4.20: DeepSpectraNetE2E Learned Features for Random Fake Audio Sample (Validation Set)

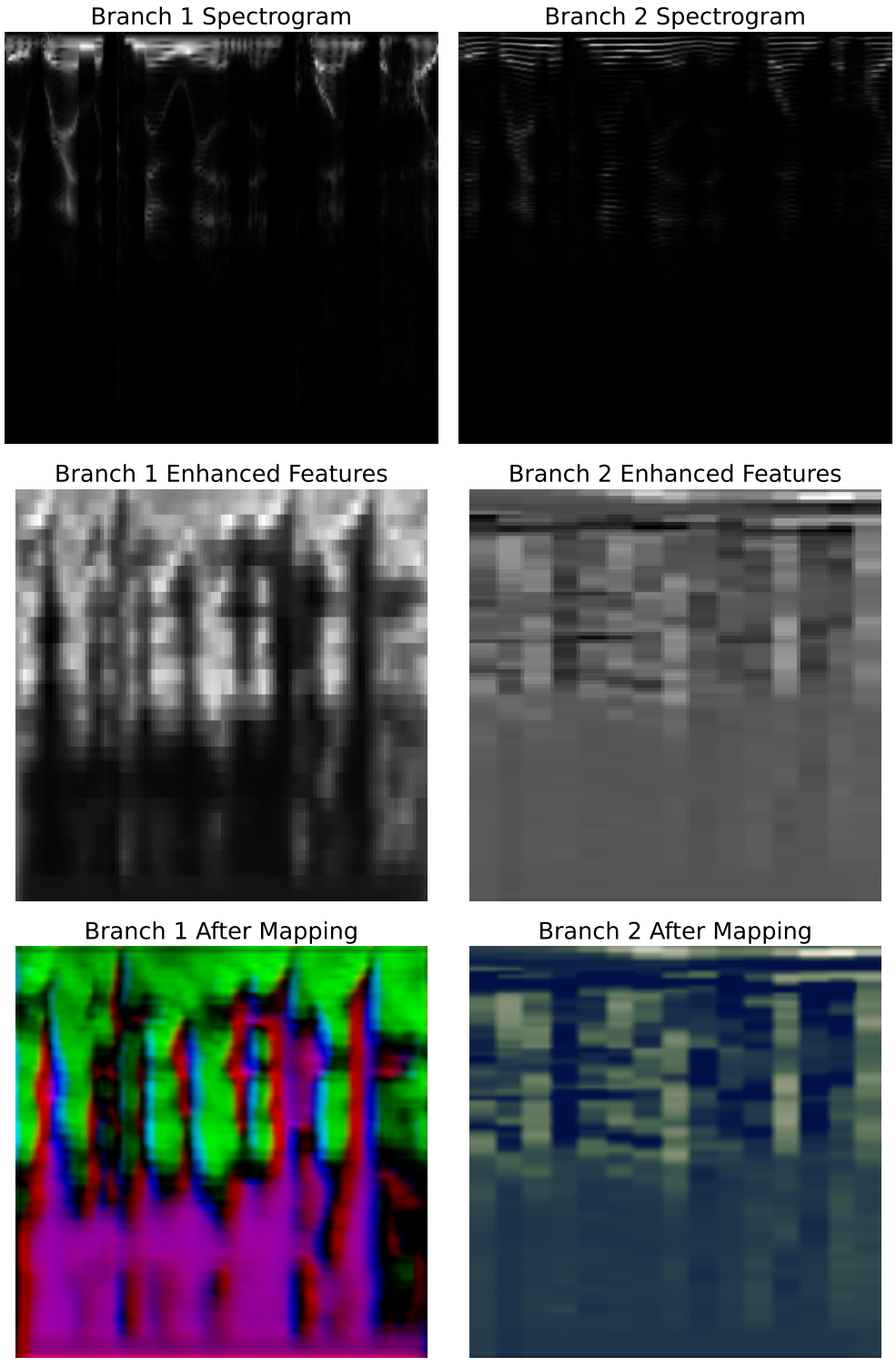


Figure 4.21: DeepSpectraNetE2E Learned Features for Random Fake Audio Sample (Test Set)

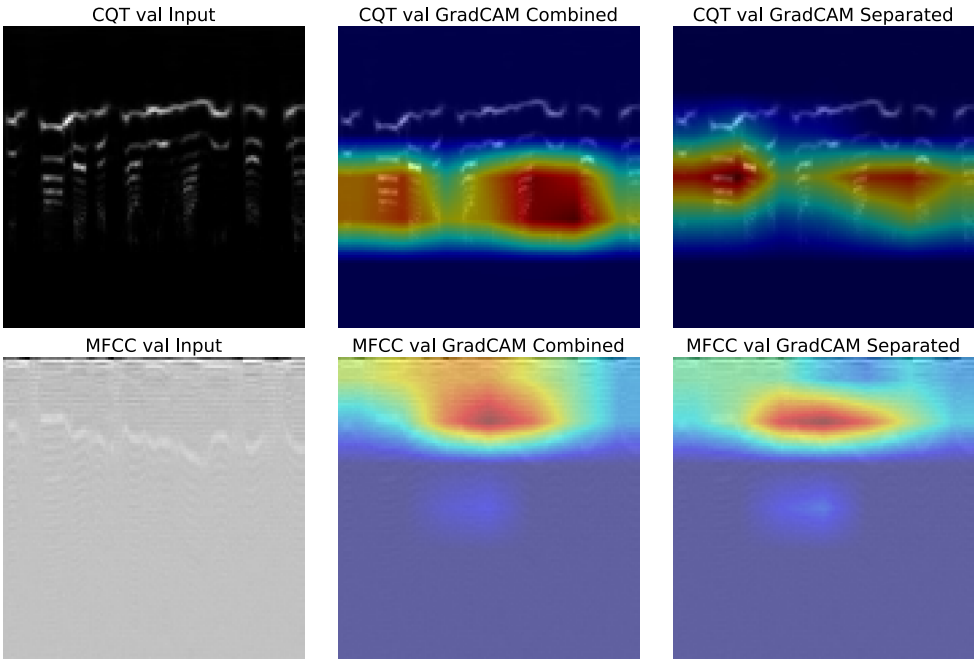


Figure 4.22: Grad-CAM Analysis of DeepSpectraNet on a Validation Set Sample. Grad-CAM Combined refers to the combined model which takes both MFCC and CQT features as input, while Grad-CAM Separated is obtained splitting the combined model into two separate models, one for each feature set.

allowing the model to refine its decision-making by shifting its attention towards different time frames and audio characteristics.

Interestingly, in both the validation and test sets, the model utilized CQT to extract information from the beginning and end of the audio clip, while MFCCs were used for the middle section. This behavior suggests that the model effectively uses both features to capture distinct temporal details.

Is the Fakeness Contained in Small Details? It is reasonable to hypothesize that the "fakeness" in audio data is captured within small, specific details, suggesting certain portions of the audio may be more likely to reveal signs of artificial generation. To test this hypothesis, the samples were divided into four categories: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). From each category, $\min(150, \text{num\_samples\_in\_class})$ samples were randomly selected and on them was calculated the average Grad-CAM activation, as well as the number of highly activated pixels (defined as pixels with values greater than 0.6). These results are presented in Figures 4.24a, 4.24b, 4.25a, and 4.25b.

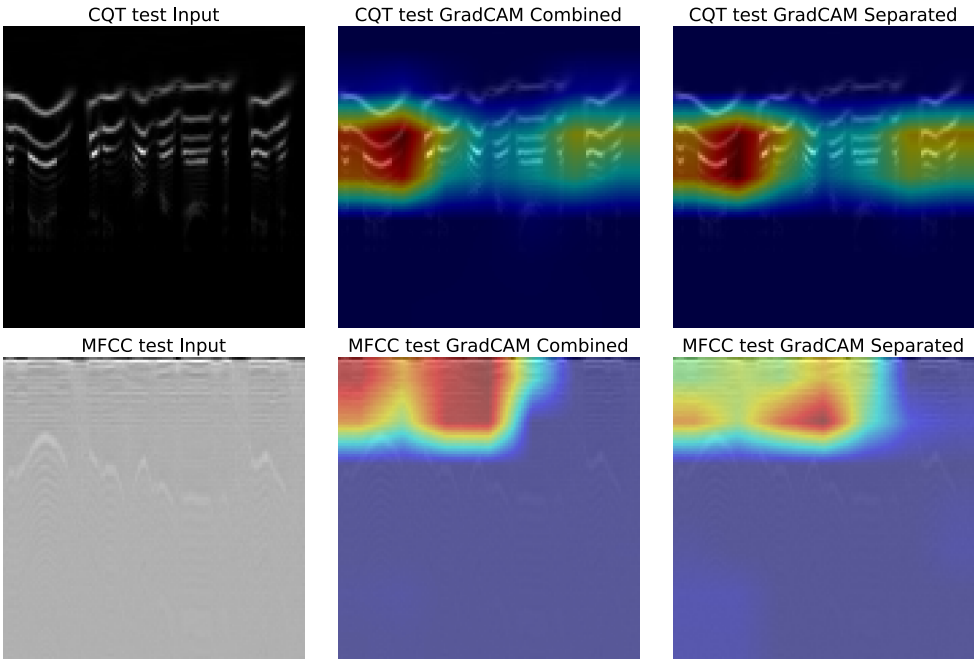
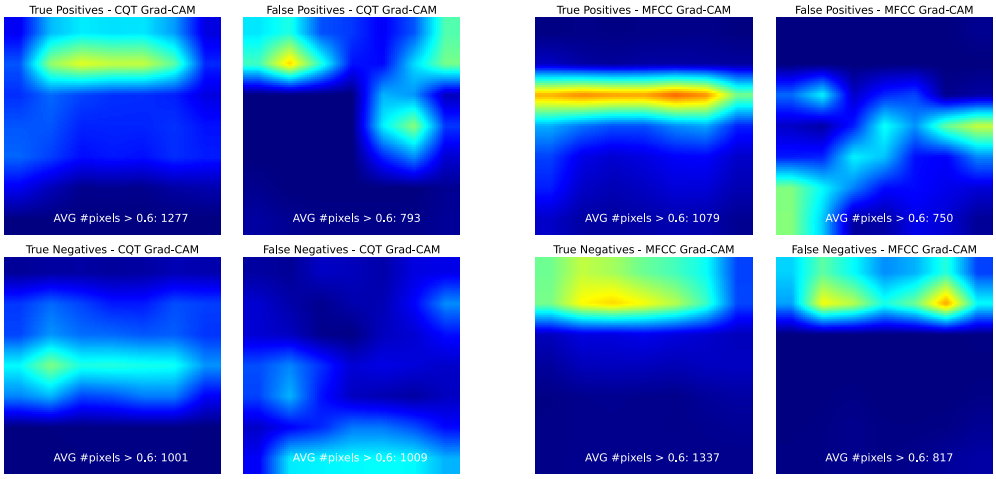


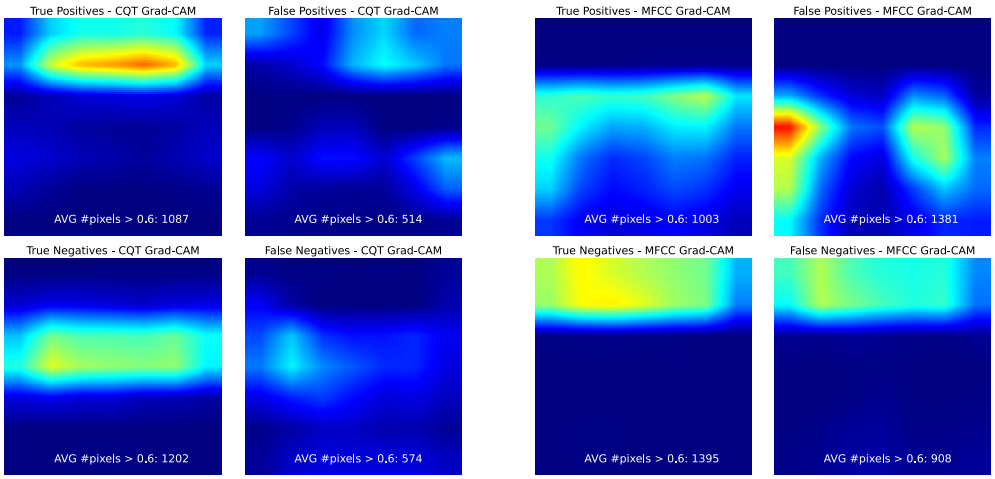
Figure 4.23: Grad-CAM Analysis of DeepSpectraNet on a Test Set Sample. Grad-CAM Combined refers to the combined model which takes both MFCC and CQT features as input, while Grad-CAM Separated is obtained splitting the combined model into two separate models, one for each feature set.



(a) (b)

Figure 4.24: Average Grad-CAM of DeepSpectraNet on Validation Set Subset splitted in TP (upper left), TN (lower left), FP (upper right), FN (lower right) samples. a) Results for the CQT features, b) Results for the MFCC features.

Contrary to the hypothesis, the results demonstrated that both the True



(a) (b)

Figure 4.25: Average Grad-CAM of DeepSpectraNet on Test Set Subset splitted in TP (upper left), TN (lower left), FP (upper right), FN (lower right) samples. a) Results for the CQT features, b) Results for the MFCC features.

Positive and True Negative classes displayed a similar number of highly activated pixels across all circumstances. This indicates that the fakeness of audio is not exclusively contained in small details, but may be dispersed across larger areas of the audio representation.

Further analysis revealed differing model behavior between real and fake audio samples. In the validation set, when analyzing CQT features, the model tended to focus on the upper part of the image for real audio (TP), whereas for fake audio (TN), the focus shifted to the middle portion. This suggests that the model identifies fakeness by analyzing higher frequencies in the CQT representation, while certain lower-frequency bands are characteristic of real audio. For MFCC features, the situation appears to be reversed: the model emphasizes broader details (lower MFCC coefficients) in fake audio, while for real audio, it concentrates on finer details (higher MFCC coefficients). These observations were consistent in the test set as well.

Notably, the *DeepSpectraNetLite* model provided distinct insights. As highlighted earlier in the Mapper analysis, this model focuses on smaller, more concentrated regions of the input. In this case, the number of highly activated pixels in the TP class was roughly double that of the TN class, supporting the idea that small, highly distinctive features play a significant role in identifying fake audio.

In conclusion, while detecting fakeness does not necessarily require focusing exclusively on small details, these subtle cues do exist, and leveraging them can enhance model performance. According to the results from *DeepSpectraNetLite*, these important details are primarily located in the higher frequencies of CQT features and in mid-range MFCC coefficients (between 15 and 50).

The Grad-CAM analysis provided valuable insights into the model's decision-making process. By analyzing the activation maps of individual feature sets (MFCC and CQT) and their combined versions, it became clear that the combination of features enabled the model to focus on a broader range of audio characteristics. While CQT features were dominant in certain cases, MFCCs still played an essential role in enhancing the model performance.

Additionally, the hypothesis that fakeness could be contained in small details was partially supported. Although fakeness detection did not rely solely on small details, models like *DeepSpectraNetLite* demonstrated a tendency to focus on specific fine-grained features. This suggests that further refinement of the model's attention to such details could potentially improve classification performance.

Chapter 5

Conclusions

Through comprehensive experimentation, this research has explored the detection of audio deepfakes using traditional machine learning and deep learning approaches. The study has contributed significantly to the field by answering seven research questions (see Section 1.3) and by proposing four novel deep learning models that substantially enhance audio deepfake detection.

- **RQ1: What are the preliminary factors that most influence correct deepfake detection?**
  The key factors influencing deepfake detection performance are the extraction interval, window length, and hop length used for spectral feature extraction. In both approaches, a 2-second extraction interval delivered the best classification performance. While the optimal window length was not definitive in traditional ML models, deep learning models consistently favored shorter window lengths, especially for MFCC and CQT features. The hop length impact difference between a quarter and a half of the window length was found to be negligible on the ML approach.

- **RQ2: What are the most important audio features for accurately identifying deepfakes?**
  Among the audio features examined, MFCC and CQT stood out as the most critical for accurately identifying deepfakes. MFCCs excelled in learning performance, while CQT features demonstrated exceptional generalization to unseen algorithms, particularly when processed by deep learning models.

- **RQ3: How effective are traditional machine learning models in detecting audio deepfakes?**

Traditional ML models like RF, SVM, Catboost, and LR performed well in learning, with Catboost achieving 99% balanced accuracy. However, they struggled to generalize, particularly on unseen algorithm data, with Logistic Regression reaching only 80%. RF proved to be the best all-around performer, and further improvement was observed combining the features and selecting the most relevant ones using Recursive Feature Elimination, where RF achieved a balanced accuracy of 92% on the unseen algorithm data.

- **RQ4: Can deep learning models represent a leap forward in audio deepfake detection?**
  The introduction of CNN-based architectures (VGG16 and MobileNetV3) significantly improved performance compared to traditional ML models, especially in handling unseen algorithm data. The proposed family of *DeepSpectraNet* models, leveraging combined MFCC and CQT features enhanced by CNN-based mappings, achieved with its top performer, balanced accuracy of 99.81% on validation and 98.37% on unseen algorithm data. Furthermore, the fully end-to-end model, *DeepSpectraNetE2E*, reached 98.69% balanced accuracy on validation and 94.20% on the test set, demonstrating the potential of automated feature extraction from raw audio.

- **RQ5: How does feature combination impact the performance of models in detecting audio deepfakes?**
  Combining multiple audio features had a clear positive impact on both ML and DL models. MFCC and CQT, in particular, offered complementary strengths, with MFCC excelling in learning performance and CQT showing superior generalization, especially in deep learning models.

- **RQ6: How can the features be improved to enhance the detection of audio deepfakes?**
  The detection performance of standard audio features can be significantly improved through feature combination, CNN-based mapping, and the addition of a CNN + Attention based preprocessing module. The dual-branch mapping strategy employed in *DeepSpectraNet* models demonstrated that enabling the network to enhance different aspects of the features contributed to substantial gains in classification accuracy.

- **RQ7: What do deep learning models focus on in their decision-making process?**

Explainability techniques such as Grad-CAM and Mapper Analysis revealed that deep learning models primarily focus on frequency-related information, capturing broad audio characteristics from lower MFCC coefficients and finer details from higher frequencies in CQT.
The best models dynamically adjusted their focus based on the task at hand, favoring MFCC for validation set data and CQT for test set data.
The lightweight *DeepSpectraNetLite* model was found to detect deepfake artifacts by focusing on small details, suggesting that further refinement in this area could enhance detection accuracy.

In conclusion, this thesis has advanced the field of audio deepfake detection by automating feature extraction and proposing four novel models that surpass existing ones in the literature. The *DeepSpectraNet* family of models, with their feature mapping and enhancement strategies, offers high detection accuracy and strong generalization performance.

While the current models show promising results, further investigation is required to enhance the generalization capabilities of the fully end-to-end models, particularly for unseen deepfake generation techniques. Another area for research is the evaluation of the models' robustness against adversarial attacks, which could potentially undermine their performance in real-world scenarios. Additionally, optimizing computational efficiency and exploring lightweight models suitable for real-time detection in resource-constrained environments will be key areas for future research. Finally, a possible extension of this work could involve the integration of multimodal features, combining audio and textual information, to provide the models with a more comprehensive understanding of the context in which the audio was generated.

Bibliography

[1] Yandex. Catboost enables fast gradient boosting on decision trees using gpus, 12 2018. Accessed: 2024-17-10.

[2] Haifeng Wang and Teng Wu. Knowledge-enhanced deep learning for wind-induced nonlinear structural dynamic analysis. *Journal of Structural Engineering*, 146, 11 2020.

[3] Sumit Saha, December 2018. Accessed: 2024-18-10.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[5] Christoph Bregler, Michele Covell, and Malcolm Slaney. Video rewrite: Driving visual speech with audio, 01 1997.

[6] Justus Thies, Michael Zollhöfer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2387–2395, 2016.

[7] Adrienne de Ruiter. The distinct wrong of deepfakes, December 2021.

[8] Ivan Perov, Daiheng Gao, Nikolay Chervoniy, Kunlin Liu, Sugasa Marangonda, Chris Umé, Mr. Dpfks, Carl Shift Facenheim, Luis RP, Jian Jiang, Sheng Zhang, Pingyu Wu, Bo Zhou, and Weiming Zhang. Deepfacelab: Integrated, flexible and extensible face-swapping framework, 2021.

[9] Yusheng Tian, Jingyu Li, and Tan Lee. Creating personalized synthetic voices from articulation impaired speech using augmented reconstruction loss. In *ICASSP 2024 - 2024 IEEE International Conference on*

Acoustics, Speech and Signal Processing (ICASSP), pages 11501–11505, 2024.

[10] Rachel Gordon. 3 questions: What you need to know about audio deepfakes, March 2024. Accessed: 2024-18-10.

[11] KnowledgeNile. Applications of deepfake technology: Positives and dangers. Accessed: 2024-18-10.

[12] Zaynab Almutairi and Hebah Elgibreen. A review of modern audio deepfake detection methods: Challenges and future directions. *Algorithms*, 15(55):155, May 2022.

[13] Telegraph Reporters. "deepfake" video shows volodymyr zelensky telling ukrainians to surrender. *The Telegraph*, March 2022.

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

[15] Heiga Zen, Andrew Senior, and Mike Schuster. Statistical parametric speech synthesis using deep neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, page 7962–7966, Vancouver, BC, Canada, May 2013. IEEE.

[16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

[17] Mohammad Reza Hasanabadi. An overview of text-to-speech systems and media applications, 2023.

[18] Zahra Khanjani, Gabrielle Watson, and Vandana P. Janeja. Audio deepfakes: A survey. *Frontiers in Big Data*, 5, 2023.

[19] Jiangyan Yi, Chenglong Wang, Jianhua Tao, Xiaohui Zhang, Chu Yuan Zhang, and Yan Zhao. Audio deepfake detection: A survey, 2023.

[20] Jiangyan Yi, Ye Bai, Jianhua Tao, Haoxin Ma, Zhengkun Tian, Chenglong Wang, Tao Wang, and Ruibo Fu. Half-truth: A partially fake audio detection dataset, December 2023. arXiv:2104.03617 [cs, eess].

[21] Abhishek Dixit, Nirmal Kaur, and Staffy Kingra. Review of audio deepfake detection techniques: Issues and prospects. *Expert Systems*, 40(8):e13322, 2023.

[22] Clara Borrelli, Paolo Bestagini, Fabio Antonacci, Augusto Sarti, and Stefano Tubaro. Synthetic speech detection through short-term and long-term prediction traces. *EURASIP Journal on Information Security*, 2021(1):2, April 2021.

[23] Massimiliano Todisco, Xin Wang, Ville Vestman, Md Sahidullah, Hector Delgado, Andreas Nautsch, Junichi Yamagishi, Nicholas Evans, Tomi Kinnunen, and Kong Aik Lee. Asvspoof 2019: Future horizons in spoofed and fake audio detection, 2019.

[24] Janavi Khochare, Chaitali Joshi, Bakul Yenarkar, Shraddha Suratkar, and Faruk Kazi. A deep learning framework for audio deepfake detection. *Arabian Journal for Science and Engineering*, 47(3):3447–3458, March 2022.

[25] Tianyun Liu, Diqun Yan, Rangding Wang, Nan Yan, and Gang Chen. Identification of fake stereo audio using svm and cnn. *Information*, 12(7), 2021.

[26] Emily R. Bartusiak and Edward J. Delp. Frequency domain-based detection of generated audio, 2022.

[27] Run Wang, Felix Juefei-Xu, Yihao Huang, Qing Guo, Xiaofei Xie, Lei Ma, and Yang Liu. Deepsonar: Towards effective and robust detection of ai-synthesized fake voices, 2020.

[28] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 11 of *KDD '16*, page 785–794. ACM, August 2016.

[29] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support, 2018.

[30] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features, 2019.

[31] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, April 1980.

[32] Ricardo Reimao and Vassilios Tzerpos. For: A dataset for synthetic speech detection. In *2019 International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, page 1–10, Timisoara, Romania, oct 2019. IEEE.

[33] Tomi Kinnunen, Md Sahidullah Sahidullah, Héctor Delgado, Massimiliano Todisco, Nicholas Evans, Junichi Yamagishi, and Kong Aik Lee. The asvspoof 2017 challenge: Assessing the limits of replay spoofing attack detection. In *Proceedings of INTERSPEECH*, 2017.

[34] Junichi Yamagishi, Xin Wang, Massimiliano Todisco, Md Sahidullah, Jose Patino, Andreas Nautsch, Xuechen Liu, Kong Aik Lee, Tomi Kinnunen, Nicholas Evans, and Héctor Delgado. Asvspoof 2021: accelerating progress in spoofed and deepfake speech detection, 2021.

[35] Jiangyan Yi, Ruibo Fu, Jianhua Tao, Shuai Nie, Haoxin Ma, Chenglong Wang, Tao Wang, Zhengkun Tian, Xiaohui Zhang, Ye Bai, Cunhang Fan, Shan Liang, Shiming Wang, Shuai Zhang, Xinrui Yan, Le Xu, Zhengqi Wen, Haizhou Li, Zheng Lian, and Bin Liu. Add 2022: the first audio deep synthesis detection challenge, 2024.

[36] Jiangyan Yi, Jianhua Tao, Ruibo Fu, Xinrui Yan, Chenglong Wang, Tao Wang, Chu Yuan Zhang, Xiaohui Zhang, Yan Zhao, Yong Ren, Le Xu, Junzuo Zhou, Hao Gu, Zhengqi Wen, Shan Liang, Zheng Lian, Shuai Nie, and Haizhou Li. Add 2023: the second audio deepfake detection challenge, 2023.

[37] Nicolas M. Müller, Pavel Czempin, Franziska Dieckmann, Adam Froghyar, and Konstantin Böttinger. Does audio deepfake detection generalize?, 2022.

[38] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[39] Farkhund Iqbal, Ahmed Abbasi, Abdul Rehman Javed, Zunera Jalil, and Jamal Al-Karaki. Deepfake audio detection via feature engineering and machine learning.

[40] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[41] Selver Ezgi Küçükbay, Adnan Yazıcı, and Sinan Kalkan. Hand-crafted versus learned representations for audio event detection. *Multimedia Tools and Applications*, 81(21):30911–30930, September 2022.

[42] Steven Camacho, Dora Maria Ballesteros, and Diego Renza. Fake speech recognition using deep learning. In Juan Carlos Figueroa-García, Yesid Díaz-Gutierrez, Elvis Eduardo Gaona-García, and Alvaro David Orjuela-Cañón, editors, *Applied Computer Sciences in Engineering*, pages 38–48, Cham, 2021. Springer International Publishing.

[43] Taiba Majid Wani, Syed Asif Ahmad Qadri, Danilo Comminiello, and Irene Amerini. Detecting audio deepfakes: Integrating cnn and bilstm with multi-feature concatenation, 2024.

[44] Karthik Sivarama Krishnan and Koushik Sivarama Krishnan. Mfaan: Unveiling audio deepfakes with a multi-feature authenticity network. In *2023 9th International Conference on Signal Processing and Communication (ICSC)*. IEEE, December 2023.

[45] Ameer Hamza, Abdul Rehman Javed, Farkhud Iqbal, Natalia Kryvinska, Ahmad Almadhor, Zunera Jalil, and Rouba Borghol. Deepfake audio detection via mfcc features using machine learning. *IEEE Access*, PP:1–1, 01 2022.

[46] Fawziya M. Rammo and Mohammed N. Al-Hamdani. Detecting the speaker language using cnn deep learning algorithm. *Iraqi Journal For Computer Science and Mathematics*, 3(1):43–52, Jan. 2022.

Appendix A

Feature Engineering Supplementary Results

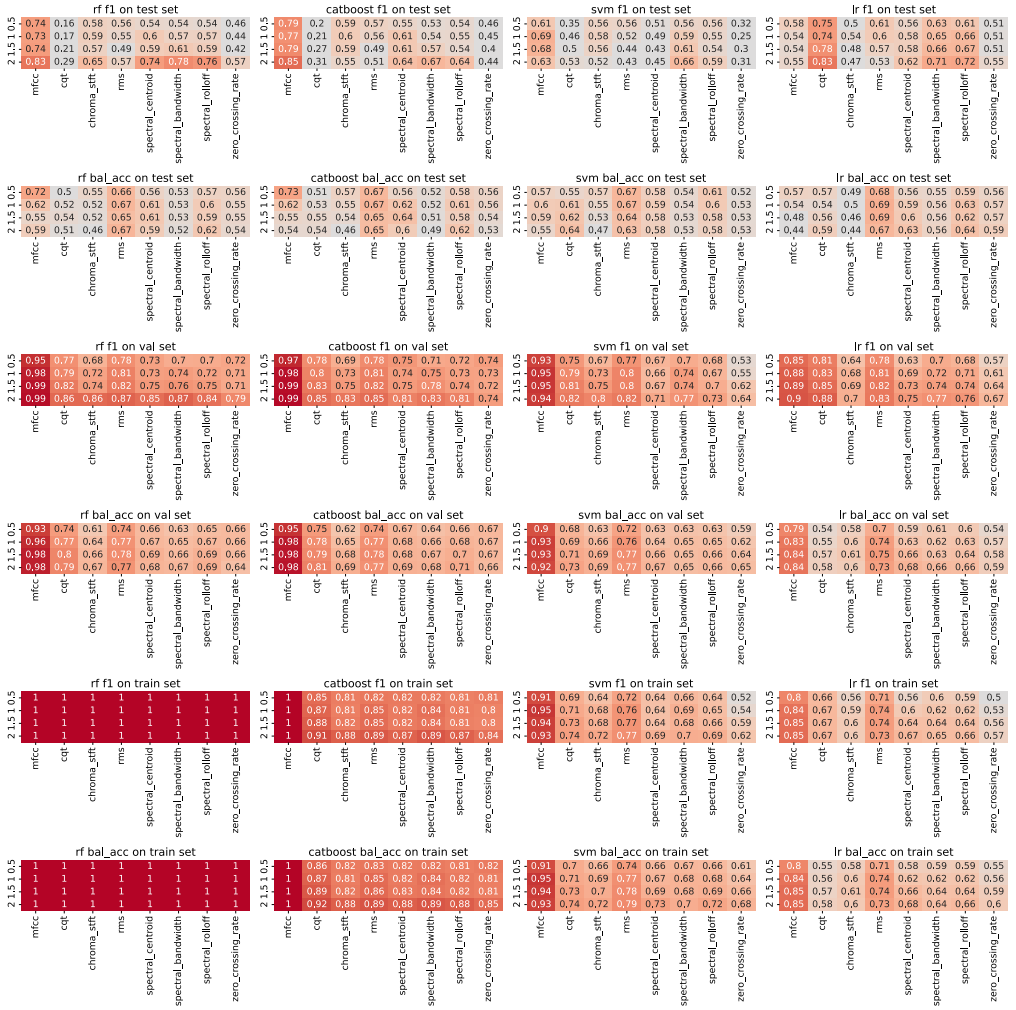


Figure A.1: Results of all the combinations of extraction intervals, feature types and models of the traditional ML approach, evaluated with 2 metrics on the three sets (train, test and validation). The num. samples is equalized across intervals. On y-axis, the extraction interval is shown.

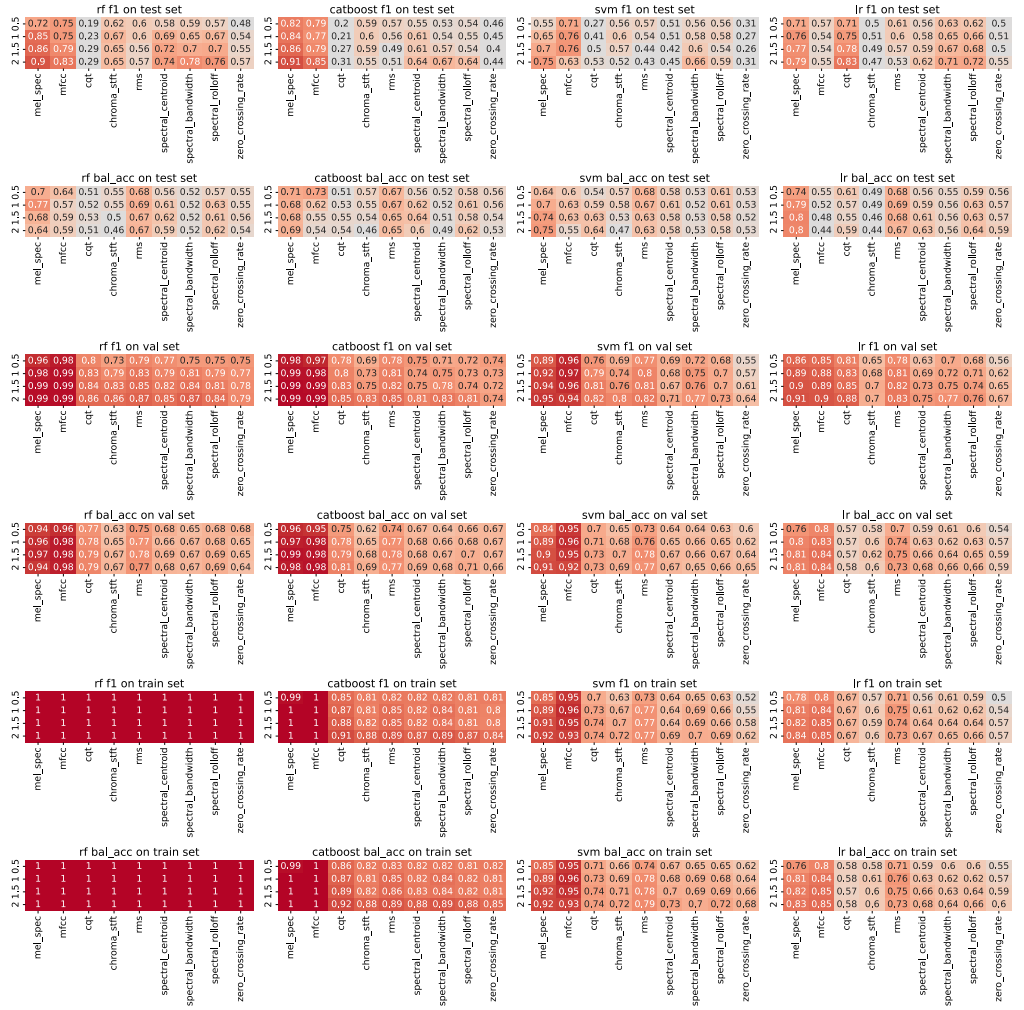


Figure A.2: Results of all the combinations of extraction intervals, feature types and models of the traditional ML approach, evaluated with 2 metrics on the three sets (train, test and validation). The num. samples is not equalized across intervals. On y-axis, the extraction interval is shown.

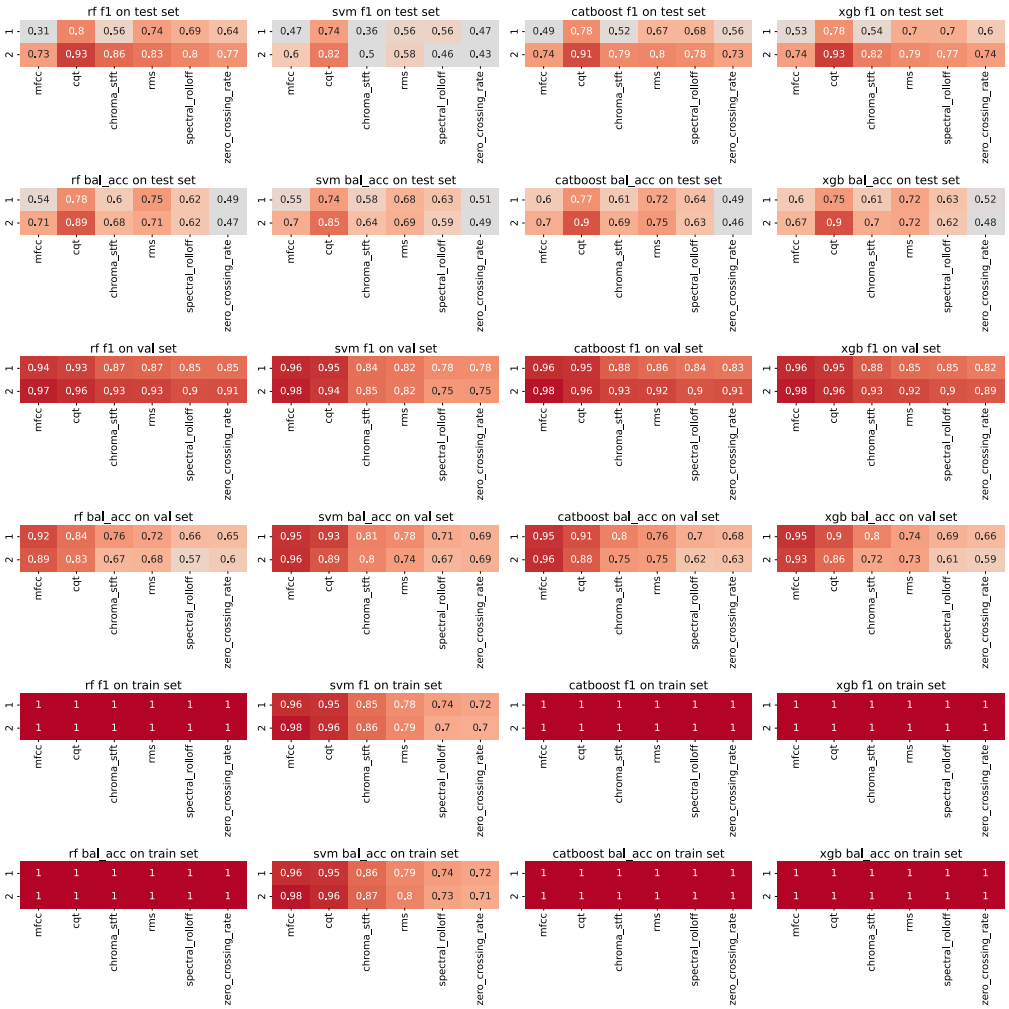


Figure A.3: Results of all the combinations of extraction intervals, feature types and models of the neural features approach based on MobileNetV3. The evaluation is made with 2 metrics on the three sets (train, test and validation). The images are normalized as illustrated in Listing 3.1. On y-axis, the extraction interval is shown.

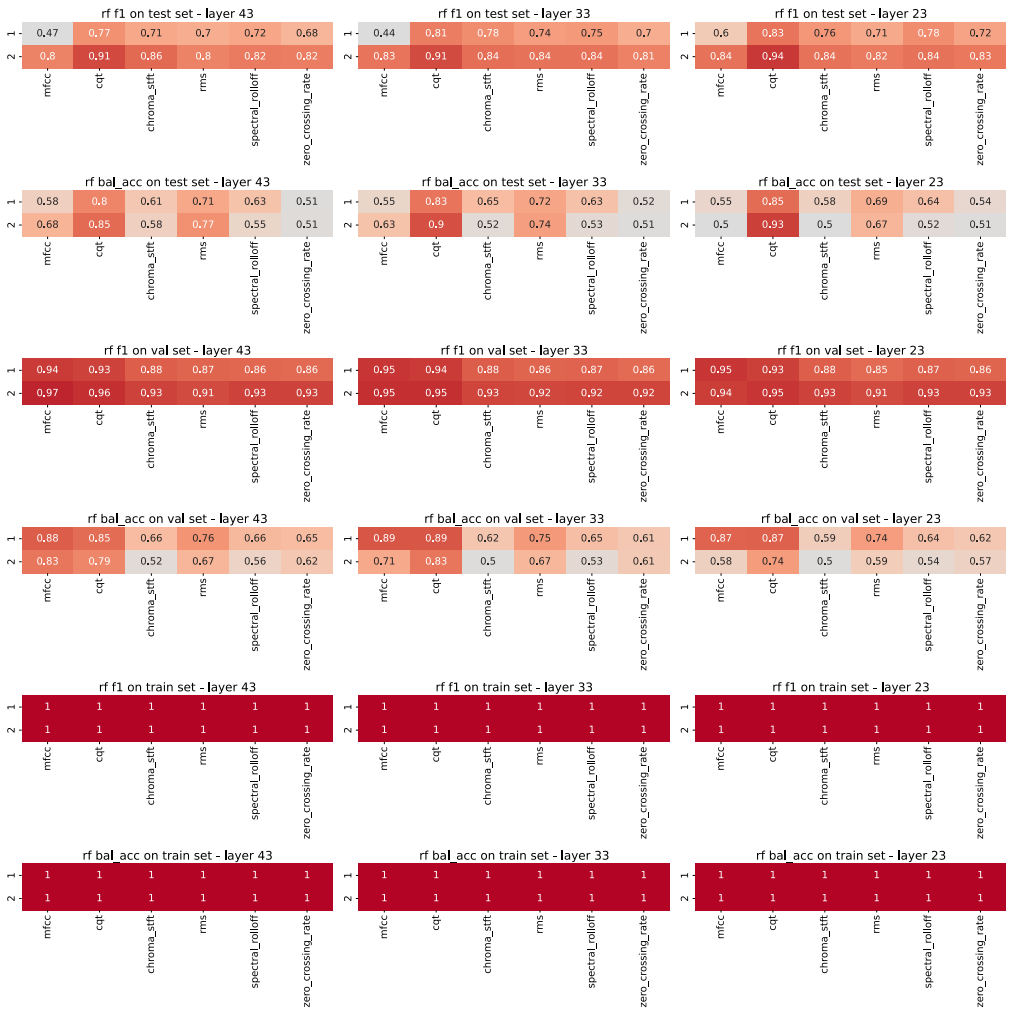


Figure A.4: Results of all the combinations of extraction intervals, feature types and models of the neural features approach based in VGG16. The evaluation is made with 2 metrics on the three sets (train, test and validation). The images are normalized as illustrated in Listing 3.1. On y-axis, the extraction interval is shown.

Appendix B

Models Supplementary Results

```
class Mapper3D(nn.Module):
    def __init__(self):
        super(Mapper3D, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1,
out_channels=64, kernel_size=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(in_channels=64,
out_channels=128, kernel_size=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(in_channels=128,
out_channels=3, kernel_size=1)
        self.bn3 = nn.BatchNorm2d(3)
        self.upsample = nn.Upsample(scale_factor=2,
mode='bilinear', align_corners=True)

    def forward(self, x): # Input: (B, 1, 112, 112)
        x = F.relu(self.bn1(self.conv1(x))) # (B, 64,
112, 112)
        x = F.relu(self.bn2(self.conv2(x))) # (B, 128,
112, 112)
        x = self.bn3(self.conv3(x)) # (B, 3, 112, 112)
        x = torch.sigmoid(x) # scale to [0, 1]
        x = self.upsample(x) # (B, 3, 224, 224)
        x = x * 255.0 # scale to [0, 255]
        return x
```

Listing B.1: 3D Mapper CNN Implementation (64-128-3-sigmoid)

```
class Mapper3D(nn.Module):
    def __init__(self):
        super(Mapper3D, self).__init__()
```

```
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(in_channels=128, out_channels=3, kernel_size=1)
        self.bn3 = nn.BatchNorm2d(3)
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.scale = nn.Parameter(torch.ones(1, 3, 1, 1))  # Learnable scale factor
        self.shift = nn.Parameter(torch.zeros(1, 3, 1, 1))  # Learnable shift factor

    def forward(self, x): # Input: (B, 1, 112, 112)
        x = F.relu(self.bn1(self.conv1(x))) # (B, 64, 112, 112)
        x = F.relu(self.bn2(self.conv2(x))) # (B, 128, 112, 112)
        x = self.bn3(self.conv3(x)) # (B, 3, 112, 112)
        x = self.scale * x + self.shift # (B, 3, 112, 112)
        x = torch.sigmoid(x) # scale to [0, 1]
        x = self.upsample(x) # (B, 3, 224, 224)
        x = x * 255.0 # scale to [0, 255]
        return x
```

Listing B.2: 3D Mapper CNN Implementation (64-128-3-learnable)

```
class Mapper3D(nn.Module):
    def __init__(self):
        super(Mapper3D, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(in_channels=128, out_channels=3, kernel_size=1)
        self.bn3 = nn.BatchNorm2d(3)
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)

```

```
        def forward(self, x): # Input: (B, 1, 112, 112)
            x = F.relu(self.bn1(self.conv1(x))) # (B, 64, 112, 112)
            x = F.relu(self.bn2(self.conv2(x))) # (B, 128, 112, 112)
            x = self.bn3(self.conv3(x)) # (B, 3, 112, 112)
            x = self.upsample(x) # (B, 3, 224, 224)
            return x
```

Listing B.3: 3D Mapper CNN Implementation (64-128-3-none)

```
class Mapper3D(nn.Module):
    def __init__(self):
        super(Mapper3D, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=5, padding=2)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=3, kernel_size=1)
        self.bn2 = nn.BatchNorm2d(3)
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)

    def forward(self, x): # Input: (B, 1, 112, 112)
        x = F.relu(self.bn1(self.conv1(x))) # (B, 64, 112, 112)
        x = F.relu(self.bn2(self.conv2(x))) # (B, 3, 112, 112)
        x = torch.sigmoid(x) # scale to [0, 1]
        x = self.upsample(x) # (B, 3, 224, 224)
        x = x * 255.0 # scale to [0, 255]
        return x
```

Listing B.4: 3D Mapper CNN Implementation (64x5-3x1-sigmoid)

```
class Mapper3D(nn.Module):
    def __init__(self):
        super(Mapper3D, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=3, kernel_size=1)
        self.bn1 = nn.BatchNorm2d(3)
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)

    def forward(self, x): # Input: (B, 1, 112, 112)
        x = F.relu(self.bn1(self.conv1(x))) # (B, 3, 112, 112)
```

```
            x = torch.sigmoid(x) # scale to [0, 1]
            x = self.upsample(x) # (B, 3, 224, 224)
            x = x * 255.0 # scale to [0, 255]
            return x
```

Listing B.5: 3D Mapper CNN Implementation (3-1-sigmoid)

Model Name	Mapping	Features	Epoch	Bal Acc (%)		
				Train	Val	Test
64-128-3-sigmoid	CNN	CQT	7	95.1	91.9	88.3
64-128-3-sigmoid	CNN	MFCC	10	98.1	97.7	59.5
64-128-3-learnable	CNN	CQT	9	98.9	96.6	89.3
64-128-3-learnable	CNN	MFCC	10	97.4	97.5	65.2
64-128-3-none	CNN	CQT	8	97.9	96.8	78.7
64-128-3-none	CNN	MFCC	10	98.6	96.3	63.3
64x5-3x1-sigmoid	CNN	CQT	8	94.5	90.7	90.1
64x5-3x1-sigmoid	CNN	MFCC	7	97.8	97.7	55.3
3-1-sigmoid	CNN	CQT	8	97.1	95.8	86.5
3-1-sigmoid	CNN	MFCC	6	96.8	92.6	76.1
5-3-random	LUT	CQT	9	98.2	96.9	83.5
5-3-random	LUT	MFCC	7	97.4	94.7	66.6
10-3-random	LUT	CQT	7	97.4	95.5	84.2
10-3-random	LUT	MFCC	8	98.2	93.4	67.1
20-3-random	LUT	CQT	10	97.4	96.3	74.5
20-3-random	LUT	MFCC	7	95.7	93.4	56.2
40-3-random	LUT	CQT	9	97.7	93.9	81.9
40-3-random	LUT	MFCC	5	93.4	91.1	64.5
10-3-linear	LUT	CQT	7	97.8	94.8	92.1
10-3-linear	LUT	MFCC	9	97.3	95.9	61.3
10-1.5-linear	LUT	CQT	8	96.4	94.7	88.9
10-1.5-linear	LUT	MFCC	10	98.2	96.9	72.1

Table B.1: Results of All Tested 3D Mapper Configurations and Features

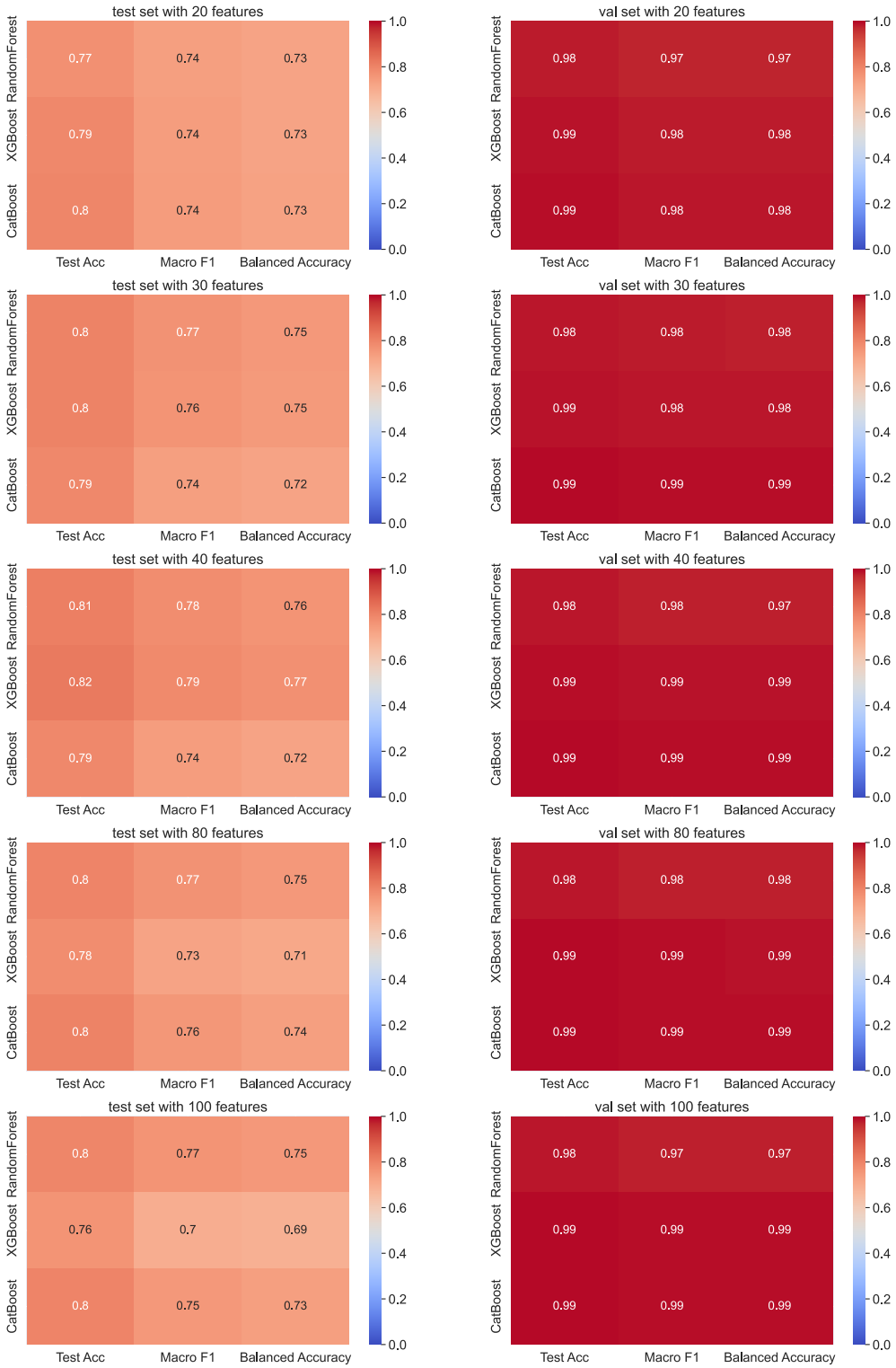


Figure B.1: Results of traditional ML, concatenating all the features (20 per type) including mel-spectrogram and reduced to 20, 30, 40, 80 and 100 features using RFE.

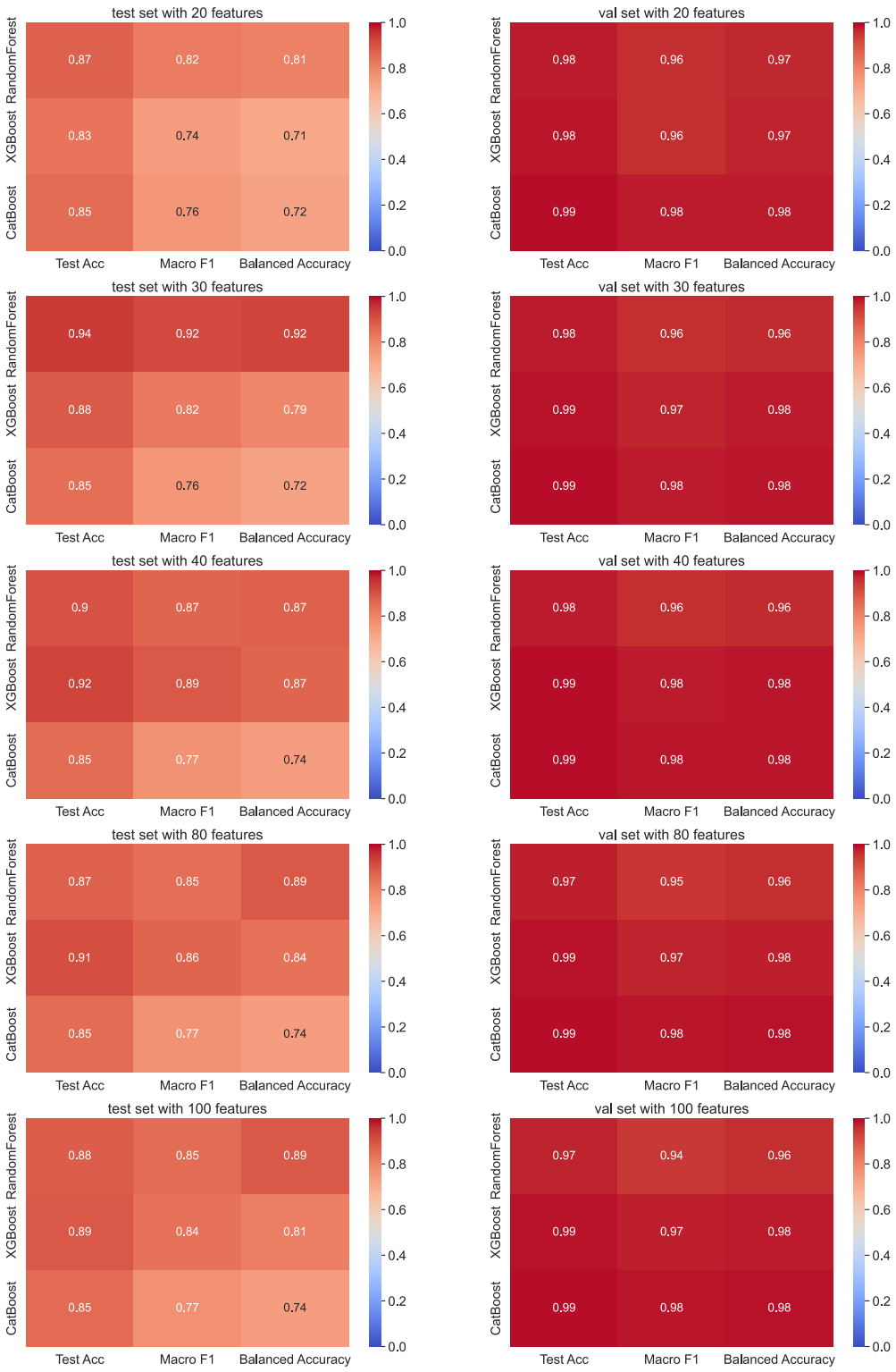


Figure B.2: Results of traditional ML, concatenating all the features (20 per type) excluding mel-spectrogram and reduced to 20, 30, 40, 80 and 100 features using RFE.

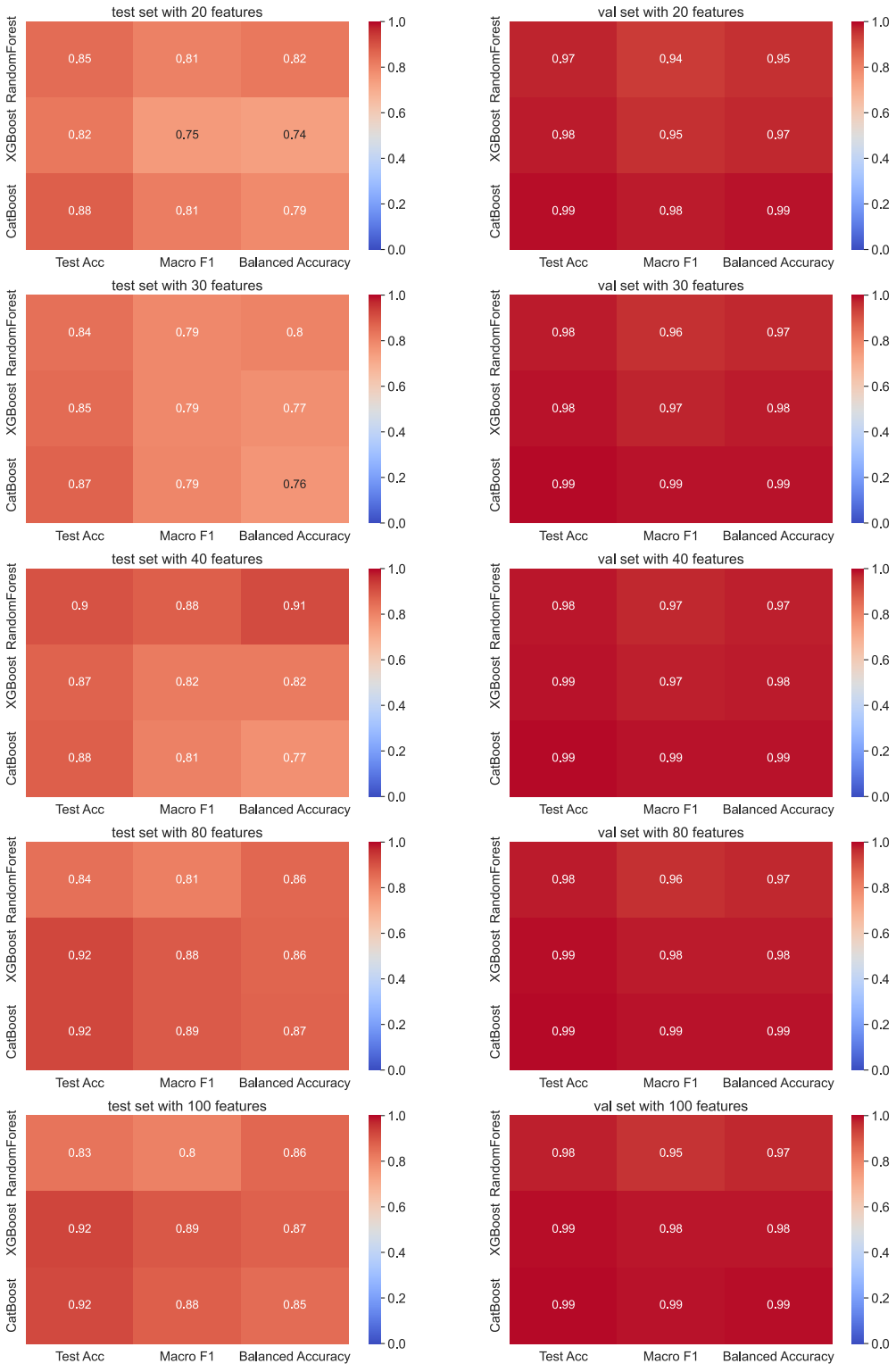
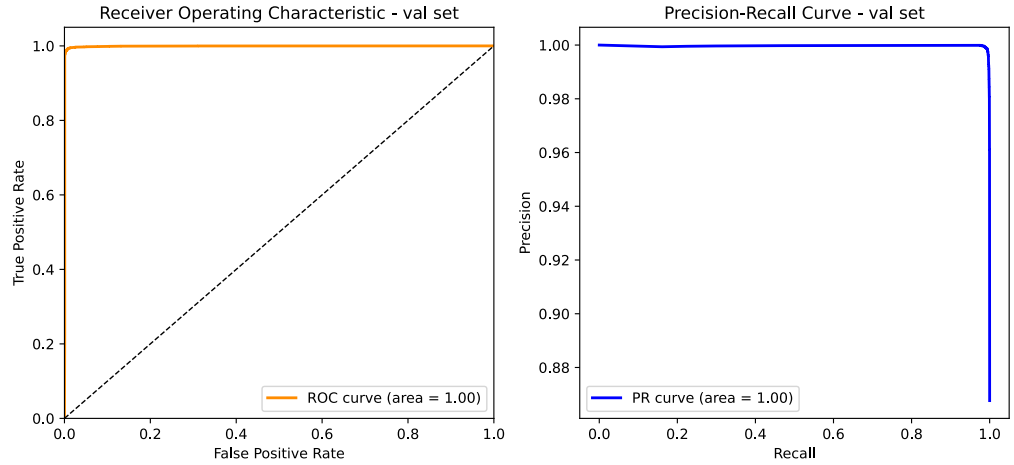


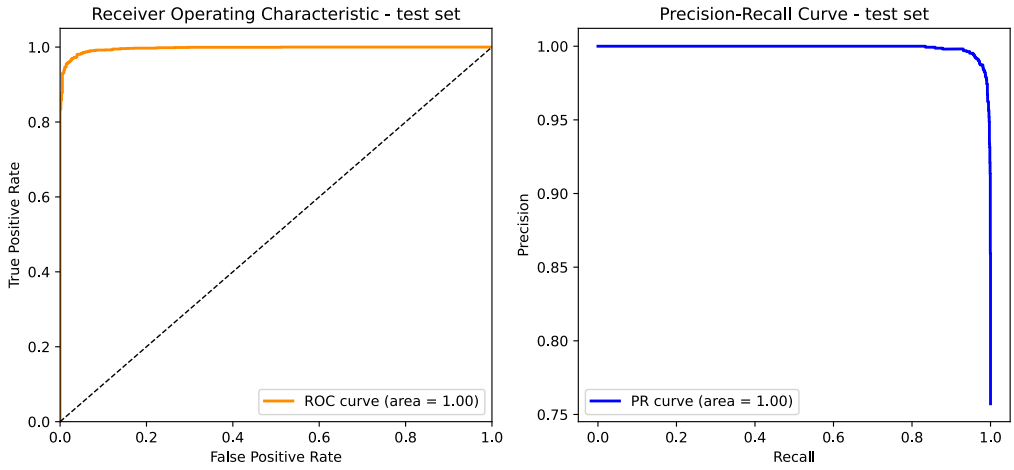
Figure B.3: Results of traditional ML, concatenating all the features (40 per type) excluding mel-spectrogram and reduced to 20, 30, 40, 80 and 100 features using RFE.

Appendix C

Explainability Supplementary Results



(a)



(b)

Figure C.1: ROC and PR curves of DeepSpectraNetLite on: a) Validation Set, b) Test Set.

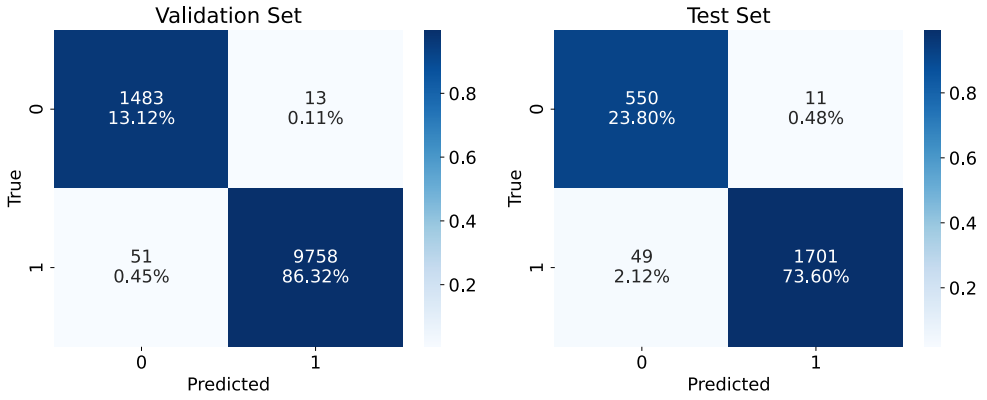
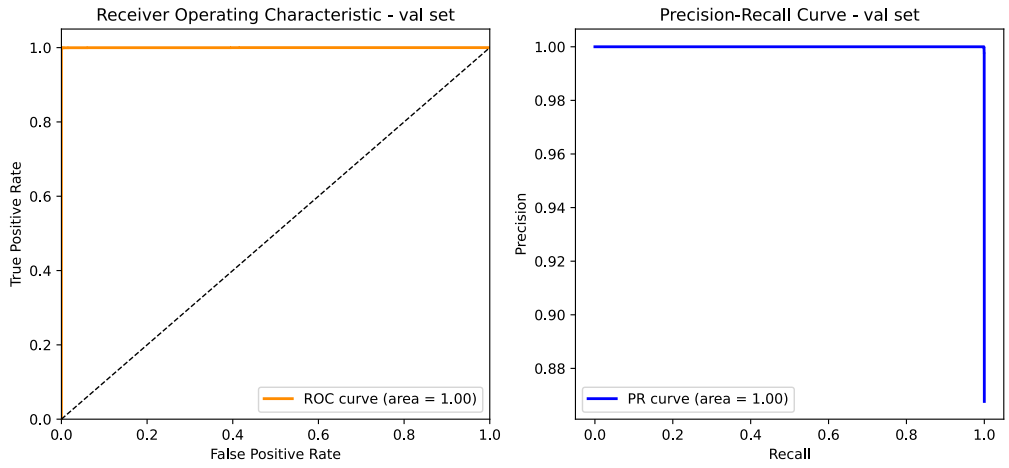
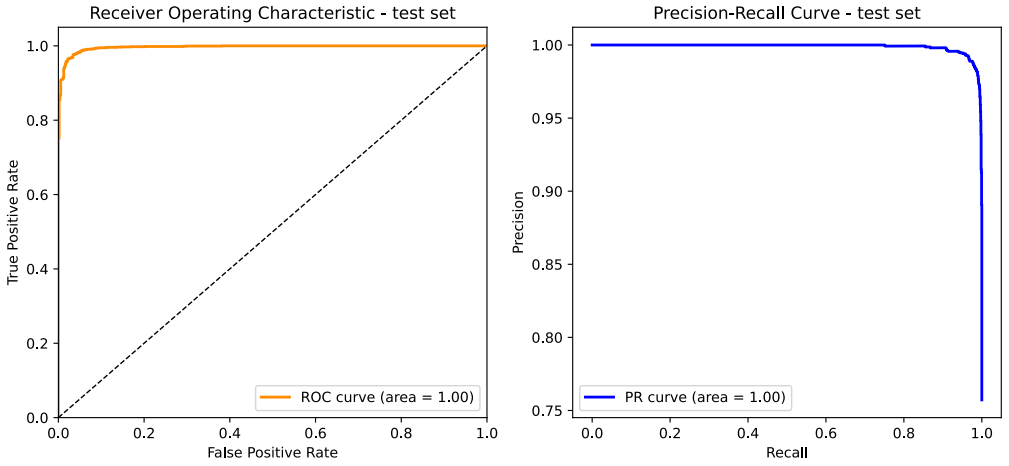


Figure C.2: Confusion Matrix of DeepSpectraNetLite



(a)



(b)

Figure C.3: ROC and PR curves of DeepSpectraNetFlex on: a) Validation Set, b) Test Set.

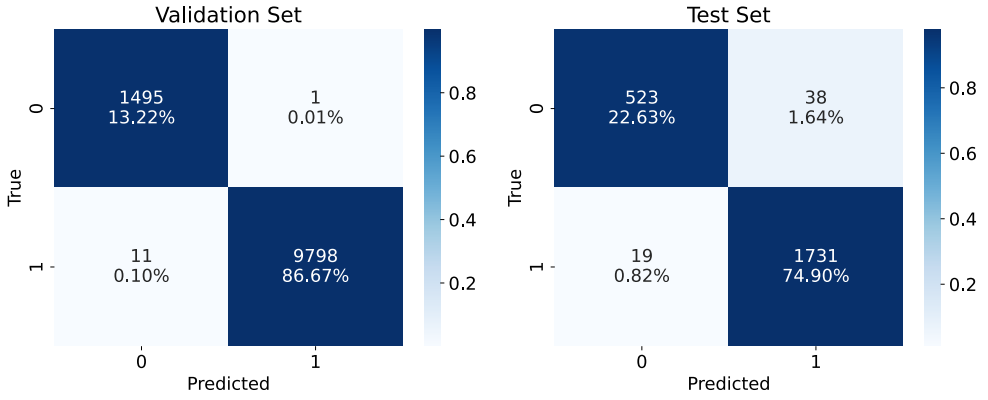
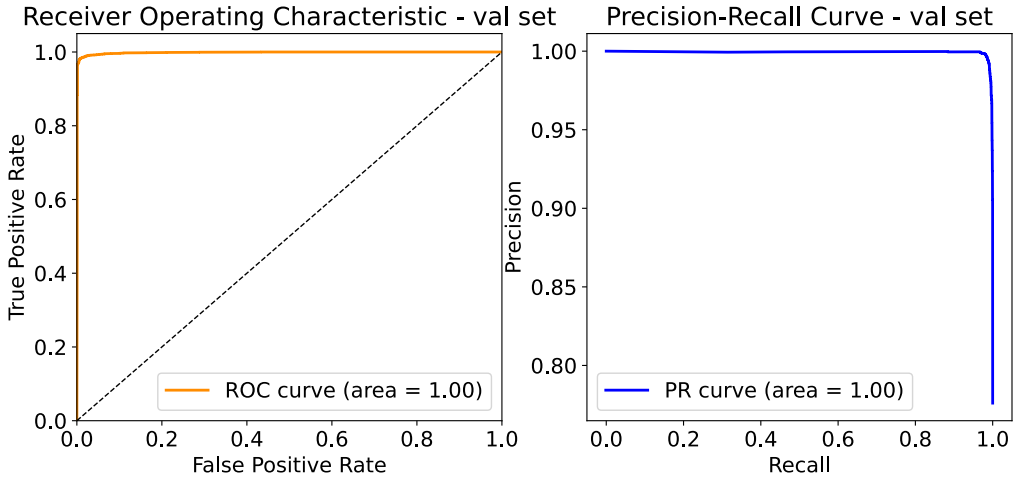
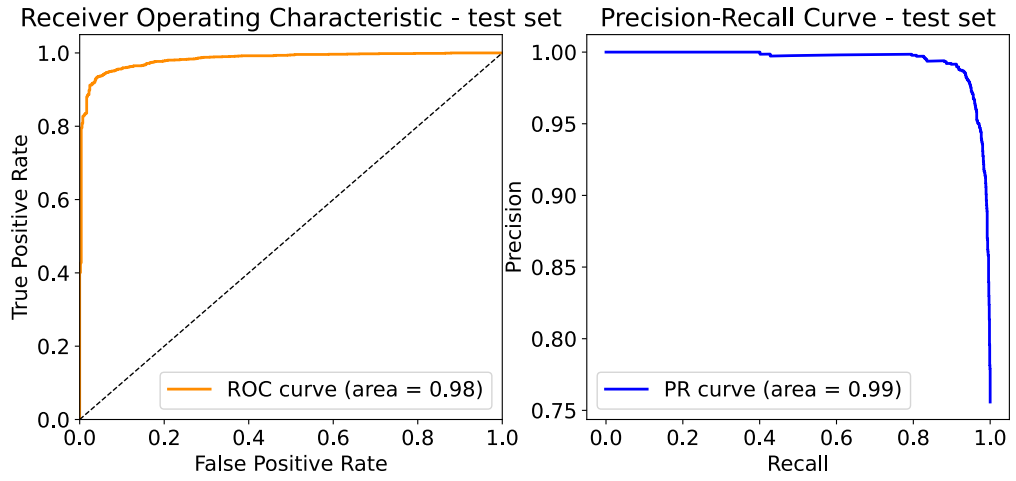


Figure C.4: Confusion Matrix of DeepSpectraNetFlex



(a)



(b)

Figure C.5: ROC and PR curves of DeepSpectraNetE2E on: a) Validation Set, b) Test Set.

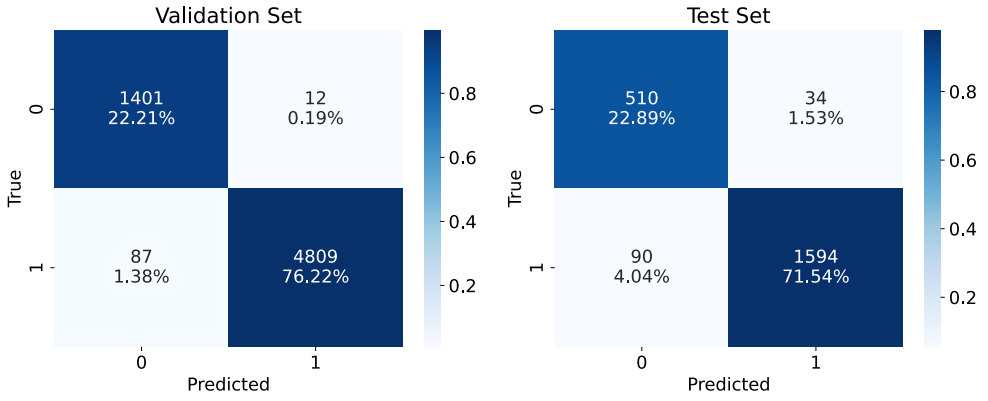


Figure C.6: Confusion Matrix of DeepSpectraNetE2E

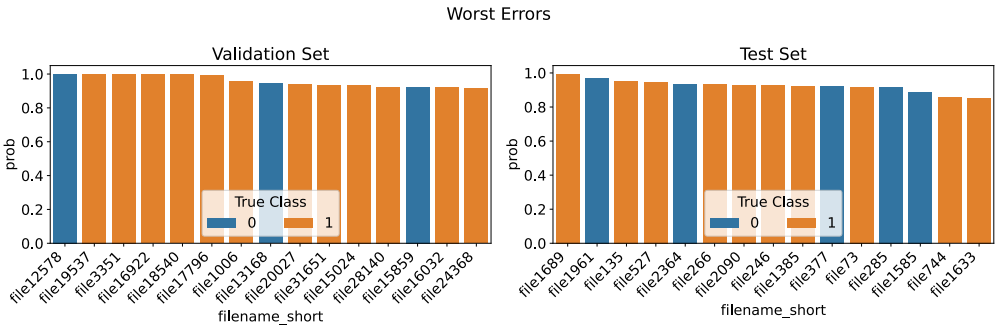


Figure C.7: Worst Errors of DeepSpectraNetLite

Figure C.8: Worst Errors of DeepSpectraNetFlex

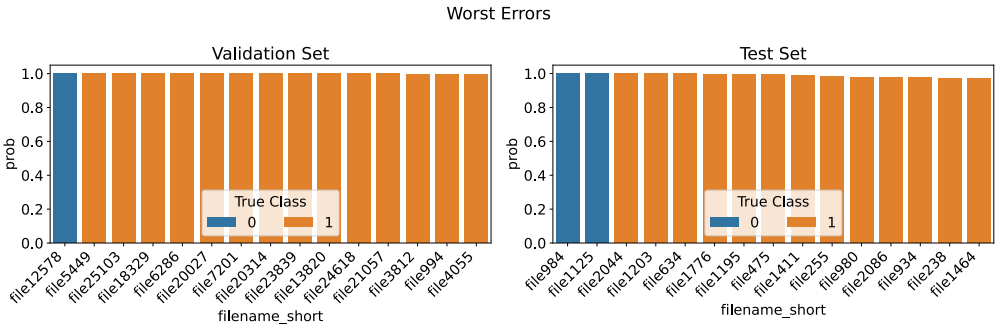


Figure C.9: Worst Errors of DeepSpectraNetE2E

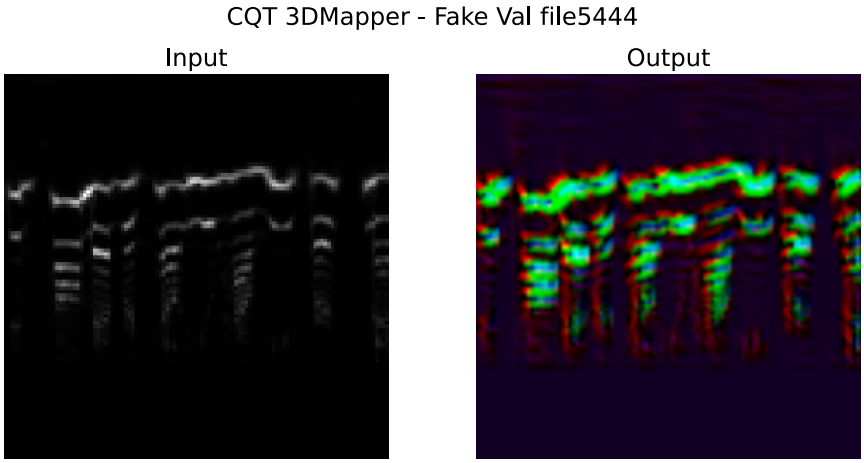


Figure C.10: DeepSpectraNetLite Mapper Output for a Random CQT Fake Audio Sample (Validation Set)

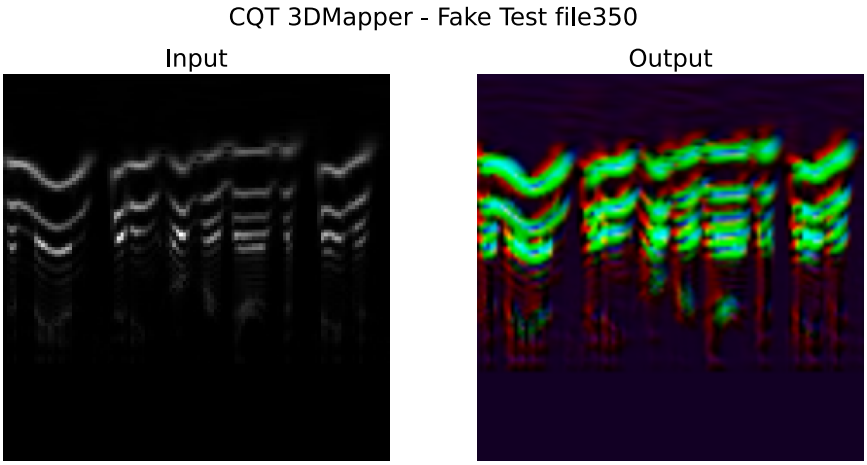


Figure C.11: DeepSpectraNetLite Mapper Output for a Random CQT Fake Audio Sample (Test Set)

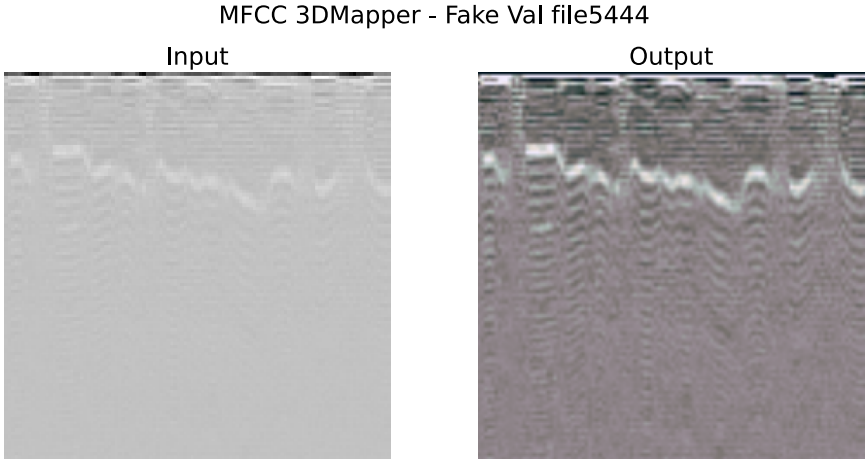


Figure C.12: DeepSpectraNetLite Mapper Output for a Random MFCC Fake Audio Sample (Validation Set)

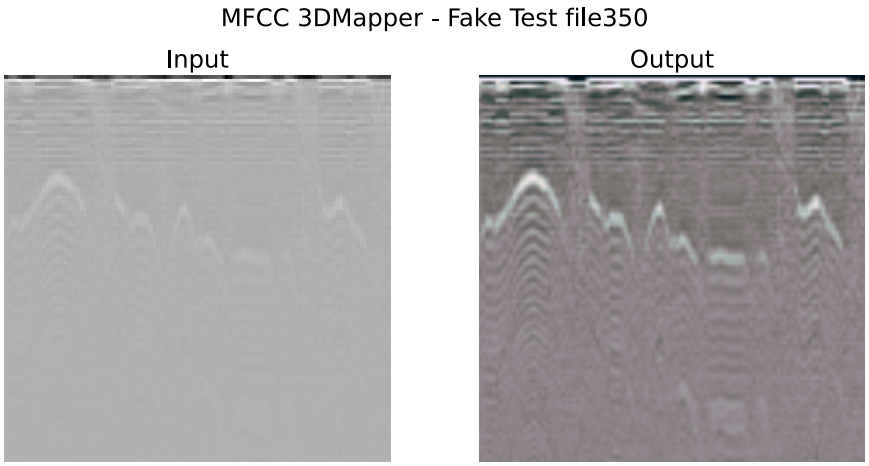


Figure C.13: DeepSpectraNetLite Mapper Output for a Random MFCC Fake Audio Sample (Test Set)

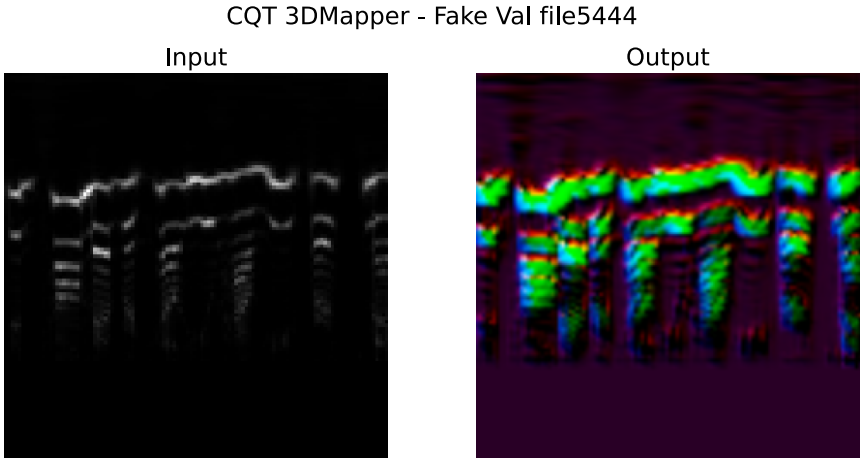


Figure C.14: DeepSpectraNetFlex Mapper Output for a Random CQT Fake Audio Sample (Validation Set)

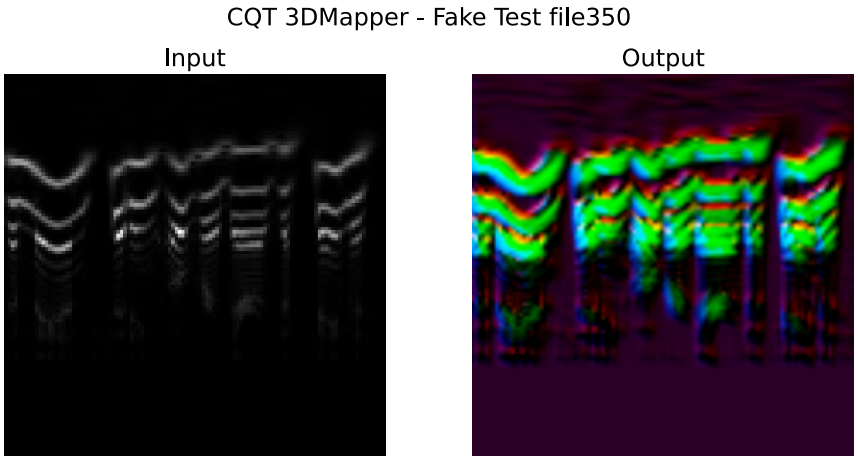


Figure C.15: DeepSpectraNetFlex Mapper Output for a Random CQT Fake Audio Sample (Test Set)

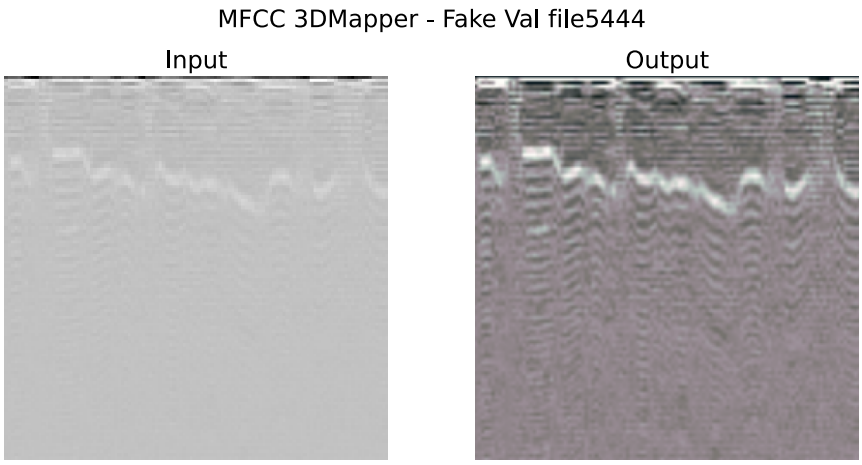


Figure C.16: DeepSpectraNetFlex Mapper Output for a Random MFCC Fake Audio Sample (Validation Set)

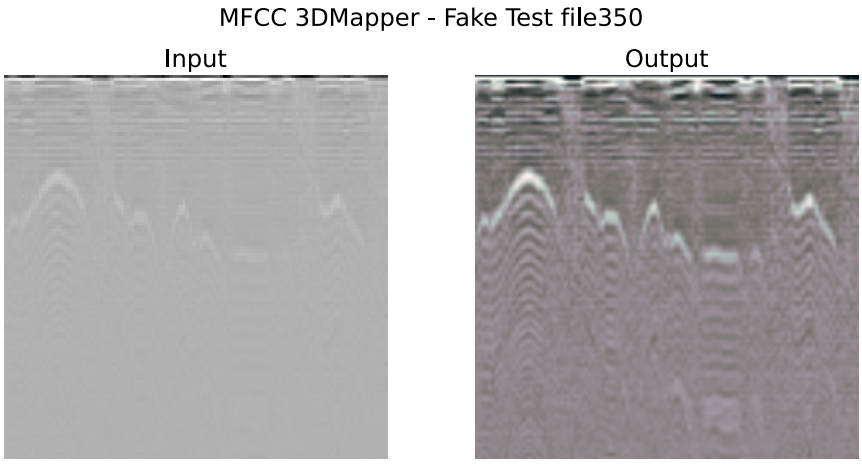


Figure C.17: DeepSpectraNetFlex Mapper Output for a Random MFCC Fake Audio Sample (Test Set)

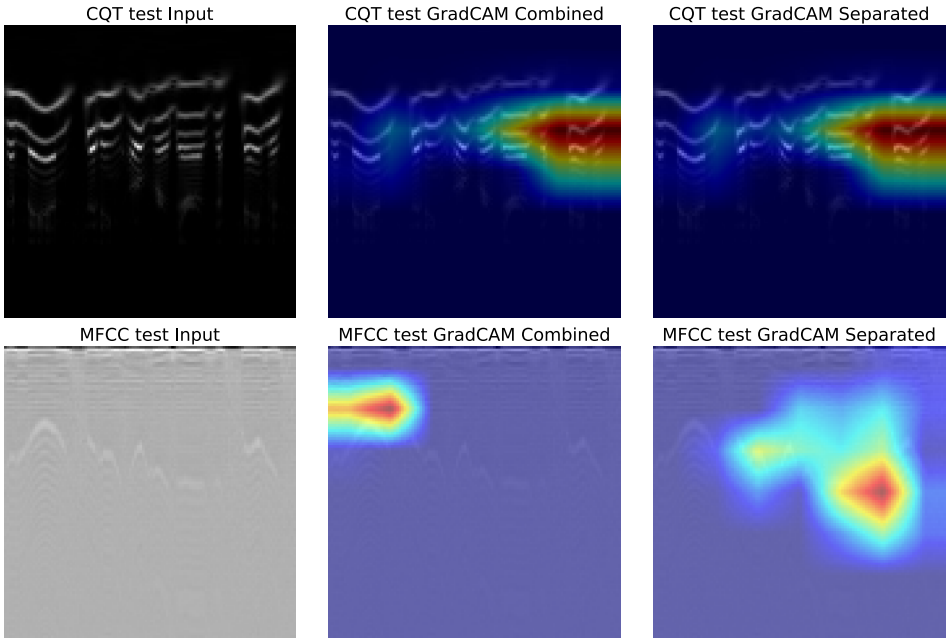


Figure C.18: Grad-CAM Analysis of DeepSpectraNetLite on a Test Set Sample. Grad-CAM Combined refers to the combined model which takes both MFCC and CQT features as input, while Grad-CAM Separated is obtained splitting the combined model into two separate models, one for each feature set.

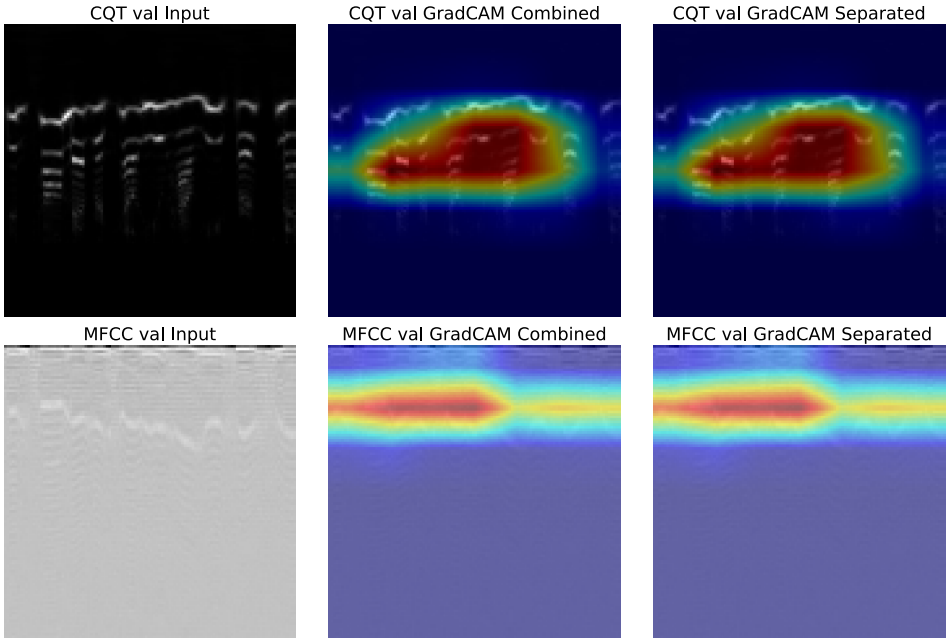


Figure C.19: Grad-CAM Analysis of DeepSpectraNetLite on a Validation Set Sample. Grad-CAM Combined refers to the combined model which takes both MFCC and CQT features as input, while Grad-CAM Separated is obtained splitting the combined model into two separate models, one for each feature set.

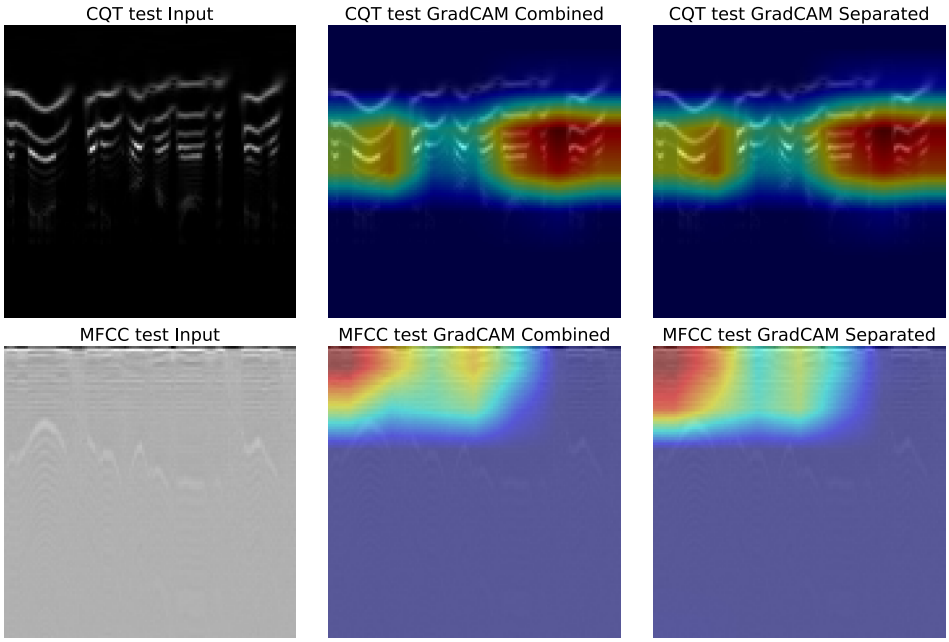


Figure C.20: Grad-CAM Analysis of DeepSpectraNetFlex on a Test Set Sample. Grad-CAM Combined refers to the combined model which takes both MFCC and CQT features as input, while Grad-CAM Separated is obtained splitting the combined model into two separate models, one for each feature set.

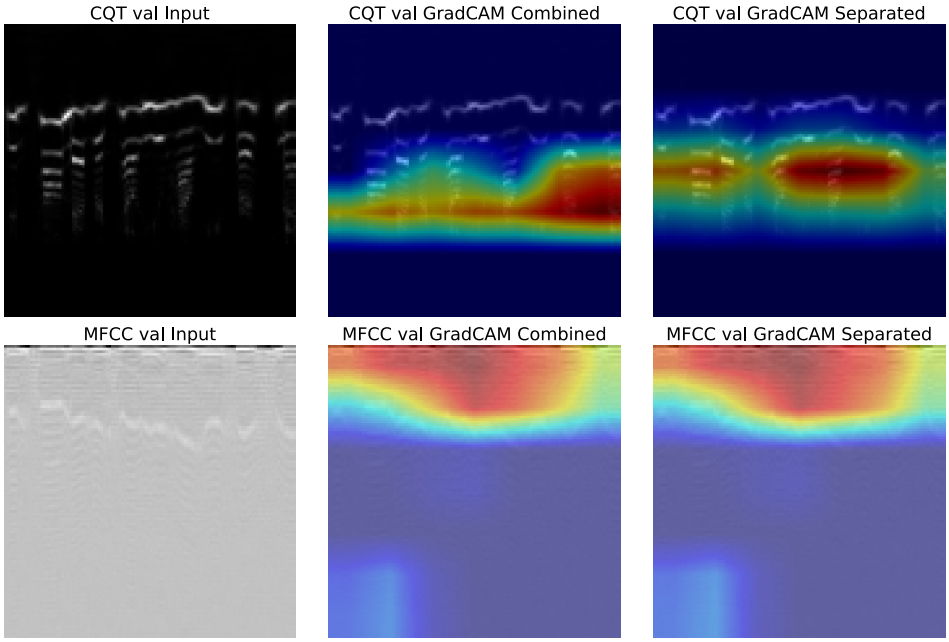
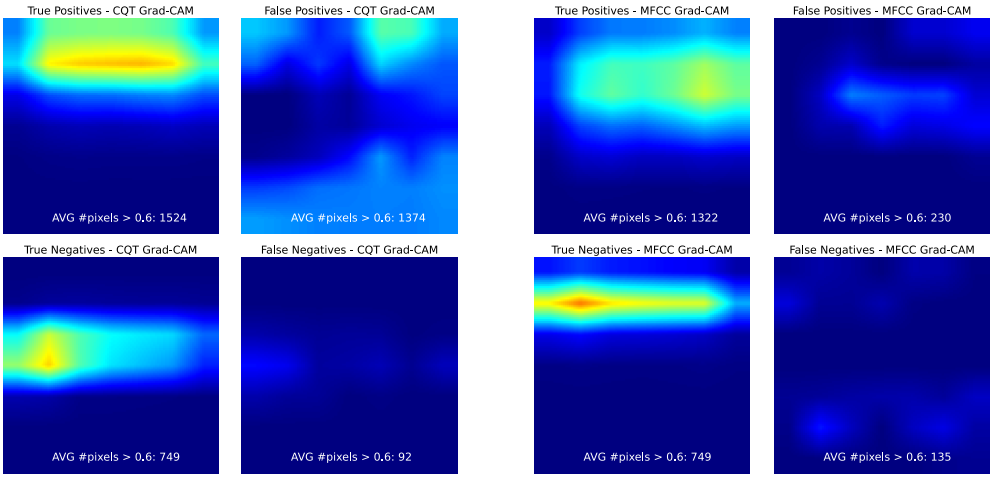
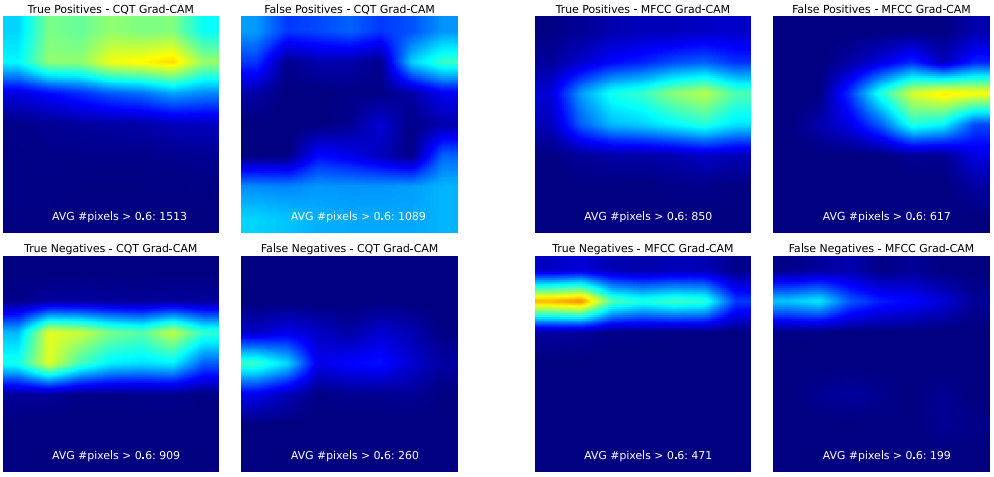


Figure C.21: Grad-CAM Analysis of DeepSpectraNetFlex on a Validation Set Sample. Grad-CAM Combined refers to the combined model which takes both MFCC and CQT features as input, while Grad-CAM Separated is obtained splitting the combined model into two separate models, one for each feature set.



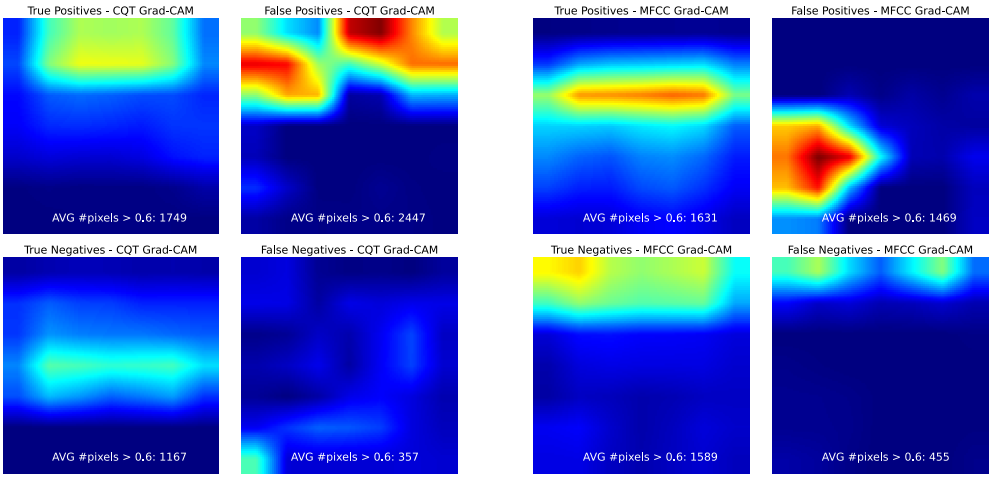
(a) (b)

Figure C.22: Average Grad-CAM of DeepSpectraNetLite on Validation Set Subset splitted in TP (upper left), TN (lower left), FP (upper right), FN (lower right) samples. a) Results for the CQT features, b) Results for the MFCC features.



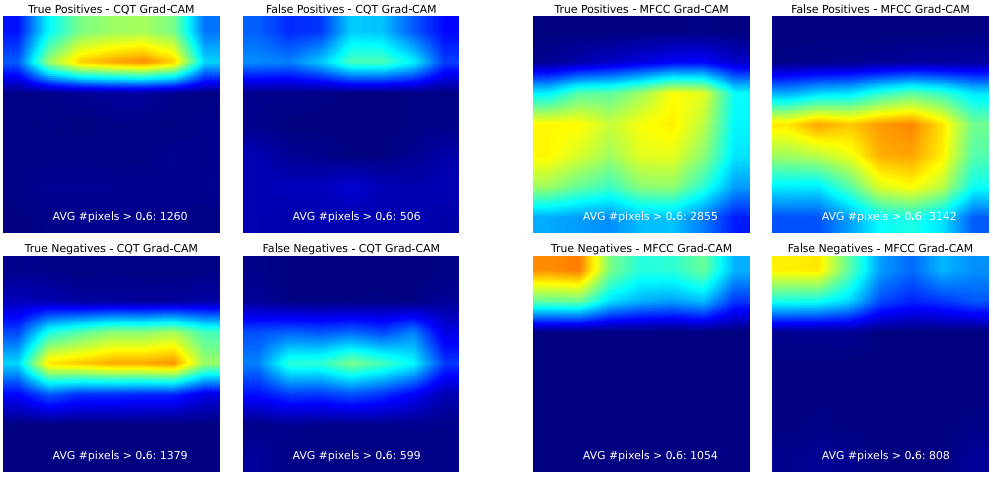
(a) (b)

Figure C.23: Average Grad-CAM of DeepSpectraNetLite on Test Set Subset splitted in TP (upper left), TN (lower left), FP (upper right), FN (lower right) samples. a) Results for the CQT features, b) Results for the MFCC features.



(a) (b)

Figure C.24: Average Grad-CAM of DeepSpectraNetFlex on Validation Set Subset splitted in TP (upper left), TN (lower left), FP (upper right), FN (lower right) samples. a) Results for the CQT features, b) Results for the MFCC features.



(a) (b)

Figure C.25: Average Grad-CAM of DeepSpectraNetFlex on Test Set Subset splitted in TP (upper left), TN (lower left), FP (upper right), FN (lower right) samples. a) Results for the CQT features, b) Results for the MFCC features.