Enhancing Real-Time Voice Authentication: Leveraging Deep Learning to Combat Spoofing and Verify Speaker Identity

Miles Herrman
California Polytechnic State University - San Luis Obispo
Computer Science Senior Project

Abstract

Voice spoofing attacks pose a significant threat to systems that rely on speaker recognition for authentication. This project investigates the feasibility of developing a real-time voice authentication system for live voice calls using deep learning techniques. Two binary classification models were trained: one to distinguish between spoofed and genuine speech, and another to verify the identity of a specific target speaker . A proof of concept was successfully implemented, demonstrating the potential of this approach for enhancing security in voice call scenarios. However, further research is needed to address challenges related to real-time implementation, model optimization, and generalization to handle real-world conditions. This work provides a foundation for future development of robust and secure voice authentication systems for telecommunication applications.

1. Introduction

1.1 Background

The landscape of communication is undergoing a significant shift. Voice-based interfaces and authentication methods are becoming increasingly prevalent, from interacting with virtual assistants on smart devices to voice verification systems in call centers and financial institutions.

While these advancements offer convenience and accessibility, they also introduce new security vulnerabilities: the ability to manipulate or fabricate voices for malicious purposes, known as voice spoofing.

Voice spoofing is the use of technology to alter, create, or imitate authentic voices (truly spoken by a human) (Boyd, Fahim, & Olukoya, 2023). This practice encompasses various techniques, including voice conversion, speech synthesis, and recorded replay attacks. We will explore each in the following section, but when used effectively, these sophisticated methods can deceive both automated systems and human listeners, posing a significant threat to security and trust in communication.

The risk posed by these practices underscores the critical importance of robust voice recognition and spoofing detection systems.

Voice recognition is a technology or process that involves identifying or verifying a person based on their unique vocal characteristics (Juang et al., 2003). It generally falls into two categories:

1. **Speaker Identification**: Determining which individual is speaking.
2. **Speaker Verification**: Confirming that the speaker's voice matches a claimed identity.

Within the realm of speaker verification, **Automatic Speaker Verification (ASV)** plays a pivotal role. ASV systems employ advanced algorithms to automatically verify whether a speaker's voice aligns with a pre-established voiceprint, ensuring accurate and reliable authentication.

Complementing these recognition capabilities is **voice spoofing detection**, a specialized technology designed to ascertain whether a voice input is genuine or has been artificially manipulated. While voice recognition systems focus on identifying or verifying the speaker, spoofing detection adds an essential layer of security by ensuring that the voice input has not been synthesized, altered, or otherwise fabricated. Together, ASV and voice spoofing detection provide a framework for secure and trustworthy voice-based authentication, safeguarding against both identity verification breaches and malicious spoofing attempts

The relevance of voice spoofing detection is especially critical in voice call environments, where voice remains the primary medium of communication. Call centers, customer service lines, and even automated phone systems increasingly rely on voice recognition technologies for authentication and service delivery (Clark, 2022). In these contexts, a successful voice spoofing attack can have immediate and severe repercussions. For instance, fraudsters might leverage spoofed voices to bypass security checks during customer support interactions, impersonating legitimate users to reset passwords, access accounts, or authorize transactions. In industries like banking, healthcare, and telecoms, the stakes are particularly high, as these sectors handle sensitive financial and personal data. Robust voice spoofing detection mechanisms are therefore essential to prevent unauthorized access and protect against financial loss, identity theft, and data breaches. Additionally, businesses that depend on voice calls for decision-making processes—such as approving large transactions or verifying executive instructions—must be particularly vigilant. Attackers using spoofed voices could manipulate these processes, leading to fraud or internal sabotage.

Furthermore, detecting spoofed speech is critical for combating social engineering scams. Attackers can impersonate trusted voices, such as those of company executives or family members, to manipulate victims into revealing confidential information or transferring funds (Espinosa & Espinosa, 2024). The ability to identify these attempts can safeguard individuals and organizations from significant financial and reputational harm. This is especially crucial for businesses that rely on voice verification for decision-making processes.

The rapid advancements in deep learning models, particularly in speech synthesis, are creating highly realistic synthetic voices. There are a variety of speech synthesis tools available to the public, such as ElevenLabs and Murf.ai. While beneficial applications exist, these advancements also pose a challenge as they facilitate more sophisticated spoofing attacks. Detecting spoofed speech is essential to mitigate the risks associated with these technological developments and ensure that voice interfaces remain trustworthy.

Main Types of Spoofing Attacks:

> **Replay** attacks are the simplest and most prevalent tactic. Attackers trick the target into uttering specific keywords or phrases during a call such as “yes”, “no”, and their full name. These recordings can then be spliced together to create a seemingly authentic voice that can be used to authorize fraudulent transactions or gain access to sensitive information.
>
> **Speech synthesis** attacks leverage text-to-speech technology to generate realistic voices. While not perfect, these synthetic voices can mimic accents, speech impediments, and even subtle vocal nuances. This makes them more convincing than simple replays, particularly for systems that rely on less complex voice analysis
>
> **Voice conversion** attacks, powered by deep learning techniques, leverage a machine learning model to transform the attacker's voice into one that closely resembles the target's. This approach utilizes a combination of autoencoders (to capture the essence of the target voice) and sequence-to-sequence neural networks (to map the attacker's voice onto that learned essence).

The rapid advancements in synthetic speech generators are producing increasingly realistic and indistinguishable audio, making it difficult to differentiate between genuine human speech and manipulated or fabricated voices (Gupta et al., 2024). Spoofers can now customize synthetic speech to mimic specific individuals or accents, further complicating detection and amplifying the risk of unauthorized access, fraud, or identity theft. As spoofing techniques continue to evolve—with the development of new methods and adversarial attacks—detection systems must also continuously adapt and improve to stay ahead of emerging threats.

The Role of AI/ML in Spoofed Speech Detection

The convergence of artificial intelligence (AI) and machine learning (ML) with cybersecurity is driving a significant shift in the way organizations detect and prevent spoofed speech attacks. AI/ML technologies have become fundamental tools in combating this threat by enabling more accurate and scalable detection methods. Traditional detection approaches struggle against the sophistication of modern voice spoofing techniques; however, AI and ML models excel at identifying subtle patterns and anomalies that are otherwise imperceptible (Khan et al., 2023).

In the context of spoofed speech detection, AI/ML algorithms can be trained on extensive datasets containing both genuine and spoofed voice samples. Through this process, models learn complex features such as intonation, pitch, and other acoustic properties that differentiate authentic speech from synthetic or altered voices. One of the primary strengths of AI/ML models in this domain is their ability to adapt and generalize—allowing them to keep up with new and evolving spoofing techniques.

Moreover, the automation enabled by AI/ML brings another key advantage: real-time detection (Costa, 2024). In scenarios such as live customer service calls, automated systems can analyze speech on the fly and flag potentially spoofed audio, minimizing the window for malicious actors to carry out attacks. This real-time capability reduces the burden on human operators, enabling faster responses and more secure communication channels.

In summary, AI/ML technologies are not only enhancing general cybersecurity but are specifically revolutionizing the fight against spoofed speech attacks. By leveraging advanced models and automating detection processes, organizations can significantly bolster their defenses, ensuring that voice-based authentication systems remain secure and trustworthy in an increasingly digital landscape.

Note: For readers who may not be familiar with audio processing, spoofing techniques, or machine learning concepts, relevant definitions and explanations can be found in the appendices at the end of this paper.

1.2 Objective

Specific Problem

The increasing sophistication of speech spoofing techniques poses a significant threat to the security and integrity of voice-based authentication and authorization processes. As these techniques advance, it becomes increasingly difficult to distinguish between genuine and spoofed audio, making it easier for malicious actors to deceive systems and individuals.

Primary Goal

The primary goal of my project is to leverage machine learning to provide speaker verification, particularly for identifying a target speaker in live voice calls, with a strong emphasis on detecting spoofed speech. I aim to create a more comprehensive and effective solution to ensure the integrity of voice-based authentication processes. This model will enhance the security and privacy of voice-based communication systems by preventing unauthorized access, fraud, and identity theft.

Leveraging Machine Learning

My strategy for using machine learning to tackle voice spoofing involves the following steps:

1. **Data Acquisition and Preprocessing:** Finding a diverse and representative dataset of both genuine and spoofed speech samples. Preprocessing the data to extract relevant features.

2. **Model Selection and Training:** Choosing a suitable machine learning algorithm, such as a convolutional neural network (CNN) or recurrent neural network (RNN), based on the nature of the extracted features. Training the model on the preprocessed dataset to learn the patterns and characteristics that distinguish genuine speech from spoofed speech.
3. **Model Evaluation and Refinement:** Evaluating the model's performance using appropriate metrics, such as accuracy, precision, and recall. Iteratively refining the model by adjusting hyperparameters, exploring different architectures, and incorporating additional features as needed.
4. **Deployment and Testing:** Deploying the trained model in a real-world environment to test its effectiveness in detecting various types of spoofed speech. Continuously monitoring and updating the model as new spoofing techniques emerge.

By following this approach, I aim to create a machine learning model that can effectively identify and prevent voice spoofing attacks, safeguarding the security and integrity of voice-based communication systems.

1.3 Scope

Speaker Identification and Spoofed Speech Detection

During my research for this project, it became clear that existing techniques for speaker identification are significantly more advanced than those for spoofed speech detection. As a result, the majority of my efforts were concentrated on the latter.

This project has a heavy focus on detecting spoofed speech generated by the three main spoofing methods mentioned above. This includes addressing the challenges associated with distinguishing between genuine human speech and highly realistic synthetic audio.

Limitations and Constraints

Several limitations and constraints influenced the scope and outcomes of my project:

- **Lack of Experience and Learning Curve**: As someone relatively new to the field of machine learning and artificial intelligence, I encountered a steep learning curve when getting started with the tools, frameworks, and methodologies required for this project. The transition from theoretical knowledge to practical implementation, especially when working with advanced frameworks like Hugging Face and AWS SageMaker (both of which I will discuss in detail later), took considerable time. This initial learning phase impacted the overall efficiency and progress of the project as I had to familiarize myself with key concepts, troubleshoot unexpected issues, and build proficiency in these new environments.

- **Computational Resources:** I was working entirely off of my personal laptop, which is fine for smaller projects, but was insufficient for the tasks necessary for this project. Fine-tuning existing models on thousands of high quality audio files requires a significant amount of storage and computational power, which was a large barrier I faced. I chose to use AWS S3 to store my data and fine-tuned models, and worked with SageMaker for the training jobs and endpoint deployment. Although AWS was successful at addressing the computational limitations, the associated costs can be significant, especially for large-scale model training. This added an extra layer of complexity and financial burden to the project.
- **Time Constraints:** The duration of this senior project imposed significant time constraints. While I focused on developing robust and accurate models, there were additional areas I would have liked to explore, such as experimenting with different deep learning architectures or incorporating different feature extraction techniques. Additionally, while testing was performed on the ASVSpoof2019 dataset, I couldn't extensively evaluate the model in real-world scenarios, such as testing against varied environments or real-time data. The short timeline also limited how much I could refine and document the process, impacting the depth of analysis and replicability. Given more time, these additional areas of research could have led to further enhancements and more robust results.

Boundaries of the Project

Given that this project focused primarily on developing a proof of concept for real-time authentication that emphasized the detection of spoofed speech, the scope was deliberately limited to exploring the feasibility and potential effectiveness of machine learning models in addressing this specific problem.

Several aspects were intentionally excluded from the project scope to maintain focus and manage complexity:

- Spoofing Attack Generation: The project did not delve into the technical details of how spoofed speech is generated, such as the specific algorithms or tools used by attackers. The primary focus was on detecting spoofed speech, not on understanding the underlying mechanisms of its creation.
- Alternative Authentication Methods: The project did not investigate alternative authentication methods, such as facial recognition or behavioral biometrics. The primary goal was to assess the viability of voice-based authentication using machine learning techniques.
- Real-world Deployment and Testing: As mentioned in the limitations section, the project did not progress to full-scale implementation and testing in a live call environment. The proof of concept was evaluated using pre-recorded audio samples, and further research is needed to assess its performance in real-world scenarios.

Contribution to Cybersecurity

This project makes several key contributions to the field of cybersecurity, particularly in the domain of voice authentication and spoofing detection:

- Proof of Concept for Real-Time Detection Using Fine-Tuning: By successfully implementing a proof of concept for spoofed speech detection in live call scenarios using fine-tuned machine learning models, this work demonstrates the effectiveness and adaptability of transfer learning techniques in addressing this complex challenge. This approach can serve as a valuable reference and starting point for future research and development efforts in this area, particularly for those seeking to leverage pre-trained models for specific security applications.
- Open-Source Resources: The project's code, fine-tuned models, and datasets can be made publicly available, fostering collaboration and reproducibility within the cybersecurity community. This can accelerate the development of more advanced and sophisticated solutions for voice spoofing detection and prevention, building upon the foundation established by this work.
- Raising Awareness: By highlighting the growing threat of voice spoofing and demonstrating the potential of machine learning, particularly transfer learning through fine-tuning, to address this challenge, this work contributes to raising awareness among researchers, developers, and policymakers. This can encourage increased investment and focus on developing effective countermeasures to protect individuals and organizations from voice-based attacks.

2. Literature Review

2.1 Existing Methods

Liveness detection is an advanced authentication technique used to verify that a person is alive and present during a transaction or interaction. It is often employed in various applications, including biometric authentication, video conferencing, and voice-based services. There are two primary approaches to liveness detection: active and passive. Both approaches can use physical and audio behaviors to dynamically authenticate the target, but for this use case we will just focus on the auditory biometric methods.

(What Is Liveness Detection? How It Helps Fraud Prevention)

Active methods require the user to perform specific actions, such as pronouncing a passphrase or uttering random words, to identify them. The Challenge-and-Response test is an example of an active liveness detection mechanism. The system analyzes the audio signal for characteristics that indicate a live person, such as variations in pitch, intonation, and energy levels. These features can be compared to pre-recorded samples or analyzed for consistency with the user's expected voice characteristics. While effective, active methods can be intrusive to the user experience, potentially limiting their adoption in certain applications.

Passive liveness detection, on the other hand, operates at the system backend without requiring any explicit user interaction. Instead, it relies on analyzing the inherent characteristics of the audio signal to determine if it is from a live person. This approach can be more seamless and user-friendly since it doesn't interrupt the natural flow of the interaction.

The existing analysis techniques fall into four broad categorizations.

Signal processing-based methods: These methods analyze the acoustic features of speech signals to identify inconsistencies or artifacts introduced during the spoofing process. Techniques such as Mel Frequency Cepstral Coefficients (MFCC) and Linear Predictive Coding (LPC) are commonly used to extract features that capture the spectral and temporal characteristics of speech. However, these methods can be susceptible to noise and variations in speaking style.

Statistical model-based methods: These methods utilize statistical models, such as Gaussian Mixture Models (GMMs) and Hidden Markov Models (HMMs), to characterize the distribution of genuine speech features. Spoofed speech is then detected by identifying deviations from these learned distributions. While effective for detecting simple spoofing attacks, these methods may struggle with more sophisticated techniques that can mimic the statistical properties of genuine speech.

Machine learning-based methods: These methods employ machine learning algorithms, such as Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs), to learn complex patterns and features from large datasets of genuine and spoofed speech. These models can adapt to new spoofing techniques and achieve high accuracy rates. However, they require large amounts of labeled data for training and may be computationally expensive.

Deep learning-based methods: These methods leverage deep learning architectures, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), to automatically extract features and learn intricate patterns from raw speech data. Deep learning models have demonstrated superior performance in spoofing detection tasks, but they also require significant computational resources and large datasets for training.

2.2 Related Work

The increasing reliance on Automatic Speaker Verification (ASV) systems for user authentication necessitates robust countermeasures against voice spoofing attacks. I explored the application of existing methods for voice spoofing detection, focusing on the following subcategories: deep learning architectures and feature engineering, and multiclass classification for spoofing attack type detection.

- **Deep Learning Architectures and Feature Engineering:**

  Deep learning models have become increasingly popular for audio spoofing detection in recent years. This shift towards deep learning-based approaches is likely due to their ability to extract complex and intricate patterns from data, leading to more accurate and robust detection models. Basu et al. (2023) explored various deep learning architectures for multiclass voice spoofing detection, classifying real audio from spoofed audio generated through replay, synthesis, and conversion. The authors investigated Convolutional Neural Networks (CNNs), WaveNet, Gated Recurrent Units (GRUs), and Long Short-Term Memory Networks (LSTMs) on a large collated dataset. Their findings highlight the suitability of CNNs and WaveNet for this task. They found that CNNs, when trained on audio data represented as graphs, achieved near-perfect classification of real vs. fake audio and provided insights into the type of spoofing attack. They also observed that WaveNet, a model originally developed for text-to-speech and music synthesis, successfully classified spoofed audio against real audio, particularly excelling in detecting synthesized attacks. Conversely, GRUs and LSTMs struggled with real audio classification, making them less suitable for this specific application.

  The researchers suggest combining the raw waveform and spectral features to build a hybrid model for multiclass voice spoofing detection. This approach would leverage the strengths of both WaveNet's raw audio processing and CNN's ability to discern patterns in spectral feature graphs. Lavrentyeva et al. (2021) highlights the importance of using both static and dynamic features in Automatic Speaker Verification (ASV) systems to improve their robustness against spoofing attacks. The researchers found that combining static-dynamic Constant Q Cepstral Coefficients (CQCC) features with hybrid deep learning models, like LSTM networks with Time Distributed Wrappers, significantly increased the accuracy of spoof detection. They also demonstrated the effectiveness of two-level spoof detection systems, particularly the user identification and verification system, which showed promising performance. Future research could focus on expanding the dataset to include a wider range of spoofing attacks like twins and mimicry. Investigating more complex deep learning model architectures, like the VGG-family, could also lead to further improvements in spoof detection.

- **Multiclass Classification for Spoofing Attack Type Detection:**

  Rishabh Ranjan , Mayank Vatsa , Richa Singh (2023) addressed the problem of voice spoofing attacks on Automatic Speaker Verification (ASV) systems, which are increasingly used for authentication in various organizations. They formulated voice spoofing detection as both a binary classification problem (real vs. fake audio) and a multiclass classification problem (to detect specific types of attacks like voice conversion, synthesis, and replay). The study investigated various audio features and evaluated several state-of-the-art deep learning algorithms, including Convolutional Neural Networks (CNNs), WaveNet, and Recurrent Neural Network (RNN) variants like Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) models. Experiments were conducted on a large dataset of 419,426 audio files.

The results show that the binary class CNN achieves a False Positive Rate (FPR) of 0.0216, while the multiclass solutions using CNN, WaveNet, LSTMs, and GRUs achieve FPRs of 0.003, 0.0260, 0.0302, and 0.0358, respectively. The authors also assess the models' performance in real-time classification using microphone voice audio and user-uploaded audio to demonstrate practical deployability.

Overall, the work contributes to the development of effective voice spoofing countermeasures using deep learning techniques for multiclass attack classification.

2.3 Challenges and Research Gaps

Detecting spoofed speech for Automatic Speaker Verification (ASV) systems remains a challenging task, despite advancements in deep learning techniques. Several key gaps and challenges persist, specifically within the deep learning-based spoofing detection:

1. Generalizable and Universal Detection Models

   The effectiveness of voice spoofing detection models often varies across different datasets, highlighting the need for comprehensive cross-corpus evaluations. Testing models on diverse datasets—beyond those used for training—can uncover biases and limitations, guiding the creation of more robust and adaptable solutions. This approach is crucial for developing language-independent models, which can overcome the constraints of current systems that rely heavily on language-specific features. Building models that generalize across multiple languages would represent a significant advancement toward truly universal voice spoofing detection systems.

   Moreover, real-world audio environments introduce complexities such as noise, interference, and background sounds that can impact the accuracy of spoofed speech detection. These environmental variations pose challenges for models, potentially affecting their robustness in practical applications.

   To address these challenges, there is a growing need for universal detection models capable of identifying a wide array of spoofing attacks, including both established and emerging techniques. Current models often focus on specific spoofing methods, such as replay attacks or synthetic speech generated by particular algorithms. As new and sophisticated spoofing methods continue to emerge, developing universal detection models that can generalize across different techniques becomes increasingly important. Ramachandra et al. (2023) highlight the necessity of creating models that can understand the core characteristics of both genuine and spoofed speech, regardless of the attack method. Such models would enhance the adaptability and security of voice-based authentication systems, better addressing the evolving landscape of voice spoofing threats.

2. Privacy-Preserving Techniques

   Privacy concerns are a major consideration in voice spoofing detection, as the collection and analysis of voice data can raise ethical and legal issues. To address these concerns, researchers are exploring privacy-preserving techniques that can protect sensitive user information while still enabling effective spoofing detection.

   Federated learning is one promising approach that allows multiple devices to collaboratively train a shared model without sharing their raw data (Martineau, 2023). In the context of voice spoofing detection, federated learning could enable the development of models that are trained on a diverse range of voice data from different users, without compromising individual privacy. This approach has the potential to improve the accuracy and generalizability of spoofing detection models while also addressing privacy concerns.

3. Diverse Datasets

   The availability of diverse and representative datasets is crucial for training effective and unbiased spoofing detection models. However, collecting such datasets can be challenging, as it requires obtaining a large number of voice samples from different speakers, representing various spoofing techniques, acoustic environments, and languages.

   To overcome this challenge, researchers are exploring various strategies for data collection, such as crowdsourcing, data augmentation, and synthetic data generation. Additionally, there is a growing interest in developing standardized datasets and evaluation protocols for voice spoofing detection, which would facilitate fair and objective comparisons between different models and algorithms.

4. Fair Spoofing Detection Models

   Ensuring fairness in voice spoofing detection is crucial to avoid discrimination against certain groups of users based on their demographic characteristics, such as age, gender, or accent. Biased models can lead to unfair outcomes, where certain individuals are more likely to be falsely accused of spoofing than others. It will be important to carefully consider the potential sources of bias in the data and algorithms and implement techniques to mitigate these biases.

3. Methodology

3.1 Dataset

For this project, I selected the ASVSpoof2019 dataset due to its relevance and comprehensive features that align closely with my objectives of detecting spoofed speech in a voice authentication setting. ASVSpoof2019 is part of an ongoing challenge series designed to drive innovation and standardize research in anti-spoofing measures for Automatic Speaker Verification systems (Wang et al., 2019). This dataset is particularly suited to the goals of my senior project, which focuses on distinguishing between genuine and spoofed speech using machine learning techniques.

The dataset consists of a large collection of labeled audio files, including both authentic and spoofed speech samples generated using various spoofing techniques such as voice conversion and speech synthesis. Each file is labeled with attributes like speaker identity, classification (genuine or spoofed), and the specific type of spoofing attack. These detailed annotations were critical in structuring the data for model training and evaluation.

In terms of preprocessing, the dataset's structure allowed for straightforward splitting into training and evaluation subsets, enabling me to efficiently experiment with various machine learning models. The speaker IDs and attack type labels also made it easier to analyze performance metrics and refine model parameters based on specific scenarios. This level of granularity and organization was instrumental in building models that could accurately classify spoofed versus genuine speech, directly addressing the problem statement of my senior project.

Overall, the ASVSpoof2019 dataset not only provided the foundation for model development but also offered the appropriate depth and breadth needed to explore the intricacies of spoof detection. Its comprehensive nature enabled detailed analysis and experimentation, ultimately supporting the development of solutions critical in enhancing security and reliability in voice-based authentication systems.

3.2 Model Selection

Selecting the right machine learning model for this project required significant iteration and experimentation. Initially, I explored a Siamese network framework, which is a type of neural network architecture designed to compare two inputs and determine their similarity.

I designed and trained a custom Siamese model using the ASVSpoof2019 dataset, with the goal of differentiating between genuine and spoofed speech samples by learning meaningful audio representations. However, during inference, the model performed poorly, with accuracy no better than random guessing.

Upon further investigation, I determined that the challenge lay in the model's inability to capture critical audio features. Unlike tasks such as image recognition, where features like edges and textures are easier for a model to detect, audio signals present unique complexities. The custom model struggled to extract relevant patterns from the raw audio data, resulting in ineffective learning and poor generalization. Recognizing the limitations of my approach, I decided to pivot to a different architecture that could better handle the nuances of audio feature extraction.

Given that the primary bottleneck was in feature representation, I shifted my focus to pretrained models specifically designed for audio classification. After extensive research, I decided to leverage the Wav2Vec2 framework. Wav2Vec2 is a state-of-the-art, self-supervised learning model capable of extracting powerful representations from raw speech audio. The architecture consists of a feature encoder, quantizer, and transformer encoder, which together process audio signals in a way that allows for robust downstream task performance. Pretrained on vast amounts of unlabeled speech data, Wav2Vec2 can be fine-tuned for specific applications, such as speaker verification and speech recognition, making it a natural fit for my binary classification task of distinguishing spoofed from genuine speech.

I specifically chose the Wav2Vec2-960h model, a pretrained version developed by Facebook that was trained on the LibriSpeech dataset, which is a collection of 960 hours of audiobooks. This model offers a well-balanced trade-off between accuracy and computational efficiency for English speech recognition, and its versatility makes it highly relevant for my project's objectives. The Wav2Vec2-960h model is particularly effective for tasks where extracting fine-grained features from audio is essential. Its architecture, which has been optimized through extensive pretraining, is capable of learning high-quality representations from raw waveform data—something critical when attempting to differentiate between nuanced differences in genuine versus spoofed audio or in distinguishing between target and non-target speakers.

For my project, which involves two key binary classification tasks (spoofed vs. genuine speech and target vs. non-target speaker identification), Wav2Vec2-960h provided several advantages. First, its pretrained feature encoder efficiently handles the complexities of audio data, enabling the model to pick up on subtle spectral and temporal patterns that are difficult to learn from scratch. This capability is crucial for detecting artifacts commonly found in synthesized or manipulated speech, which are often subtle and require sophisticated feature extraction to identify. Additionally, this ability would make the model suitable for adapting to the task of target versus non-target speaker identification.

Overall, transitioning to Wav2Vec2-960h allowed me to focus on fine-tuning and optimizing the model for the specific requirements of spoof detection and speaker identification, ultimately leading to more reliable and accurate performance. The model's proven performance, combined with its flexibility in adapting to different speech-related tasks, made it an ideal choice for achieving the goals of my senior project.

3.4 Tools and Technologies

Throughout this project, I leveraged a range of tools, frameworks, and cloud services to develop and deploy my machine learning solution. Each tool played a crucial role in different phases of the project, and I encountered both strengths and challenges along the way.

Wav2Vec2

The backbone of my project was the Wav2Vec2 model architecture, which is specifically designed for speech recognition and feature extraction from raw audio data. This model features a built-in feature extractor that converts audio waveforms into dense representations, which are then processed by transformer layers. Such an architecture is particularly suited for tasks requiring fine differentiation of subtle speech variations.

Hugging Face (Transformers and Estimators)

Throughout this project, I relied heavily on the Hugging Face ecosystem, particularly their Transformers library, which is integral for working with advanced models like Wav2Vec2. Hugging Face provides a user-friendly interface for loading pretrained models, customizing training scripts, and managing deployment steps. Their Estimator API, used in conjunction with AWS SageMaker, was invaluable for managing hyperparameters and automating training jobs. One of the standout aspects of Hugging Face is its extensive documentation and the active community support available through forums and GitHub repositories. This support proved crucial for resolving issues related to library compatibility and configuration settings. Additionally, the seamless integration with cloud services like AWS SageMaker streamlined my workflow, allowing me to focus more on model performance rather than dealing with low-level configurations. However, it is important to manage version compatibility across libraries carefully to avoid potential issues.

Amazon S3 (Simple Storage Service)

Amazon S3 (Simple Storage Service) served as my primary storage solution for both data and model artifacts. Its scalability, cost-effectiveness, and seamless integration with other AWS services like SageMaker made it an ideal choice. I utilized S3 to store training datasets and save fine-tuned models, which streamlined the entire workflow from data loading to model deployment. The affordability and reliability of S3, coupled with its easy management through the AWS console, were significant advantages. However, I faced challenges with configuring permissions and IAM roles to ensure secure and accessible S3 buckets for SageMaker. Additionally, the requirement for S3 and SageMaker to be in the same region led to some initial setup difficulties. I also encountered outdated examples and deprecated packages in many tutorials, which necessitated a thorough search for relevant, current guidance.

AWS SageMaker

AWS SageMaker was the core platform I used for training and deploying my models. SageMaker offers a managed environment for running machine learning jobs, and it's incredibly powerful once configured correctly. I appreciated the way SageMaker brings everything together—training jobs, model deployments, and monitoring—into a single interface. Having a built-in code editor that essentially functions like VSCode in the cloud made it easy to iterate on scripts without needing to switch platforms.

Despite its strengths, SageMaker presented its own set of challenges. One pain point during the project was discovering that certain GPU instances essential for model training are only available in specific regions, necessitating careful alignment between S3 and SageMaker regions. I learned this the hard way, leading to a time-consuming migration of data and retraining efforts. Additionally, I spent some time dealing with leftover instances running in different regions, which led to unexpected costs—$140 in charges for an instance I was unaware of. Configuring access to specific GPU instances and managing IAM permissions also required meticulous attention to avoid costly mistakes and ensure smooth operation.

AWS CloudWatch

Amazon CloudWatch was an indispensable tool for logging and monitoring throughout the project. It provided a centralized view of all my logs, whether from training jobs, endpoint deployments, or real-time predictions. One of the standout features is that CloudWatch logs never expire, so I could always go back and analyze historical data from previous runs. This helped immensely with debugging issues, particularly when dealing with configuration errors or permission problems.

TensorBoard: TensorBoard played a pivotal role in monitoring and visualizing the training process in real time. By enabling the logging of various metrics such as loss, accuracy, learning rates, and other performance indicators at regular intervals, it provided an intuitive and interactive way to track the model's progress. This tool was instrumental in quickly identifying trends and diagnosing potential issues, such as overfitting or underfitting, by allowing me to compare different runs, zoom in on specific epochs, and visualize the distribution of weights and gradients. The ability to inspect and compare these metrics visually helped in making informed decisions about model adjustments and optimizations throughout the training process.

3.5 Feature Extraction

In this project, the feature extraction process was a critical component for accurately distinguishing between genuine and spoofed speech, as well as identifying target versus non-target speakers. Given the inherent challenges of working with raw audio data—such as complex temporal patterns and subtle differences in acoustic features—it was clear that relying on traditional feature engineering techniques would be insufficient. Instead, I opted for a more advanced approach by utilizing the feature extractor provided by the Wav2Vec2 model. This decision was aligned with my goal of leveraging state-of-the-art methods to maximize the effectiveness of the classification tasks.

Wav2Vec2 is a self-supervised learning framework specifically designed to extract meaningful representations directly from raw speech audio, eliminating the need for manual feature engineering (Baevski et al., 2020). Unlike conventional approaches where features like MFCCs (Mel-Frequency Cepstral Coefficients) or spectral features must be explicitly defined, Wav2Vec2 automates the extraction process by learning high-quality features from the data itself. This shift from manual to automated feature extraction provided a significant advantage by allowing the model to focus on capturing complex and nuanced patterns that are often crucial for detecting spoofed audio or verifying speaker identity.

The feature extraction process in Wav2Vec2 can be broken down into several key stages:

1. **Raw Audio Input**: The input to the model consists of raw speech signals, which are passed directly into the Wav2Vec2 feature encoder. Unlike traditional methods that require pre-processing steps such as resampling, windowing, or filtering, the model accepts audio in its raw form. This capability is essential for tasks involving diverse and unstructured audio data, as it reduces the overhead associated with manual preprocessing.
2. **Feature Encoding**: After receiving the raw audio input, the Wav2Vec2 model processes it through a convolutional neural network (CNN) in the feature encoder. This network is designed to capture local dependencies in the audio signal by applying multiple layers of convolutional filters. Each layer progressively transforms the raw waveform into a more compact and meaningful representation. The output of this process is a sequence of high-level feature vectors that encapsulate the important information from the raw audio. By abstracting the complex raw audio into a form that the model can more effectively analyze and learn from, we can eliminate the need for extensive manual feature engineering.
3. **Quantization**: After feature encoding, the extracted features are still in a continuous form. Quantization involves converting these continuous features into discrete units or categories, reducing the complexity of the data while retaining the most significant information. By mapping the continuous audio features to a set of discrete representations, the model simplifies the data, making it easier to identify relevant patterns during downstream tasks.
4. **Contextual Representation via Transformer Encoding**: After quantization, the features are passed through a transformer encoder, which captures contextual relationships across the audio signal. The transformer architecture is particularly well-suited for sequential data like speech because it can model long-range dependencies and understand how different segments of audio relate to each other. This is critical in tasks such as spoof detection, where the model must discern not just isolated features but how they fit together within the broader context of the speech

signal. For example, a convincing spoof might have realistic individual phonemes, but the overall cadence and rhythm could reveal inconsistencies.

5. **Self-Supervised Learning**: One of the standout aspects of Wav2Vec2 is its self-supervised learning approach. During pre-training, the model learns to predict masked portions of the audio input based on the surrounding context, allowing it to learn meaningful speech representations without requiring labeled data. This capability not only reduces the need for manual feature engineering but also enables the model to be fine-tuned on a relatively small labeled dataset for specialized tasks like binary classification. By leveraging the pre-learned representations, I was able to focus my efforts on fine-tuning the model for my specific tasks without having to reinvent the feature extraction process.

In summary, the Wav2Vec2 feature extraction pipeline provided an end-to-end solution that aligned perfectly with the objectives of this project. By automating the extraction of meaningful features directly from raw audio, the model allowed me to focus on optimizing the classification process without getting bogged down in manual feature engineering. The combination of convolutional feature encoding, quantization, and transformer-based contextual learning ensured that even subtle differences between genuine and spoofed speech were captured effectively, leading to improved model performance across both of my binary classification tasks.

3.6 Training Process

The training process was a critical phase in bringing together the various components of my project. After finalizing the dataset, model selection, and feature extraction pipeline, I focused on fine-tuning the Wav2Vec2 model to adapt it specifically for the tasks of spoofed versus genuine speech classification and target versus non-target speaker detection. Due to the high computational requirements and memory constraints of this project, I opted to use AWS SageMaker for training, combined with Hugging Face's robust model integration and Amazon S3 for data storage. This setup provided the scalability and flexibility needed for iterating on model configurations, especially given the large dataset size and complex architecture of the Wav2Vec2 model.

Training Environment and Infrastructure

Given the limitations of my local machine in terms of storage, memory, and GPU power, AWS SageMaker was essential for running large-scale training jobs. SageMaker seamlessly integrates with Hugging Face, which allowed me to leverage pre-existing tools and utilities while gaining access to powerful cloud-based resources. The initial setup involved getting accustomed to managing storage with S3, monitoring training metrics with CloudWatch, and utilizing SageMaker's built-in support for distributed training and hyperparameter optimization.

Model Fine-Tuning Workflow

The first step in the training process was defining an estimator in SageMaker, which specified the necessary hyperparameters, including the learning rate, batch size, number of training epochs, and the dataset version. The estimator also determined the specific compute resources (like instance type and number of instances) and the integration with Hugging Face's model repository. The hyperparameters and configurations in this estimator file were passed to the training script that executed within SageMaker, launching a training job that utilized the specified dataset from S3.

Hyperparameter Optimization

The primary hyperparameters I focused on were learning rate, number of training epochs, batch size, warmup steps, and weight decay.

> Learning Rate: The learning rate controls the size of weight updates during training. A higher learning rate accelerates convergence but can cause instability, while a lower learning rate ensures more stable updates but may require more training epochs to achieve the desired performance.
>
> Number of Training Epochs: The number of training epochs is the count of complete passes through the entire training dataset. It affects how thoroughly the model learns from the data and impacts training duration and convergence.
>
> Batch Size: Batch size defines the number of training examples processed before updating the model's weights. It affects the stability of training and the speed at which the model converges.
>
> Warmup Steps: Warmup steps involve gradually increasing the learning rate from a low value to its target during the initial phase of training. This helps stabilize the learning process and prevents large, destabilizing weight updates early on.
>
> Weight Decay: Weight decay is a regularization technique that adds a penalty to the loss function for large weights. This encourages the model to adopt simpler weights and helps prevent overfitting.

Finding the right balance between these parameters was crucial to achieving optimal model performance without overfitting, which occurs when a model becomes too closely tailored to the training data, capturing not only the underlying patterns but also the noise and specific details that do not generalize well to new, unseen data. This results in a model that performs exceptionally well on the training data but poorly on validation or test data, as it fails to generalize beyond the examples it was trained on.

Monitoring and Evaluation

During training, I frequently logged key metrics, including loss, accuracy, precision, and recall, using TensorBoard for real-time monitoring. This allowed me to visualize how the model was learning and compare trends across multiple training jobs. To assess a model's effectiveness and compare different models, I examined both training and evaluation metrics. These metrics provide insights into the model's performance on both the training data it has seen and the unseen evaluation data.

Training loss measures how well the model fits the training data, with a lower value indicating effective learning of data patterns. However, if training loss is very low while evaluation loss is high, it suggests overfitting, where the model has memorized the training data but struggles to generalize to unseen data. Evaluation loss, on the other hand, reflects the model's performance on new, unseen data, such as a validation or test set, with a lower value indicating better generalization. By comparing evaluation loss across different models, I could determine which model handles new data more effectively and is better suited for generalizing beyond the training set.

Accuracy measures the proportion of correctly predicted instances out of the total instances, providing a straightforward indicator of overall model performance, especially when the dataset is balanced. However, accuracy alone can be misleading, as it might not reflect the model’s performance on minority classes. In such scenarios, precision and recall become crucial metrics. Precision indicates how many of the predicted positive instances were actually positive, highlighting the model's ability to avoid false positives. Recall measures how many actual positive instances were correctly identified by the model. Here’s a more in-depth breakdown of the three.

1. **Accuracy**: Accuracy measures the proportion of correctly classified instances out of the total number of instances. It is defined as $Accuracy\ =\ \frac{True\ Positives + True\ Negative}{Total\ Number\ of\ Instances}$. It provides a general overview of how well the model performs across all classes.
2. **Precision**: Precision, also known as positive predictive value, indicates the proportion of true positive predictions among all positive predictions made by the model. It is defined as $Precision\ =\ \frac{True\ Positives}{True\ Positives + False\ Positives}$. Precision is crucial when the cost of false positives is high.
3. **Recall**: Recall, also known as sensitivity or true positive rate, measures the proportion of true positives correctly identified by the model out of all actual positive instances. It is defined as $Recall\ =\ \frac{True\ Positives}{True\ Positives + False\ Negatives}$. Recall is important when the cost of missing positive instances is significant.

4. Implementation

4.1 Initial Attempts

The initial phases of this project were marked by considerable experimentation, learning, and pivoting, as I aimed to build a robust system capable of detecting spoofed speech and distinguishing between target and non-target speakers. Like many machine learning projects, the path from concept to implementation was not linear; it involved numerous false starts and iterations before I arrived at a solution that worked.

Attempting a Custom-Built Siamese Network

My first approach was to design a custom Siamese network from scratch. The idea behind this architecture was to leverage pairwise comparisons between audio samples to determine whether two samples belonged to the same speaker or class (genuine vs. spoofed). Conceptually, the Siamese model was a good fit for tasks involving fine-grained differentiation, such as distinguishing between similar voices or subtle artifacts in spoofed audio. The architecture involved feeding pairs of speech samples into identical subnetworks, with a contrastive loss function aimed at minimizing the distance between embeddings of similar samples while maximizing the distance between dissimilar ones.

However, despite its conceptual promise, this custom approach quickly became an uphill battle. Implementing a Siamese model from scratch introduced a host of complexities:

1. **Data Pairing and Labeling**: Because of a large imbalance in training data, preparing the dataset for pairwise input required a decent amount of preprocessing to generate balanced pairs for training. I oversampled the minority class (genuine samples in this case) to ensure balanced training classes.
2. **Model Convergence Issues**: Training stability was another significant problem. The model exhibited poor convergence. The custom-built model struggled to capture the nuanced distinctions between similar audio samples, and frequent exploding gradients made the training process erratic.
3. **Debugging and Code Complexity**: The manual implementation of the Siamese architecture introduced a high level of complexity in the codebase, which made debugging an exhausting process. I encountered numerous bugs related to data pipeline management, gradient flow issues, and edge cases in backpropagation. Despite significant troubleshooting efforts, these challenges persisted and hindered any meaningful progress.

Transitioning to a Pre-Trained Model: Wav2Vec2

After grappling with the limitations of the custom Siamese network, I decided to pivot to a more established solution: fine-tuning a pre-trained Wav2Vec2 model. The decision to switch was largely motivated by the realization that state-of-the-art pre-trained models, particularly those designed for speech processing, offered robust feature extraction capabilities out of the box. Rather than spending weeks debugging a custom model with questionable performance, I could leverage the strengths of Wav2Vec2's self-supervised learning framework and focus my efforts on fine-tuning and task-specific optimization.

This transition marked a significant turning point in the project, as it allowed me to make rapid progress:

- **Improved Stability and Convergence**: Right from the initial fine-tuning runs, the Wav2Vec2 model showed promising results, with consistent convergence and stable learning curves. The pre-trained model's ability to capture nuanced speech features reduced the complexity of downstream tasks and provided a solid foundation for further experimentation.
- **Simplified Workflow**: By adopting a pre-trained model, I could rely on the extensive documentation and community support available for Hugging Face's Wav2Vec2 implementation. This drastically reduced the time spent debugging low-level issues and allowed me to focus on more impactful aspects of the project, such as hyperparameter optimization and data augmentation.

Challenges Faced During Fine-Tuning

Even with the more stable foundation provided by Wav2Vec2, the early fine-tuning experiments were not without their challenges:

1. **Overfitting in Initial Epochs**: In the initial runs, I noticed that the model would achieve near-perfect accuracy on the training set after just a single epoch. Additionally, the training loss was dropping to almost zero after a couple epochs, which signals that the model learned the training data very quickly. While this might sound ideal, it was an immediate red flag for overfitting, indicating that the model was memorizing the training data rather than learning generalizable patterns.
2. **Data Preprocessing Pitfalls**: Another major issue involved inconsistencies in data preprocessing. Early on, I discovered that mismatched padding and normalization across different datasets caused erratic training behavior. Even subtle differences in how audio samples were preprocessed (e.g., inconsistent padding lengths, varying normalization methods, or using different package versions) led to significant discrepancies in model performance. Once I standardized these preprocessing steps, training stability improved substantially.
3. **Hyperparameter Exploration**: Given the high capacity of the Wav2Vec2 model, controlling the learning dynamics was crucial. In this process, I tested different combinations of hyperparameters such as learning rate, batch size, number of epochs, and weight decay to control the model's learning behavior. Finding the right combination was a matter of trial and error, guided by TensorBoard visualizations of loss curves, accuracy metrics, and validation performance. This helped identify patterns, like how a learning rate that's too high can cause unstable training, or how excessive training epochs can lead to overfitting. By iterating on these hyperparameters and comparing

their results, I could zero in on the combination that minimized both training and validation loss while maximizing accuracy and generalization to unseen data.

Leveraging TensorBoard for Diagnostics

One of the most valuable tools during this phase was TensorBoard, which I used extensively to monitor and diagnose training runs. By logging metrics at regular intervals, I could track how the model's performance evolved over time, allowing me to pinpoint exactly when and where overfitting began. The ability to visualize loss and accuracy trends across both training and validation datasets helped me make informed decisions on which hyperparameters to tweak. I will analyze the TensorBoard data in a later section.

Learning from Failures

The lessons learned from my initial failures with the Siamese network and the challenges of early fine-tuning were invaluable:

- **Don't Reinvent the Wheel**: The time and effort I invested in manually building a custom model were disproportionate to the gains I achieved. Leveraging a pre-trained model allowed me to skip over many low-level implementation details and focus on adding value at a higher level.
- **Standardization is Key**: Ensuring consistent data preprocessing was crucial to achieving stable and reliable training. Inconsistent preprocessing can introduce subtle but significant issues that lead to unreliable results.
- **Monitoring is Essential**: Regularly visualizing and analyzing metrics using tools like TensorBoard can catch issues early and guide more targeted experimentation.

4.2 Dual Model Design

I chose to train two separate models because they served distinct tasks: detecting spoofed vs. genuine speech and identifying the target vs. non-target speaker. These tasks, while related, required different data and learning objectives. By isolating each task into its own model, I was able to simplify the training process and optimize them individually. This approach also allowed for easier refinement and troubleshooting since each model focused on one clear classification goal.

Spoofed vs. Genuine Model: The spoofed vs. genuine model was designed to classify audio samples as either spoofed or authentic. This binary classification task involved labeling each audio file based on its origin: spoofed audio, or real, genuine speech. By employing binary classification, I simplified the data processing pipeline, focusing on distinguishing between two clear categories. This approach streamlined both the data labeling and the model's learning process. Given its critical role in detecting spoofed speech, I prioritized the development of this model, ensuring its training process was robust and fully functional before moving on to the next model.

Target vs. Non-Target Model: The target vs. non-target model aimed to determine if the speaker in the audio was me or someone else, which also involved binary classification. However, this task presented unique challenges, primarily due to the limited amount of training data available. I initially recorded seventy five audio clips of my own voice to represent the target class. The small dataset posed difficulties in training the model effectively, as it struggled to generalize from such a limited sample. Due to time constraints, I was unable to record enough recordings of myself for this model. Instead, I pivoted to using audio clips from a different speaker within the ASVSpoof2019 dataset as the target class, thereby expanding the available data for training. This adjustment allowed me to leverage a slightly larger dataset, which I hoped would improve the model's performance and generalization capability. Despite these challenges, the adapted approach enabled the development of a functional proof of concept, which is detailed further in subsequent sections.

Although the data for each task was different, I was able to make similar refinements to both models, such as optimizing learning rates, batch sizes, and model architecture. This two-model approach allowed me to streamline each training process while keeping the models modular, so they could easily be combined during inference for real-time speech authentication.

4.3 Refinements

The refinement stage of the project involved a methodical and data-driven approach to improve my model's performance after overcoming the initial hurdles. While the initial bugs and setup issues were resolved, I soon encountered more nuanced challenges that required fine-tuning, careful analysis, and iterative experimentation.

Handling Imbalanced Data and Weighting

After initial fixes, the model appeared to train successfully. The loss decreased steadily, and the accuracy metrics showed promising improvement. However, when I tested the model on real-time predictions, the outputs were heavily biased toward classifying almost every input as "spoofed." Despite favorable metrics during training, the model's real-world performance was skewed, which indicated deeper issues beyond simple overfitting.

This misclassification pattern hinted at a different problem: class imbalance. The ASVSpoof2019 training dataset contained roughly ten times more spoofed audio samples compared to genuine ones. This imbalance introduced a risk of the model becoming biased towards predicting the majority class, leading to poor generalization and potentially significant misclassification errors. To address this issue, I experimented with several strategies:

1. **Class Weights**: I experimented with adjusting class weights in the loss function. By assigning higher weights to the minority class (genuine speech), I aimed to make the model focus more on those underrepresented samples during training. I implemented a custom loss function that emphasized the genuine class, expecting this would counterbalance the overwhelming number of spoofed samples. However, despite this weighting, the results remained skewed towards the spoofed class, indicating that the

class imbalance was still impacting the model's ability to properly differentiate genuine speech.

2. **Oversampling and Undersampling**: I also explored oversampling genuine speech and undersampling spoofed speech in the training dataset to achieve a more balanced input distribution. Oversampling involves increasing the number of instances of the minority class by duplicating or generating synthetic samples. This ensures that the model sees more examples of the underrepresented class. Undersampling, on the other hand, reduces the number of instances in the majority class by randomly discarding some of its samples. This balances the dataset by making both classes more equal in size. This strategy had the biggest positive effect on my models training, significantly improving the model's ability to generalize and reduced the bias towards predicting the majority class.

Fine-Tuning Hyperparameters

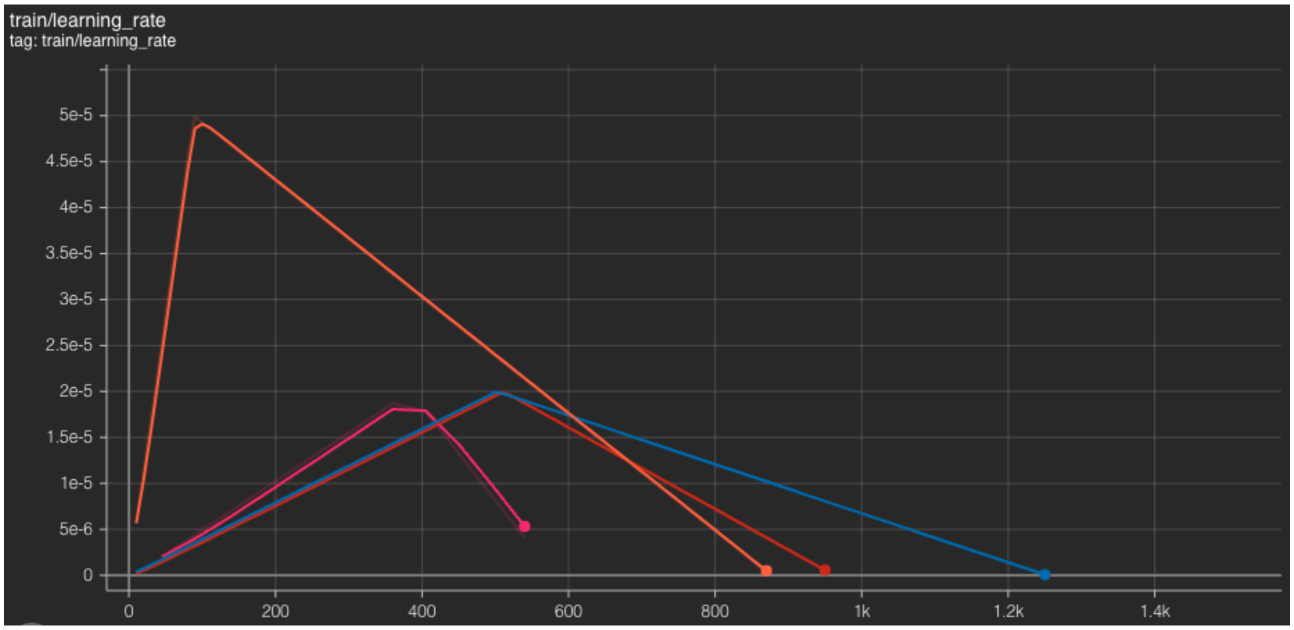
A **hyperparameter** is a parameter whose value is set before the learning process begins and controls the training process of a machine learning model (GeeksforGeeks, 2023). Hyperparameter tuning played a critical role in this refinement phase. The iterative process involved experimenting with various configurations of epochs, learning rates, and batch sizes. Given that the model's early training showed rapid accuracy improvements but later plateaued or overfitted, finding the optimal number of training epochs was crucial.

I experimented with different ranges for these parameters, adjusting them based on trends observed in TensorBoard:

1. **Learning Rate Adjustments**: The learning rate is a critical hyperparameter that determines the magnitude of weight updates during each iteration of training, based on the gradient of the loss function. A high learning rate accelerates the model's learning process by making larger updates to the weights. However, this can cause the model to miss finer adjustments and overshoot optimal solutions. On the other hand, a low learning rate makes more gradual updates to the model's weights, which allows the model to converge more precisely. The downside of this is a slower overall learning process, as it usually requires more epochs to reach optimal performance. I tested a range of learning rates, and found that the optimal solution to be a moderately low learning rate combined with warm up steps, allowing the model to gradually stabilize before fully engaging in fine-tuning.
2. **Epoch Fine-Tuning**: Given the large size of the dataset, an extensive number of epochs was unnecessary. Instead, I employed a trial-and-error approach to monitor and manage overfitting. By closely observing the training and validation loss curves, I adjusted the number of epochs to ensure that the model did not overfit to the training data. Specifically, I terminated training if the validation loss showed signs of increasing, indicating that the model was starting to memorize the training data rather than generalizing well. This approach allowed the model to avoid overfitting and make effective use of the large dataset, balancing the need for sufficient training while maintaining generalization performance.

3. **Batch Size Optimization**: Given the computational constraints and variability in audio input lengths, choosing an appropriate batch size was important for stable training. I experimented with various batch sizes and found that while larger batch sizes sped up the training process, it was not the most crucial factor for my model's performance. I ultimately selected a batch size that balanced efficiency with adequate updates per epoch.
4. **Warmup Steps:** Warmup steps helped stabilize the training process by initiating the model with smaller, more controlled weight updates. This gradual adjustment improved convergence and overall model performance. For the spoofed vs. genuine model, which had a significantly larger dataset and thus required more training steps, I used a greater number of warmup steps to facilitate a smoother and more effective learning process.
5. **Weight Decay:** This technique discourages the model from learning overly complex patterns by penalizing large weight values, which promotes simpler and more generalizable models. Although I initially experimented with both using and omitting this parameter, I ultimately chose to include it, as it led to improved model performance and better generalization on unseen data.

Below is a TensorBoard graph illustrating the learning rate trajectories for four completed training jobs, plotted against training steps. The triangular pattern observed reflects a learning rate schedule comprising a warm up phase followed by a decay phase. During the warmup phase, the learning rate gradually increases to stabilize model updates and avoid instability at the start of training. Subsequently, the learning rate decreases slowly, as shown by the downward slope of the triangle, allowing for refined adjustments to the model parameters and ensuring stable convergence. This strategy effectively balances the need for rapid learning with the precision necessary for optimal model performance, thereby enhancing the overall training process.



Comparing Model Variants in TensorBoard

TensorBoard was instrumental throughout this refinement process. I monitored multiple metrics simultaneously, such as training and validation loss, accuracy, precision, and recall scores across different training runs. By visualizing these metrics side-by-side, I could identify trends like overfitting, training plateaus, and class imbalance issues early on. This guided decisions such as when to halt training, which hyperparameters to adjust next, and how to reconfigure the dataset for better results. The ability to overlay different runs allowed me to systematically compare the impact of each change.

Key Takeaways

1. **Class Imbalance Requires Strategic Solutions**: Simply adjusting the dataset size isn't always enough. Combining weighted loss functions with data augmentation provided a robust solution for improving performance on minority classes without sacrificing overall model accuracy.
2. **Hyperparameter Tuning is a Continuous Process**: Finding the right combination of learning rate, batch size, and epochs is not a one-time task. Iterative adjustments based on real-time metrics allowed me to hone in on configurations that maximized performance while avoiding overfitting.
3. **Visualization and Monitoring Drive Informed Decisions**: The insights gained from TensorBoard visualizations were indispensable. Tracking key metrics in real-time and across multiple runs enabled data-driven decisions rather than relying on trial-and-error alone.

The refinements in this phase ultimately allowed the model to transition from an overfitted, biased system into one that could accurately differentiate between genuine and spoofed speech, providing reliable and consistent performance across a variety of test scenarios.

5. Technical Code Overview

My process is broken down into three main sections, each critical to the overall success of the project:

1. **Data Preparation:** This involved collecting, labeling, and pre-processing the audio data to ensure it was ready for model training.
2. **Model Training:** I fine-tuned a pre-trained Wav2Vec2 model on the ASVSpoof dataset, carefully optimizing hyperparameters and evaluating performance to avoid overfitting.
3. **Model Deployment:** After training, the model was deployed for real-time inference, leveraging AWS SageMaker to manage resources and scale efficiently.

Throughout this process, Ying Hou's article was an invaluable resource, providing key insights that informed many of the technical decisions made during implementation.

5.1 Data Preparation

Protocol File Analysis: I start by examining the protocols file from the ASVSpoof2019 data repository. These include a list of files, their speaker IDs, classification types, and attack type. Understanding these files helps me organize and manage the data effectively. Below are a few example lines from this file.

LA_0079 LA_T_3187715 - - bonafide
LA_0095 LA_T_3775083 - A02 spoof
LA_0091 LA_T_2758730 - A05 spoof

Each of these lines corresponds to an audio file encoded using the Free Lossless Audio Codec (FLAC). The training dataset has roughly twenty five thousand .flac files, the majority of which are spoofed. All of these audios were less than seconds long, with an even distribution around six to seven seconds. I set a maximum length of 8 seconds for the audio inputs, padding any shorter audio clips to ensure consistency in input size. This approach was aimed to standardize the data fed into the model, hopefully making it easier for the model to learn and generalize from the audio features.

Filename Scraping and Labeling: I iterate through the entire protocol file and compile each of the files, along either their classification, into a CSV file. The file path is stored so I can preprocess the data in the next step, and the labeling is crucial for supervised learning, as it provides the ground truth for evaluating the model.

```
def create_spoof_csv(protocol_file_path, flac_file_directory):
   data = []
   with open(protocol_file_path, "r") as file:
       lines = file.readlines()
       for line in lines:
           parts = line.split()
           if len(parts) >= 5:
               classification = parts[4]
               if classification == "bonafide":
                   path = flac_file_directory+ parts[1] + ".flac"
                   data.append([classification, path])
               elif classification == "spoof":
                   path = flac_file_directory + parts[1] + ".flac"
                   data.append([classification, path])
   return data
```

Data Preprocessing: Using the Wav2Vec2 feature extractor, I preprocess the data in batches. This step involves converting raw audio into meaningful feature representations that the model can learn from. The Wav2Vec2 model simplifies the data by extracting high-level features from the audio, which enhances the model's ability to learn effectively.

I define the feature extractor, using the Wav2Vec2FeatureExtractor import from the transformers package.

```
feature_extractor = Wav2Vec2FeatureExtractor(
       feature_size=1, # Defines the number of dimensions
       sampling_rate=16000, # Must be 16 kHz for Wav2Vec
       padding_value=0.0, # Pad sequences to the same length
       do_normalize=True, # Normalize audio data to a standard range
       return_attention_mask=False # Used for handling padding tokens
       )
```

I then create a preprocess function that accepts a batch of audio arrays, passes them through the feature extractor, and returns the inputs, which are formatted to be passed into the Wav2Vec2 model during training.

```
def preprocess_function(samples):
   audio_arrays = [x["array"] for x in samples["audio"]]
   inputs = feature_extractor(
       audio_arrays,
       sampling_rate=16000, # Must be 16 kHz for Wav2Vec
       return_tensors="pt", # Returns tensors in PyTorch format
       padding=True, # Pads sequences to the maximum length
       max_length=16000*8, # Maximum length of the sequences
       do_normalize=True # Normalize audio data to a standard range
   )
   return inputs
```

The next step is to pass all of the .flac files through this function, and format the dataset to be simple and functional.

```
# Load the csv dataset
dataset = load_dataset(
   "csv",
   data_files="datasets/spoof_genuine.csv",
   split="train"
   )
# Rename the 'classification' column to 'label' for clarity
dataset = dataset.rename_column("classification", "label")
# Convert the 'label' values to binary format:
# 'bonafide' becomes 0 and 'spoof' become 1
dataset = dataset.map(lambda x:
                       {"label": 0 if x["label"] == "bonafide" else 1})
# Cast the 'audio' column to the Audio type
dataset = dataset.cast_column("audio", Audio(sampling_rate=16_000))
# Shuffle the dataset to randomize the order of samples
dataset = dataset.shuffle(seed=42)
# Apply the 'preprocess_function' to the dataset in batches
dataset = dataset.map(
   preprocess_function,
   batched=True,
   remove_columns=["audio"],
   batch_size=4
   )
```

Data Upload: I upload the preprocessed data to my S3 bucket. This centralized storage allows me to efficiently manage and access the data during the training phase.

```

# Split the dataset into training (80%) and testing (20%) sets
dataset = dataset.train_test_split(test_size=0.2)

# Save the two parts of the dataset
train_dataset = dataset['train']
test_dataset = dataset['test']

# Upload the training data to s3
training_input_path = f's3://{BUCKET}/spoof_train/'
train_dataset.save_to_disk(
   training_input_path,
   storage_options=s3.storage_options
   )

# Upload the test data to s3
test_input_path = f's3://{BUCKET}/spoof_test/'
```

```
test_dataset.save_to_disk(
   test_input_path,
   storage_options=s3.storage_options
   )
```

5.2 Model Training

The training code is split up into two main parts: the training script, and the estimator. A Hugging Face Estimator instance is an essential tool for integrating Hugging Face models with AWS SageMaker, enabling seamless management of training jobs for machine learning tasks. It allows you to define and run training processes on AWS's scalable infrastructure by specifying key components like the pre-trained model, training script, and necessary compute resources. The Estimator takes in a custom training script that outlines how to load your dataset, set up the model, compute metrics, and conduct the training, making it highly flexible for fine-tuning and retraining models.

Training Script Creation: The training script automates the model training process through a series of structured steps.

Initially, the script sets up logging mechanisms to monitor training progress and performance. It then reads in configuration arguments provided by the Estimator instance, which are essential for customizing the training parameters and ensuring the script aligns with the specific requirements of the training job.

```

# Set up logging
logger = logging.getLogger(__name__)
logging.basicConfig(
   level=logging.getLevelName("INFO"),
   handlers=[logging.StreamHandler(sys.stdout)],
   format="%(asctime)s - %(name)s - %(levelname)s - %(message)s",
)

# Parse command-line arguments
parser = argparse.ArgumentParser()

# Hyperparameters for training
parser.add_argument("--epochs", type=int, default=5)
parser.add_argument("--train_batch_size", type=int, default=8)
parser.add_argument("--eval_batch_size", type=int, default=8)
parser.add_argument("--warmup_steps", type=int, default=500)
parser.add_argument("--model_name", type=str)
parser.add_argument("--learning_rate", type=str, default=2e-5)

# Data, model, and output directories
parser.add_argument("--output_data_dir", type=str,
default=os.environ["SM_OUTPUT_DATA_DIR"])
parser.add_argument("--model_dir", type=str,
default=os.environ["SM_MODEL_DIR"])
parser.add_argument("--training_dir", type=str,
default=os.environ["SM_CHANNEL_TRAIN"])
parser.add_argument("--test_dir", type=str,
default=os.environ["SM_CHANNEL_TEST"])

args, _ = parser.parse_known_args()
```

Next, the script loads the datasets from the specified S3 bucket.

```
# Load the datasets from S3
s3 = s3fs.S3FileSystem()
test_dataset = load_from_disk(dataset_path=args.test_dir,
storage_options=s3.storage_options)
train_dataset = load_from_disk(dataset_path=args.training_dir,
storage_options=s3.storage_options)
```

Following that, I define the feature extractor and relevant metrics. The feature extractor is configured to process the raw audio data, converting it into a format suitable for model training. Additionally, I specify the metrics to monitor during training, which helps in evaluating the model's performance and making necessary adjustments.

```
# Define Wav2Vec Feature Extractor
feature_extractor = Wav2Vec2FeatureExtractor(
   feature_size=1, # Defines the number of dimensions
   sampling_rate=16000, # Must be 16 kHz for Wav2Vec
   padding_value=0.0, # Pad sequences to the same length
   do_normalize=True, # Normalize audio data to a standard range
   return_attention_mask=False # Used for handling padding tokens
   )

# Define necessary metrics
```

```
accuracy_metric = evaluate.load("accuracy")
precision_metric = evaluate.load("precision")
recall_metric = evaluate.load("recall")
```

After this, I create a function to compute the metrics during training. This function evaluates the model's performance against the defined metrics, providing real-time feedback on its accuracy, precision, recall, and other relevant measures.

```
# Compute metrics for evaluation
   def compute_metrics(pred):
       labels = pred.label_ids
       preds = pred.predictions.argmax(-1)

       accuracy = accuracy_metric.compute(
           predictions=preds, references=labels)
       precision = precision_metric.compute(
           predictions=preds, references=labels)
       recall = recall_metric.compute(
           predictions=preds, references=labels)

       return {"accuracy": accuracy["accuracy"],
               "precision": precision["precision"],
               "recall": recall["recall"]}
```

Then, I load the Wav2Vec2 pretrained model. This involves initializing the model with pre-trained weights to leverage its existing knowledge for fine-tuning.

```
model_name = "facebook/wav2vec2-base-960h"
model = Wav2Vec2ForSequenceClassification.from_pretrained(
   model_name, num_labels=2)
```

Finally, I define all training arguments and instantiate a `Trainer` object, which provides a high-level API for managing the training and evaluation process. I then call the `.train()` method on the `Trainer` instance to initiate the training job, allowing the model to be trained according to the specified parameters and configuration.

```
# Define training args
training_args = TrainingArguments(
   output_dir=args.model_dir,
   evaluation_strategy="steps",
   eval_steps=40,
   save_strategy="no",
   learning_rate=args.learning_rate,
   per_device_train_batch_size=args.train_batch_size,
   per_device_eval_batch_size=args.eval_batch_size,
   gradient_accumulation_steps=8,
   num_train_epochs=args.epochs,
   fp16=True,
   weight_decay=0.1,
   logging_steps=25,
   logging_dir='model/logs',
   warmup_steps=args.warmup_steps
)

# Create Trainer instance
trainer = Trainer(
   model=model,
   args=training_args,
   train_dataset=train_dataset,
   eval_dataset=test_dataset,
   tokenizer=feature_extractor,
   compute_metrics=compute_metrics,
   callbacks=[TensorBoardCallback()]
)

# Train model
trainer.train()
```

The final steps involve documenting the evaluation results and saving the fine-tuned model to the S3 bucket for future use.

```
# Evaluate final model
eval_result = trainer.evaluate(eval_dataset=test_dataset)

# Write eval results to text file
with open(os.path.join(args.output_data_dir, "eval_results.txt"), "w") as
writer:
    print(f"***** Eval results *****")
    for key, value in sorted(eval_result.items()):
       writer.write(f"{key} = {value}\n")

# Save the model to S3 bucket
trainer.save_model(args.model_dir)
```

Estimator Instance: the Estimator integrates AWS infrastructure settings, allowing you to configure instance types (e.g., GPU or CPU), instance counts, and hyperparameters super conveniently.

In my `estimator.py` file, the process begins by retrieving essential environment variables, defining paths to the S3 bucket, and setting up TensorBoard logging.

```
# Grab necessary environment variables
role = os.getenv("SAGEMAKER_ROLE")
source = os.getenv("SOURCE")
bucket = os.getenv("BUCKET")
log_dir = os.getenv("LOG_DIR")

training_input_path = f's3://{bucket}/train'
test_input_path = f's3://{bucket}/test'
output_path = f's3://{bucket}/output'

# TensorBoard configuration
tensorboard_output_config = TensorBoardOutputConfig(
   s3_output_path=os.path.join(f"s3://{bucket}", "output", 'tensorboard'),
   container_local_output_path=log_dir
   )
```

Next, I define the hyperparameters for training and establish metric definitions. Although the metrics might appear complex, they are designed to extract and track essential performance values from the logs, ensuring effective monitoring of the training process.

```
# Define model hyperparameters
hyperparameters = {'epochs': 5,
               'train_batch_size': 8,
               'model_name': "facebook/wav2vec2-base-960h",
               'training_dir': training_input_path,
               'test_dir': test_input_path,
               'output_dir': output_path,
                'warmup_steps': 500,    # Add if you want to control warmup
                }

# Define metrics definitions
metric_definitions=[
   {'Name': 'loss',
       'Regex': "'loss': ([0-9]+(.|e\-)[0-9]+),?"},
   {'Name': 'learning_rate',
       'Regex': "'learning_rate': ([0-9]+(.|e\-)[0-9]+),?"},
   {'Name': 'eval_loss',
       'Regex': "'eval_loss': ([0-9]+(.|e\-)[0-9]+),?"},
   {'Name': 'eval_accuracy',
       'Regex': "'eval_accuracy': ([0-9]+(.|e\-)[0-9]+),?"},
   {'Name': 'eval_precision',
       'Regex': "'eval_precision': ([0-9]+(.|e\-)[0-9]+),?"},
   {'Name': 'eval_recall',
       'Regex': "'eval_recall': ([0-9]+(.|e\-)[0-9]+),?"},
   {'Name': 'eval_runtime',
       'Regex': "'eval_runtime': ([0-9]+(.|e\-)[0-9]+),?"},
   {'Name': 'eval_samples_per_second',
       'Regex': "'eval_samples_per_second': ([0-9]+(.|e\-)[0-9]+),?"},
   {'Name': 'epoch',
       'Regex': "'epoch': ([0-9]+(.|e\-)[0-9]+),?"}]
```

Finally, I create an instance of the Hugging Face Estimator and initiate the training job using the `.fit()` method. This step launches the training process, leveraging the specified configuration to train the model.

```
# Create estimator
huggingface_estimator = HuggingFace(
   entry_point='train.py',
   source_dir=source,
   output_path= output_path,
   instance_type='ml.p3.2xlarge', # Must request access from AWS
   instance_count=1,
   transformers_version='4.36.0',
   pytorch_version='2.1.0',
   py_version='py310',
   role = role,
   hyperparameters = hyperparameters,
   metric_definitions = metric_definitions,
   tensorboard_output_config=tensorboard_output_config
   )

# Start the training job
huggingface_estimator.fit(
   {'train': training_input_path,
   'test': test_input_path}
)
```

Requirements: Configuring package versions to ensure compatibility between each other and with the SageMaker interface proved more challenging than anticipated. Below is the `requirements.txt` file reflecting the package versions used at the time of writing.

```
datasets==2.20.0
numpy>=1.17, <2.0
transformers==4.36.0
evaluate==0.4.2
s3fs==2024.5.0
fsspec==2024.5.0
tensorboard==2.17.0
pandas==2.2.2
```

5.3 Model Deployment and Inference

Deploying the model as an endpoint involves creating a persistent service that can accept incoming requests and provide predictions based on the trained model. In the code, this process begins with configuring the deployment environment, such as selecting the appropriate instance type and specifying resource requirements. Once the model is deployed, it is accessible via a URL, which serves as the endpoint for making inferences.

To make inferences, the code sends data to this endpoint using HTTP requests. The input data is formatted according to the model's requirements and is then transmitted to the endpoint. The deployed model processes the data and returns predictions, which are captured and interpreted by the application. This setup allows for real-time interaction with the model, enabling it to perform tasks such as classifying audio samples or predicting outcomes based on new input data.

Deployment: The model deployment process comprises two critical components: a deployment script and a separate file containing custom functions for inference tasks.

The deployment script coordinates the model deployment by first retrieving essential environment variables, initializing a Hugging Face model instance, and then invoking the `.deploy()` method to establish the endpoint. This streamlined process ensures that the model is accurately configured and made accessible for inference on the specified instance type within SageMaker.

```
# Grab necessary environment variables
role = os.getenv("SAGEMAKER_ROLE")
model_1 = os.getenv("MODEL_1")
entry_point = os.getenv("INFERENCE")

# create Hugging Face Model Class
huggingface_model = HuggingFaceModel(
  model_data=model_1, # path to fine-tuned model
  entry_point=entry_point, # path to inference file
  role=role,
  transformers_version="4.37.0",
  pytorch_version="2.1.0",
  py_version='py310',
)
# deploy model to SageMaker Inference
predictor = huggingface_model.deploy(
  initial_instance_count=1,
  instance_type="ml.g4dn.xlarge",
  endpoint_name='finetuned-wav2vec-endpoint-spoof'
)
```

The `inference.py` file encompasses all custom inference functions tailored for the deployment of the model. This file is integral to the deployment process as it is supplied to the Hugging Face model instance.

In the deployment process, I adapted several functions to enhance the model's inference capabilities.

1. **`model_fn`**: This function is responsible for loading the model from a specified directory. It utilizes a helper function to determine the appropriate device (GPU or CPU) and loads the model to that device.

2. **`input_fn`**: This function handles incoming data, specifically JSON requests. It extracts the audio data, processes it using the preprocessing function, and moves it to the appropriate device for inference.
3. **`predict_fn`**: This function takes the preprocessed input data and the loaded model to generate predictions. It performs inference by passing the input through the model and determining the predicted class based on the output logits.
4. **`output_fn`**: This function formats the model's predictions into JSON format, facilitating easy consumption by external systems or applications.
5. **`get_device`**: This helper function checks for the availability of a GPU and returns the appropriate device identifier, ensuring that the model operates on the optimal hardware.

These functions work together to ensure that the model is properly loaded, data is correctly processed, predictions are made accurately, and results are formatted for easy integration.

```
def model_fn(model_dir, context=None):
       device = get_device()
       model =
Wav2Vec2ForSequenceClassification.from_pretrained(model_dir).to(device)
       return model

def input_fn(json_request_data, content_type='application/json'):
   device = get_device()
   input_data = json_request_data['audio_array']
   inputs = preprocess_function(input_data).to(device)

   return inputs

def predict_fn(input_data, model):
   with torch.no_grad():
       logits = model(input_data["input_values"]).logits
   prediction = torch.argmax(logits, dim=-1).item()

   return prediction

def output_fn(predictions, accept='application/json'):
   return json.dumps(predictions)

def get_device():
   device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
   return device
```

Inference: The inference script is designed for straightforward interaction with the deployed model to make predictions. It initializes a `Predictor` instance, specifying the model's endpoint and configuring the serializer for handling input data. The script then reads the audio data from a specified file. This data is formatted into a JSON-compatible structure and passed to the model endpoint using the `predictor.predict()` method.

```
s3 = s3fs.S3FileSystem()
endpoint="finetuned-wav2vec-endpoint-spoof"
predictor = Predictor(endpoint_name=endpoint, serializer=NumpySerializer())

audio_data, sampling_rate = sf.read(test_file)
input_json = {"audio_array": audio_data}
with torch.no_grad():
   prediction = predictor.predict(data=input_json)
print("Prediction:", prediction)
```

6. Results

6.1 Performance Metrics

The performance evaluation of my models shows distinct strengths and areas for improvement across two key tasks: classifying spoofed versus genuine speech, and identifying target versus non-target speakers.

Although I would have loved to spend a lot more time refining hyperparameters and generating the highest-performing model possible, training these models didn't come without cost. Training on the ml.p3.2xlarge instance, essential for models of this size, incurs a cost of $3.825 per hour. Each training job took between upwards of 2 hours, making continuous experimentation impractical. Consequently, I ran each model a handful of times and selected the best-performing variant for detailed analysis.

This analysis involved an evaluation of the models' performance on an evaluation dataset, which consisted of data that the models had not encountered during training. This separation ensured that the metrics provided an unbiased measure of how well the models generalized to new, unseen data. Key performance metrics such as accuracy, precision, and recall were scrutinized to understand the models' effectiveness in differentiating between spoofed and genuine speech, as well as in identifying target versus non-target speakers. These metrics were described in depth in the Training Process section.

Additionally, the evaluation loss was analyzed as an indicator of the models' convergence during training. This loss metric reflects how well the model's predictions align with the actual labels in the evaluation set. By monitoring changes in evaluation loss, I could assess whether the models had effectively learned the critical features necessary for accurate classification and whether they had reached a point of stable performance. This thorough examination provided insights into the models' strengths and weaknesses and informed decisions for potential improvements.

An in-depth analysis of both models will be presented in the subsequent sections, where each model's performance will be thoroughly examined and evaluated.

The graphs below illustrate the training progress of the models, with the x-axis representing training steps and the y-axis displaying the relevant metrics. The graph for the spoofed vs. genuine model features a larger number of training steps due to the larger dataset and more epochs utilized.

For the spoofed vs. genuine classification model **(Model 1)**, the best performing model was trained on a balanced dataset comprising twenty-six thousand audio samples across four epochs. By maintaining an even distribution of samples between the two classes, the model benefited from more stable and effective training. The performance of this model was exceptional, which will be detailed further in the following graphs.

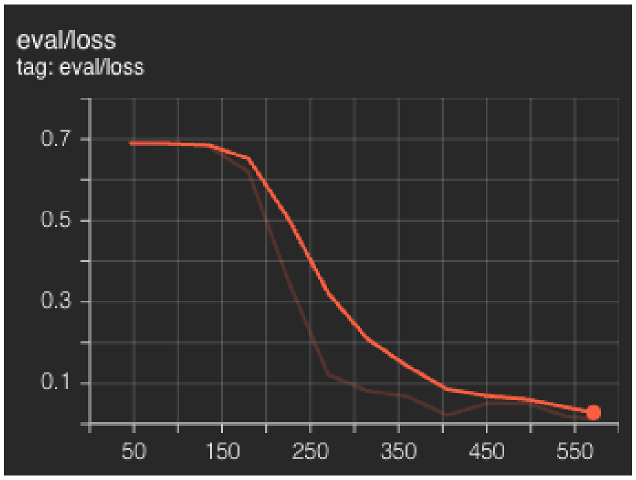


Figure 1: Evaluation Loss of Model 1

Started at 0.6896, and ended at 0.0356.

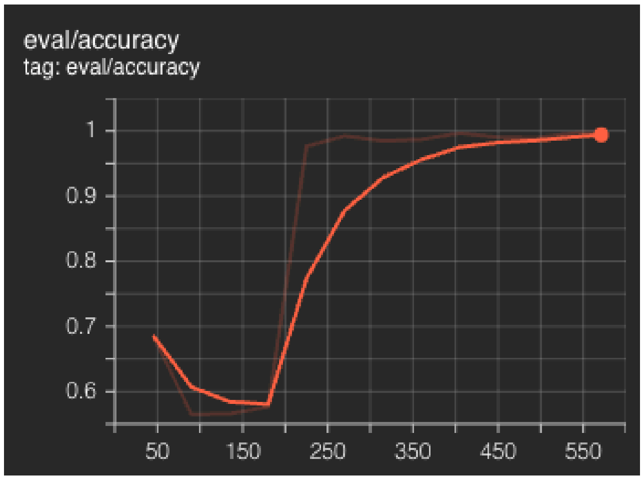


Figure 2: Evaluation Accuracy of Model 1

Lowest was 58.24%, and ended at 99.76%.

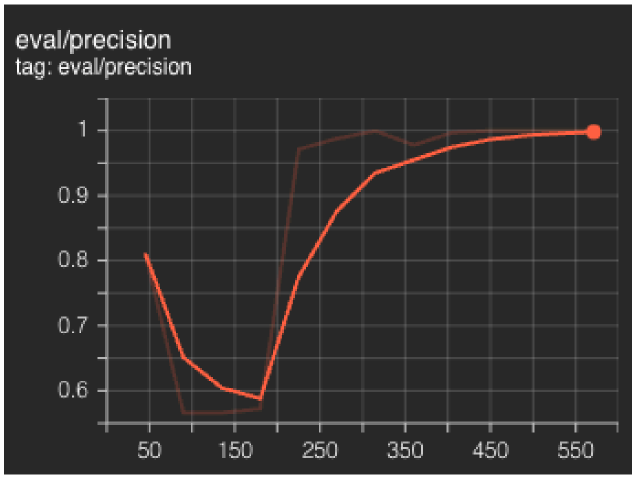


Figure 3: Evaluation Precision of Model 1

Lowest was 59.29%, and ended at 99.97%.

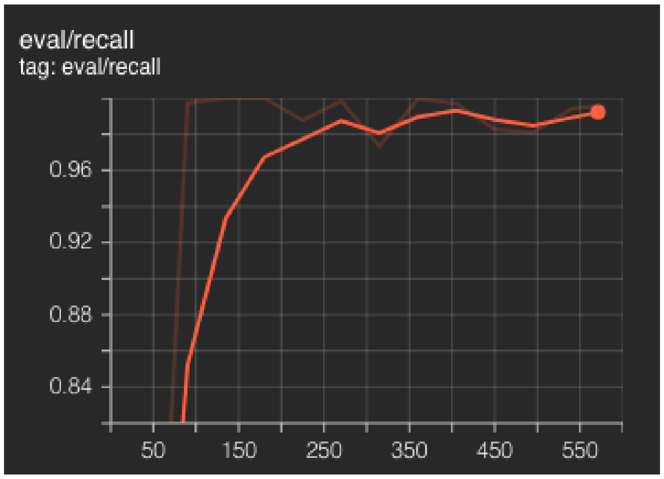


Figure 4: Evaluation Recall of Model 1

Started at 57.93%, and ended at 99.34%.

The target versus non-target speaker classification model (**Model 2**) was trained on a substantially smaller dataset, consisting of three thousand samples per class over five epochs. Despite using an equal number of samples for each class, with the aim of replicating the success of Model 1, the results fell short of expectations. The reduced dataset size presented considerable challenges, impacting the model's ability to generalize and perform as effectively as anticipated.

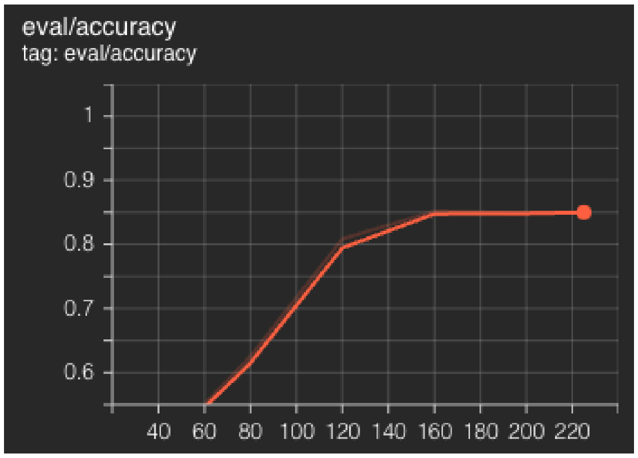


Figure 5: Evaluation Loss of Model 2

Started at 0.6937, and ended at 0.4244.

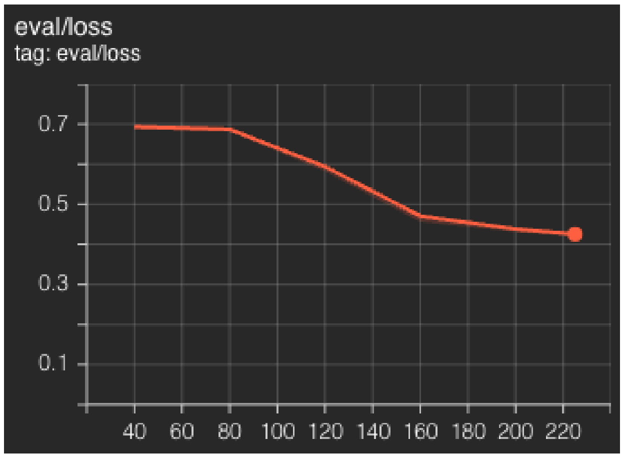


Figure 6: Evaluation Accuracy of Model 2

Started at 47.50%, and ended at 82.16%.

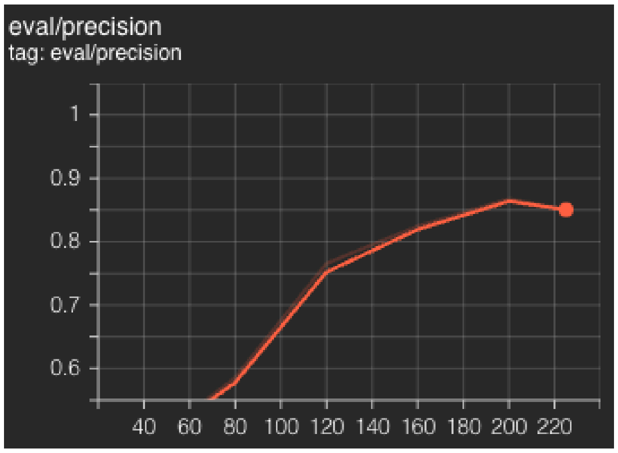


Figure 7: Evaluation Precision of Model 2

Lowest was 47.50%, and ended at 84.95%.

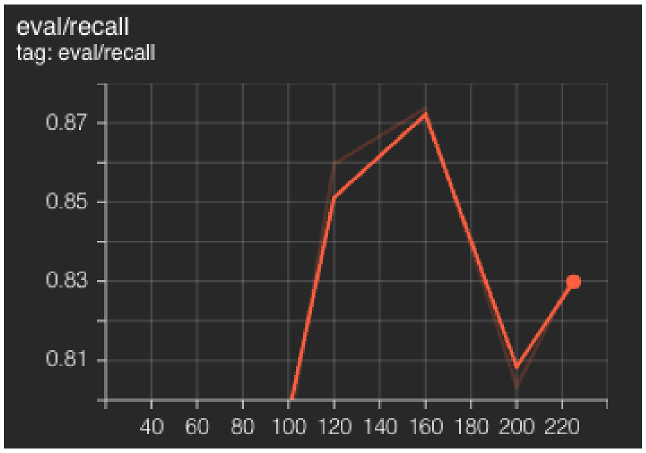


Figure 8: Evaluation Recall of Model 2

Lowest was 72.28%, and ended at 83.16%.

6.2 Analysis of Model 1

The performance metrics of the two models offer valuable insights into their effectiveness and areas for enhancement, particularly in the context of their respective tasks—classifying spoofed versus genuine speech and identifying target versus non-target speakers.

Evaluation Loss and Accuracy Analysis:

Figure 1 illustrates the evaluation loss of Model 1, starting at 0.6896 and decreasing to 0.0356. Evaluation loss is a crucial metric that measures the discrepancy between the model's predictions and the actual labels. Initially, the high loss indicates that the model was making predictions that were significantly off from the true values. This is expected in the early stages of training when the model is still learning. As the training progresses, the sharp decrease in loss to 0.0356 signifies that the model has successfully adjusted its parameters to better align its predictions with the actual labels. A lower loss at the end of training suggests effective model convergence, meaning the model has learned to make more accurate predictions and fit the training data well.

Complimenting this, Figure 2 shows the evaluation accuracy starting at 58.24% and rising to an impressive 99.76%. The model's performance was suboptimal at the beginning of training, with a significant number of misclassifications. However, the dramatic improvement to 99.76% suggests that the model achieved near-perfect performance by the end of the training period. This high accuracy indicates that the model became highly effective at distinguishing between spoofed and genuine speech, correctly classifying almost all instances.

Evaluation Precision and Recall Analysis:

Figure 3 shows that the evaluation precision of Model 1 increased significantly from 59.29% to 99.97%. Precision measures the proportion of correctly identified positive instances out of all instances classified as positive. The substantial improvement in precision indicates that as training progressed, the model became highly adept at correctly identifying spoofed speech when it predicted a sample as spoofed, thereby minimizing false positives. Initially, the model struggled with precision, but it achieved near-perfect precision by the end of training.

Similarly, Figure 4 displays the evaluation recall, which rose from 57.93% to 99.34%. Recall measures the proportion of actual positive instances that the model successfully identified. The sharp increase in recall demonstrates that the model's ability to detect all relevant instances of spoofed speech improved significantly over time, reducing false negatives. Early in the training, the model missed many spoofed samples, but with continued training, it became highly effective at recognizing almost all spoofed instances. This combination of high precision and recall by the end of the training indicates that the model not only correctly identified spoofed speech with high accuracy but also managed to detect almost all spoofed samples, making it highly effective for the classification task.

Overall Model Effectiveness:

The substantial improvement across all metrics—loss, accuracy, precision, and recall—demonstrates the effectiveness of the training process. The model successfully refined its ability to distinguish between spoofed and genuine speech, as evidenced by the dramatic reduction in loss and the sharp increase in accuracy, precision, and recall. This overall performance indicates that the model not only became highly proficient at classification but also learned to generalize well from the training data, making it a robust tool for identifying spoofed speech.

6.3 Analysis of Model 2

Evaluation Loss and Accuracy Analysis

Figure 5 illustrates the evaluation loss for the target versus non-target speaker model, starting at 0.6937 and decreasing to 0.4244. This reduction in loss signifies that the model improved its error minimization over the training period. However, the final loss value of 0.4244 indicates that the model did not achieve a sufficiently low loss, implying that there were still notable inaccuracies in its predictions. This higher end loss suggests that despite improvement, the model was still not performing optimally.

Figure 6 shows the evaluation accuracy, which started at 47.50% and increased to 82.16%. The substantial improvement in accuracy is indicative of the model's learning progression. However, the initial accuracy of 47.50% is concerning, as it reflects performance only marginally better than random guessing. Although reaching 82.16% is an improvement, it still indicates that the model struggled to consistently classify target versus non-target speakers effectively. The combination of high final loss and moderate accuracy suggests that while the model learned and improved, its performance was not consistently reliable.

Precision and Recall Analysis

Figure 7 depicts the evaluation precision, ranging from 47.50% to 84.95%. Precision measures how well the model identifies target speakers among all predicted positives. The significant increase from 47.50% to 84.95% shows that the model improved its ability to accurately identify target speakers over time. However, the initial low precision indicates a high rate of false positives early in training. Although the improvement is notable, the model's initial precision was insufficient, reflecting early difficulties in correctly identifying target speakers.

Figure 8 presents the evaluation recall, ranging from 72.28% to 83.16%. Recall measures the model's ability to identify all actual positive cases (target speakers). The increase in recall suggests that the model became better at capturing target speakers over time. Nevertheless, a final recall of 83.16% indicates that there were still missed instances of target speakers The moderate final recall suggests that while the model improved, it was not fully effective in capturing all target speaker instances.

Overall Model Effectiveness

In summary, the target versus non-target speaker model demonstrated improvement during training, but it faced notable challenges. The evaluation loss and accuracy indicate that the model's performance was initially weak and only moderately improved. The precision and recall metrics show that while the model's ability to identify target speakers improved, it still struggled with accuracy and completeness. Overall, the model's effectiveness in its current state is not sufficient for reliable real-world applications, suggesting the need for further refinement and potentially more training data.

7. Conclusion

7.1 Summary

This project aimed to develop a real-time voice authentication system to combat the rising threat of spoofing scams in telecommunications. The primary objective was to leverage machine learning to provide speaker verification, particularly for identifying a target speaker in live voice calls, with a strong emphasis on detecting spoofed speech.

Within the project's timeframe, I was able to create two distinct but related models trained to perform binary classification tasks. The first model focused on detecting spoofed speech versus genuine speech, while the second model was designed to differentiate between a specific target speaker and non-target speakers. In evaluating the performance of the models developed for speaker authentication, significant insights emerge regarding their alignment with the initial project goal and their real-world applicability.

For the spoofed versus genuine speech model, the training results show a high degree of effectiveness in distinguishing between authentic and spoofed audio. The model demonstrated a substantial improvement in key metrics over time, reflecting its capability to accurately classify speech and detect spoofing attempts. The decrease in evaluation loss and the substantial increase in accuracy indicate that the model successfully refined its predictions and achieved near-perfect classification. Furthermore, high precision and recall values suggest that the model not only correctly identified spoofed speech with minimal false positives but also captured nearly all instances of spoofed speech, making it highly effective for the intended authentication task.

On the other hand, the target versus non-target speaker model exhibited notable improvements during training, though it faced some challenges. While there was a clear enhancement in performance metrics such as loss, accuracy, precision, and recall, the model did not reach the same level of effectiveness as the spoofed versus genuine model. The final performance, although better than initial results, indicates that the model still struggled with accurately identifying target speakers and may have missed some instances. This suggests that further refinement and additional training data could enhance its reliability.

The models were effectively trained and deployed as endpoints, allowing for real-time inferences on audio input. However, due to time constraints, the integration of these models into a live call environment for real-world testing was not achieved.

Despite the limitations, this project offers valuable contributions to the field of voice spoofing detection. The successful development of the proof of concept highlights the potential of machine learning, particularly deep learning techniques, for addressing this critical security challenge. The project also underscores the importance of further research and development in this area, particularly in exploring real-time implementation and testing in dynamic environments.

7.2 Future Work

This project serves as a foundation for future research in voice spoofing detection and real-time authentication. The following areas are open for further exploration to enhance the system and address its limitations:

Speaker Identification with Minimal Data: In the realm of speaker identification and verification, existing models excel at differentiating between speakers. However, a crucial challenge remains in ensuring that a speaker is accurately verified as who they claim to be. This project addressed this challenge by attempting to provide real-time voice authentication. Unfortunately, the model encountered a major issue - it needed substantial amounts of data for each target speaker to achieve reliable identification. This demand for extensive data presents practical difficulties, as accumulating hours of speech data per individual is not always feasible or efficient. Future work in this area could focus on optimizing the data requirements for speaker identification. This might involve developing techniques that require less data to achieve accurate verification or incorporating advanced methods that leverage limited data more effectively. Additionally, exploring innovative approaches in machine learning and speaker modeling could enhance the system's robustness and applicability in real-world scenarios.

Continuous Learning with Diverse Datasets: To ensure the system remains effective against evolving spoofing techniques, a continuous learning framework should be established. This involves regularly updating the training dataset with new and diverse voice samples, including a wider range of spoofing attacks, accents, languages, and acoustic environments. Implementing active learning strategies could help prioritize the most informative samples for labeling and training, further improving model performance and generalization.

By addressing these key areas, future research can build upon this project's foundation and contribute to the development of robust, adaptable, and user-friendly voice authentication systems for real-world applications.

8. References

Antispoofing. (2023, August 1). Voice anti-spoofing: origin and methods — Antispoofing Wiki. *Antispoofing Wiki*. https://antispoofing.org/voice-antispoofing-origin-types-and-preventive-techniques/

Aws. (n.d.). *GitHub - aws/sagemaker-huggingface-inference-toolkit*. GitHub. https://github.com/aws/sagemaker-huggingface-inference-toolkit/tree/main

Baevski, A., Zhou, H., Mohamed, A., & Auli, M. (2020, June 20). *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*. arXiv.org. https://arxiv.org/abs/2006.11477

Benhur, S. (2022, January 25). *A friendly introduction to Siamese networks*. Built In. https://builtin.com/machine-learning/siamese-network

Boyd, J., Fahim, M., & Olukoya, O. (2023). Voice spoofing detection for multiclass attack classification using deep learning. *Machine Learning With Applications*, *14*, 100503. https://doi.org/10.1016/j.mlwa.2023.100503

Clark, S. (2022, July 28). How digital voice technology is changing customer service. *CMSWire.com*. https://www.cmswire.com/customer-experience/how-digital-voice-technology-is-changin g-customer-service/

Costa, J. (2024, May 14). Counter AI Attacks with AI Defense. *Palo Alto Networks Blog*. https://www.paloaltonetworks.com/blog/2024/05/counter-with-ai-defense/

Dutt, A. (2022, August 1). Siamese Networks Introduction and Implementation - towards Data Science. *Medium*. https://towardsdatascience.com/siamese-networks-introduction-and-implementation-2140 e3443dee

Espinosa, C., & Espinosa, C. (2024, February 13). Understanding vishing tactics and prevention - Blue Goat Cyber. *Blue Goat Cyber*. https://bluegoatcyber.com/blog/understanding-vishing-tactics-and-prevention/

Fine-tune and deploy a Wav2Vec2 model for speech recognition with Hugging Face and Amazon SageMaker | Amazon Web Services. (2022, May 25). Amazon Web Services. https://aws.amazon.com/blogs/machine-learning/fine-tune-and-deploy-a-wav2vec2-mode l-for-speech-recognition-with-hugging-face-and-amazon-sagemaker/

GeeksforGeeks. (2023, December 7). *Hyperparameter tuning*. GeeksforGeeks. https://www.geeksforgeeks.org/hyperparameter-tuning/

Google Colab. (n.d.). https://colab.research.google.com/github/m3hrdadfi/soxan/blob/main/notebooks/Emotion_recognition_in_Greek_speech_using_Wav2Vec2.ipynb

Gupta, P., Patil, H. A., & Guido, R. C. (2024). Vulnerability issues in Automatic Speaker Verification (ASV) systems. *EURASIP Journal on Audio Speech and Music Processing*, *2024*(1). https://doi.org/10.1186/s13636-024-00328-8

Juang, B. H., Sondhi, M., & Rabiner, L. R. (2003). Digital Speech Processing. In *Elsevier eBooks* (pp. 485–500). https://doi.org/10.1016/b0-12-227410-5/00178-2

Khan, A., Mahmood Malik, K., Ryan, J., & Saravanan, M. (2023). Battling Voice Spoofing: A review, comparative analysis, and generalizability evaluation of state‐of‐the‐art voice spoofing counter measures. *Artificial Intelligence Review*. https://par.nsf.gov/servlets/purl/10427026

Marcel, S., Fierrez, J., & Evans, N. (2023a). Handbook of Biometric Anti-Spoofing. In *Advances in computer vision and pattern recognition*. https://doi.org/10.1007/978-981-19-5288-3

Marcel, S., Fierrez, J., & Evans, N. (2023b). Handbook of Biometric Anti-Spoofing. In *Advances in computer vision and pattern recognition*. https://doi.org/10.1007/978-981-19-5288-3

Martineau, K. (2023, August 18). *What is federated learning?* IBM Research. https://research.ibm.com/blog/what-is-federated-learning

Park, H., & Kim, T. (2022, October 10). *User Authentication Method via Speaker Recognition and Speech Synthesis Detection*. Wiley. https://onlinelibrary.wiley.com/doi/10.1155/2022/5755785

Saha, S. (2023a, April 8). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. *Medium*.

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Saha, S. (2023b, April 8). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. *Medium*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Wang, X., Yamagishi, J., Todisco, M., Delgado, H., Nautsch, A., Evans, N., Sahidullah, M., Vestman, V., Kinnunen, T., Lee, K. A., Juvela, L., Alku, P., Peng, Y., Hwang, H., Tsao, Y., Wang, H., Maguer, S. L., Becker, M., Henderson, F., . . . Ling, Z. (2019, November 5). *ASVspoof 2019: A large-scale public database of synthesized, converted and replayed speech*. arXiv.org. https://arxiv.org/abs/1911.01601

What is Hyperparameter Tuning? - Hyperparameter Tuning Methods Explained - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/hyperparameter-tuning/

Zhou, J., Hai, T., Jawawi, D. N. A., Wang, D., Ibeke, E., & Biamba, C. (2022). Voice spoofing countermeasure for voice replay attacks using deep learning. *Journal of Cloud Computing Advances Systems and Applications*, *11*(1). https://doi.org/10.1186/s13677-022-00306-5

9. Appendices

Accuracy: The proportion of correct predictions among the total number of predictions.

Argmax: Finds the index of the maximum value in a tensor or array.

Automatic Speaker Verification (ASV): A biometric system that verifies a speaker's identity based on their voice.

Binary Classification: A type of classification task that involves two classes (e.g., spoofed vs. genuine speech).

Bonafide Speech: Genuine, natural speech used for comparison in voice authentication systems.

Convolutional Neural Network (CNN): A type of deep learning model commonly used for image or audio processing, utilizing convolutional layers.

Decay: A technique to gradually reduce the learning rate during training.

Deep Learning: A subset of machine learning that involves neural networks with multiple layers.

Embedding: A dense vector representation of data.

Epoch: One complete pass through the entire training dataset.

Evaluation Metrics: Measurements used to assess the performance of a machine learning model (e.g., accuracy, precision, recall).

Feature Extraction: The process of extracting relevant features from raw data (e.g., audio signals).

Feature Extractor: A component that extracts relevant features from raw audio data.

FLAC: Free Lossless Audio Codec, a lossless audio compression format.

Generalization: The ability of a model to perform well on new, unseen data.

Gradient: The rate of change of a function with respect to its input variables.

Hyperparameter Tuning: The process of adjusting hyperparameters (e.g., learning rate) to optimize model performance.

Learning rate: A hyperparameter that controls the step size during gradient descent.

Logits: Raw output scores from a neural network before applying an activation function.

Loss function: A mathematical function that quantifies the error between the predicted and actual values.

Machine Learning Model: A computational model that learns from data to make predictions or decisions.

Neural Network: A machine learning model inspired by the structure of the human brain, composed of interconnected layers of nodes.

Normalize: Scaling data to a specific range (e.g., 0 to 1).

Overfitting: A scenario where a model performs well on training data but poorly on new, unseen data.

Precision: The proportion of correct positive predictions out of all positive predictions.

Real-Time Detection: Detecting events (e.g., voice spoofing) as they occur, without significant delay.

Recall: The proportion of actual positive cases correctly identified.

Recurrent Neural Network (RNN): A type of neural network that processes sequences, such as time-series data or audio signals.

Replay Attack: A type of attack where a previously recorded voice is played back to spoof an authentication system.

Speaker Verification: The process of verifying a speaker's identity based on their voice.

Speech Synthesis: The artificial production of human speech.

Spoofing Detection: Detecting attempts to deceive voice authentication systems using fake or altered voices.

Synthetic Voice: A computer-generated voice used in speech synthesis or for spoofing.

Training loss: The loss calculated on the training data during each epoch.

Voice Authentication: The process of verifying the identity of a speaker using their voice.

Voice Conversion: A technique that modifies one speaker's voice to sound like another's.

Voiceprint: A digital representation of a person's voice used for identification or authentication.

Wav2Vec: A pre-trained deep learning model designed for self-supervised learning of speech representations.