Sanjana M Santhosh

Semester VII
AI-ML , Computer Science and Engineering Dept. ,
Sree Chitra Thirunal College of Engineering ,
Pappanmkode

ASVspoof2019 Audio Spoofing Detection

Repo : https://github.com/Sanjana-Santhosh/spoofed_audio_classifier

June 27, 2025

Overview

The goal of this project was to develop a robust system to **detect spoofed** (fake) audio samples using deep learning. This is an important area in biometric security, since voice-based authentication systems are **vulnerable to attacks** via **synthetic or replayed speech**. I worked with the **ASVspoof2019 Logical Access (LA) dataset** to implement and train a classifier capable of distinguishing between bonafide (genuine) and spoofed audio recordings.

Throughout this work, I explored various approaches for feature extraction before settling on a modern **end-to-end pipeline using Wav2Vec 2.0**. I have documented my entire development process, along with the codebase, at my project's GitHub repository:

https://github.com/Sanjana-Santhosh/spoofed_audio_classifier

Dataset Exploration and Preparation

To begin, I spent time understanding the **ASVspoof2019 LA dataset's directory structure** and protocol. The dataset comprises around **25,000 audio files** in **.flac** format, and each file is paired with a label in a dedicated **protocol file**. Initially, I made sure all the audio files were properly unzipped and present in the required flac/ folder.

Reading and organizing the data was a key initial task. I created scripts that parsed the protocol files, efficiently **mapping each audio filename** to its corresponding label—**'bonafide' or 'spoof'**. During this process, I also wrote checks to **identify and handle any missing** or **corrupted** files gracefully, logging their absence to avoid pipeline errors later on.

Since the dataset is quite large, I decided to **limit the number of files** used for experiments initially. This was done mostly due to the **limited computational resources** available on my laptop. Still, I ensured that both classes were sufficiently represented for meaningful experimentation.

Feature Engineering Attempts

Before settling on an end-to-end model, I tried out classical **feature extraction strategies**. My first thought was to extract **Mel-Frequency Cepstral Coefficients (MFCCs)** and **spectrograms** from each audio file using the librosa library.

For each sample, I computed a matrix of **13 MFCC features** across multiple time frames. The resulting matrix captured what sounds are in the audio and how those sounds change from beginning to end. This helps the model decide if the audio is real or fake. and was initially intended as **input to a convolutional neural network (CNN)**. Although spectrograms were also considered, I didn't pursue them further since the MFCC approach was representative and widely used in audio classification.

From this phase, I observed that classical feature extraction works but comes with **limitations**—especially in terms of **requiring domain knowledge to select and tune features**. And as per your **recommendation** looked into modern neural approaches that can learn audio features directly from raw waveforms.

Model Architecture and Approach

Why Wav2Vec 2.0?

To move beyond hand-crafted features, I decided to leverage the **Wav2Vec 2.0 framework**. Wav2Vec 2.0 is a state-of-the-art self-supervised model developed by Facebook AI, designed to learn powerful representations directly from raw audio. I used the pretrained version available from **Hugging Face**: **"facebook/wav2vec2-base-960h"**.

The main idea here was to feed the raw waveform into the model and let it **automatically extract** and learn the **relevant features** needed for spoofing detection. No manual feature extraction was necessary. This not only simplified preprocessing but also let the model learn more features in the audio signals.

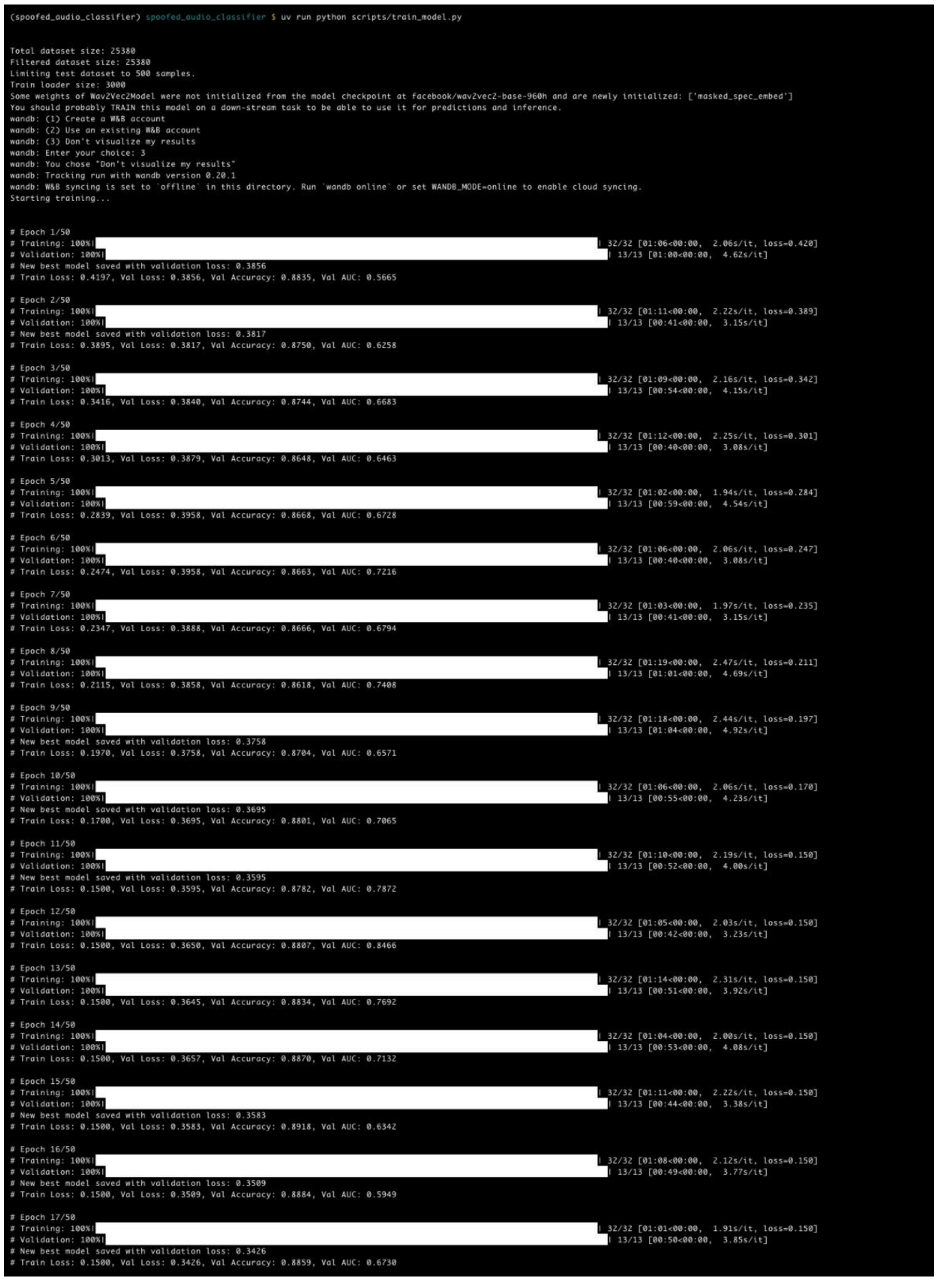
Model Implementation

The final architecture consisted of two main parts:

- **Feature Extractor**: This was the **frozen Wav2Vec 2.0 model,** which converted waveforms into high-dimensional embeddings. For efficiency, I froze the feature extractor and sometimes the initial transformer layers during training.
- **Classification Head**: On top of the Wav2Vec 2.0 embeddings, I built a small neural network with **two fully connected layers (with ReLU and dropout) that produced a 2-class output (bonafide or spoofed).**

I used **mean pooling over the time dimension** on the **final hidden states**, followed by the **classification head**. This setup allowed the model to summarize the temporal information from the audio into a **single feature vector** per file, which was then classified.

Training Strategy

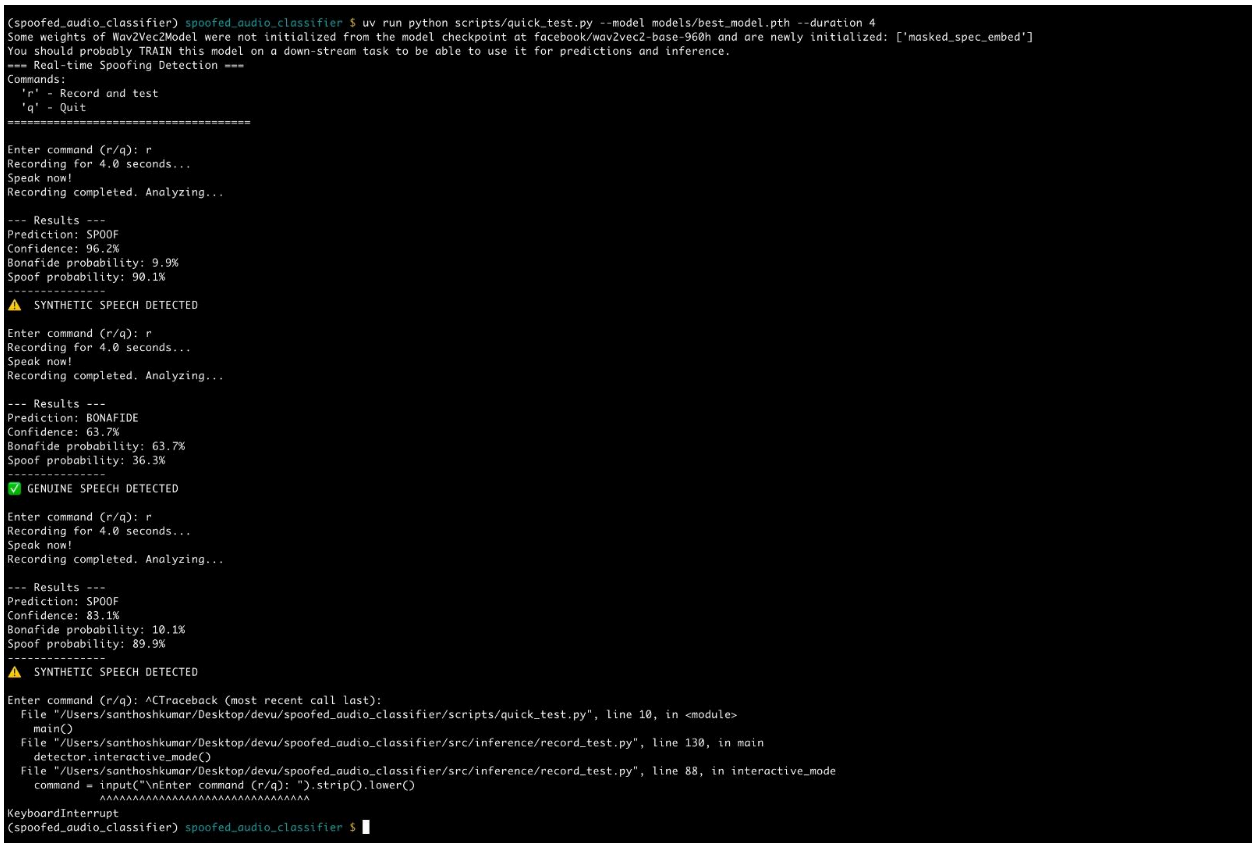


```
(spoofed_audio_classifier) spoofed_audio_classifier $ uv run python scripts/train_model.py


Total dataset size: 25380
Filtered dataset size: 25380
Limiting test dataset to 500 samples.
Train loader size: 3000
Some weights of Wav2Vec2Model were not initialized from the model checkpoint at facebook/wav2vec2-base-960h and are newly initialized: ['masked_spec_embed']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
wandb: (1) Create a W&B account
wandb: (2) Use an existing W&B account
wandb: (3) Don't visualize my results
wandb: Enter your choice: 3
wandb: You chose "Don't visualize my results"
wandb: Tracking run with wandb version 0.20.1
wandb: W&B syncing is set to `offline` in this directory. Run `wandb online` or set WANDB_MODE=online to enable cloud syncing.
Starting training...


# Epoch 1/50
# Training: 100%| | 32/32 [01:06<00:00,  2.06s/it, loss=0.420]
# Validation: 100%| | 13/13 [01:00<00:00,  4.62s/it]
# New best model saved with validation loss: 0.3856
# Train Loss: 0.4197, Val Loss: 0.3856, Val Accuracy: 0.8835, Val AUC: 0.5665

# Epoch 2/50
# Training: 100%| | 32/32 [01:11<00:00,  2.22s/it, loss=0.389]
# Validation: 100%| | 13/13 [00:41<00:00,  3.15s/it]
# New best model saved with validation loss: 0.3817
# Train Loss: 0.3895, Val Loss: 0.3817, Val Accuracy: 0.8750, Val AUC: 0.6258

# Epoch 3/50
# Training: 100%| | 32/32 [01:09<00:00,  2.16s/it, loss=0.342]
# Validation: 100%| | 13/13 [00:54<00:00,  4.15s/it]
# Train Loss: 0.3416, Val Loss: 0.3840, Val Accuracy: 0.8744, Val AUC: 0.6683

# Epoch 4/50
# Training: 100%| | 32/32 [01:12<00:00,  2.25s/it, loss=0.301]
# Validation: 100%| | 13/13 [00:40<00:00,  3.08s/it]
# Train Loss: 0.3013, Val Loss: 0.3879, Val Accuracy: 0.8648, Val AUC: 0.6463

# Epoch 5/50
# Training: 100%| | 32/32 [01:02<00:00,  1.94s/it, loss=0.284]
# Validation: 100%| | 13/13 [00:59<00:00,  4.54s/it]
# Train Loss: 0.2839, Val Loss: 0.3958, Val Accuracy: 0.8668, Val AUC: 0.6728

# Epoch 6/50
# Training: 100%| | 32/32 [01:06<00:00,  2.06s/it, loss=0.247]
# Validation: 100%| | 13/13 [00:40<00:00,  3.08s/it]
# Train Loss: 0.2474, Val Loss: 0.3958, Val Accuracy: 0.8663, Val AUC: 0.7216

# Epoch 7/50
# Training: 100%| | 32/32 [01:03<00:00,  1.97s/it, loss=0.235]
# Validation: 100%| | 13/13 [00:41<00:00,  3.15s/it]
# Train Loss: 0.2347, Val Loss: 0.3888, Val Accuracy: 0.8666, Val AUC: 0.6794

# Epoch 8/50
# Training: 100%| | 32/32 [01:19<00:00,  2.47s/it, loss=0.211]
# Validation: 100%| | 13/13 [01:01<00:00,  4.69s/it]
# Train Loss: 0.2115, Val Loss: 0.3858, Val Accuracy: 0.8618, Val AUC: 0.7408

# Epoch 9/50
# Training: 100%| | 32/32 [01:18<00:00,  2.44s/it, loss=0.197]
# Validation: 100%| | 13/13 [01:04<00:00,  4.92s/it]
# New best model saved with validation loss: 0.3758
# Train Loss: 0.1970, Val Loss: 0.3758, Val Accuracy: 0.8704, Val AUC: 0.6571

# Epoch 10/50
# Training: 100%| | 32/32 [01:06<00:00,  2.06s/it, loss=0.170]
# Validation: 100%| | 13/13 [00:55<00:00,  4.23s/it]
# New best model saved with validation loss: 0.3695
# Train Loss: 0.1700, Val Loss: 0.3695, Val Accuracy: 0.8801, Val AUC: 0.7065

# Epoch 11/50
# Training: 100%| | 32/32 [01:10<00:00,  2.19s/it, loss=0.150]
# Validation: 100%| | 13/13 [00:52<00:00,  4.00s/it]
# New best model saved with validation loss: 0.3595
# Train Loss: 0.1500, Val Loss: 0.3595, Val Accuracy: 0.8782, Val AUC: 0.7872

# Epoch 12/50
# Training: 100%| | 32/32 [01:05<00:00,  2.03s/it, loss=0.150]
# Validation: 100%| | 13/13 [00:42<00:00,  3.23s/it]
# Train Loss: 0.1500, Val Loss: 0.3650, Val Accuracy: 0.8807, Val AUC: 0.8466

# Epoch 13/50
# Training: 100%| | 32/32 [01:14<00:00,  2.31s/it, loss=0.150]
# Validation: 100%| | 13/13 [00:51<00:00,  3.92s/it]
# Train Loss: 0.1500, Val Loss: 0.3645, Val Accuracy: 0.8834, Val AUC: 0.7692

# Epoch 14/50
# Training: 100%| | 32/32 [01:04<00:00,  2.00s/it, loss=0.150]
# Validation: 100%| | 13/13 [00:53<00:00,  4.08s/it]
# Train Loss: 0.1500, Val Loss: 0.3657, Val Accuracy: 0.8870, Val AUC: 0.7132

# Epoch 15/50
# Training: 100%| | 32/32 [01:11<00:00,  2.22s/it, loss=0.150]
# Validation: 100%| | 13/13 [00:44<00:00,  3.38s/it]
# New best model saved with validation loss: 0.3583
# Train Loss: 0.1500, Val Loss: 0.3583, Val Accuracy: 0.8918, Val AUC: 0.6342

# Epoch 16/50
# Training: 100%| | 32/32 [01:08<00:00,  2.12s/it, loss=0.150]
# Validation: 100%| | 13/13 [00:49<00:00,  3.77s/it]
# New best model saved with validation loss: 0.3509
# Train Loss: 0.1500, Val Loss: 0.3509, Val Accuracy: 0.8884, Val AUC: 0.5949

# Epoch 17/50
# Training: 100%| | 32/32 [01:01<00:00,  1.91s/it, loss=0.150]
# Validation: 100%| | 13/13 [00:50<00:00,  3.85s/it]
# New best model saved with validation loss: 0.3426
# Train Loss: 0.1500, Val Loss: 0.3426, Val Accuracy: 0.8859, Val AUC: 0.6730
```

Due to **hardware constraints**, I trained the model on a subset of about **2,000 audio files**, with **batches of size 16** and a **learning rate of 1e-4.** The dataset was **split into training, validation, and testing** sets. I also set up checkpoints to save the model whenever the validation performance improved.

Early in training, I froze the Wav2Vec 2.0 layers to speed up convergence. Later, with more resources, it would be valuable to unfreeze more layers and fine-tune the full model.

Experimental Results and Observations



```
(spoofed_audio_classifier) spoofed_audio_classifier $ uv run python scripts/quick_test.py --model models/best_model.pth --duration 4
Some weights of Wav2Vec2Model were not initialized from the model checkpoint at facebook/wav2vec2-base-960h and are newly initialized: ['masked_spec_embed']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
=== Real-time Spoofing Detection ===
Commands:
  'r' - Record and test
  'q' - Quit
====================================

Enter command (r/q): r
Recording for 4.0 seconds...
Speak now!
Recording completed. Analyzing...

--- Results ---
Prediction: SPOOF
Confidence: 96.2%
Bonafide probability: 9.9%
Spoof probability: 90.1%
---------------
⚠️  SYNTHETIC SPEECH DETECTED

Enter command (r/q): r
Recording for 4.0 seconds...
Speak now!
Recording completed. Analyzing...

--- Results ---
Prediction: BONAFIDE
Confidence: 63.7%
Bonafide probability: 63.7%
Spoof probability: 36.3%
---------------
✅ GENUINE SPEECH DETECTED

Enter command (r/q): r
Recording for 4.0 seconds...
Speak now!
Recording completed. Analyzing...

--- Results ---
Prediction: SPOOF
Confidence: 83.1%
Bonafide probability: 10.1%
Spoof probability: 89.9%
---------------
⚠️  SYNTHETIC SPEECH DETECTED

Enter command (r/q): ^CTraceback (most recent call last):
  File "/Users/santhoshkumar/Desktop/devu/spoofed_audio_classifier/scripts/quick_test.py", line 10, in <module>
    main()
  File "/Users/santhoshkumar/Desktop/devu/spoofed_audio_classifier/src/inference/record_test.py", line 130, in main
    detector.interactive_mode()
  File "/Users/santhoshkumar/Desktop/devu/spoofed_audio_classifier/src/inference/record_test.py", line 88, in interactive_mode
    command = input("\nEnter command (r/q): ").strip().lower()
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
KeyboardInterrupt
(spoofed_audio_classifier) spoofed_audio_classifier $ 
```

Training on the full dataset was not feasible locally; it would have taken **more than 2 hours per epoch and a total of 50 epochs**. By working with a 2k sample subset, I could iterate more quickly and observe trends in model performance.

After about **11 epochs of training**, I observed that the **training loss plateaued at around 0.15,** with a best **validation accuracy of approximately 87.8%**.

The classifier showed **high confidence and accuracy on spoofed audio files (with 96.2% confidence in some cases)**, but somewhat **lower performance on bonafide files (reaching only about 63.7% confidence)**.

From these results, I suspect there was a **class imbalance in my sample selection**, with more spoofed than bonafide files. This would explain the strong bias towards detecting spoofed audio. In future iterations, ensuring balanced sampling and perhaps employing data augmentation could help address this.

Challenges and Learning

The biggest challenge by far was **computational**. Wav2Vec 2.0 is **resource-intensive**, and with limited hardware I had to compromise on dataset size and model tuning. However, this constraint also forced me to engineer a more efficient workflow and to focus on tracking clear validation metrics.

I also learned the importance of careful data selection and shuffling, as early results may be misleading due to sampling artifacts. Even so, working with Wav2Vec 2.0 gave me good exposure to modern representation learning techniques, and I appreciated how the model could essentially learn relevant features from raw data.

Conclusion and Future Directions

In summary, I was able to developed a working pipeline to detect audio spoofing on the ASVspoof2019 dataset using an end-to-end Wav2Vec 2.0 model. While the results are promising for spoofed audio, there is room for improvement in bonafide detection. I am confident that, with access to better hardware and more balanced training data, the system can be further improved.

For future work, I would like to:

- **Train on the full dataset** for more representative results.
- Explore **fine-tuning more layers** of Wav2Vec 2.0.
- Investigate other **transformer-based or ensemble approaches** (from a couple of websites I visited but didn't get time to explore further)..
- **Address class imbalance** and try **different data processing methods**.