CS478 Project Report

Deepfake Audio Detection with Neural Networks using Audio Features

Gabriel Tellez Ornelas
telle056@csusm.edu

Cristian Enriquez Tapia
enriq115@csusm.edu

Aiman Madan
madan005@csusm.edu

California State University San Marcos
Instructor: *Dr. Gutta*
sgutta@csusm.edu

Spring, 2025

Problem Statement:
Create a speech spoofing detection system based on Convolutional neural networks using different audio features.

The paper “Deepfake Audio Detection with Neural Networks using Audio Features” details a spoof detection system that utilizes audio data from the ASVspoof competition. The paper zooms in on the feature extraction processes for converting audio files into visual representations. The paper also details a CNN model for classifying the samples into genuine or spoof samples. The model is trained various times with different audio features: Spectrogram, Mel Spectrogram, Mel-Frequency Cepstral Coefficients, Fast Fourier Transform, and Short Time Fourier Transform. Allowing the features themselves to be merited. The paper found that MFCC features provided the best performance and alluded to the vibrancy of the image as the reason for that. It also found that Spectrograms provided the worst performance even though they are commonly used for analyzing audio.

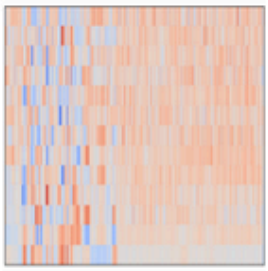
Dataset:

The paper uses the ASVspoof 2017 Version 2.0 dataset. The data comprises over 18 thousand audio samples of various lengths and speakers. The splits come predefined (as per the competition specifications) as follows:

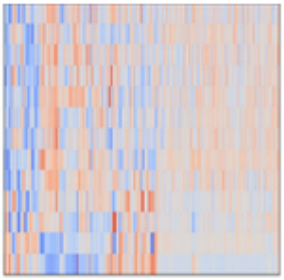
- Training set has a total of 3014 samples, 1507 genuine and 1507 spoof samples.
- Validation set has a total of 1710 samples, 760 genuine and 950 spoof samples.
- Testing set has a total of 13,306 samples, 1298 genuine and 12008 spoof samples.

Preprocessing

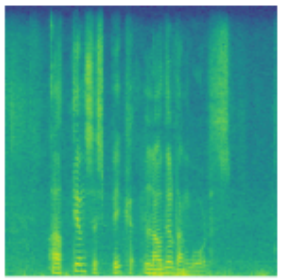
To prepare our dataset for use within our CNN models , we convert these samples into **MFCC** (Figure 1.0, 1.1) and **Spectrogram** (Figure 2.0,2.1) representations and standardize the size of each image to 200x200x3 to have a consistent input layer in the models.



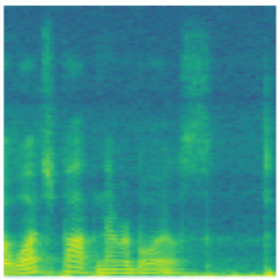
MFCC genuine Figure 1.0



MFCC genuine Figure 1.1



Spectrogram genuine Figure 2.0



Spectrogram genuine Figure 2.1

Original Implementation:

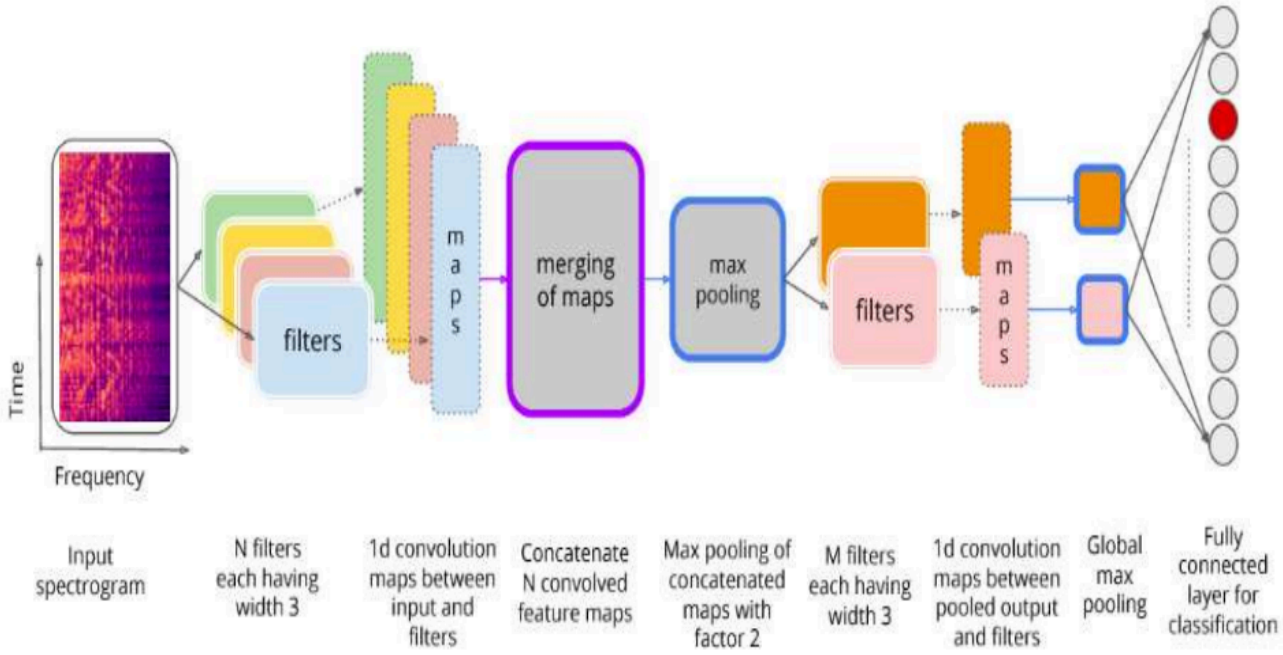


Figure 3.0 Diagram of the Paper's CNN model

The original Convolutional Neural Network model (Figure 3.0) from the paper has the following specifications:

Model Architecture

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 198, 198, 16)	448
max_pooling2d (MaxPooling2D)	(None, 99, 99, 16)	0
conv2d_1 (Conv2D)	(None, 97, 97, 8)	1,160
global_max_pooling2d (GlobalMaxPooling2D)	(None, 8)	0
dense (Dense)	(None, 1)	9

Figure 3.1 Summary of our CNN reimplementation

Training Setup

The model is trained for 30 epochs with 8 steps per epoch. The paper does not specify the optimizer so we defaulted to the Adam optimizer. Binary cross entropy is used as the loss function and accuracy is the metric tracked.

Results

Although our reimplementation slightly underperformed when compared to the paper's model -which got an accuracy of 70.86% on the Spectrograms and an accuracy of 75.97% on the MFCC on unseen data as demonstrated in their table 2-, we managed to obtain similar conclusions; That being, MFCC features resulted in better performance than Spectrogram features.

For our model, the Spectrograms resulted in an 80% accuracy on training and 54% on the testing set (Figure 3.2 and Figure 3.3). The MFCC training resulted in a 71% accuracy on the training set and 69% on the testing set (Figure 3.4 and Figure 3.5). Something that is interesting to note here is that although Spectrograms obtained the higher score on the training score, it also got the lowest score on the testing set; This is a 26% difference between seen and unseen data that signifies overfitting. Meanwhile, MFCC only has a 2% gap between the seen and unseen data meaning that there is little to no overfitting, a clear demonstration that MFCC performed better than Spectrograms.

```
=== Classification Report — Train ===
              precision    recall  f1-score   support

     Genuine       0.83      0.75      0.79      1507
       Spoof       0.77      0.84      0.81      1507

    accuracy                           0.80      3014
   macro avg       0.80      0.80      0.80      3014
weighted avg       0.80      0.80      0.80      3014
```

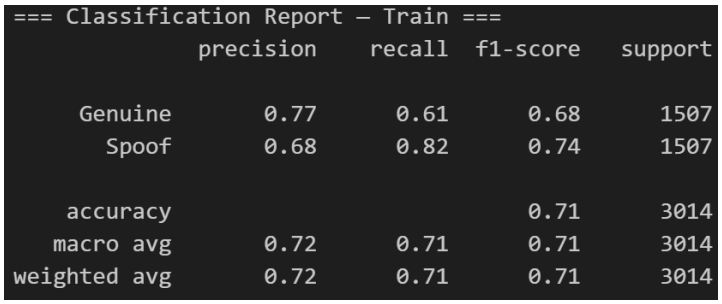
Figure 3.2
Spectrogram Results On Training Data

```
=== Classification Report — Test ===
              precision    recall  f1-score   support

     Genuine       0.11      0.54      0.19      1298
       Spoof       0.92      0.54      0.68     12008

    accuracy                           0.54     13306
   macro avg       0.52      0.54      0.44     13306
weighted avg       0.84      0.54      0.64     13306
```

Figure 3.3
Spectrogram Results on Test Data

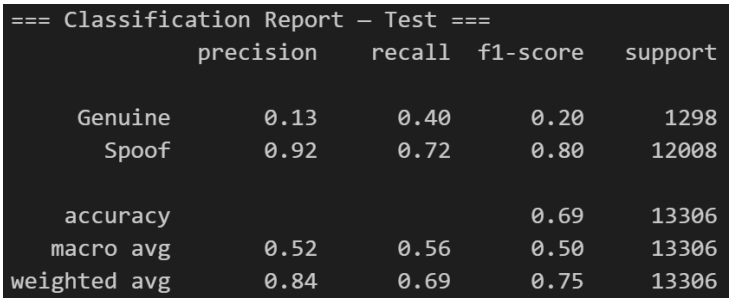


```
=== Classification Report — Train ===
              precision    recall  f1-score   support

     Genuine       0.77      0.61      0.68      1507
       Spoof       0.68      0.82      0.74      1507

    accuracy                           0.71      3014
   macro avg       0.72      0.71      0.71      3014
weighted avg       0.72      0.71      0.71      3014
```

Figure 3.4
MFCC Results On Training Data



```
=== Classification Report — Test ===
              precision    recall  f1-score   support

     Genuine       0.13      0.40      0.20      1298
       Spoof       0.92      0.72      0.80     12008

    accuracy                           0.69     13306
   macro avg       0.52      0.56      0.50     13306
weighted avg       0.84      0.69      0.75     13306
```

Figure 3.5
MFCC Results on Test Data

Alternate Implementations:

Wav2Vec:

Preprocessing

Rather than using the MFCC and Spectrogram images extracted from the raw audios, this approach takes advantage of the Wav2Vec pretrained model, a transformer that extracts high level features also known as embeddings. These high level features are then used as input within the CNN model. Additionally, since we are using embeddings rather than MFCC or Spectrogram images, the input shape is different with it being (768, 1) instead of (200,200,3).

Model Architecture & Training Setup

For the Wav2Vec implementation, both the CNN model and the training setup is the same as that of the original paper with the only difference being in using Convolutional 1D layers rather than Convolutional 2D layers due to the one dimensional nature of the embeddings. The goal for using the same CNN model is to compare how the results of using embeddings differ from using the MFCC and Spectrograms. The summary of this model can be seen in Figure 4.0 below.

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 766, 16)	64
max_pooling1d (MaxPooling1D)	(None, 383, 16)	0
conv1d_1 (Conv1D)	(None, 381, 8)	392
global_max_pooling1d (GlobalMaxPooling1D)	(None, 8)	0
dense (Dense)	(None, 1)	9

Figure 4.0 Summary of CNN model using Wav2Vec

Results

When it came to predicting on the testing dataset, it got the highest accuracy compared to MFCC and Spectrograms with 81%. However, when predicting on the testing dataset, this approach performed the worst with a 82% accuracy on the training dataset and a 51% accuracy on the testing dataset (Figure 4.1 and Figure 4.2). Not only did it obtain lower accuracies, the huge gap in accuracy between seen and unseen data of 30% hints at a greater overfitting issue compared to the other two features. Thus, these results point to the conclusion that using embeddings rather than extracting the MFCC and Spectrograms is not a sound strategy.

=== Classification Report – Train ===

	precision	recall	f1-score	support
Genuine	0.83	0.81	0.82	1507
Spoof	0.81	0.84	0.82	1507
accuracy			0.82	3014
macro avg	0.82	0.82	0.82	3014
weighted avg	0.82	0.82	0.82	3014

Figure 4.1
Wav2Vec Results On Training Data

=== Classification Report – Test ===

	precision	recall	f1-score	support
Genuine	0.13	0.73	0.23	1298
Spoof	0.94	0.49	0.64	12008
accuracy			0.51	13306
macro avg	0.54	0.61	0.43	13306
weighted avg	0.86	0.51	0.60	13306

Figure 4.2
Wav2Vec Results on Test Data

Modified CNN:

Since the model provided in the paper was not clear, we added a few layers in the replicated CNN.

Model Architecture:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 198, 198, 128)	3,584
max_pooling2d (MaxPooling2D)	(None, 99, 99, 128)	0
batch_normalization (BatchNormalization)	(None, 99, 99, 128)	512
conv2d_1 (Conv2D)	(None, 97, 97, 64)	73,792
max_pooling2d_1 (MaxPooling2D)	(None, 48, 48, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 48, 48, 64)	256
conv2d_2 (Conv2D)	(None, 46, 46, 32)	18,464
max_pooling2d_2 (MaxPooling2D)	(None, 23, 23, 32)	0
batch_normalization_2 (BatchNormalization)	(None, 23, 23, 32)	128
flatten (Flatten)	(None, 16928)	0
dense (Dense)	(None, 256)	4,333,824
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 1)	257

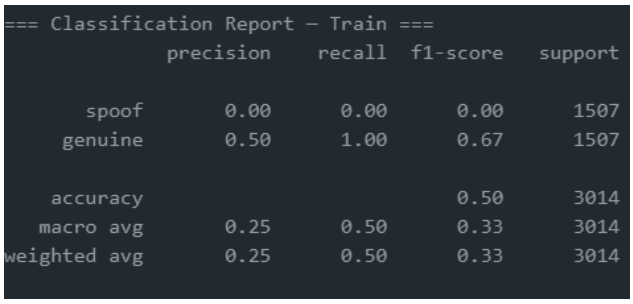
Figure 5.0 Modified CNN

Results:

The Alternate Modified CNN was run on the MFCC performance was quite poor. The model seems to be consistently predicting one class (genuine) for all inputs, as indicated by:

0 precision, recall, and F1-score for the "spoof" class across (train (figure 5.1), validation, and test (figure 5.2)).
The model never correctly identifies a spoof sample.

Perfect recall for the Genuine class across all datasets, but with very low precision on the validation and test sets. This suggests that while it labels all genuine samples as genuine, it also incorrectly labels many (or all) spoof samples as genuine.

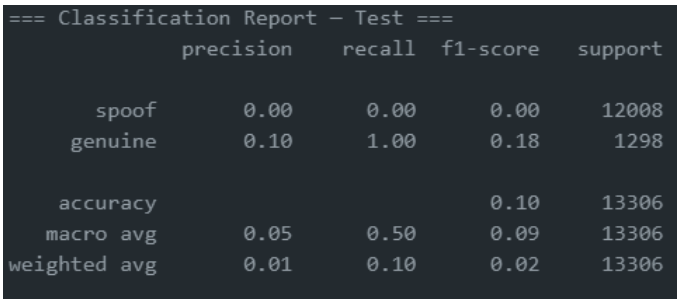


```
=== Classification Report - Train ===
              precision    recall  f1-score   support

       spoof       0.00      0.00      0.00      1507
     genuine       0.50      1.00      0.67      1507

    accuracy                           0.50      3014
   macro avg       0.25      0.50      0.33      3014
weighted avg       0.25      0.50      0.33      3014
```

Figure 5.1
Modified CNN Train Results



```
=== Classification Report - Test ===
              precision    recall  f1-score   support

       spoof       0.00      0.00      0.00     12008
     genuine       0.10      1.00      0.18      1298

    accuracy                           0.10     13306
   macro avg       0.05      0.50      0.09     13306
weighted avg       0.01      0.10      0.02     13306
```

Figure 5.2
Modified CNN Test Results

Transfer Learning (MobileNetV2):
This approach utilized the same preprocessed MFCC images (200x200x3). The images were fed directly into the MobileNetV2 model, which expects an input shape compatible with standard image data.

Preprocessing:
To augment the training data and improve model generalization, image augmentations were applied to the training set

Model Architecture & Training Setup:
A pre-trained MobileNetV2 model, with weights pre-trained on ImageNet, was employed as the convolutional base. The include_top parameter was set to False to remove the original classification layer, and input_shape was set to (200, 200, 3). The pooling='avg' argument was used to add a GlobalAveragePooling2D layer after the convolutional base.(Figure 6.0)

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 200, 200, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 1280)	2,257,984
dense (Dense)	(None, 64)	81,984
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

Figure 6.0 Classification Head

Phase 1: Training the custom head

The layers of the MobileNetV2 base model were initially frozen so that only the weights of the newly added classification head would be updated.

The model was compiled with the Adam optimizer using an initial learning rate of 1e-4.

Training was performed for up to 50 epochs using a batch size of 32.

An EarlyStopping callback was employed to monitor val_loss, with a patience of 5 epochs, restoring the best weights found during training to prevent overfitting.

Phase 2: Fine-tuning

After the initial training phase, the MobileNetV2 base model was made trainable.

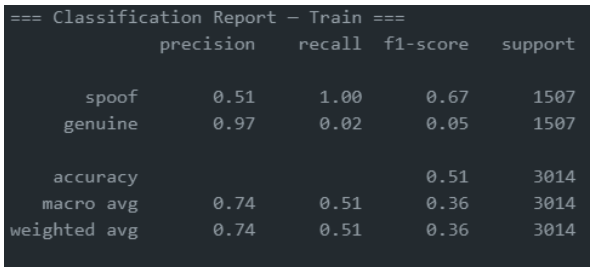
However, to avoid destroying the learned features and to allow for more subtle adjustments, only the top 20 layers of the MobileNetV2 base were unfrozen for fine-tuning. The earlier layers remained frozen.

The model was re-compiled with the Adam optimizer, using a lower learning rate of 1e-5 for fine-tuning.

The model was then trained for an additional period of up to 20 epochs, again using the EarlyStopping callback with the same configuration (monitoring val_loss, patience of 5) and a batch size of 32.

Results

The detailed classification performance on the training and testing datasets for the MFCC-based MobileNetV2 model can be seen in Figure 6.1 and Figure 6.2, respectively.

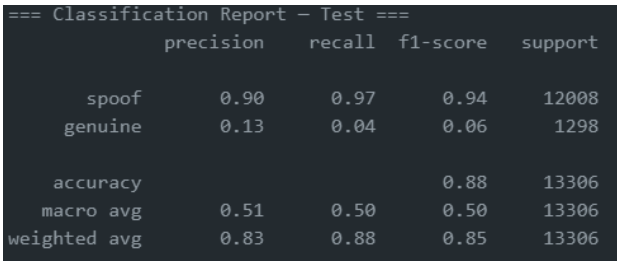


```
=== Classification Report - Train ===
              precision    recall  f1-score   support

       spoof       0.51      1.00      0.67      1507
     genuine       0.97      0.02      0.05      1507

    accuracy                           0.51      3014
   macro avg       0.74      0.51      0.36      3014
weighted avg       0.74      0.51      0.36      3014
```

Figure 6.1 Train Report



```
=== Classification Report - Test ===
              precision    recall  f1-score   support

       spoof       0.90      0.97      0.94     12008
     genuine       0.13      0.04      0.06      1298

    accuracy                           0.88     13306
   macro avg       0.51      0.50      0.50     13306
weighted avg       0.83      0.88      0.85     13306
```

Figure 6.2 Test Report

The performance of the MobileNetV2 model using MFCC features presented a complex picture. The training accuracy was low at 51% (Figure 6.1), where the model almost exclusively identified "spoof" samples (recall of 1.00 for spoof) while failing to recognize "genuine" samples (recall of 0.02 for genuine). This indicates a significant bias learned during training, despite the training set being balanced. The validation accuracy remained similarly low at 52%, with slightly improved but still poor detection of "genuine" samples.

The test accuracy reached 88% (Figure 6.2). However, this high overall accuracy is misleading. It is largely driven by the model's strong performance on the "spoof" class (F1-score of 0.94), which forms the vast majority of the imbalanced test set (12,008 spoof vs. 1,298 genuine samples). Conversely, the model performed extremely poorly on the "genuine" class in the test set, with an F1-score of only 0.06 (precision 0.13, recall 0.04).

These results suggest that the model did not generalize well. Instead of learning distinguishing features between the two classes, it heavily relied on the prevalence of the "spoof" class, particularly in the imbalanced test environment.

Multi-modal Transfer Learning with Dual MobileNetV2:

This approach explores a multi-modal architecture, leveraging transfer learning by using two separate MobileNetV2 backbones to process Spectrogram and MFCC image features simultaneously. The extracted features are then combined for final classification.

Preprocessing:
The training utilized a **tf.data.Dataset pipeline** with the **multimodal_data_generator** for efficient data loading and augmentation, prefetching data for performance.

Model Architecture & Training Setup:

Layer (type)	Output Shape	Param #	Connected to
spectrogram_input (InputLayer)	(None, 200, 200, 3)	0	-
mfcc_image_input (InputLayer)	(None, 200, 200, 3)	0	-
mobilenet_for_spec (Functional)	(None, 1280)	2,257,984	spectrogram_inpu…
mobilenet_for_mfcc (Functional)	(None, 1280)	2,257,984	mfcc_image_input…
concatenated_featu… (Concatenate)	(None, 2560)	0	mobilenet_for_sp… mobilenet_for_mf…
dense (Dense)	(None, 512)	1,311,232	concatenated_fea…
batch_normalization (BatchNormalizatio…	(None, 512)	2,048	dense[0][0]
dropout (Dropout)	(None, 512)	0	batch_normalizat…
dense_1 (Dense)	(None, 256)	131,328	dropout[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 256)	1,024	dense_1[0][0]
dropout_1 (Dropout)	(None, 256)	0	batch_normalizat…
dense_2 (Dense)	(None, 64)	16,448	dropout_1[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 64)	256	dense_2[0][0]
dropout_2 (Dropout)	(None, 64)	0	batch_normalizat…
output (Dense)	(None, 1)	65	dropout_2[0][0]

Figure 7.0 Dual MobileNetV2 Classification Head

This model features a dual-branch design:

Spectrogram Branch: A MobileNetV2 model pre-trained on Image net, with its top classification layer removed and global average pooling.

MFCC Branch: A separate MobileNetV2 model, also pre-trained on Image net, with the same setup .

Initially, all layers in both MobileNetV2 backbones were frozen.

Feature Fusion:

The output features after global average pooling from the Spectrogram branch and the MFCC branch were concatenated along the feature axis.

The concatenated features then passed through a the classification head (Figure 7.0)

The overall model takes two image inputs Spectrogram & MFCC and produces a single binary output.

Phase 1: Training the custom head

The MobileNetV2 backbones for both Spectrogram and MFCC branches remained frozen. Only the custom classification head was trained.

Training was performed for 30 epochs. EarlyStopping callback was employed to monitor val_loss, with a patience of 5 epochs, restoring the best weights found during training to prevent overfitting.

Phase 2: Fine-tuning

The top 10 layers of both the spec_branch and the mfcc_branch were unfrozen and made trainable.

The earlier layers in both backbones remained frozen.

Optimizer: The model was re-compiled with the Adam optimizer, using a lower learning rate of 1e-5.

Results:

The multi-modal approach yielded a training accuracy of 84%, with the model demonstrating good F1-scores for both classes on the balanced training data (0.86 for spoof, 0.81 for genuine). This suggests the architecture could learn from the combined Spectrogram and MFCC features during training, exhibiting high recall for "spoof" (0.99) and reasonable recall for "genuine" (0.69). The Phase 1 and Phase 2 training plots show the learning dynamics, with validation accuracy appearing to plateau or degrade relative to training accuracy, hinting at emerging overfitting even during these phases.

However, the performance on the test set dropped dramatically to an accuracy of 49%. This stark difference between training and test accuracy points to significant overfitting, meaning the model did not generalize well to unseen data.

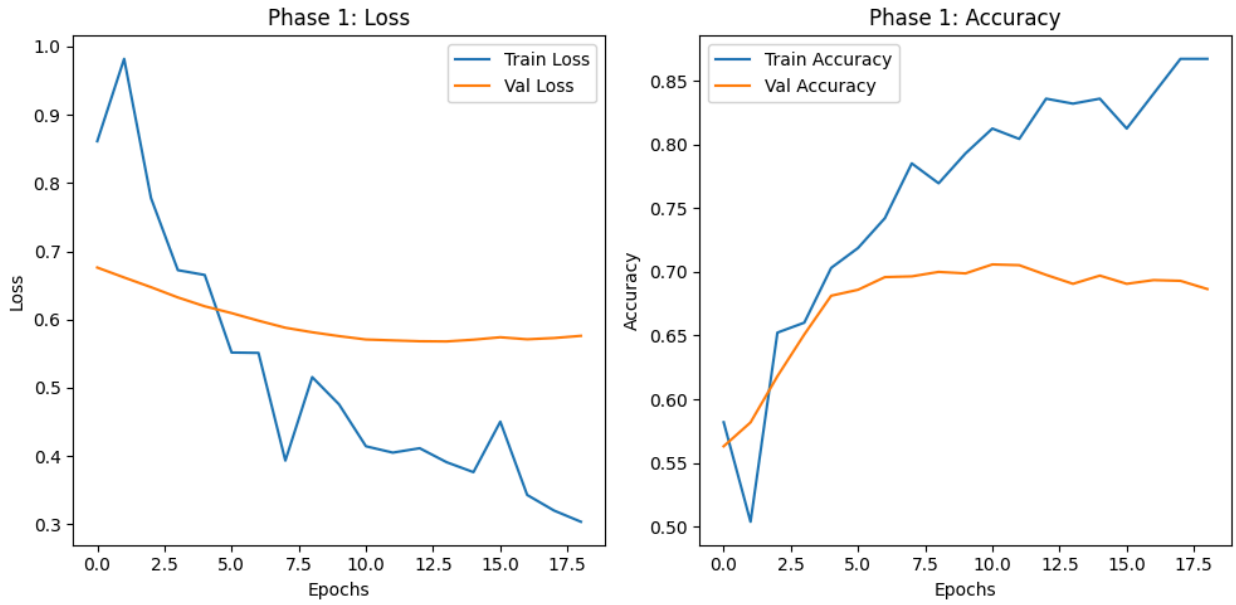


Figure 8.0 Phase One

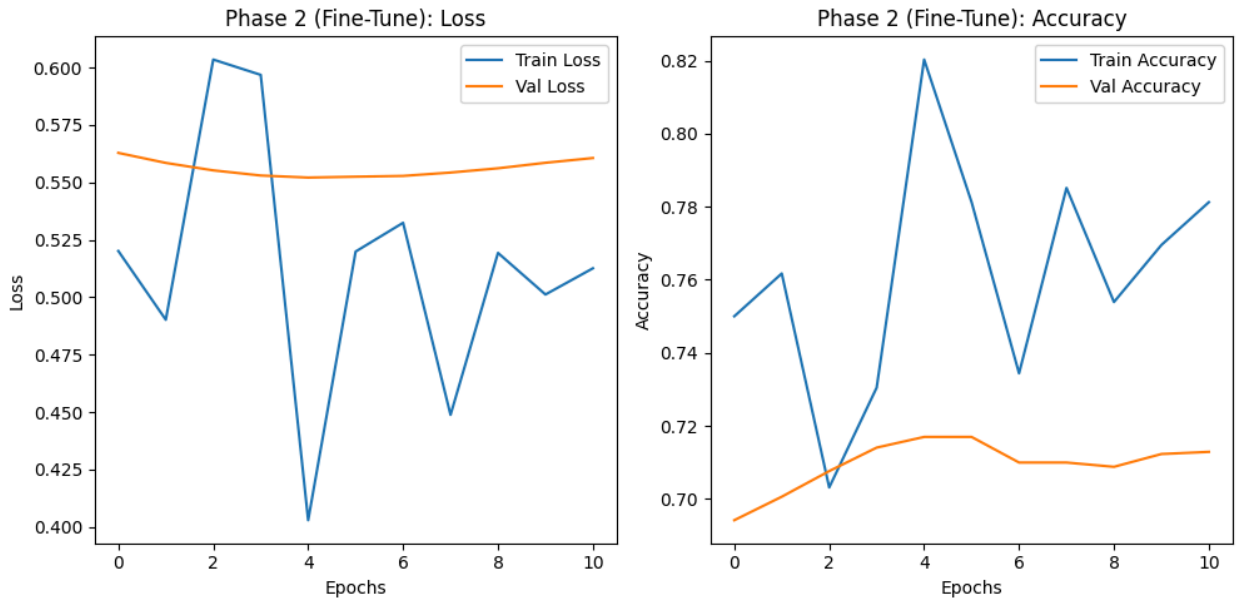


Figure 9.0 Phase Two

Challenges

The project encountered several challenges spanning data preprocessing, model implementation:

- **Replicating Paper's Feature Extraction:** A primary hurdle was accurately replicating the feature extraction methods detailed in the reference paper. Many libraries mentioned were outdated, and critical hyperparameters for extraction processes like MFCC were not specified. For instance, the visual representation of MFCCs in the paper differed from what standard library functions produced initially, necessitating a trial-and-error approach to approximate the paper's examples. Spectrogram extraction was comparatively more straightforward due to its more standardized nature.

- **Efficient Feature Generation:** Converting the entire dataset of over 18 thousand audio files into their respective visual feature representations (200x200x3 images like Spectrograms and MFCCs) was computationally intensive. Initial attempts to convert just 3,000 audio samples into MFCC

images took over 9 hours. To overcome this, the feature extraction code was optimized, and a multiprocessing strategy was implemented, distributing the workload across 10 CPU cores to perform conversions in parallel. This significantly reduced the processing time to approximately 2 hours for the entire dataset.

- **Reimplementing Paper's Model:** Ambiguities in the original paper's model architecture and training specifications posed another challenge. Details such as the use of 1D convolutions for image-like inputs (which might have been a common practice at the time of the paper's release) and the lack of specifics on model compilation (optimizer, learning rate) required us to make informed assumptions and default to common practices in deep learning.

- **Dataset Imbalance:** A major difficulty inherent to the ASVspoof 2017 dataset is the significant class imbalance. While the training set is balanced (1507 genuine vs. 1507 spoof), the validation set has a slight imbalance (760 genuine vs. 950 spoof), and the testing set is severely imbalanced (1298 genuine vs. 12008 spoof). This skew, especially in the test set, makes robust model training and evaluation challenging, as models can easily achieve high accuracy by favoring the majority class, leading to poor performance on the minority class and difficulties in true generalization. This was evident in the results of most of our implementations.

- **Limited Training Data Size and Overfitting:** The training dataset, with only 3014 samples, is relatively small for training deep convolutional neural networks (CNNs) or fine-tuning large pre-trained models effectively. This limited data makes it difficult for models to learn a comprehensive set of distinguishing features, often leading to memorization of the training examples. Consequently, many of our models, including the custom CNNs and transfer learning approaches, exhibited signs of overfitting, performing well on training data but poorly on unseen test data.

- **Spectrogram Processing and Model Sensitivity:** While Spectrograms are a common audio feature, achieving optimal performance with them was

challenging. The visual characteristics of Spectrograms can vary widely based on extraction parameters (e.g., FFT window size, hop length, frequency scaling). Fine-tuning these parameters and then designing a CNN architecture that is sensitive to the subtle discriminative patterns within them, without overfitting to noise or irrelevant details, proved difficult.

- **Modified CNN Design and Performance:** Developing the "Modified CNN" involved iterating on architectural choices (number of layers, filter sizes, activation functions, regularization) to improve upon the baseline or original paper's model. A key challenge was to create a model complex enough to capture relevant features but not so complex that it would drastically overfit the small training set. As seen in its results, this model struggled significantly, consistently predicting one class and failing to identify spoof samples, indicating issues with learning discriminative features or getting stuck in a suboptimal local minimum during training.

- **Transfer Learning Single MobileNetV2:** Implementing transfer learning with MobileNetV2 (for MFCCs) required careful decisions regarding which layers to freeze versus fine-tune, and selecting appropriate learning rates for the two-phase training. Despite these strategies and data augmentation, the model still showed significant overfitting, and its performance on the minority "genuine" class in the imbalanced test set was very poor, highlighting the difficulty of adapting a model pre-trained on general images (ImageNet) to the specific domain of audio-derived images under dataset limitations.

- **Multi-modal Dual MobileNetV2:** The multi-modal approach, using separate MobileNetV2 branches for Spectrograms and MFCCs, introduced further complexity. Challenges included:
    - **Data Handling:** Managing and augmenting two separate input streams correctly and feeding them into the model required a custom data generator.

- **Architecture Complexity:** Designing the fusion mechanism for the features from the two branches.

Future Work

Two of the main issues described in the previous section included having very little data along with a huge imbalance in it, which can lead to overfitting. Thus, an area of improvement to solve this issue is finding ways to augment our data. One method to achieve this is to record more genuine data along with generating spoof data from scratch. However, we have also proposed the following two ideas to generate synthetic audio:

- Changing the pitch of the audio to see if our CNN models can learn to identify spoof and genuine features at different frequencies.
- Creating multiple audio files from one big audio file with each of them starting and ending at different time stamps (this would be similar to augmenting image data by shifting or rotating it).
- Attention-like fusion: instead of pure concatenation, we try a small attention module that learns to weigh each modality features.

The second area of improvement is fine-tuning the pretrained Wav2Vec model to get better results when extracting embeddings. With our current implementation, Wav2Vec extracts the model using the Tensorflow library which limits our ability to fine tune due to the limited support. Thus, rather than using Tensorflow we plan on fine-tuning the Wav2Vec by using PyTorch.

Contribution

This project was a collaborative effort, with each team member taking lead responsibility for specific components and contributing to the overall development, experimentation, and analysis:

- **Gabriel Tellez Ornelas:** Led the initial research analysis to define the project's direction and select appropriate methodologies based on existing literature. Spearheaded the data preprocessing efforts, specifically developing and implementing the pipeline for MFCC feature extraction from the raw audio data. Defined the evaluation metrics framework and was responsible for specifying how model performance would be assessed and reported.

- **Cristian Enriquez Tapia:** Took charge of reimplementing the baseline CNN model described in the reference paper, forming a benchmark for comparison. Developed and experimented with the Wav2Vec transformer-based model, focusing on extracting and utilizing audio embeddings for the spoof detection task. Contributed significantly to the iterative model development lifecycle and troubleshooting technical challenges across different implementations.

- **Aiman Madan:** Managed the end-to-end Spectrogram processing pipeline, including feature extraction and preparation for model input. Designed, implemented, and systematically evaluated the custom "Modified CNN" architecture. Led the exploration and implementation of transfer learning techniques, which included developing and training the single MobileNetV2 model (for both MFCC and Spectrogram inputs) and the more complex multi-modal dual MobileNetV2 architecture that processed both feature types concurrently.

All team members actively participated in discussions, problem-solving sessions, results interpretation, and the final compilation of the project report.