Deepfake Detection

End-Eval

Group 6

Group: What’s in a sound?

Shashwat, Samarth,
Arnav, Rigved, Mahir

Mentors

Arin Dhariwal
Deham Rajvanshi
Divyam Agarwal
Shreyash Dwivedi

INTRODUCTION -DEEPFAKE DETECTION

Deepfake technology poses significant risks, including misinformation, identity theft, and the spread of fabricated content that can damage reputations and manipulate public opinion. It is widely exploited for fraud, political propaganda, and cybercrime. Deepfake detection plays a crucial role in mitigating these threats by identifying manipulated media, ensuring authenticity, and preventing misuse, thereby safeguarding individuals, organizations, and society from deception, financial loss, and ethical concerns.

Description of the Dataset

The ASVspoof dataset is a collection of audio samples used for developing and evaluating automatic speaker verification (ASV) systems to detect spoofing attacks. It includes various attack types such as Speech Synthesis (SS), Voice Conversion (VC), and Replay Attacks (RA). The dataset is available in WAV file format, with most recordings sampled at 16 kHz or 8 kHz. The ASVspoof 2015 version contains about 3,400 samples with a file size of approximately 1.5 GB, while the larger ASVspoof 2021 includes over 30,000 samples with a file size of around 20-25 GB. The dataset is essential for advancing robust anti-spoofing techniques in voice-based authentication systems.

Preprocessing Of the Dataset

```
!pip install librosa
import os
import tqdm
import numpy as np
import librosa
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2
D, Flatten, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical
```

```
# Define paths and parameters
DATASET_PATH = "/kaggle/input/asvpoof-2019-dataset/LA/LA/ASVsp
oof2019_LA_train/flac"
LABEL_FILE_PATH = "/kaggle/input/asvpoof-2019-dataset/LA/LA/AS
Vspoof2019_LA_cm_protocols/ASVspoof2019.LA.cm.train.trn.txt"
NUM_CLASSES = 2  # Number of classes (bonafide and spoof)
SAMPLE_RATE = 16000  # Sample rate of your audio files
DURATION = 5  # Duration of audio clips in seconds
N_MELS = 128  # Number of Mel frequency bins
```

The Preprocessing Part begins with loading of data and importing all the relavant modules

```
    # Extract Mel spectrogram using librosa
    mel_spectrogram = librosa.feature.melspectrogram(y=audio,
sr=SAMPLE_RATE, n_mels=N_MELS)
    mel_spectrogram = librosa.power_to_db(mel_spectrogram, ref
=np.max)

    # Ensure all spectrograms have the same width (time steps)
    if mel_spectrogram.shape[1] < max_time_steps:
        mel_spectrogram = np.pad(mel_spectrogram, ((0, 0), (0,
max_time_steps - mel_spectrogram.shape[1])), mode='constant')
    else:
        mel_spectrogram = mel_spectrogram[:, :max_time_steps]

    X.append(mel_spectrogram)
    y.append(label)
```

The audio dataset has been preprocessed by extracting Mel spectrograms, normalizing, padding/truncating, and mapping them to labels for model training.

The spectrograms are then used to train the model

Why Mel Spectrograms?

- Emphasize speech-relevant frequencies, making detection more effective.
- STFT vs. Mel Spectrograms: STFT treats all frequencies equally, whereas Mel spectrograms align with human hearing and focus on important speech-related frequencies.
- Capture deepfake artifacts that are hard to notice in raw waveforms.
- Work well with CNNs, treating spectrograms as images for feature extraction.

Training The Model

CNN ARCHITECTURE

```
x = Conv2D(32, kernel_size=(3, 3), activation='relu')(model_input)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(64, kernel_size=(3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
model_output = Dense(NUM_CLASSES, activation='softmax')(x)
```

CNN Model Architecture

1️⃣ Feature Extraction:

Conv2D (32, 3×3, ReLU): Detects patterns in the Mel spectrogram.

MaxPooling (2×2): Reduces spatial dimensions.

Conv2D (64, 3×3, ReLU) + MaxPooling: Extracts deeper features.

2️⃣ Flattening & Fully Connected Layers:

Flatten(): Converts 2D features to 1D.

Dense (128, ReLU) + Dropout (50%): Prevents overfitting and learns high-level features.

3️⃣ Output Layer:

Dense(NUM_CLASSES, Softmax): Classifies audio as real or deepfake.

OPTIMIZATION AND TRAINING

```
model = Model(inputs=model_input, outputs=model_output)
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Train the Model
model.fit(X_train, y_train, batch_size=32, epochs=10, validation_data=(X_val, y_val))
```

This code defines, compiles, and trains a Convolutional Neural Network (CNN) for deepfake audio classification. The model consists of convolutional layers (Conv2D) with ReLU activation for feature extraction, max pooling layers for downsampling, and fully connected layers for classification. The final layer uses softmax activation for multi-class classification. The model is compiled using the Adam optimizer and categorical cross-entropy loss, and it is trained with a batch size of 32 for 10 epochs while validating on a separate dataset.

Visualization

Predicting using the model

```
# Convert list to numpy array
X_test = np.array(X_test)

# Predict using the loaded model
y_pred = model.predict(X_test)

# Convert probabilities to predicted classes
y_pred_classes = np.argmax(y_pred, axis=1)

y_pred
```

```
protocol_true_labels = np.array([true_labels[file_name] for file_name in valid_files])
```

Compare with true labels

Understanding accuracy using confusion matrix

A confusion matrix is a simple table that shows how well a classification model is performing by comparing its predictions to the actual results. It breaks down the predictions into four categories: correct predictions for both classes (true positives and true negatives) and incorrect predictions (false positives and false negatives)

	Predicted	Predicted
Actual	True Positive (TP)	False Negative (FN)
Actual	False Positive (FP)	True Negative (TN)

2x2 confusion matrix for binary classification

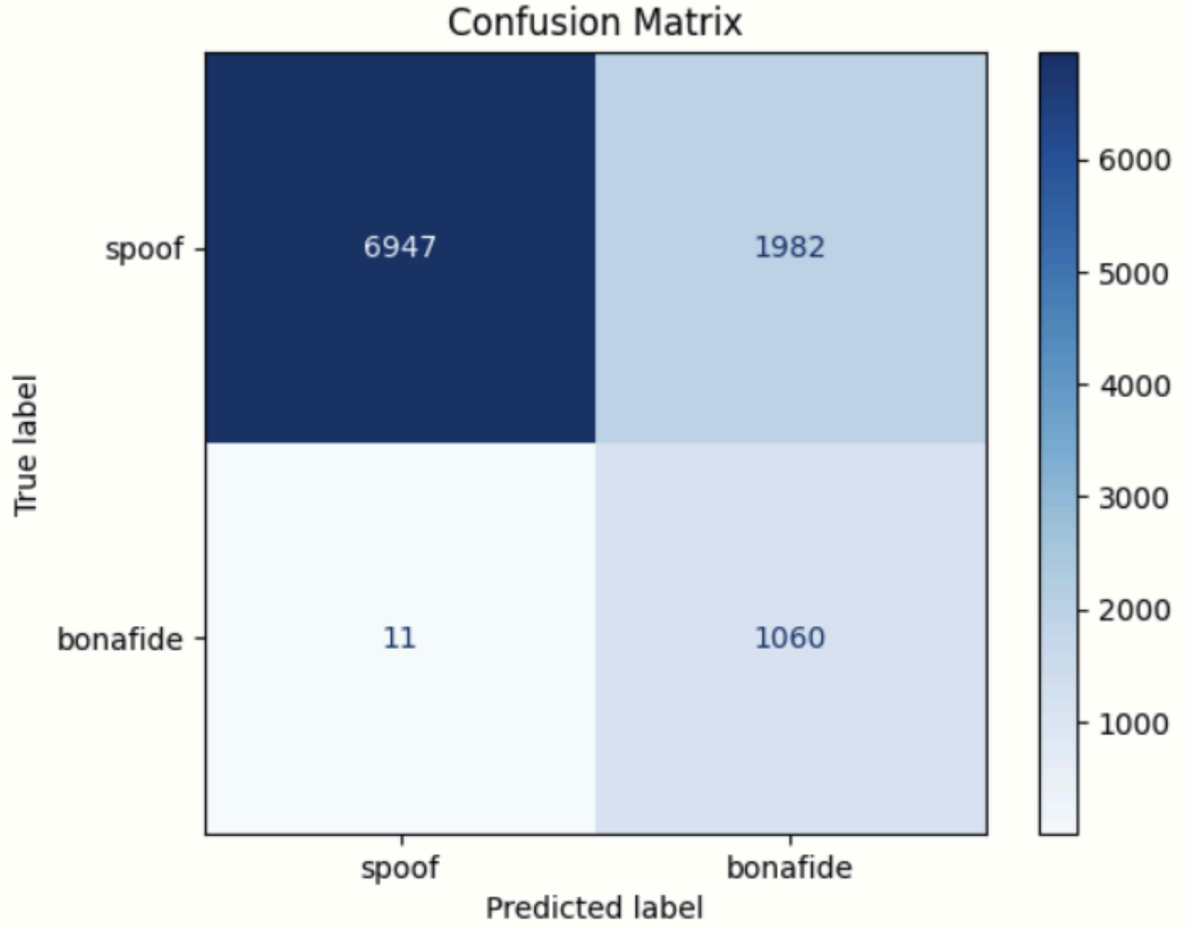
```python
cm = confusion_matrix(protocol_true_labels[:100], y_pred_classes[:100])

# Display the confusion matrix
classes = ["spoof", "bonafide"]
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```

TP = 6947

FN = 1982

FP = 11

TN = 1060



Different metrics for visualizing the results

1. **Accuracy**

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

= (6947+1060)/10000
= 80.07%

2. **Precision**

$$\text{Precision} = \frac{TP}{TP+FP}$$

= 6947/6958
= 99.84%

3. **Recall**

$$\text{Recall} = \frac{TP}{TP+FN}$$

=6947/8929
=77.8%

4. F1-score

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

= (2*0.9984*0.78) / 1.7784
=0.88

5. Specificity

$$\text{Specificity} = \frac{TN}{TN + FP}$$

= 1060 /(1060+ 11)
=0.99

6. Type- 1 and 2 errors

$$\text{Type 1 Error} = \frac{FP}{TN + FP}$$

=11/1071= 0.011

$$\text{Type 2 Error} = \frac{FN}{TP + FN}$$

=1982/8929= 0.22

THANK YOU!!!!