🔍 FakeSoundDetector - Audio Deepfake Detection

Candidate Name: Navneet Naman
Project Name: FakeSoundDetector
GitHub Repository: https://github.com/NavneetNaman/FakeSoundDetector
Dataset Used: Kaggle Audio Deepfake Detection Dataset
Frontend: React
Backend: Node.js (Express)
Model Type: Deep Neural Network (TensorFlow, Librosa)
Model Accuracy: 71%
Model File: Saved model used during inference (no training required at runtime)

🧪 Part 1: Research & Model Selection

We studied three forgery detection models from the curated Audio-Deepfake-Detection GitHub repo, based on their relevance to detecting AI-generated speech and suitability for near real-time detection.

1. RawNet2

- **Key Technical Innovation**: End-to-end deep model using raw waveform input instead of spectrograms.
- **Performance**: Achieved high accuracy in the ASVspoof 2019 challenge.
- **Why Promising**: No need for feature extraction, and directly learns from the raw waveform.
- **Limitations**: Heavy model; slow inference; harder to deploy on lightweight setups.

2. X-vector + PLDA

- **Key Technical Innovation**: Speaker embeddings (x-vectors) with Probabilistic Linear Discriminant Analysis (PLDA) for classification.
- **Performance**: Performed well in speaker verification challenges.

- **Why Promising**: Good at capturing speaker-specific characteristics.
- **Limitations**: Requires well-aligned datasets; complex pipeline; not real-time friendly.

3. SpecRNet (Spectrogram-based CNN) – ✅ Selected Approach

- **Key Technical Innovation**: Uses spectrograms (visual representation of audio) with CNN layers for forgery detection.
- **Performance**: Balanced accuracy with good generalization to unseen attacks.
- **Why Promising**: Efficient with low computational cost; easy to scale and deploy.
- **Limitations**: May need careful preprocessing and tuning for spectrogram consistency.

We selected **SpecRNet** as our base approach and implemented a simplified version using **Mel-spectrograms + Deep Neural Network (DNN)** for ease of training and deployment.

⚙️ Part 2: Implementation

🛠 Model Architecture

python

CopyEdit

```
model = tf.keras.Sequential([
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

🔗 Training Setup

- **Feature Extraction**: Mel-spectrogram (via Librosa)
- **Optimizer**: Adam
- **Loss Function**: Binary Crossentropy
- **Epochs**: 10
- **Batch Size**: 32

📊 Model Results

Metric	Score
Accuracy	71%
Precision	0.74 (weighted avg)
F1-score	0.69
Recall (Real)	0.90
Recall (Fake)	0.51

- The model performs well on **real audio**, but struggles more with detecting **fake audio**, which is common in many deepfake detection systems.

📌 Part 3: Documentation & Analysis

💡 Implementation Process

- **Audio files** were preprocessed using **Librosa** to generate **Mel-spectrograms**.
- Features were flattened and normalized using **StandardScaler**.
- We built a **lightweight DNN model** with dropout layers to avoid overfitting.
- After training, the model was saved (.h5) and reused for inference.
- The backend was built using **Node.js (index.js)** to connect with the frontend.

🚧 Challenges Encountered

1. **Class Imbalance in Detection**: The model had a high recall on real audio (90%) but struggled to recall fake audio (51%). This imbalance affected the overall F1-score.
2. **Audio Feature Variability**: Differences in duration and quality across fake samples made consistent feature extraction difficult.
3. **Model Overfitting**: Without dropout, the model overfit quickly. We addressed this by using two dropout layers.
4. **Inference Latency**: We avoided complex CNNs to keep response time fast on local systems.
5. **Data Volume**: The full dataset (over 1.5 GB) was used during training. For GitHub, we removed the training/validation data and kept only the test files, reducing the size to 479 MB.

📈 Why We Selected SpecRNet

- **Lightweight and interpretable**: Spectrograms give visual insight into audio signals.
- **Easier to implement and debug** than end-to-end raw audio models.
- **Faster inference** due to smaller model size.
- **Real-time capable** after optimization and model saving.

🔬 How It Works (High-level)

1. Audio is converted into a Mel-spectrogram.
2. The spectrogram is flattened into a 1D array.
3. The array is scaled and passed to the DNN.
4. The DNN predicts whether the audio is real (0) or fake (1).
5. Results are shown in the frontend with percentage and color-coded output.

🧠 **Reflection**

1. What were the most significant challenges in implementing this model?

- The biggest challenge was **low recall for fake audio**, meaning the model sometimes classified fake audio as real.
- Handling **inconsistent audio formats** and lengths also required careful feature extraction and cleaning.

2. How might this approach perform in real-world conditions?

- The model performs reasonably well on structured test data, but **may struggle with noisy or low-quality audio in real-world settings**.
- More robust preprocessing and data augmentation would be needed.

3. What additional data or resources would improve performance?

- Including more **diverse fake audio samples**, especially from newer AI models (e.g., ElevenLabs, Play.ht).
- Using **data augmentation** like noise addition, pitch shift, and time-stretching.
- Trying **CNN or transformer-based architectures** for deeper feature extraction.

4. How would you approach deploying this model in a production environment?

- Convert the trained model to **TensorFlow Lite or ONNX** for faster deployment.
- Use **Docker** to containerize the backend.
- Deploy on a cloud service (e.g., AWS Lambda or GCP Cloud Run) for scalability.
- Add **rate limiting, file size restrictions**, and **logging** for monitoring usage.