Deepfake Detection Using Frame and Temporal Features

1. Introduction

This project explores the problem of **deepfake video detection** using machine learning. The core objective is to classify whether a video is real or deepfake based on extracted **frame-level**, **temporal**, and **combined features**.

Throughout the course of this project, I gained hands-on experience in:

- Designing neural network architectures using PyTorch
- Extracting and combining different modalities of features
- Performing comparative evaluation of multiple models
- Diagnosing performance bottlenecks and overfitting

2. Thought Process

To approach the deepfake detection problem, I began by conducting thorough research and analyzing existing deepfakes. One observation was that **audio discrepancies** often provide early indicators—AI-generated voices, while improving, still retain distinguishable artifacts that can be detected with proper analysis. Another line of thought involved generating a **deepfake version of the input video** under examination and then computing the **triplet loss** between the generated and original video embeddings. If the loss exceeds a certain margin, the video is likely authentic; if it's low, it suggests a potential deepfake due to high similarity in identity representation.

But the dataset provided wa without audio so i focused on visual artifacts then. (would need some more time to research and code properly the apt solution)

The task was to **develop a binary classifier** to distinguish between real and manipulated content in videos.

Literature Review:

- Prior works utilize CNNs on image frames and RNNs or 3D CNNs for temporal dynamics.
- Frame-based approaches can capture visual artifacts; temporal-based methods observe consistency between frames.

Design Decision:

- Three branches were created:
    - One for static **frame features**
    - One for **temporal embeddings**
    - One **combined model** using both
- A simple yet regularized **fully connected neural network (MLP)** was used for classification.

3. Blockers

Key challenges faced during the project included:

- Low performance of the combined features model, which was expected to outperform the individual ones
- Difficulty in hyperparameter tuning due to inconsistent performance and overfitting
- Libraries installation and GPU requirements were a hurdle as well as the huge dataset which took time to load.Videos afteral.
- Training time was initially high due to large feature dimensions

4. Approach

Preprocessing:

- Extracted embeddings from videos using pretrained models
- Normalized all feature sets
- Split datasets into training and validation sets (80-20 split)

Model Architecture:

- class DeepfakeDetectionModel(nn.Module):
-     def __init__(self, input_size, hidden_sizes=[512, 256, 128], dropout=0.3):
- Layers: Linear → ReLU → BatchNorm → Dropout → Linear
- Output: Final Linear layer to 2 classes (real/fake)

Training Setup:

- Optimizer: Adam
- Epochs: 30
- Batch size: 16
- Dropout: 0.3
- Evaluation Metrics: Accuracy, Precision, Recall, F1-Score

Comparative Study

	Model	Accuracy	Precision	Recall	F1-Score
0	Frame Features	0.500	0.500000	0.6	0.545455
1	Temporal Features	0.525	0.518519	0.7	0.595745
2	Combined Features	0.450	0.454545	0.5	0.476190

- The **combined model** underperformed, likely due to feature mismatch or overfitting
- **Temporal features** yielded the best results, highlighting the importance of motion consistency

Strengths:

- Fast training
- Interpretable structure
- Good recall with temporal model

Limitations:

- Shallow model architecture might miss deep patterns
- Feature fusion in the combined model needs better handling

6. Results

Evaluation Metrics: The **temporal model** consistently outperformed the others across all evaluation criteria.

7. Future Prospects

Potential directions to improve the current approach:

- Use **Transformer-based temporal encoders** such as TimeSformer or ViT
- Apply **late fusion techniques** instead of early concatenation for multimodal inputs
- Experiment with **contrastive learning** to improve representation quality
- Integrate **attention mechanisms** to highlight relevant spatial-temporal patterns
- Ensembling predictions across multiple models or video segments

8. Appendix

Model Architecture Summary

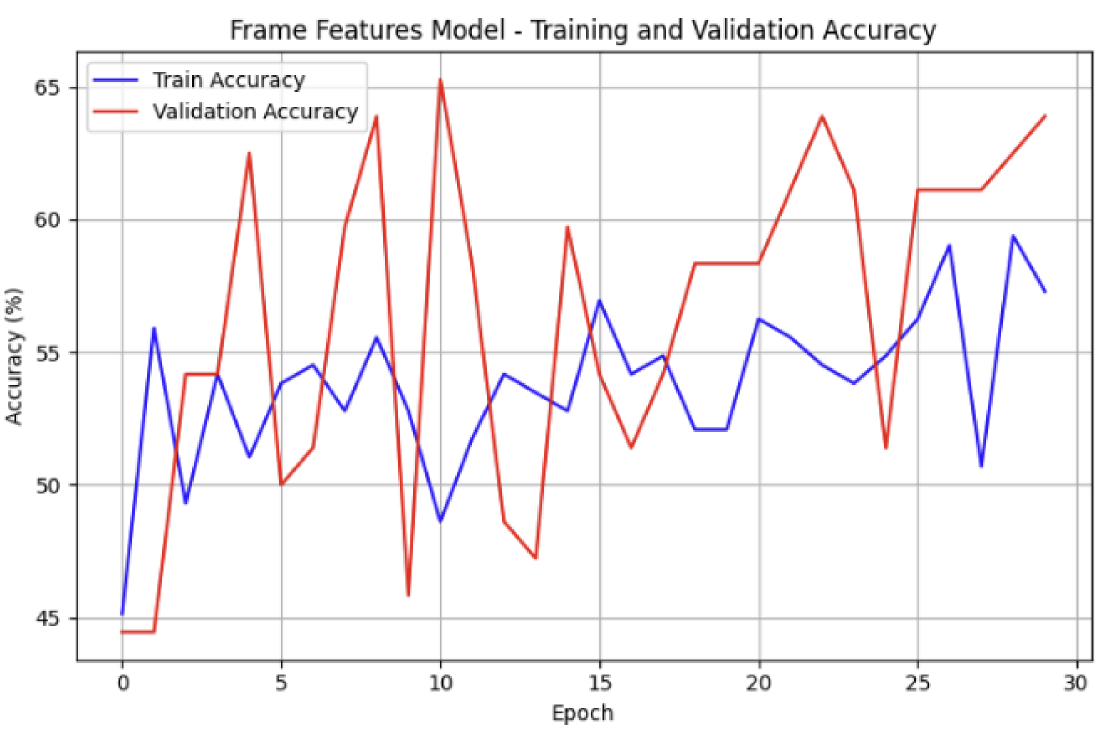
- Input → [Linear → ReLU → BatchNorm → Dropout] × N → Linear → Output

Screenshots:

```
def __init__(self, input_size, hidden_sizes=[512, 256, 128], dropout=0.3):
    super(DeepfakeDetectionModel, self).__init__()

    layers = []
    prev_size = input_size

    for hidden_size in hidden_sizes:
        layers.extend([
            nn.Linear(prev_size, hidden_size),
            nn.ReLU(),
            nn.BatchNorm1d(hidden_size),
            nn.Dropout(dropout)
        ])
        prev_size = hidden_size

    layers.append(nn.Linear(prev_size, 2))

    self.network = nn.Sequential(*layers)

def forward(self, x):
    return self.network(x)
```

- Evaluation metric results

	Model	Accuracy	Precision	Recall	F1-Score
0	Frame Features	0.500	0.500000	0.6	0.545455
1	Temporal Features	0.525	0.518519	0.7	0.595745
2	Combined Features	0.450	0.454545	0.5	0.476190



- Feature visualization or extraction diagrams (if applicable)