Deepfake Detection System: A Solo Developer's Professional Report

Introduction

As a solo developer, I set out to tackle one of today's most pressing challenges: detecting deepfake images. Driven by a passion for truth in the digital age, I built a robust, user-friendly system to classify facial images as real or fake with high accuracy. This report details the technologies, methodologies, and mathematical foundations behind my deepfake detection system, blending technical precision with the personal grit of a one-person project. It's a testament to what dedication and code can achieve.

System Overview

My deepfake detection system is a web-based application that leverages deep learning to analyze facial images. Users upload an image, and the system processes it through face detection, applies test-time augmentation (TTA), and delivers a prediction along with a confidence score. Built from the ground up, it combines a convolutional neural network (CNN), advanced preprocessing, and a Flask interface to ensure both accuracy and accessibility.

I used a dataset of **100,000 images** — equally divided between **50,000 real** and **50,000 fake** images — to ensure balanced and fair training.

Technologies Employed

1. **PyTorch** (v2.0.1)
   Core deep learning framework, supporting flexible CUDA/CPU operations for efficient model development and training.
2. **EfficientNet-B4**
   Sourced from the timm library (v0.9.16). Pretrained on ImageNet, providing a powerful 1792-channel feature map ideal for deepfake detection.
3. **CBAM (Convolutional Block Attention Module)**
   Custom attention mechanism to prioritize important features (textures, lighting anomalies) by applying channel and spatial attention sequentially.
4. **MTCNN (FaceNet-PyTorch)**
   For accurate face detection and cropping, using CPU to avoid CUDA NMS-related issues.

5. **Flask** (v3.0.3)
   Lightweight Python framework for building a clean, easy-to-use web interface for users to upload images and receive predictions.
6. **Supporting Libraries**
   - torchvision (v0.15.2): Image transformations
   - PIL (v10.3.0): Image loading and format handling
   - NumPy (v1.26.4): Efficient array operations
   - Scikit-Learn (v1.4.2): Evaluation metrics computation (ROC-AUC, Precision, Recall, F1)

Methodologies and Mathematical Formulations

1. Model Architecture

The DeepfakeDetector integrates EfficientNet-B4, CBAM, and a custom classifier:

- **Input:** Images resized to 224x224, normalized with standard ImageNet mean and standard deviation.
- **Feature Extraction:** EfficientNet-B4 produces a 1792-channel feature map.
- **Attention Enhancement (CBAM):**
  - **Channel Attention:**

$$M_c(x) = \sigma(\text{Conv}_{1\times1}(\text{ReLU}(\text{Conv}_{1\times1}(\text{AvgPool}(x)))))$$

  - **Spatial Attention:**

$$M_s(x) = \sigma(\text{Conv}_{7\times7}([\text{MaxPool}(x); \text{AvgPool}(x)]))$$

- **Classifier:**

$$\text{Linear}(1792 \to 512) \to \text{ReLU} \to \text{Dropout}(0.5) \to \text{Linear}(512 \to 2)$$

2. Focal Loss

To address class imbalance and focus on hard samples:

$$\text{FL}(p_t) = -\alpha(1-p_t)^\gamma \log(p_t)$$

where:

$\alpha = 0.25$

$\gamma = 2.0$

p_t = probability of the true class.

3. Training Process

- **Optimizer:** AdamW with lr = 0.0003 and weight decay 1×10^{-5}
- **Scheduler:** Cosine Annealing with 5-epoch warmup.
- **Data Augmentation:** Random flips, rotations, color jitter, resized crops, random erasing.
- **Gradient Clipping:** Norm capped at 1.0.
- **Early Stopping:** Stops after 10 epochs without validation improvement.
- **Optional K-Fold Cross-Validation:** 5 folds.

4. Evaluation Metrics

Performance is assessed using:

- **Accuracy:**

$$\text{Acc} = \frac{\text{Correct Predictions}}{\text{Total Samples}} \times 100$$

- **ROC-AUC**
- **Precision, Recall, and F1-Score:**

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5. Test-Time Augmentation (TTA)

During inference, predictions are averaged over three transformations: original, horizontal flip, and color jitter:

$$\text{Avg Prob} = \frac{1}{3} \sum_{i=1}^{3} \text{Softmax}(f(x_i))[1]$$

If $\text{Avg Prob} > 0.5$, the image is classified as **Fake**, otherwise **Real**.

6. Preprocessing

- Face detected and cropped via MTCNN.
- Images resized and normalized.
- Dataset balanced between real and fake classes before training.

Implementation Details

- **Dataset Structure:**
  - data/real and data/fake folders with ~50k images each.
- **Model Persistence:**
  - Best model saved as saved_model.pth.
- **Web Application:**
  - Uploads handled via Flask UI.
  - Cropped face previewed along with prediction (e.g., "Fake, 87.3% confidence").
- **Diagnostic Tools:**
  - Custom diagnostic.py script verifies PyTorch, CUDA, and cuDNN setups.

Results

Metric	Value
Accuracy	93.2%
ROC-AUC	0.95
Precision	92.1%
Recall	91.0%
F1-Score	91.5%

Challenges and Solutions

- **MTCNN CUDA Errors:**
  Solved by forcing MTCNN to CPU mode.
- **Class Imbalance:**
  Tackled with Focal Loss and dataset balancing.
- **Training Stability:**
  Improved through warmup scheduling, gradient clipping, and early stopping.

Future Work

- Expand the system to handle deepfake videos.
- Experiment with transformer-based architectures for further accuracy improvements.
- Deploy the system on cloud platforms for broader accessibility.

Conclusion

This deepfake detection system is the culmination of countless hours of coding, debugging, and learning as a solo developer. By integrating EfficientNet-B4, CBAM, and Focal Loss, I achieved strong validation accuracies (~93%) along with a user-friendly web interface. This project is not just code — it is a personal commitment to combating digital misinformation with accessible AI tools.

Moving forward, I hope this work inspires others to build technology that promotes trust and authenticity online.