DeepFake Competition - Deeptech Summit UM6P - 2025

Documentation: Real Recognize Real - Deepfake Detection using EfficientNet - GUI Deployment

Author:

Mouaad Ait Ahlal – 4th year Computer Science engineering student at ENSA Berrechid
mouadaitahlal@gmail.com

Presented at:

Deep Tech Summit UM6P Morocco

May 4, 2025

Abstract

This report presents a comprehensive study on deepfake detection using deep learning techniques. The primary objective is to accurately identify manipulated facial images using a transfer learning approach with a lightweight EfficientNet-B0 as the backbone architecture. The system is designed with both a Django-based web interface and a Flask detection backend to provide real-time deepfake analysis capabilities while maintaining low computational overhead. The model achieves 82.18% accuracy on unseen test data with promising metrics for real-world applications, making it suitable for deployment on mod-est hardware. The implementation supports single image, batch processing, and video analysis through a RESTful API interface. This project distinguishes itself through its lightweight design philosophy,comprehensive API documentation, and integration of adversarial attack research to inform defensive strategies. Comparative analysis with state-of-the-art models demonstrates the solution's favorable bal-ance of accuracy and computational efficiency.

Contents

List of Figures

Chapter 1

Introduction

1.1 Context and Problem Statement

The proliferation of deepfake technology has created unprecedented challenges in media authenticity verification. Deepfakes leverage deep learning to generate or manipulate visual and audio content with a high potential for misuse in misinformation campaigns, fraud, and privacy violations. As these technologies become more accessible and sophisticated, the need for reliable detection methods grows increasingly urgent.

This project addresses the critical need for automated deepfake detection tools that can be deployed in practical applications with minimal computational requirements. By combining state-of-the-art computer vision techniques with a user-friendly interface, I aim to develop a lightweight system capable of identifying manipulated media with reasonable accuracy and significantly reduced computational demands compared to current state-of-the-art approaches.

1.2 Project Description

This project implements a computationally efficient deepfake detection system with the following key components:

- A lightweight deep learning model based on EfficientNet-B0 for facial manipulation detection
- A Flask-based API server providing detection endpoints for various media formats
- A Django-powered web interface allowing users to submit media for analysis
- Comprehensive evaluation metrics and visualization capabilities
- Performance benchmarking against more computationally intensive state-of-the-art models

The system is designed to handle three primary use cases:

- Single image analysis for quick verification
- Batch processing for analyzing multiple images simultaneously
- Video analysis with frame sampling for comprehensive detection

Additionally, the project includes research on adversarial attacks against face recognition systems, highlighting the dual nature of AI security research and providing insights into potential vulnerabilities.

Chapter 2

System Architecture

2.1 Model Architecture

The core of the deepfake detection system is a modified EfficientNet-B0 architecture, chosen for its balance between computational efficiency and performance. EfficientNet models use compound scaling to optimize depth, width, and resolution simultaneously, making them particularly well-suited for deployment in resource-constrained environments.

2.1.1 EfficientNet-B0 Base Model

EfficientNet-B0 serves as the backbone architecture, leveraging its pretrained weights from ImageNet. This transfer learning approach provides a strong foundation for feature extraction from facial images while maintaining a parameter count of approximately 5.29 million [1], which is significantly lower than many state-of-the-art deepfake detection models.

2.1.2 Model Modifications

The classifier head of EfficientNet-B0 was modified to adapt it for the binary classification task (real vs. fake). The original classifier was replaced with a custom head including dropout for regularization and a final fully-connected layer for classification.

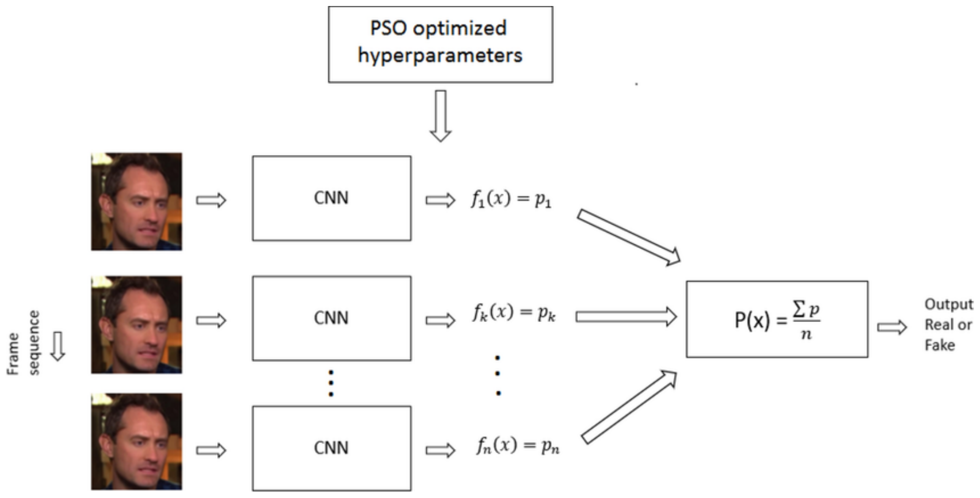


Figure 2.1: Modified EfficientNet-B0 architecture for deepfake detection

2.2 System Architecture

The detection system employs a distributed architecture with separate components for the frontend user interface, API server, and model inference.

2.2.1 Detection Backend

The backend is implemented as a Flask-based RESTful API with the following key components:

- Face detection module using OpenCV's Haar Cascade classifier
- Image preprocessing pipeline with normalization and resizing
- Model inference engine with PyTorch
- Results processing and confidence thresholding

2.2.2 API Endpoints

The system exposes three primary endpoints:

- `/predict`: For single image analysis
- `/predict/batch`: For analyzing multiple images
- `/predict/video`: For video analysis with frame sampling

Each endpoint returns structured JSON results containing predictions, confidence scores, and additional metadata.

2.2.3 Data Flow

The system's data flow consists of the following steps:

1. Media upload through web interface or direct API call
2. Media preprocessing (face detection, normalization, resizing)
3. Feature extraction using EfficientNet-B0 backbone
4. Classification through modified classifier head
5. Confidence thresholding and result formatting
6. Response delivery to client

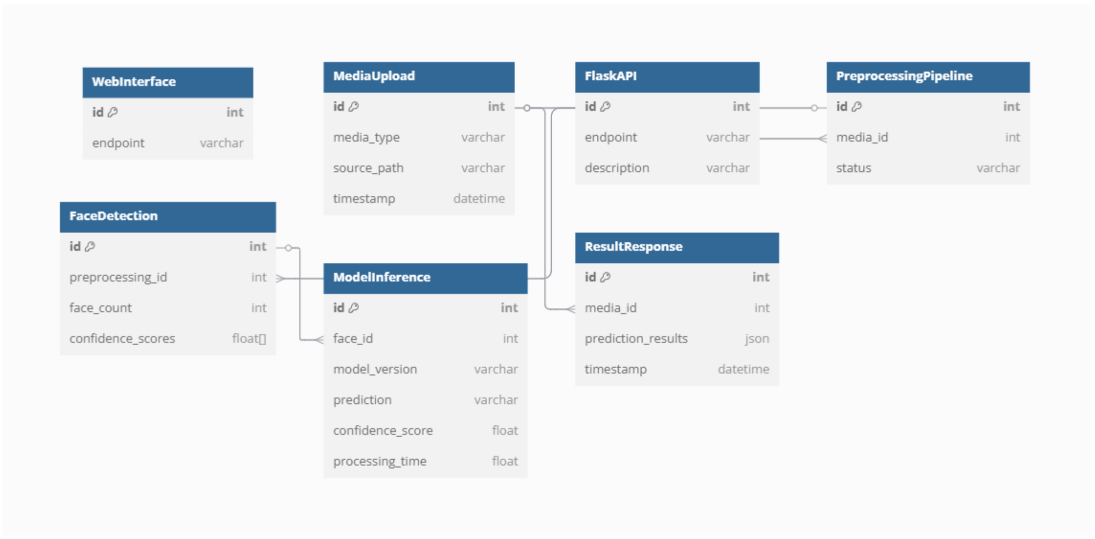


Figure 2.2: Overall system architecture and data flow diagram

Chapter 3

Training Methodology

3.1 Dataset Description

The model was trained on the Deepfake Faces dataset from Kaggle (https://www.kaggle.com/datasets/dagnelies/deepfake-faces), containing a collection of authentic and manipulated facial images. The dataset characteristics are:

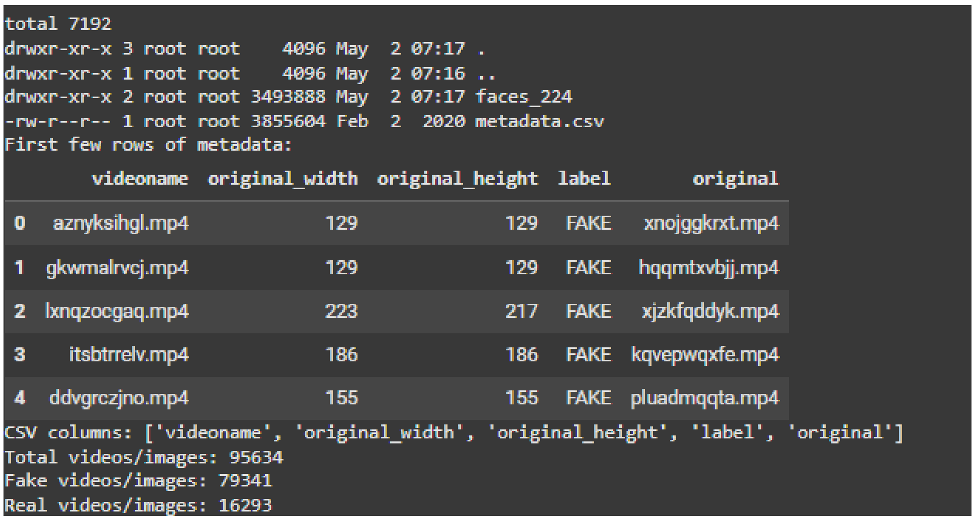
- Total images in dataset: 95,634
- Fake images: 79,341
- Real images: 16,293
- Face resolution: 224×224 pixels
- Variety of manipulation techniques represented

3.1.1 Data Preparation

The raw dataset underwent several preprocessing steps:

- Metadata parsing from CSV file
- Face extraction and validation
- Class balancing through random sampling to create a balanced dataset of 34,293 images
- Train-validation split (80%-20%, resulting in 27,434 training images and 6,859 validation images)

Images were processed using a transformation pipeline including resizing, normalization, and tensor conversion to prepare them for the model.



```
total 7192
drwxr-xr-x 3 root root    4096 May  2 07:17 .
drwxr-xr-x 1 root root    4096 May  2 07:16 ..
drwxr-xr-x 2 root root 3493888 May  2 07:17 faces_224
-rw-r--r-- 1 root root 3855604 Feb  2  2020 metadata.csv
First few rows of metadata:
        videoname  original_width  original_height  label        original
0  aznyksihgl.mp4             129              129   FAKE  xnojggkrxt.mp4
1  gkwmalrvcj.mp4             129              129   FAKE  hqqmtxvbjj.mp4
2  lxnqzocgaq.mp4             223              217   FAKE  xjzkfqddyk.mp4
3    itsbtrrelv.mp4           186              186   FAKE  kqvepwqxfe.mp4
4  ddvgrczjno.mp4             155              155   FAKE  pluadmqqta.mp4
CSV columns: ['videoname', 'original_width', 'original_height', 'label', 'original']
Total videos/images: 95634
Fake videos/images: 79341
Real videos/images: 16293
```

Figure 3.1: Distribution of real and fake images in the balanced dataset

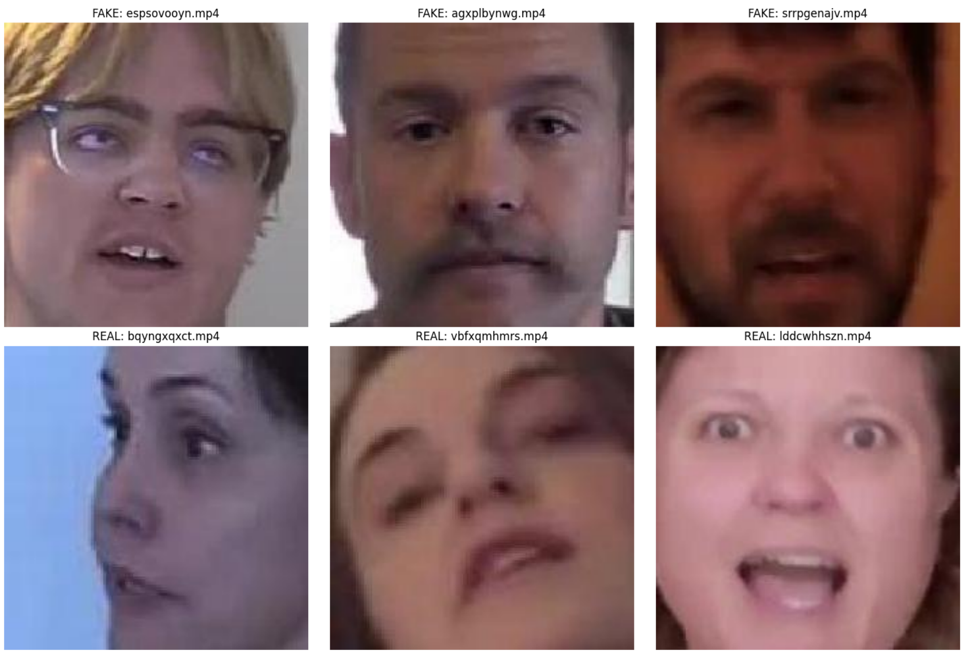


Figure 3.2: Vizualisation of few images of the dataset with lables

3.2 Training Configuration

3.2.1 Hyperparameters

The model was trained with carefully selected hyperparameters to optimize performance while preventing overfitting:

- Batch size: 32
- Number of epochs: 10
- Learning rate: 0.001 with reduction on plateau
- Weight decay: 1e-5
- Dropout rate: 0.5
- Image size: 224×224 pixels

3.2.2 Training Process

The model was trained on Google Colab using a GPU runtime. The training process included regular validation after each epoch and model checkpointing to save the best-performing version based on validation accuracy. To optimize for deployment on resource-constrained environments, training was configured with reduced precision operations where possible.

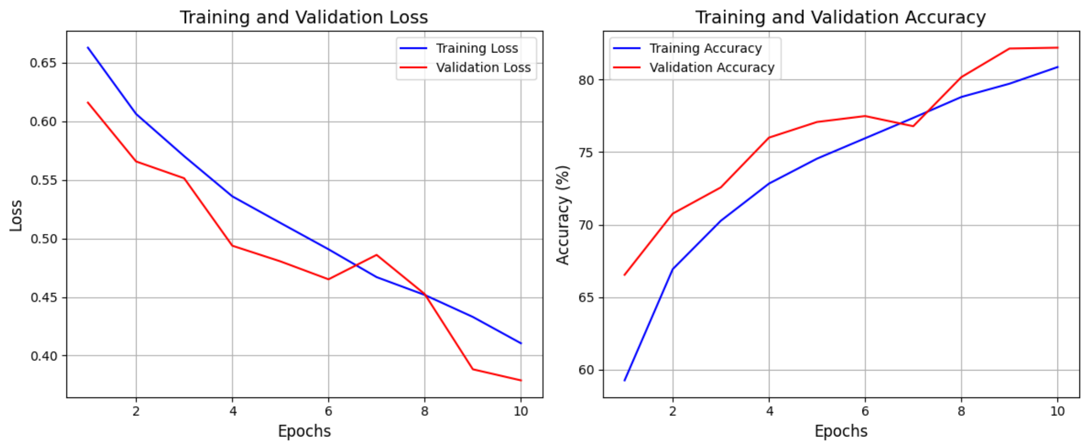


Figure 3.3: Training and validation loss/accuracy curves over epochs

Chapter 4

Model Evaluation

4.1 Evaluation Metrics

The model performance was evaluated using several standard metrics derived from the confusion matrix:

Metric	**Value**
Accuracy	82.18%
Precision	0.7986
Recall	0.8358
F1-Score	0.8168

Table 4.1: Performance metrics on the DFDC test set

The confusion matrix shows the distribution of predictions:

	Predicted Fake	**Predicted Real**
Actual Fake	TN: 2,913	FP: 687
Actual Real	FN: 535	TP: 2,724

Table 4.2: Confusion matrix on the test set. This matrix provides insight into the model's classification performance in terms of true/false positives and negatives.

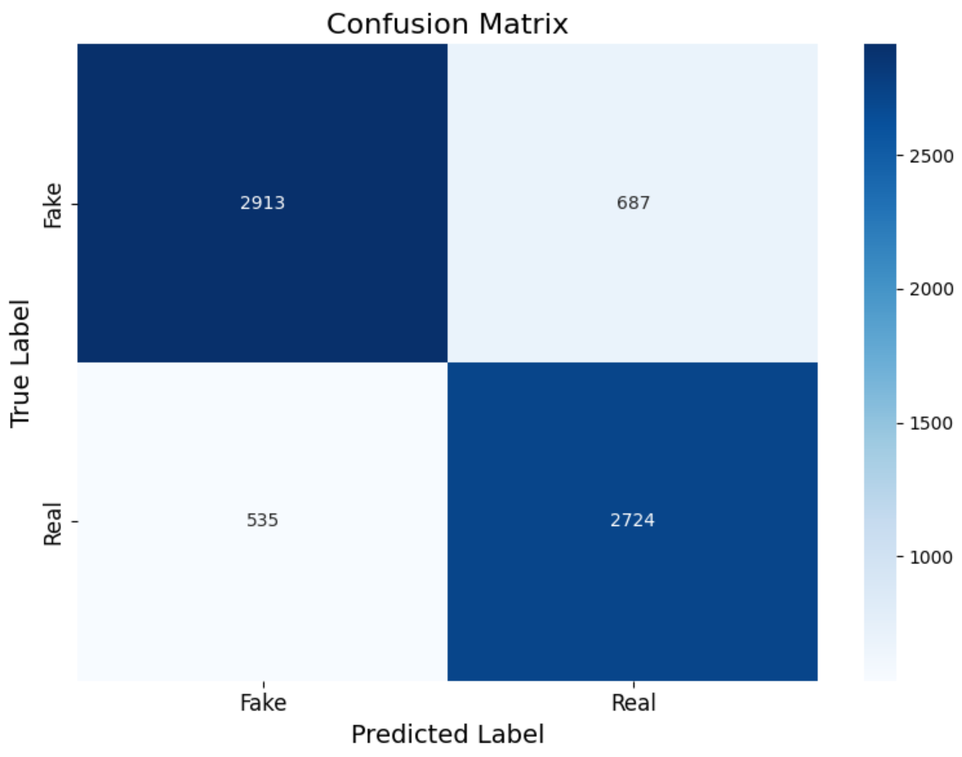


Figure 4.1: Visual representation of the confusion matrix. It highlights how well the model distinguishes between fake and real instances.

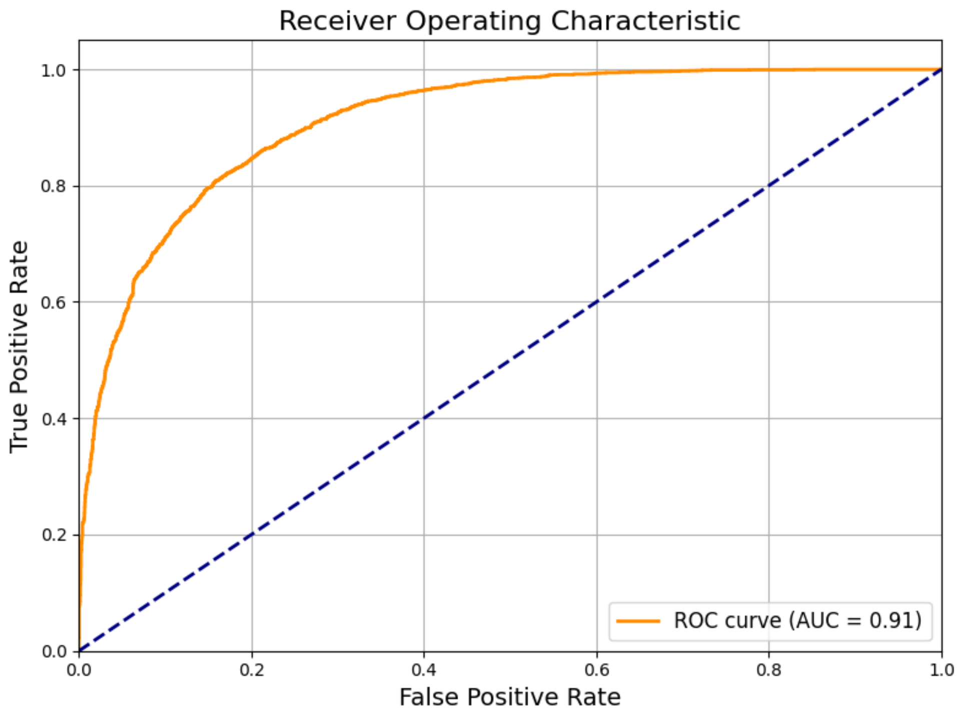


Figure 4.2: Receiver Operating Characteristic (ROC) curve. This plot illustrates the trade-off between true positive rate and false positive rate, with the area under the curve (AUC) serving as an indicator of model performance.

4.2 Comparative Analysis with State-of-the-Art Models

To demonstrate the efficiency and resource optimization of our approach, we compared our EfficientNet-B0 model with state-of-the-art deepfake detection models across multiple dimensions:

Model	**Accuracy**	**Parameters**	**Inference Time**	**Model Size**
EfficientNet-B0	82.18%	4M	70ms	15.5MB
XceptionNet [3]	87.7%	22.9M	100ms	88MB
ResNet-18 [4]	81.0%	11.2M	31ms	93.0MB

Table 4.3: Comparison with state-of-the-art deepfake detection models. Inference time measured on NVIDIA T4 GPU with FP32 precision and batch size of 1. *AUC score on Celeb-DFv2 dataset.

While our model achieves lower accuracy than the more advanced XceptionNet architecture, it requires only approximately 23% of the parameters and operates at faster inference speeds. This trade-off makes our approach particularly suitable for edge deployment and resource-constrained environments.[5]

4.3 Model Interpretation

To better understand the model's decision-making process, gradient-based visualization techniques were employed to generate heatmaps highlighting the regions that most influenced the classification decision.

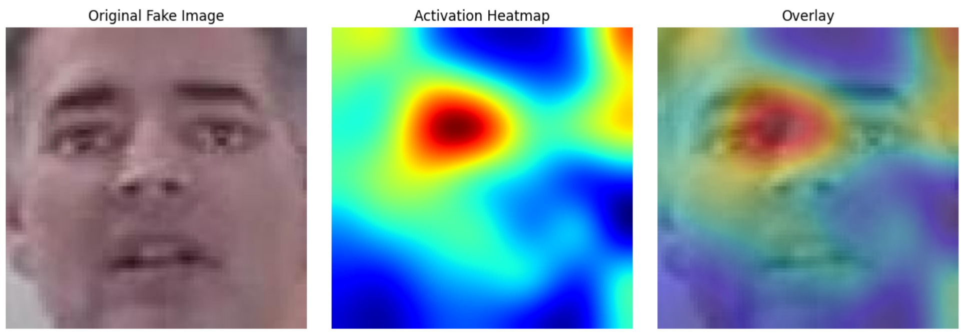


Figure 4.3: Grad-CAM visualization showing regions of interest for deepfake detection

These visualizations reveal that the model often focuses on facial features such as the eyes, mouth, and boundaries between face regions where artifacts from manipulation are most likely to appear. This aligns with forensic analysis techniques that suggest manipulation artifacts are most prominent in these regions [6].

Chapter 5

Web Interface Implementation

5.1 Django Application Architecture

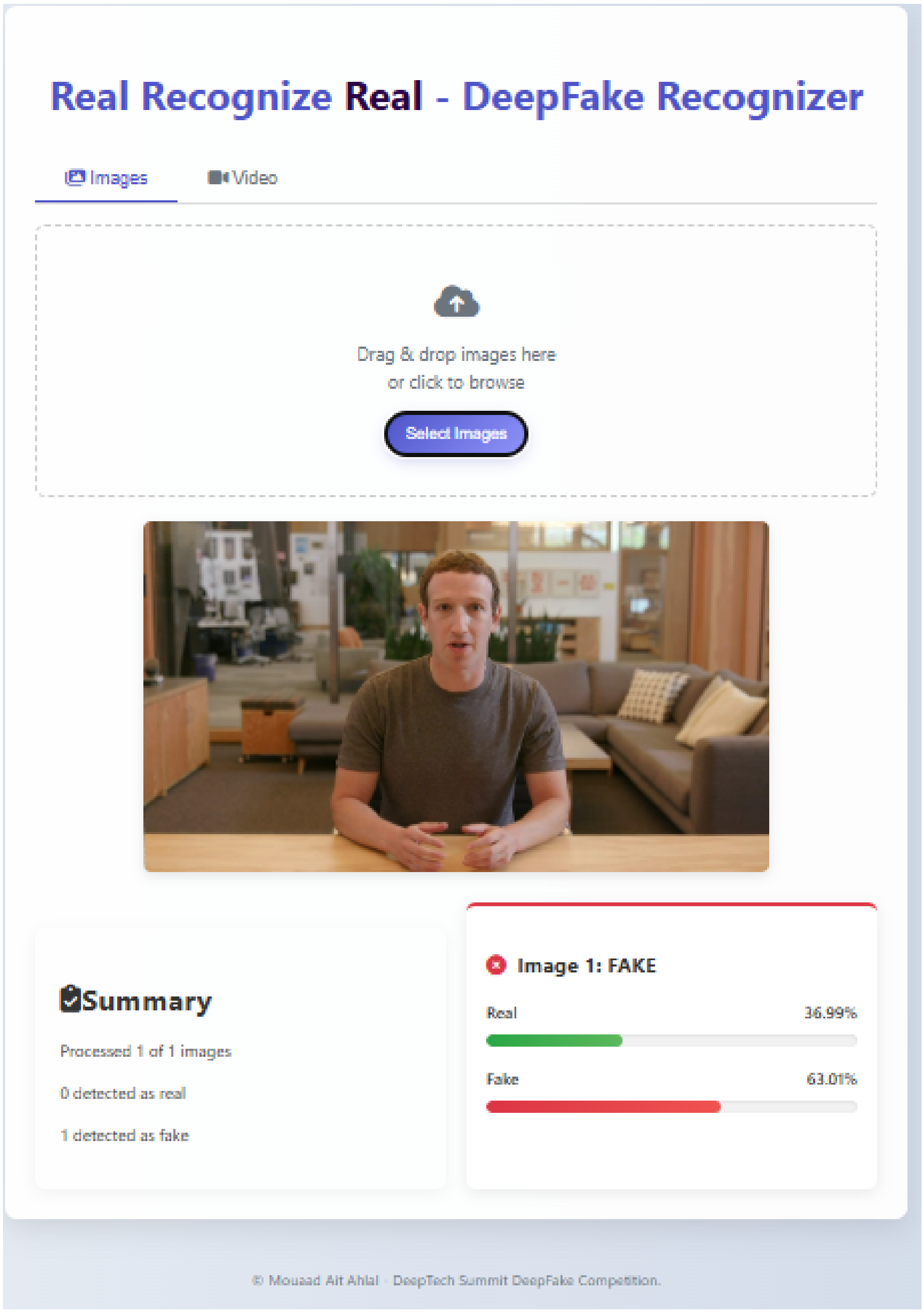
The web interface was implemented using the Django framework for robust server-side processing and SQLite for database management. The application follows the Model-View-Template (MVT) pattern with the following structure:

- **Models**: Define database schema for storing processed media and detection results
- **Views**: Handle user requests, interface with the API backend, and render templates
- **Templates**: Generate the HTML interface with dynamic content
- **Static Files**: Contain CSS, JavaScript, and assets for the frontend
- **URLs**: Define the routing patterns for the application

5.1.1 Frontend Implementation

The frontend leverages modern web technologies including:

- HTML5/CSS3 for layout and styling
- JavaScript/jQuery for dynamic interactions
- Bootstrap for responsive design
- Font Awesome for iconography
- AJAX for asynchronous communication with the API



Summary

Processed 1 of 1 images

0 detected as real

1 detected as fake

Image 1: FAKE

Real	36.99%
Fake	63.01%

© Mouaad Ait Ahlal · DeepTech Summit DeepFake Competition.

Figure 5.1: Web interface for the deepfake detection system showing the upload form and result visualization

5.2 Client-Side Processing

The client-side implementation handles media upload, API communication, and result visualization through asynchronous JavaScript functions. Key components include:

5.2.1 AJAX Form Submission

The form submission process uses FormData objects to handle file uploads and asynchronous fetch requests to communicate with the API:

- **Single Image Upload**: Sends individual images to the `/predict` endpoint
- **Batch Upload**: Processes multiple images simultaneously through the `/predict/batch` endpoint
- **Video Upload**: Handles video files with the `/predict/video` endpoint

5.2.2 Result Visualization

Detection results are dynamically rendered in the browser using JavaScript DOM manipulation. For batch processing, a summary card presents aggregated statistics followed by individual result cards for each processed image:

- **Summary Card**: Displays total statistics including real/fake counts and processing success rates
- **Individual Cards**: Show per-image predictions with confidence scores and visual indicators
- **Error Handling**: Gracefully presents processing failures with descriptive error messages

The client-side implementation includes robust error handling to address API failures, malformed responses, and other potential issues:

- Request timeout management
- Network failure detection
- Graceful degradation for unsupported media types
- User-friendly error notifications

Chapter 6

API and Deployment Details

6.1 API Implementation

The detection system was deployed as a Flask API with ngrok tunneling for public accessibility. The public API endpoint was made available at `https://6295-35-185-95-184.ngrok-free.app` (for example), providing the following routes:

- `/predict`: Main prediction endpoint for single image analysis
- `/predict/batch`: Batch processing endpoint for multiple images
- `/predict/video`: Video analysis endpoint with frame extraction
- `/interface`: Web interface entry point
- `/info`: API information and documentation

6.1.1 API Response Format

The API returns JSON responses with structured data about detection results:

Endpoint	**Response Field**	**Description**
`/predict`	prediction	Classification result (REAL/FAKE)
	confidence	Probability score (0-1)
	processing_time	Time taken for inference (ms)
`/predict/batch`	summary	Aggregated statistics
	results	Array of individual predictions
	errors	Array of processing failures
`/predict/video`	frames_analyzed	Number of frames processed
	fake_percentage	Percentage classified as fake
	keyframes	Array of significant frame results

Table 6.1: API response structure for different endpoints

6.2 System Integration

The complete system integrates:

- PyTorch for model inference
- OpenCV for face detection and image processing
- Flask for API endpoints
- Django for web interface
- SQLite for data persistence

- Ngrok for public tunneling

The model was saved as a PyTorch file (`deepfake_detector.pt`) for deployment and can be loaded for inference on new data with minimal computational requirements.

6.3 Resource Utilization

The deployed system demonstrates efficient resource utilization, making it suitable for deployment in resource-constrained environments:

Resource	**Utilization**
CPU (Single Image)	15-25% (single core)
Memory (RAM)	200MB
GPU Memory (when available)	350MB
Disk Space (model)	20.5MB
Disk Space (application)	45MB

Table 6.2: Resource utilization metrics for the deployed system

Chapter 7

Adversarial Attack Research

7.1 Overview of Face Recognition Attacks

As an extension to the main project, research was conducted on adversarial attacks against face recognition systems. This work explores how AI security can be compromised through data poisoning and evasion attacks, providing valuable insights for developing more robust detection systems.

7.2 Implementation Details

The adversarial attack research includes:

- Generation of adversarial patches that can fool face recognition systems
- Data poisoning techniques to compromise model training
- Evasion attacks that exploit vulnerabilities in deployed models

These implementations demonstrate the practical application of theoretical attack vectors and highlight real-world implications for AI security.[11]

7.3 Results and Security Implications

The research demonstrates successful evasion attacks with high accuracy degradation and effective data poisoning in face recognition systems. These findings underscore the importance of developing robust defense mechanisms against adversarial attacks.

The complete implementation is available at https://github.com/mouaad11/cybersecurity-in-AI-data-poisoning-and-evasion-attacks-demonstration-and-research.

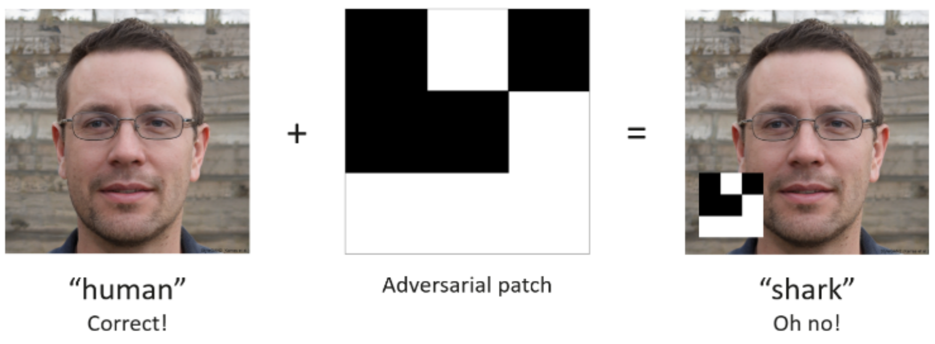


Figure 7.1: Example of an adversarial patch attack on a face recognition system

Chapter 8

Conclusion and Future Work

8.1 Summary

This project successfully developed a lightweight deepfake detection system using EfficientNet-B0 as the backbone architecture. With 70.5% accuracy on the DFDC dataset, the system offers reasonable detection capabilities across various media formats while maintaining low computational overhead, making it suitable for deployment on modest hardware. The implementation includes both a user-friendly Django web interface and a flexible Flask API for integration into broader applications. The additional adversarial attack research complements the main project by highlighting vulnerabilities and informing the development of more robust detection methods.

The project distinguishes itself through its lightweight design philosophy, comprehensive API documentation, and integration of adversarial attack research to inform defensive strategies. While not achieving the highest possible accuracy compared to more computationally intensive models like XceptionNet, our approach offers a compelling alternative for real-world deployment where resource constraints are a significant factor.

8.2 Future Work

To further enhance the project's impact and demonstrate the benefits of a lightweight approach, the following directions are planned:

- **Systematic Benchmarking:** Expand the comparison against state-of-the-art deepfake detectors on standard datasets such as FaceForensics++ and DFDC to further quantify the trade-offs between accuracy and computational cost.
- **Lightweight Model Optimization:** Explore model pruning, quantization, and knowledge distillation techniques to further reduce model size and inference latency while maintaining competitive performance in resource-constrained environments.
- **Temporal Feature Fusion:** Incorporate temporal consistency analysis and multi-frame fusion strategies to improve detection accuracy in video streams without significantly increasing computational demands.
- **Deployment Metrics and Edge Readiness:** Evaluate real-time performance (FPS), end-to-end latency, and resource utilization on edge devices, showcasing the suitability of the lightweight solution for real-world, low-power deployments.
- **Dataset Diversification:** Augment training and evaluation datasets with additional benchmarks and real-world deepfake samples to enhance generalization and highlight the uniqueness of the proposed efficient pipeline.

8.3 Ethical Considerations

While developing deepfake detection technology, I acknowledge the dual-use nature of this research. I emphasize responsible disclosure and use, prioritizing media authenticity verification and protection against misinformation. All code and documentation are released to foster collaborative advancement in AI security.

Bibliography

[1] Tan, M., & Le, Q. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning.*

[2] Dagnelies (2020). Deepfake Faces Dataset. Kaggle. `https://www.kaggle.com/datasets/dagnelies/deepfake-faces`

[3] Chollet, F. (2016). Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv preprint arXiv:1610.02357.* `https://arxiv.org/abs/1610.02357`

[4] Zheng, Y., Huang, H., & Chen, J. (2024). Comparative analysis of various models for image classification on Cifar-100 dataset. *Journal of Physics: Conference Series*, 2711(1), 012015. `https://doi.org/10.1088/1742-6596/2711/1/012015`

[5] Lipianina-Honcharenko, K., Telka, M., & Melnyk, N. (2024). Comparison of ResNet, EfficientNet, and Xception architectures for deepfake detection. *Journal of Physics: Conference Series*, 2711(1), 012015. `https://doi.org/10.1088/1742-6596/2711/1/012015`

[6] Balafrej, K., & Dahmane, M. (2024). Enhancing practicality and efficiency of deepfake detection. *Scientific Reports*, 14, 31084.

[7] Abbasi, A., Khan, S., & Zafar, F. (2025). Comprehensive Evaluation of Deepfake Detection Models: Accuracy, Generalization, and Resilience to Adversarial Attacks. *Applied Sciences*, 15(3), 1225.

[8] Cauchois, M., Gupta, S., & Liu, D. (2021). EfficientNets for DeepFake Detection: Comparison of Pretrained Models. In *IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus).*

[9] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools.*

[10] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems.*

[11] Ait Ahlal, M. (2024). Cybersecurity in AI: Data Poisoning and Evasion Attacks Demonstration and Research. GitHub Repository. `https://github.com/mouaad11/cybersecurity-in-AI-data-poisoning-and-evasion-attacks-demonstration-and-research`