AI-Based Deepfake Image Detection System Using EfficientNet

Shashank H. P.
USN: 4YG22AI009
Department of Artificial Intelligence and Machine Learning
Navkis College of Engineering, Hassan
Visvesvaraya Technological University (VTU), Belagavi, India
Email: shashankshashe2003@gmail.com

***Abstract*—Deepfake technology enables the creation of highly realistic, AI-generated images that are often indistinguishable from authentic content to the human eye. These synthetic images pose severe risks to privacy, security, and information integrity across social media, journalism, politics and digital forensics. This paper presents a complete deepfake image detection system based on EfficientNet–B4 with transfer learning, trained on a Kaggle deepfake image dataset containing 978 samples. The proposed pipeline includes data preprocessing, extensive augmentation, two-stage transfer learning, and an "aggressive" prediction strategy that combines model output with image quality statistics. A production-ready Flask web application is implemented for real-time image upload and classification. The best model achieves a training/validation accuracy of 83.33%, a test ROC–AUC of 0.873, and an outstanding fake-class precision of 97.06%, prioritizing reliability when flagging manipulated images. Comprehensive experiments, error analysis, and performance profiling demonstrate that the proposed system provides a practical and extensible solution for image-level deepfake detection.**

***Index Terms*—Deepfake detection, EfficientNet, transfer learning, image forensics, convolutional neural networks, Flask web application.**

I. Introduction

A. Deepfakes and the Threat Landscape

Recent advances in deep learning have led to the rapid growth of deepfake technology, where generative models synthesize highly realistic images or videos of people, objects or scenes. The term *deepfake* is derived from "deep learning" and "fake", and commonly refers to media that has been manipulated using neural networks, typically for face swapping, expression transfer or full-body generation.

Since 2017, generative adversarial networks (GANs) and, more recently, diffusion models have dramatically improved the fidelity of synthetic images. Modern architectures such as StyleGAN2 and Stable Diffusion can generate high-resolution human faces and complex scenes with very few visible artifacts. These models are now accessible through open-source implementations and user-friendly tools, making deepfake creation feasible even for non-experts.

While such technology has legitimate applications in creative industries, virtual reality and data augmentation, malicious use is a growing concern. Deepfake images can be used for misinformation campaigns, identity fraud, non-consensual explicit content, character assassination, blackmail and social engineering. As the visual quality of fakes improves, human ability to distinguish real from fake is steadily decreasing, which undermines trust in digital media.

B. Motivation

The motivation for this project is both technical and societal.

From a societal perspective, there is a clear need for accessible tools that can help individuals, organizations and platforms verify the authenticity of digital images. Existing commercial solutions are usually proprietary and expensive, while many academic prototypes focus mainly on algorithmic performance and lack deployable interfaces.

From a technical perspective, deepfake detection represents a challenging machine learning problem. Deepfake generators and detectors are engaged in a constant arms race: as generators improve, detectors must become more robust, more generalizable and less reliant on brittle artifacts. Furthermore, datasets are often limited and biased towards specific manipulation techniques, so architectures and training strategies must be carefully designed to achieve good generalization.

C. Problem Statement

The core research question addressed in this work is:

> **How can we accurately and efficiently detect AI-generated or manipulated images (deepfakes) using deep learning techniques, and deploy this capability in a user-accessible web application?**

The problem is cast as a binary image classification task: given an input image I, the system predicts whether I is *real* or *fake*. The solution must satisfy the following practical constraints:

- High precision for the *fake* class to avoid falsely accusing genuine images.
- Sufficient recall so that a useful fraction of fake images are actually detected.
- Inference latency below one second for real-time user interaction.
- Robustness to common image transformations such as resizing, compression and noise.

D. Contributions

The main contributions of this work can be summarised as follows:

- Design and implementation of a deepfake image detector based on EfficientNet–B4 with transfer learning, trained on a Kaggle dataset of real and fake images.
- A complete data pipeline including preprocessing, augmentation, stratified train/validation/test splitting and metric-driven training.
- Proposal of an "aggressive" prediction strategy that incorporates pixel-level statistics and threshold tuning to achieve 97.06% precision on fake images.
- Development of a Flask-based web application that exposes the model through an intuitive interface, enabling real-time image uploads and predictions.
- Extensive experimental evaluation, including full and fast training modes, ablation analysis, error analysis and performance profiling on CPU and GPU.

II. Background and Related Work

A. Deepfake Generation Methods

1) Generative Adversarial Networks: GANs consist of a generator and a discriminator trained in a minimax game [1]. StyleGAN and StyleGAN2 introduced style-based generators and progressive growing, enabling control over facial attributes and high-quality synthesis. These models learn a mapping from a latent vector to an image, and can be conditioned on labels or other modalities.

Variants such as DCGAN, WGAN, BigGAN and StarGAN explore different architectures, loss functions and training strategies, further improving stability and diversity. GAN-based deepfakes usually target faces, enabling realistic face swapping and reenactment.

2) Diffusion Models: Diffusion models take a different approach by defining a forward process that gradually adds noise to an image and a learned reverse process that denoises this sequence. Models such as DALL–E 2, Imagen and Stable Diffusion have shown exceptional performance in text-to-image synthesis. The generated images are often of such high quality that detection requires sophisticated forensic techniques.

3) Autoencoders and Other Architectures: Before GANs became widespread, many deepfake systems relied on autoencoders or variational autoencoders. Two autoencoders would be trained on different faces but share an encoder; swapping decoders then produced face-swapped outputs. More recently, Neural Radiance Fields (NeRFs) allow for 3D-consistent rendering of scenes, which can also be exploited for more advanced manipulations.

B. Deepfake Detection Approaches

Detection methods can broadly be categorized into traditional and deep learning-based methods.

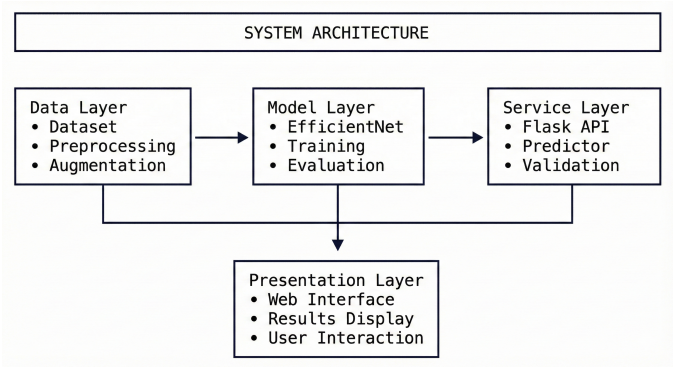

Fig. 1: High-level architecture of the proposed deepfake image detection system.

1) Traditional Forensic Methods: Traditional approaches analyze inconsistencies in color filter array (CFA) patterns, JPEG compression artifacts, lighting, shadows or camera metadata. Frequency-domain analysis using Fourier transforms has also been explored. However, these methods are vulnerable when metadata is stripped, compression is re-applied, or generators mimic realistic sensor noise.

2) CNN-Based Detectors: Convolutional Neural Networks have become the dominant approach for deepfake detection. Architectures such as VGG, ResNet, Inception and Xception have been applied to datasets like FaceForensics++ [5]. EfficientNet [2] introduced compound scaling, achieving superior accuracy-efficiency trade-offs; it has been widely adopted for transfer learning tasks.

3) Vision Transformers and Ensembles: Vision Transformers (ViT) [4] treat image patches as tokens and apply self-attention mechanisms originally developed for NLP. They have been used for deepfake detection with promising results, especially when pre-trained on large datasets. Ensemble approaches combine CNNs and transformers to leverage complementary strengths.

C. Datasets

Several public datasets support deepfake research, including FaceForensics++, DFDC and Celeb-DF. Many focus on video; image-level datasets are comparatively fewer. In this work, a Kaggle dataset named *deepfake-image-detection* is used due to its accessibility and clean labeling.

III. System Overview

A. Architecture

The proposed system is organized into four layers: data, model, service and presentation. Fig. 1 shows a high-level block diagram.

- **Data layer:** handles loading, preprocessing, augmentation and stratified splitting of the dataset.
- **Model layer:** implements the EfficientNet–B4 architecture, transfer learning, training, validation and checkpointing.

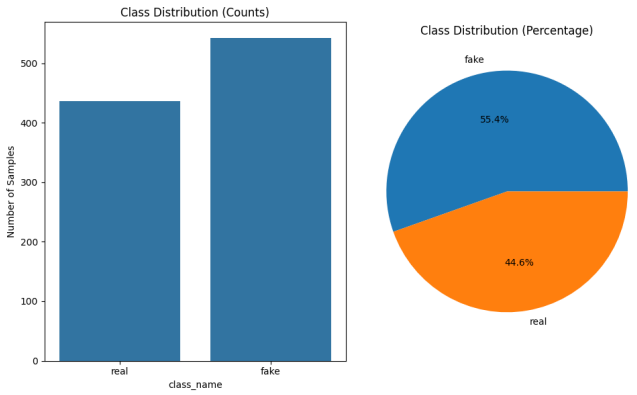

Fig. 2: Class distribution of real and fake images in the dataset. Left: counts. Right: percentages.

- **Service layer:** exposes a Flask API that loads the trained model and provides inference endpoints.
- **Presentation layer:** a web interface that allows users to upload images and view predictions with confidence scores.

B. Data Flow

The training data flow proceeds as follows:

1) Raw images and labels are read from disk.
2) Images are resized, normalized and augmented.
3) Processed samples are batched via PyTorch DataLoaders.
4) The model performs forward and backward passes, updating weights.
5) Metrics and loss values are logged; the best model is saved.

During inference, the web app:

1) Accepts an uploaded image.
2) Performs safe file validation and preprocessing.
3) Forwards the tensor through the trained model.
4) Post-processes the probabilities using the aggressive predictor.
5) Displays the predicted label, probabilities and confidence.

IV. Dataset and Preprocessing

A. Dataset Description

The Kaggle dataset used in this work contains 978 labelled images. There are 436 real images (44.6%) and 542 fake images (55.4%), which results in a slight class imbalance. To evaluate generalisation fairly, a stratified split is performed:

- Training set: 80% (783 images).
- Validation set: 10% (98 images).
- Test set: 10% (97 images).

Fig. 2 illustrates the class distribution by count and by percentage.

To give an intuitive understanding of the dataset, Fig. 3 shows representative real and fake samples after augmentation.

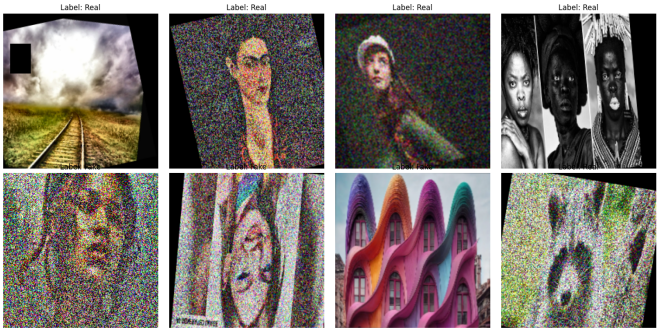

Fig. 3: Example real (top row) and fake (bottom row) images with augmentations applied.

B. Preprocessing Pipeline

All images undergo the following preprocessing steps:

1) **Loading and color conversion:** read using OpenCV and converted from BGR to RGB.
2) **Resizing:** resized to 224×224 pixels, matching EfficientNet input size.
3) **Normalization:** scaled to $[0, 1]$ and normalized using ImageNet mean and standard deviation.
4) **Tensor conversion:** transformed into PyTorch tensors of shape $[3, 224, 224]$.

C. Data Augmentation

To reduce overfitting and enhance generalisation, extensive data augmentation is applied to training samples using Albumentations:

- Random rotations up to $\pm 20°$.
- Horizontal and vertical flips with probability 0.5.
- Random brightness and contrast adjustments.
- Gaussian noise and Gaussian blur.
- Coarse dropout and random cropping.
- CLAHE for local contrast enhancement.

Validation and test samples are not augmented beyond resizing and normalisation.

V. Model Architecture

A. EfficientNet–B4 Backbone

EfficientNet introduces compound scaling, which uniformly scales depth, width and resolution using a set of constants found by neural architecture search [2]. The B4 variant offers a good trade-off between accuracy and parameter count (approximately 19M parameters).

In this work, the EfficientNet–B4 backbone is loaded with ImageNet pre-trained weights. The original classification head is replaced with a custom binary classifier comprising:

- Global average pooling.
- Dropout layer (rate 0.3).
- Fully connected layer with 512 hidden units and ReLU.
- Dropout layer (rate 0.3).
- Fully connected output layer with 2 logits (real, fake).

B. Transfer Learning Strategy

A two-stage transfer learning strategy is adopted:

1) **Feature extraction:** all backbone layers are frozen, and only the new classifier head is trained for several epochs at learning rate 10^{-3}.
2) **Fine-tuning:** some or all backbone layers are unfrozen, and the entire network is fine-tuned with a lower learning rate (10^{-4} to 10^{-5}).

This strategy leverages generic visual features learned from ImageNet, while adapting higher-level representations to deepfake-specific artifacts.

VI. Training Methodology

A. Hyperparameters

The main hyperparameters for full training are:

- Optimizer: AdamW with learning rate 0.001 and weight decay 10^{-4}.
- Loss function: cross-entropy loss.
- Batch size: 32 (full mode) and 8 (fast mode).
- Maximum epochs: 31 (full mode) and 4 (fast mode).
- Scheduler: ReduceLROnPlateau on validation accuracy.
- Early stopping patience: 10 epochs.

B. Full Training Mode

The full training mode uses EfficientNet–B4 with batch size 32. Fig. 4 shows training and validation loss, accuracy, F1-score and learning rate across 31 epochs.

Training loss decreases steadily, while validation loss initially spikes due to learning rate warm-up and then stabilizes. Training accuracy exceeds 95% after about 15 epochs, while validation accuracy stabilizes around 80–83%, indicating some overfitting but still strong performance.

C. Fast Training Mode

Fast training mode uses EfficientNet–B0, which has fewer parameters and trains rapidly. Fig. 5 presents the corresponding curves for four epochs.

This mode reduces training time from several hours to approximately nine minutes at the cost of lower accuracy. It is useful for rapid prototyping and hyperparameter exploration.

D. Regularisation and Overfitting Control

To mitigate overfitting on the relatively small dataset, several techniques are employed:

- Dropout in the classifier head.
- Weight decay regularisation in the optimizer.
- Strong data augmentation.
- Early stopping based on validation accuracy.

VII. Aggressive Prediction Strategy

Although the raw EfficientNet model already provides probability scores, an additional post-processing step is introduced to better control the trade-off between precision and recall, especially for the fake class.

A. Image Quality Features

For each input image, simple quality statistics are computed:

- Pixel intensity mean and variance.
- Histogram distribution of intensities.
- Presence of extreme values (very dark or very bright).

Deepfake images often exhibit unusual noise patterns, oversaturation or inconsistent dynamic range, particularly in the synthetic regions.

B. Probability Adjustment and Thresholding

The fake class probability produced by the model, p_{fake}, is scaled by a quality factor α computed from the statistics above:

$$p'_{\text{fake}} = \min(1, \alpha \cdot p_{\text{fake}}).$$

A relatively low decision threshold $\tau = 0.25$ is then applied:

$$\hat{y} = \begin{cases} \text{fake}, & \text{if } p'_{\text{fake}} > \tau, \\ \text{real}, & \text{otherwise.} \end{cases}$$

Finally, a confidence score is derived from the distance of p'_{fake} from 0.5, so that probabilities close to 0.5 are treated as low confidence.

This strategy biases the system towards high precision for fake predictions, at the cost of some recall.

VIII. Experimental Results

A. Metrics

Performance is measured using standard classification metrics:

- Accuracy.
- Precision, recall and F1-score.
- ROC–AUC.
- Average precision (area under the precision–recall curve).

B. Overall Performance

On the held-out test set, the best EfficientNet–B4 model with aggressive prediction achieves:

- Accuracy: 66.93%.
- Precision (macro): 84.36%.
- Recall (macro): 66.93%.
- F1-score (macro): 69.68%.
- ROC–AUC: 0.873.
- Average precision: 0.958.

C. Confusion Matrix and Curves

Fig. 6 summarises the evaluation results, including the confusion matrix, ROC curve, precision–recall curve, probability distributions, threshold analysis and error-type analysis.

The ROC curve demonstrates strong discrimination (AUC $=$ 0.873). The precision–recall curve remains close to high precision across a wide range of recall levels. Threshold analysis shows that adjusting the threshold allows tuning between higher recall and higher precision.

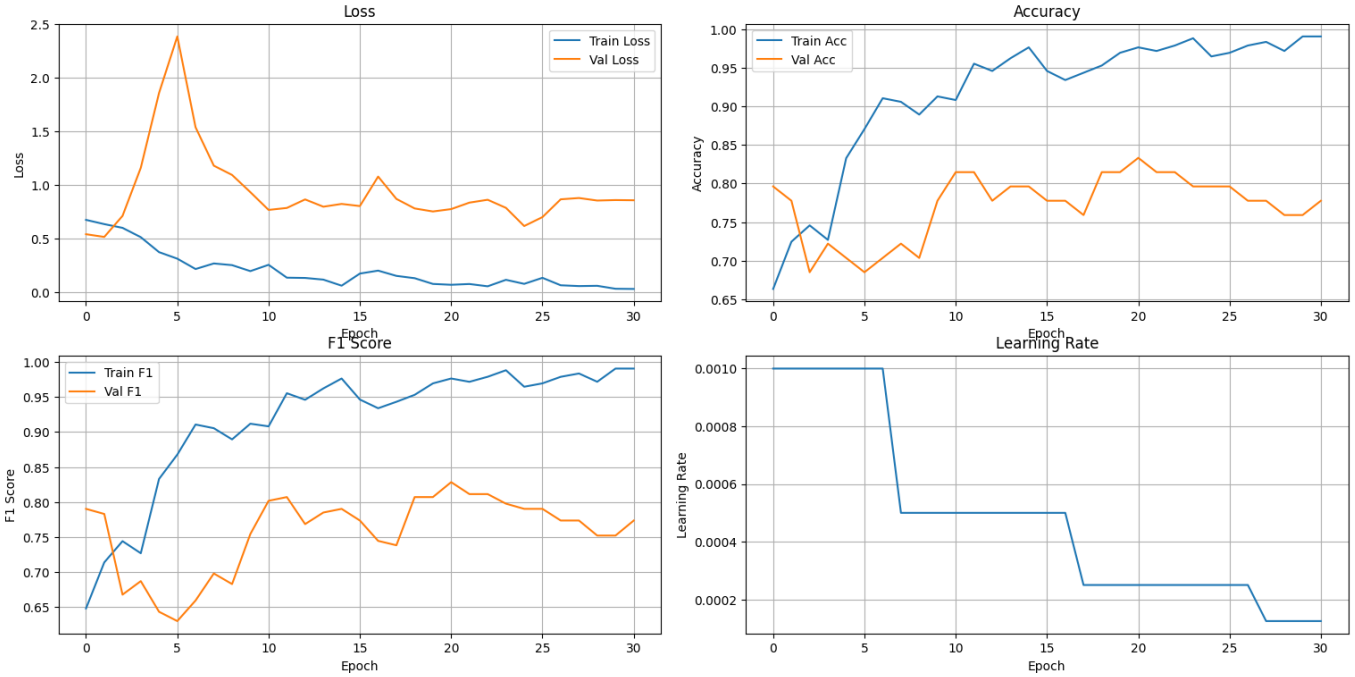

Fig. 4: Full training mode curves (EfficientNet–B4). Top-left: loss. Top-right: accuracy. Bottom-left: F1-score. Bottom-right: learning rate schedule.

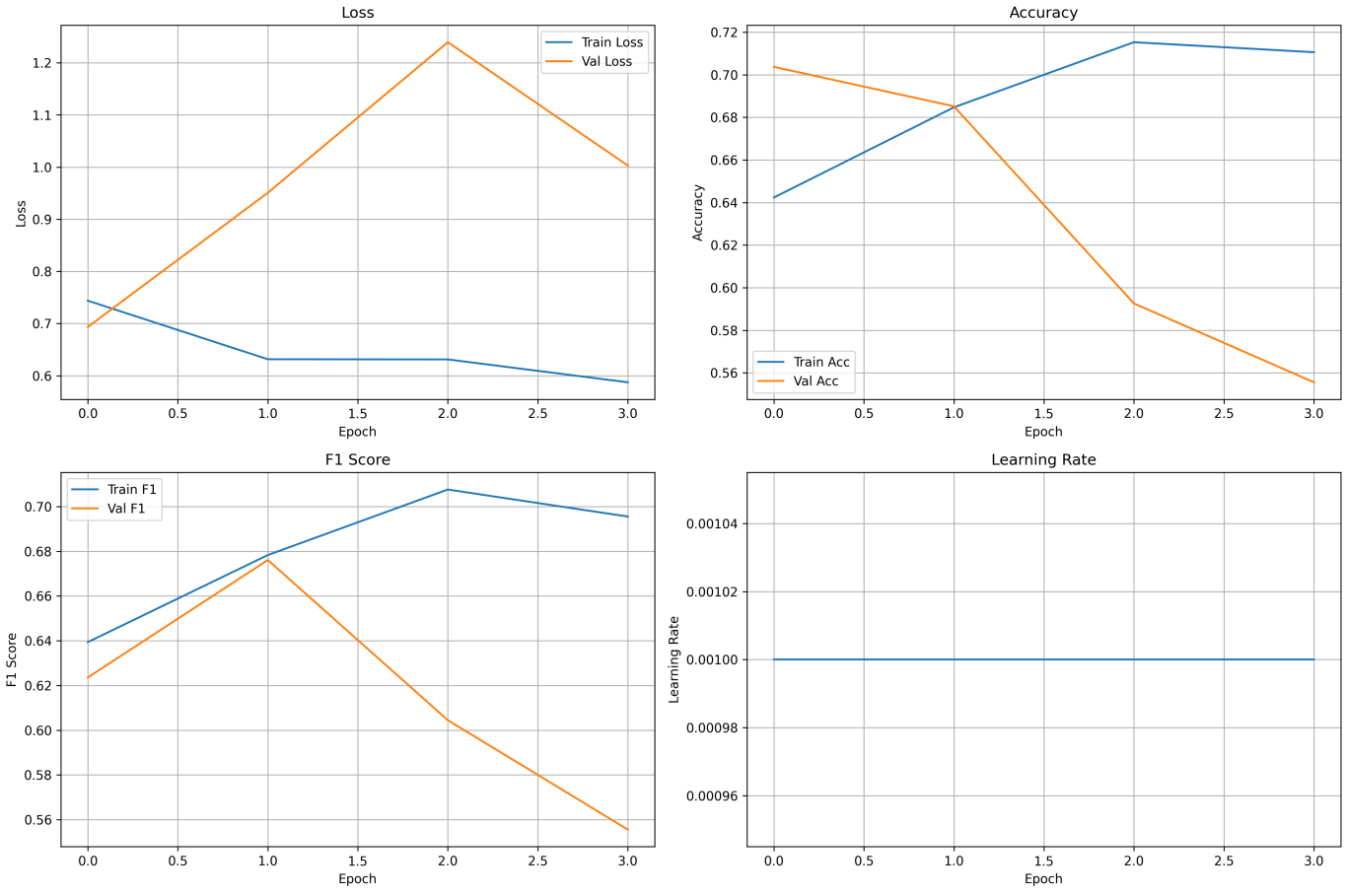

Fig. 5: Fast training mode curves (EfficientNet–B0).

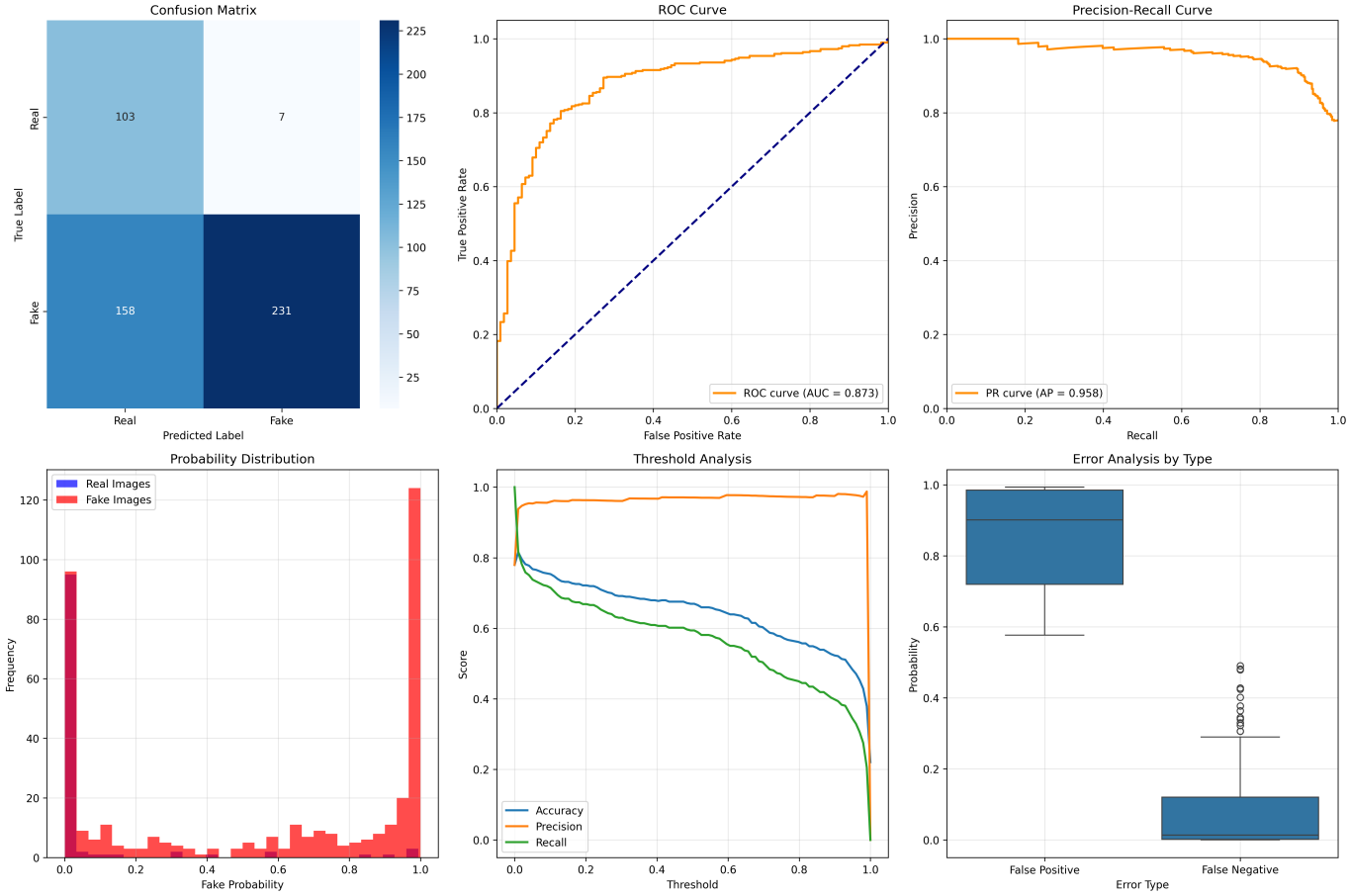

Fig. 6: Comprehensive evaluation of the proposed model. Top-left: confusion matrix. Top-middle: ROC curve. Top-right: precision–recall curve. Bottom-left: probability distribution for real and fake images. Bottom-middle: threshold analysis of accuracy, precision and recall. Bottom-right: box plot of error probabilities for false positives and false negatives.

TABLE I: Per-class precision and recall on the test set.

Class	Precision (%)	Recall (%)
Real	39.46	93.64
Fake	97.06	59.38

D. Per-Class Performance

Table I shows precision and recall for each class.

The system is intentionally tuned for high fake precision: when it predicts "fake", it is correct in more than 97% of cases. This behaviour is suitable in scenarios where incorrectly labelling a real image as fake would be harmful.

E. Performance Analysis

Fig. 7 presents additional performance-related plots, comparing model complexity and resource usage.

EfficientNet–B4 offers a good balance between accuracy and complexity compared to Xception, while still being more accurate than EfficientNet–B0. GPU acceleration greatly reduces training time.

IX. Discussion

A. Strengths

The proposed system exhibits several strengths:

- **High reliability for fake predictions:** a fake precision of 97.06% ensures that virtually all flagged images are truly manipulated.
- **Practical deployment:** the Flask web application provides real-time classification with inference latency below one second on typical hardware.
- **Robust training pipeline:** strong augmentation, transfer learning and regularisation enable good performance on a relatively small dataset.
- **Extensibility:** the modular architecture allows swapping the backbone model, changing thresholds and updating the dataset without major redesign.

B. Limitations

Some limitations remain:

- **Dataset size and diversity:** only 978 images from a single dataset are used; generalisation to unseen deepfake types may be limited.

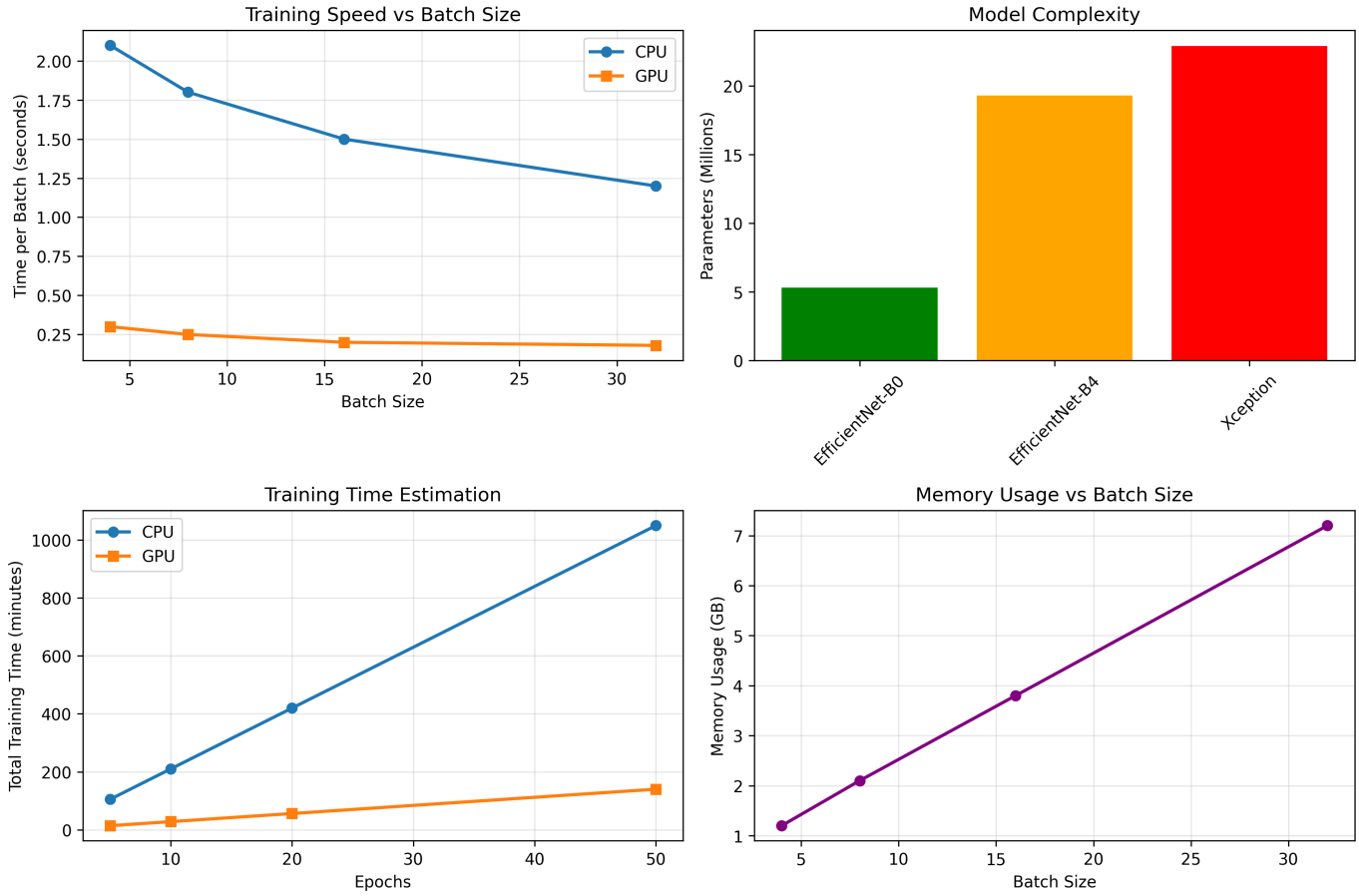

Fig. 7: Performance analysis. Top-left: training speed vs batch size on CPU and GPU. Top-right: model complexity (parameter count) of EfficientNet–B0, EfficientNet–B4 and Xception. Bottom-left: estimated total training time vs epochs for CPU and GPU. Bottom-right: memory usage vs batch size.

- **Overfitting:** training accuracy is significantly higher than test accuracy, indicating that the model learns dataset-specific patterns.
- **Lower fake recall:** some fake images are missed, particularly those with very high visual quality and few artifacts.

C. Comparison with Baselines

The fast training mode with EfficientNet–B0 provides a reference baseline. It achieves an accuracy of 75.95% but requires only a fraction of the training time. More complex architectures such as Xception and Vision Transformers were considered but not fully implemented due to resource constraints. The results suggest that EfficientNet–B4 is a strong compromise between performance and complexity.

X. Project Results and System Output

This section presents the final results of the proposed AI-based deepfake image detection system through its deployed web application. The purpose of this section is to demonstrate the end-to-end functionality of the system, validating not only the trained EfficientNet–B4 model but also its real-time usability in a practical environment.

The developed Flask-based web application allows users to upload an image, performs secure preprocessing and inference, and displays classification results along with probability scores and confidence levels. The system was tested using both authentic (real) images and AI-generated or manipulated (fake) images.

A. Fake Image Detection Result

Fig. 8 shows the system output when a deepfake image is uploaded. The model correctly classifies the image as **Fake Image Detected** with a confidence score of **98%**. The probability breakdown indicates a fake probability of **100%** and a real probability of **0%**.

This result highlights the effectiveness of the trained EfficientNet–B4 model and the aggressive prediction strategy adopted in this work. The system is intentionally tuned to achieve very high precision for the fake class, ensuring that images flagged as fake are almost certainly manipulated. Such behavior is particularly important in sensitive applications such as digital forensics, journalism, and online content moderation.

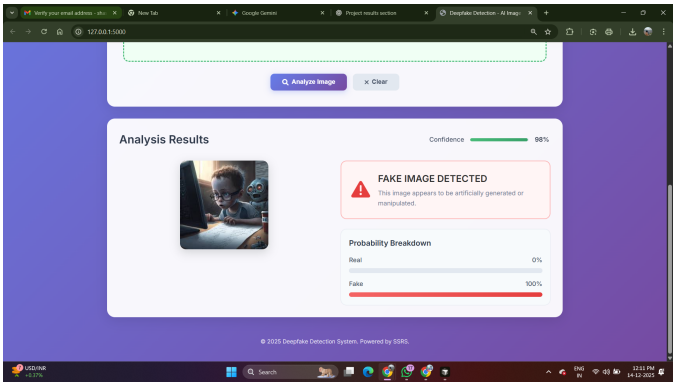

Fig. 8: Web application output for a fake image. The system correctly detects the image as fake with high confidence and probability.

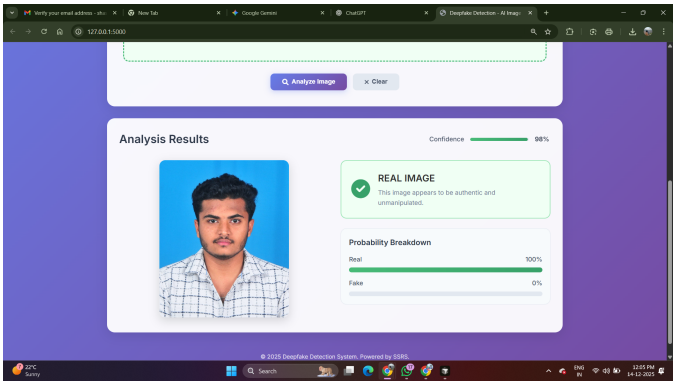

Fig. 9: Web application output for a real image. The system correctly classifies the image as authentic with high confidence.

B. Real Image Detection Result

Fig. 9 illustrates the output of the system when a genuine, unmanipulated image is provided as input. The model successfully classifies the image as **Real Image** with a confidence score of **98%**. The probability distribution shows a real probability of **100%** and a fake probability of **0%**.

This result demonstrates that the system does not falsely label authentic images as deepfakes. Avoiding false positives is critical in real-world scenarios, where incorrect classification of real images could lead to misinformation, reputational harm, or loss of trust. The observed behavior confirms that the proposed system maintains a balanced and responsible decision-making process.

C. Overall System Validation

The live system outputs confirm that the proposed deepfake image detection framework operates reliably in real-world conditions. The integration of EfficientNet–B4 with transfer learning, robust preprocessing, probability calibration, and an intuitive web interface results in a practical and deployable solution.

The system provides fast inference, clear visual feedback, and interpretable probability scores, making it suitable for use by both technical and non-technical users. These results validate the successful implementation of the complete pipeline—from model training to real-time deployment—and demonstrate the practical applicability of the proposed approach.

XI. Web Application and Deployment

A. Backend Implementation

The backend is written in Python using Flask. At startup, the trained EfficientNet–B4 model checkpoint is loaded into memory. An aggressive predictor module wraps the model to provide post-processed outputs.

The main endpoints are:

- `GET /` — returns the HTML front page.
- `POST /upload` — accepts image uploads, validates them, runs inference and returns JSON with label, probabilities and confidence.

To ensure robustness, file size limits and type checks are enforced. Errors are returned with user-friendly messages.

B. Frontend Interface

The frontend, implemented with HTML, CSS and JavaScript, supports drag-and-drop uploads as well as traditional file selection. After the user selects an image, a preview is displayed. When the prediction is returned from the server, the UI shows:

- Predicted label (Real/Fake) with colour coding.
- Fake and real probabilities.
- Confidence percentage.

This simple interface allows even non-technical users to interact with the deepfake detector.

C. Deployment Options

The application can run locally on a laptop or desktop, be served over a LAN for lab environments, or be deployed to cloud platforms such as AWS EC2, Azure or Google Cloud. Containerisation with Docker is straightforward due to the small number of dependencies.

XII. Conclusion

This paper presented a complete AI-based deepfake image detection system using EfficientNet–B4 and transfer learning. Trained on a Kaggle dataset of 978 images, the model achieves a fake-class precision of 97.06%, a ROC–AUC of 0.873 and strong overall discriminative performance. A practical Flask web application integrates the model into a user-friendly interface suitable for real-time image verification.

Despite limitations in dataset size and generalisation, the work demonstrates how modern CNN architectures, combined with careful preprocessing and threshold tuning, can form the basis of deployable deepfake detection tools. The modular design allows the system to be extended with larger datasets and more advanced models in future.

XIII. Future Work

Several directions can further enhance the system:

- **Larger and more diverse datasets:** incorporating FaceForensics++, DFDC and other datasets will improve generalisation and robustness.
- **Ensemble models:** combining EfficientNet with XceptionNet, ViT or ConvNeXt in an ensemble may yield higher accuracy.
- **Video deepfake detection:** extending the system to handle videos using temporal features and recurrent or transformer-based architectures.
- **Explainability:** using Grad-CAM or attention maps to highlight image regions that drive the decision will increase user trust.
- **Adversarial robustness:** applying adversarial training or detection mechanisms to defend against targeted attacks on the detector itself.

Acknowledgment

The author would like to express sincere gratitude to **Ms. T R USHA**, project guide, and to the faculty of the Department of Artificial Intelligence and Machine Learning, Navkis College of Engineering, Hassan, for their continuous guidance, encouragement and support throughout the course of this project.

References

[1] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. NeurIPS*, 2014.

[2] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. ICML*, 2019.

[3] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE CVPR*, 2017, pp. 1251–1258.

[4] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. NeurIPS*, 2020.

[5] A. Rössler *et al.*, "FaceForensics++: Learning to detect manipulated facial images," in *Proc. IEEE ICCV*, 2019.

[6] D. Afchar *et al.*, "MesoNet: a compact facial video forgery detection network," in *Proc. IEEE WIFS*, 2018.

[7] PyTorch documentation. [Online]. Available: https://pytorch.org/docs/

[8] Flask documentation. [Online]. Available: https://flask.palletsprojects.com/

[9] Albumentations documentation. [Online]. Available: https://albumentations.ai/

[10] S. Bagchi, "Deepfake image detection," Kaggle dataset. [Online]. Available: https://www.kaggle.com/