Image Authenticity Detection Using Statistical Image Forensics

---

1. Project Overview

With the rapid growth of **image editing tools and AI image generators**, it has become increasingly difficult to distinguish between **real**, **edited**, and **AI-generated images**. Human visual inspection is unreliable, and most existing systems rely heavily on **deep learning models**, which are computationally expensive and lack explainability.

This project proposes a **lightweight, explainable image authenticity detection system** that uses **statistical image forensics** instead of deep convolutional neural networks. The system analyzes numerical patterns extracted from images and applies machine learning to classify images as **REAL**, **FAKE**, or **LIKELY MANIPULATED**, along with a confidence score.

---

2. Problem Statement

- Image manipulation and AI generation tools are widely accessible
- Visual inspection cannot reliably detect manipulations
- Deep learning–based solutions:
    - Require large datasets and GPUs
    - Act as black boxes
    - Use rigid, static thresholds
- There is a need for:
    - Explainable decisions
    - Lightweight computation
    - Confidence-aware classification

---

3. Proposed Solution

The system detects image authenticity by:

- Extracting **statistical and forensic features** from images
- Training a **machine learning model** on numerical features
- Introducing a **confidence-aware decision layer**

- Using **threshold calibration inspired by reinforcement learning** to improve reliability

Unlike CNNs, this approach:

- Does not learn visual patterns
- Learns **numerical irregularities caused by manipulation**
- Is transparent and interpretable

---

4. Datasets Used

To ensure robustness, multiple real-world datasets are combined:

4.1 CASIA v2 Dataset (CASIA 2.0 Image Tampering Detection Dataset)

- **Au** → Authentic (REAL)
- **Tp** → Tampered (FAKE)

4.2 PS-Battles Dataset (PS-Battles)

- Original images → REAL
- Photoshopped images → FAKE

4.3 Deepfake Dataset (deepfake and real images)

- AI-generated synthetic images → FAKE

Final Dataset

- Balanced dataset
- Binary classification used for training:
    - REAL = 0
    - FAKE = 1

---

5. Feature Engineering (Statistical Parameters)

Each image is converted into a **numerical feature vector**.
The following features are extracted:

5.1 Entropy

- Measures randomness in pixel intensity distribution
- Manipulated images often show abnormal entropy levels

5.2 Noise Statistics

- Estimates sensor and smoothing noise
- Editing and AI generation alter natural noise patterns

5.3 Texture (GLCM Features)

- Contrast, homogeneity, energy, correlation
- Detects disruption in spatial pixel relationships

5.4 Frequency Domain (FFT)

- Separates low-frequency and high-frequency energy
- AI images show spectral imbalance

5.5 Color Channel Statistics

- Mean and standard deviation of R, G, B channels
- AI images often show unnatural color distributions

5.6 Edge Density

- Measures sharpness and boundary irregularities
- Edited regions often contain abnormal edges

5.7 JPEG Artifact Score

- Detects compression inconsistencies
- Editing introduces recompression artifacts

---

6. Methodology (System Workflow)

The system follows a structured pipeline:

1. **Image Upload**
   User uploads an image through the frontend UI
2. **Preprocessing**
   - Resize
   - Normalize
   - Convert color spaces
3. **Feature Extraction**
   Extract statistical forensic features from the image

4. **Machine Learning Classification**
    - Model: **Random Forest**
    - Input: Numerical feature vector
    - Output: Probability scores
5. **Confidence-Aware Decision Layer**
   Final label is decided using calibrated thresholds

---

7. Machine Learning Model

Model Used

- **Random Forest Classifier**

Why Random Forest?

- Handles non-linear relationships
- Robust to noise
- Works well with tabular numerical data
- Provides probability outputs (important for confidence scoring)

Performance (Balanced Dataset)

- Accuracy: ~71–72%
- Precision, Recall, F1-Score are balanced across classes
- Confusion matrix shows reasonable separation between REAL and FAKE images

---

8. Decision Layer & Threshold Calibration

Instead of using a fixed decision rule, the system uses a **confidence-aware decision layer**.

Decision Rules:

- **Confidence ≥ 0.75** → FAKE
- **0.60 ≤ Confidence < 0.75** → Likely Manipulated
- **Confidence < 0.60** → REAL

Threshold Calibration

- Inspired by **Reinforcement Learning**

- Thresholds are tuned based on prediction behavior
- Goal: maximize correct decisions while reducing false positives

This improves trustworthiness, especially for borderline cases

Image Authenticity Detection Us…

.

---

9. System Architecture

Frontend

- React (Vite)
- Lovable-generated UI
- Image upload, preview, animations, result display

Backend

- Python
- FastAPI
- REST API for prediction

Machine Learning & Image Processing

- Pandas
- NumPy
- Scikit-learn
- OpenCV

---

10. Key Advantages of This Approach

- Lightweight and fast
- No GPU dependency
- Explainable decisions
- Confidence-aware outputs
- Robust across edited and AI-generated images
- Suitable for real-time deployment

---

11. Limitations & Future Improvements

- Accuracy depends on dataset quality
- Very high-quality AI images remain challenging
- Future work:
    - Hybrid statistical + deep learning approach
    - Continuous threshold learning
    - Localized manipulation heatmaps
    - Larger and more diverse datasets

---

12. Conclusion

This project demonstrates that **statistical image forensics combined with machine learning** can effectively detect image authenticity without relying on deep neural networks. By focusing on numerical irregularities and introducing a confidence-aware decision layer, the system provides an **interpretable, efficient, and practical solution** for real-world image authenticity verification.

Root Folder Structure Explanation

P1-IMAGEAUTH/

├ backend/

├data/

├frontend/

├ image2stats/

├ notebooks/

├ scripts/

└── README.md

Each folder has a **clear responsibility**, explained below.

3️⃣ Backend Folder (/backend)

backend/

├── app.py

├── feature_extractor.py

├── requirements.txt

└── threshold.txt

🔹 app.py

Purpose:

- FastAPI application
- Loads trained ML model (.pkl)
- Accepts image uploads
- Extracts features
- Applies confidence thresholds
- Returns prediction (REAL / FAKE / LIKELY MANIPULATED)

Key responsibilities:

- /predict API endpoint
- Model inference
- Threshold-based decision layer

🔹 **feature_extractor.py**

Purpose:

- Core **statistical feature extraction logic**
- Converts raw image bytes → numerical feature vector

Features extracted include:

- Entropy
- Noise statistics
- Texture (GLCM)
- Frequency (FFT)
- Color channel statistics
- Edge density
- JPEG artifact score

⚠️ This file **must match** the feature columns used during training.

🔹 **requirements.txt**

Purpose:
Defines Python dependencies required to run the backend.

Example:

fastapi

uvicorn

numpy

opencv-python

scikit-learn

scikit-image

pillow

scipy

python-multipart

joblib

🔹 **threshold.txt**

Purpose:
Stores calibrated confidence thresholds used during inference.

Example:

FAKE_HIGH=0.75

FAKE_MEDIUM=0.60

This allows **easy tuning without retraining the model**.

4️⃣ **Data Folder (/data)**

data/

└── features_15k.csv

🔹 **features_15k.csv**

Purpose:

- Final structured dataset
- Contains extracted features for ~15,000 images
- Used for ML training and evaluation

Columns include:

- Entropy
- Noise
- GLCM
- FFT
- Color statistics
- Labels

5️⃣ Image Cleaning & Dataset Scripts (/image2stats)

image2stats/

├── casia_clean.py

├── deepfake_clean.py

└── ps_battles_clean.py

🔹 Purpose of this folder

Handles **raw dataset normalization** and organization.

🔹 Script roles

- casia_clean.py → CASIA v2 (real vs tampered)
- deepfake_clean.py → AI-generated images
- ps_battles_clean.py → Photoshop Battles (edited images)

Each script:

- Downloads / cleans images
- Removes corrupt files
- Standardizes folder structure

6️⃣ Notebooks Folder (/notebooks)

notebooks/

├── 02_ml_training_confidence.ipynb

└── 03_rl_threshold_optimization.ipynb

This folder is **critical**.

🔹 02_ml_training_confidence.ipynb

Purpose:

- Loads features_15k.csv
- Trains ML models (Random Forest / XGBoost etc.)

- Evaluates accuracy & confusion matrix
- Saves trained model as .pkl

📌 **This is where the final model is created.**

🔹 **03_rl_threshold_optimization.ipynb**

Purpose:

- Uses Reinforcement Learning / reward-based logic
- Tunes confidence thresholds
- Optimizes decision boundaries for:
    - REAL
    - LIKELY MANIPULATED
    - FAKE

📌 **This improves reliability, not raw accuracy.**

7️⃣ **How to Generate the Model .pkl File**

✅ **Step-by-step**

1. Open notebook:
2. notebooks/02_ml_training_confidence.ipynb
3. Run all cells:
    - Feature loading
    - Train/test split
    - Model training
    - Evaluation
4. Final cell saves model:
5. import joblib
6. joblib.dump(model, "image_auth_model.pkl")
7. Move the file to:
8. backend/image_auth_model.pkl

9. Ensure app.py loads it:
10. model = joblib.load("image_auth_model.pkl")

8️⃣ Scripts Folder (/scripts)

scripts/

├── extract_features.py

└── rgb_analysis.py

🔹 **extract_features.py**

- Batch feature extraction
- Converts raw images → CSV rows

🔹 **rgb_analysis.py**

- Exploratory analysis on RGB distributions
- Used during feature design phase

9️⃣ Frontend Folder (/frontend)

frontend/

├── src/

├── public/

├── package.json

├── vite.config.ts

└── tailwind.config.ts

✔ Frontend files are **not explained individually** as requested.

Frontend responsibilities:

- Image upload
- Scanning animation
- Result visualization
- Confidence bar

- Feedback buttons

End-to-End Execution Flow

1. User uploads image (Frontend)
2. Image sent to /predict (FastAPI)
3. Features extracted (statistics)
4. ML model predicts probabilities
5. Threshold calibration applied
6. Result returned:
    - REAL
    - LIKELY MANIPULATED
    - FAKE
7. UI displays result + confidence

⚙️ Setup Instructions

This section explains how to set up and run the **Image Authenticity Detection System** on a local machine.

1️⃣ Prerequisites

Ensure the following are installed:

🔹 System Requirements

- Operating System: Windows / Linux / macOS
- Minimum RAM: 8 GB (recommended)
- Disk Space: ~5–10 GB (datasets + dependencies)

🔹 Software Requirements

- **Python 3.10 or 3.11**
- **Node.js ≥ 18**
- **npm** (comes with Node.js)
- **Git**

- **VS Code** (recommended)

2️⃣ Project Clone or download

git clone <your-repository-url>

cd P1-IMAGEAUTH

3️⃣ Backend Setup (FastAPI + ML)

🔹 Step 1: Create Virtual Environment

cd backend

python -m venv venv

Activate it:

Windows

venv\Scripts\activate

Linux / macOS

source venv/bin/activate

🔹 Step 2: Install Python Dependencies

pip install -r requirements.txt

🔹 Step 3: Ensure Model File Exists

You must have a trained model file:

backend/image_auth_model.pkl

If not present:

1. Open:
2. notebooks/02_ml_training_confidence.ipynb
3. Run all cells
4. The final cell saves the model
5. Copy the .pkl file into:

6. backend/

🔹 **Step 4: Start Backend Server**

python -m uvicorn app:app --reload

Backend runs at:

http://127.0.0.1:8000

Test:

http://127.0.0.1:8000/docs

4️⃣ Frontend Setup (Vite + React)

🔹 **Step 1: Move to Frontend**

cd ../frontend

🔹 **Step 2: Install Dependencies**

npm install

🔹 **Step 3: Run Frontend**

npm run dev

Frontend runs at:

http://localhost:5173

5️⃣ Connecting Frontend with Backend

🔹 **Backend API Used**

POST http://127.0.0.1:8000/predict

🔹 **Request Format**

- Method: POST
- Type: multipart/form-data

- Field name: file
- Value: image file

🔹 **Response Format**

```
{
 "prediction": "REAL",
 "confidence": 0.62
}
```

6️⃣ Dataset Preparation (Optional – For Training)

If retraining is required:

1. Clean datasets using:
2. image2stats/
3. Extract features using:
4. scripts/extract_features.py
5. Generate CSV:
6. data/features_15k.csv

7️⃣ Threshold Calibration (Optional)

To optimize confidence thresholds:

1. Open:
2. notebooks/03_rl_threshold_optimization.ipynb
3. Run all cells
4. Update values in:
5. backend/threshold.txt

8️⃣ Common Issues & Fixes

❌ ModuleNotFoundError: skimage

pip install scikit-image

❌ **python-multipart missing**

pip install python-multipart

❌ **npm run dev not found**

Ensure package.json contains:

"scripts": {

"dev": "vite"

}

🚦 **Final Verification Checklist**

✅ Backend running on port 8000
✅ Frontend running on port 5173
✅ Image upload works
✅ Prediction & confidence displayed
✅ Model .pkl loaded successfully

✅ **Setup Complete**

You can now:

- Upload images
- View authenticity prediction
- Observe confidence-based decision output