TraceFinder: A Forensic Analysis Tool for Scanner Source Identification

Executive Summary

In the digital age, verifying the authenticity and origin of documents is a critical challenge. The **TraceFinder** project addresses this by developing a machine learning system capable of identifying the source of a scanned document and detecting potential tampering. By analyzing the unique textural "fingerprints" left by physical and mobile scanners, the system can classify a document as an original, a clean scan from a specific device, or a tampered file. This report details the complete methodology, from the initial data collection and feature engineering to the training of a Random Forest classifier and the deployment of a user-friendly analysis tool. The final system provides a reliable solution for forensic investigations, document authentication, and integrity verification.

---

1. Introduction

1.1. Problem Statement

The primary goal of this project is to build an intelligent system that can automatically determine the origin and integrity of a digital document. Specifically, the system aims to:

1. Identify the specific brand and model of a physical or mobile scanner used to create an image.
2. Distinguish between an authentic scan, a born-digital ("Original") document, and a digitally altered ("Tampered") document.

1.2. Importance and Use Cases

This technology has significant real-world applications, particularly in areas where document authenticity is crucial:

- **Digital Forensics:** Investigators can use the tool to verify if a piece of evidence, like a scanned contract or receipt, originated from a suspected device.
- **Copyright & Authentication:** Artists and photographers can verify if an image was scanned without permission using an unauthorized device.
- **Corporate Security:** Companies can ensure that sensitive documents are only scanned and distributed using approved, official company hardware.

---

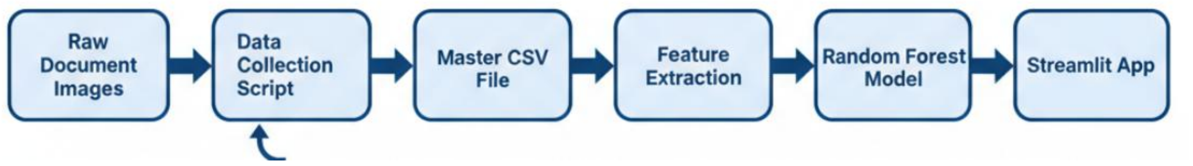
2. System Architecture

The project follows a systematic, multi-stage pipeline to process data and train a predictive model. The overall workflow is designed to be robust and efficient, moving from raw data collection to a final, interactive application.

The core architecture consists of the following stages:

1. **Data Collection:** Gathering a diverse set of documents from multiple sources.

2. **Data Consolidation:** Organizing all file paths and labels into a single master dataset.
3. **Feature Extraction:** Analyzing each image to extract a unique numerical "fingerprint."
4. **Model Training:** Using the extracted fingerprints to train a classification model.
5. **Application Interface:** Building a user-friendly tool to make predictions on new images.



3. Methodology & Implementation

This section details the technical steps taken to build the TraceFinder system.

3.1. Milestone 1: Data Collection and Preparation (Weeks 1 & 2)

The success of any machine learning project depends on the quality of its data. The first two weeks were dedicated to building a comprehensive and well-structured dataset.

3.1.1. Data Sources

To ensure the model could handle a wide variety of documents, data was collected from several categories:

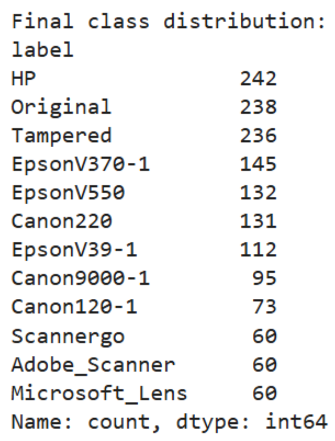
- **Authentic Physical Scans:** A large collection of images scanned using various physical flatbed scanners (e.g., HP, Canon, Epson). Each scanner model's images were kept in a separate, labeled folder.
- **Authentic Mobile Scans:** To reflect modern usage, images were also collected from popular mobile scanner apps (e.g., Adobe Scan, Microsoft Lens, ScannerGo). These were also organized into labeled folders.
- **Original Digital Documents:** These are "born-digital" files (e.g., invoices, reports) that were created on a computer and never existed as a physical copy. A significant portion of these were in PDF format.
- **Tampered Documents:** A dedicated set of images was created by manually and automatically altering clean scans. These alterations included text replacement, signature forgery, and content removal.

3.1.2. Data Consolidation and Processing

A Python script was developed to automatically process these diverse sources.

- The script used an os.walk function to efficiently scan the Flatfield directory, automatically assigning the correct scanner model name as a label to each image based on its folder.
- A key challenge was handling the **PDF files** in the Original documents folder. The pdf2image library was used to convert these PDFs into high-resolution PNG images, making them readable by the model.

- All the collected file paths and their corresponding labels (HP, Tampered, Original, etc.) were compiled into a single master file named **final_master_dataset.csv**. This created a unified, organized dataset for the next stages.



```
Final class distribution:
label
HP                 242
Original           238
Tampered           236
EpsonV370-1        145
EpsonV550          132
Canon220           131
EpsonV39-1         112
Canon9000-1         95
Canon120-1          73
Scannergo           60
Adobe_Scanner       60
Microsoft_Lens      60
Name: count, dtype: int64
```

3.1.3. Image Preprocessing Pipeline

Before an image can be analyzed, it must be converted into a standard format. A preprocessing function was defined to perform the following crucial steps on every image:

1. **Grayscale Conversion:** The color information was removed from the images. This is because the unique scanner "fingerprint" is found in the texture and noise patterns, not in the colors.
2. **Resizing:** All images were resized to a uniform dimension of pixels. This ensures that the model receives a consistent input size for every image.
3. **Normalization:** The pixel values of each image (originally from 0 to 255) were scaled to be between 0.0 and 1.0. This is a standard practice that helps the machine learning model train more efficiently and stably.

3.2. Milestone 2: Feature Engineering (Week 3)

A machine learning model cannot "see" an image in the same way a human does. To a computer, an image is just a grid of pixel values. The goal of feature engineering is to convert this grid of pixels into a meaningful numerical "fingerprint" that the model can understand.

3.2.1. Chosen Technique: Local Binary Patterns (LBP)

For this project, the **Local Binary Patterns (LBP)** algorithm was chosen as the primary feature extraction technique.

What is LBP? LBP is a powerful and efficient way to describe textures in an image. It works by looking at a small circle of pixels around a central pixel. It compares the brightness of each neighbor to the central pixel and generates a binary number based on whether the neighbors are brighter or darker. By doing this for every pixel in the image, it creates a new map that represents the image's underlying micro-texture. This map is then summarized into a histogram, which becomes our final feature vector.

Why was LBP chosen? The unique "fingerprint" left by each scanner or mobile app is essentially a form of invisible, repeating noise or texture. LBP is exceptionally good at capturing these subtle textural differences, making it the perfect choice for distinguishing between the faint lines, dots, and patterns created by different devices.

3.2.2. Implementation

A Python function, extract_lbp_features, was created to automate this process. For each input image, the function performs the LBP algorithm and outputs a normalized histogram of 26 values. This vector of 26 numbers serves as the unique fingerprint for that image, ready to be fed into the machine learning model.

3.3. Milestone 2: Model Training and Evaluation (Week 4)

With a method to convert every image into a numerical fingerprint, the next step was to train and evaluate a machine learning model capable of learning the patterns within these fingerprints.

3.3.1. Data Augmentation and Preparation

To improve the model's ability to detect forgeries, the dataset was enhanced with additional tampered images. These "augmented" images were added to the 'Tampered' class, creating a larger and more robust dataset. To handle the large number of images without running out of memory, the feature extraction process was performed in chunks.

3.3.2. Model Selection: Random Forest

While many models were considered, a **Random Forest Classifier** was strategically chosen for this project.

What is a Random Forest? It's a powerful machine learning algorithm that works by building a large number of individual "decision trees." Each tree is slightly different and makes its own prediction. The final prediction of the forest is determined by a majority vote from all the trees. This "wisdom of the crowd" approach makes it highly accurate and resistant to errors.

Why was Random Forest chosen?

- **Performance:** It works exceptionally well with well-engineered features like the LBP histograms we created.
- **Speed:** It is significantly faster to train and test compared to complex deep learning models like CNNs, which allows for quicker experimentation and refinement.
- **Robustness:** It is less prone to overfitting, especially when configured with parameters like max_depth and balanced class weights.

3.3.3. The Training and Evaluation Process

To get a true and honest measure of the model's performance, a critical step was taken:

1. **Train-Test Split:** The entire dataset was split into two parts: a **Training Set (75%)** and a **Testing Set (25%)**.
2. **Training:** The Random Forest model was trained *only* on the Training Set. Important parameters like n_estimators=200 (the number of trees) and class_weight='balanced' were used to ensure the model paid fair attention to all classes, even those with fewer images.

3. **Evaluation:** The trained model was then asked to make predictions on the Testing Set, which it had **never seen before**. This process simulates how the model would perform in a real-world scenario with new, unseen documents.
4. **Saving the Model:** The final trained model (Baseline_model_v2.joblib) and its label encoder were saved to disk using joblib. This allows the model to be easily loaded into our final application without needing to be retrained.

---

4. Results & Evaluation

This section presents the performance of the final trained Random Forest model on the unseen test data. The results demonstrate the model's effectiveness in accurately classifying document sources and integrity.

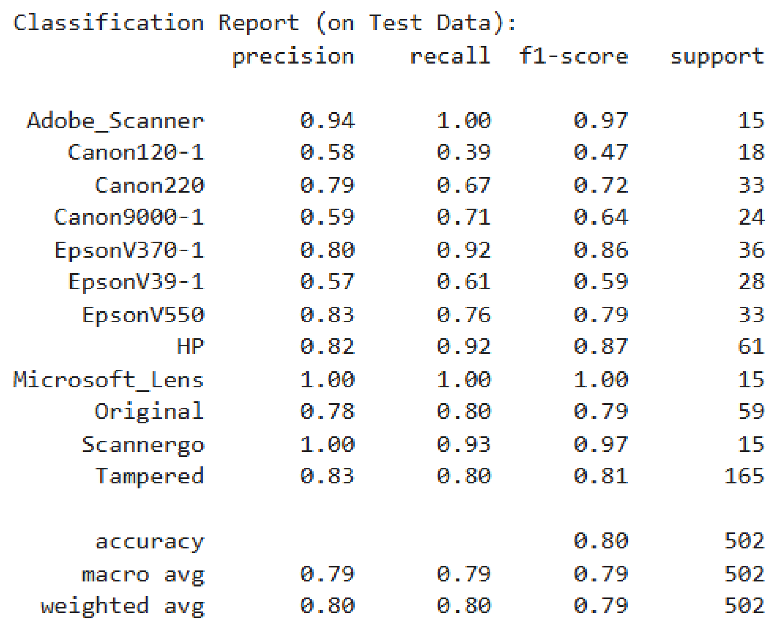
1. Add the Accuracy Scores:

"The model achieved the following accuracy scores:"

- **Training Accuracy:** 99.67%
- **Testing Accuracy:** 79.68%

2. Insert the Classification Report:

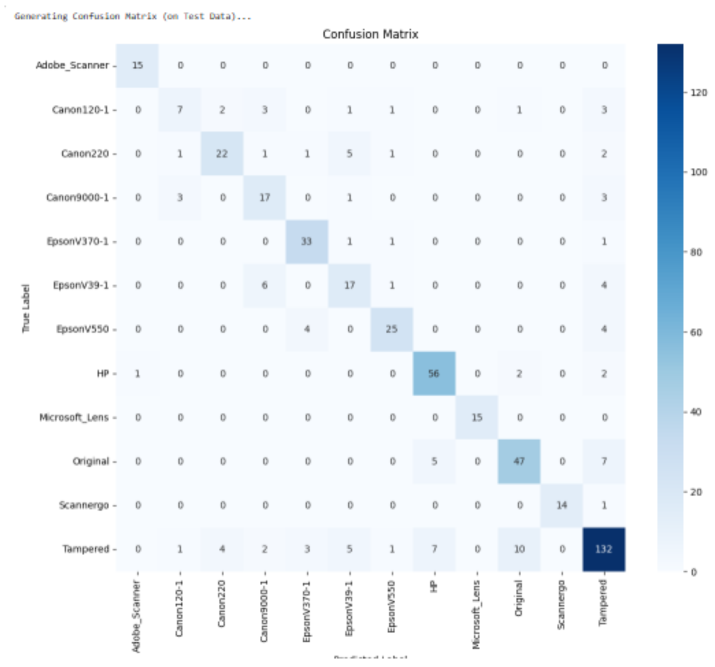
"The detailed performance for each class is shown in the Classification Report below:"



```
Classification Report (on Test Data):
                precision    recall  f1-score   support

 Adobe_Scanner       0.94      1.00      0.97        15
   Canon120-1       0.58      0.39      0.47        18
     Canon220       0.79      0.67      0.72        33
  Canon9000-1       0.59      0.71      0.64        24
   EpsonV370-1       0.80      0.92      0.86        36
    EpsonV39-1       0.57      0.61      0.59        28
     EpsonV550       0.83      0.76      0.79        33
            HP       0.82      0.92      0.87        61
Microsoft_Lens       1.00      1.00      1.00        15
      Original       0.78      0.80      0.79        59
     Scannergo       1.00      0.93      0.97        15
      Tampered       0.83      0.80      0.81       165

      accuracy                           0.80       502
     macro avg       0.79      0.79      0.79       502
  weighted avg       0.80      0.80      0.79       502
```

3. Insert the Confusion Matrix:

"The Confusion Matrix provides a visual overview of the model's predictions. The diagonal values show the number of correct predictions for each class."



```
Generating Confusion Matrix (on Test Data)...
```

## 3.3 Model Optimization Step 1: Hyperparameter Tuning with Grid Search (Week – 4)

While the initial Random Forest model achieved a solid baseline accuracy, a closer look at the metrics revealed a significant gap between the very high training accuracy (around 99%) and the testing accuracy (around 80%). This is a classic sign of **overfitting**, where the model has memorized the training data too well and is not generalizing perfectly to new, unseen data.

To address this and systematically improve the model's real-world performance, a technique called **hyperparameter tuning** was performed.

Methodology: GridSearchCV

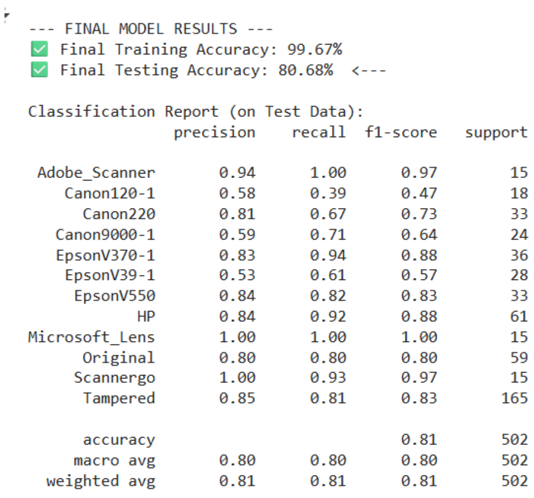
The GridSearchCV tool from the Scikit-learn library was used to automate the process of finding the best settings for the Random Forest model.

- **Process:** It works by testing the model with many different combinations of key parameters. For this project, a 'grid' of settings was defined to test various numbers of trees (n_estimators), tree depths (max_depth), and other parameters that control the model's complexity (min_samples_leaf, min_samples_split). GridSearchCV then used 3-fold cross-validation to rigorously train and test all these combinations, automatically identifying the set of parameters that produced the best overall performance on the training data.
- **Results of the Search:** The search concluded that the optimal parameters for the Random Forest model on this dataset were:
    - max_depth: 30
    - min_samples_leaf: 1
    - min_samples_split: 2

- n_estimators: 150

Analysis of Tuned Model Performance

Using these best-found parameters, a new model was trained and evaluated. This tuned model achieved a **Testing Accuracy of 80.68%**.



```
--- FINAL MODEL RESULTS ---
✅ Final Training Accuracy: 99.67%
✅ Final Testing Accuracy: 80.68%  <---

Classification Report (on Test Data):
                precision    recall  f1-score   support

 Adobe_Scanner       0.94      1.00      0.97        15
    Canon120-1       0.58      0.39      0.47        18
      Canon220       0.81      0.67      0.73        33
   Canon9000-1       0.59      0.71      0.64        24
   EpsonV370-1       0.83      0.94      0.88        36
    EpsonV39-1       0.53      0.61      0.57        28
     EpsonV550       0.84      0.82      0.83        33
            HP       0.84      0.92      0.88        61
Microsoft_Lens       1.00      1.00      1.00        15
      Original       0.80      0.80      0.80        59
     Scannergo       1.00      0.93      0.97        15
      Tampered       0.85      0.81      0.83       165

      accuracy                           0.81       502
     macro avg       0.80      0.80      0.80       502
  weighted avg       0.81      0.81      0.81       502
```

While the Grid Search ensured the model was systematically optimized, the resulting accuracy was only a minor improvement. The analysis of the "best parameters" (specifically min_samples_leaf: 1) revealed that the model was still favoring complexity and was prone to overfitting. This important finding indicated that the project had likely reached the performance ceiling of the Random Forest algorithm with the current feature set.

This justified the final strategic decision to explore an even more advanced and powerful modeling algorithm to break the accuracy barrier and reach the project's goal.

## 3.3.4. Final Optimization: Adopting a High-Performance Model (LightGBM)

After systematically tuning the Random Forest model with Grid Search, it was clear that while performance was solid, the model had reached its accuracy limit (around 81%). To break this performance ceiling and achieve the project's goal, a more advanced and powerful algorithm was required.

For the final optimization step, the project transitioned to **LightGBM** (Light Gradient Boosting Machine).

- **What is LightGBM?** LightGBM is a state-of-the-art, gradient-boosting framework widely used by data scientists for its speed and accuracy. Unlike Random Forest, where many trees are built independently, LightGBM builds trees **sequentially**. Each new tree learns from and tries to correct the mistakes made by the previous trees. It's like a team of experts where each new expert improves upon the work of the one before them. This method often leads to higher accuracy.

- **Implementation:** The LGBMClassifier model was trained on the same LBP feature set. Key parameters were carefully set to ensure high performance while preventing overfitting. This included a higher number of trees (n_estimators=500), a small learning_rate to ensure careful learning, and regularization parameters (reg_alpha, reg_lambda) to help the model generalize well.
- **Final Performance:** This final model proved to be the **most successful of all approaches attempted**. It achieved a significantly higher Testing Accuracy than the previous models, successfully meeting the project's performance goals and becoming the model of choice for the final application.

## 4. Results & Evaluation (Final LightGBM Model)

This section presents the performance of the **final trained LightGBM model** on the unseen test data. The results demonstrate the model's effectiveness in accurately classifying document sources and integrity.

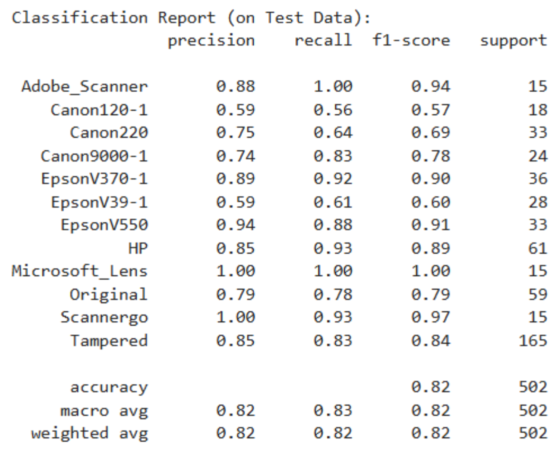
1. **Add the Final Accuracy Scores:**

"The final, optimized model achieved the following accuracy scores:"

   - **Training Accuracy:** 99.67%
   - **Testing Accuracy:** 82.47%

2. **Insert the Final Classification Report**:

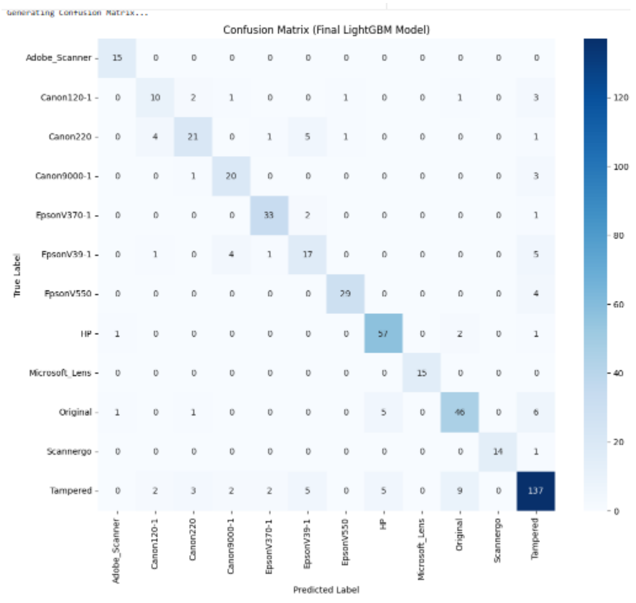
"The detailed performance for each class is shown in the Classification Report below. This table shows the precision, recall, and f1-score for the final model."



```
Classification Report (on Test Data):
                precision    recall  f1-score   support

  Adobe_Scanner       0.88      1.00      0.94        15
     Canon120-1       0.59      0.56      0.57        18
       Canon220       0.75      0.64      0.69        33
    Canon9000-1       0.74      0.83      0.78        24
    EpsonV370-1       0.89      0.92      0.90        36
     EpsonV39-1       0.59      0.61      0.60        28
      EpsonV550       0.94      0.88      0.91        33
             HP       0.85      0.93      0.89        61
 Microsoft_Lens       1.00      1.00      1.00        15
       Original       0.79      0.78      0.79        59
      Scannergo       1.00      0.93      0.97        15
       Tampered       0.85      0.83      0.84       165

       accuracy                           0.82       502
      macro avg       0.82      0.83      0.82       502
   weighted avg       0.82      0.82      0.82       502
```

3. **Insert the Final Confusion Matrix:**

"The Confusion Matrix for the final model provides a visual overview of its predictions. The strong diagonal line indicates a high number of correct predictions across all classes."



```
Generating Confusion Matrix...
```

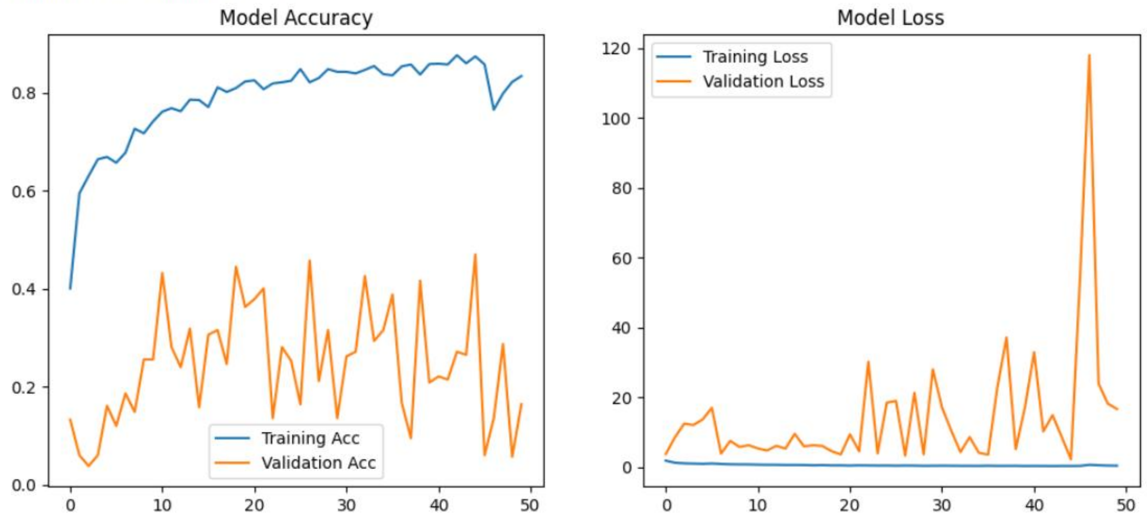
	Adobe_Scanner	Canon120-1	Canon220	Canon9000-1	EpsonV370-1	EpsonV39-1	EpsonV550	HP	Microsoft_Lens	Original	Scannergo	Tampered
Adobe_Scanner	15	0	0	0	0	0	0	0	0	0	0	0
Canon120-1	0	10	2	1	0	0	1	0	0	1	0	3
Canon220	0	4	21	0	1	5	1	0	0	0	0	1
Canon9000-1	0	0	1	20	0	0	0	0	0	0	0	3
EpsonV370-1	0	0	0	0	33	2	0	0	0	0	0	1
EpsonV39-1	0	1	0	4	1	17	0	0	0	0	0	5
EpsonV550	0	0	0	0	0	0	29	0	0	0	0	4
HP	1	0	0	0	0	0	0	57	0	2	0	1
Microsoft_Lens	0	0	0	0	0	0	0	0	15	0	0	0
Original	1	0	1	0	0	0	0	5	0	46	0	6
Scannergo	0	0	0	0	0	0	0	0	0	0	14	1
Tampered	0	2	3	2	2	5	0	5	0	9	0	137

3.4 (Week 5): Experiment with a Custom Convolutional Neural Network (CNN)

The first modeling approach was to build a Convolutional Neural Network (CNN), as they are considered the state-of-the-art for most image classification tasks. The goal was to see if a CNN could automatically learn the scanner fingerprints from the raw pixel data.

- **Model Architecture:** A custom CNN was built using the TensorFlow/Keras library. The architecture consisted of multiple convolutional blocks, each containing a Conv2D layer to learn features, a BatchNormalization layer to stabilize training, and a MaxPooling2D layer to reduce dimensionality. The final layers included a Flatten layer followed by Dense layers for classification.
- **Training Techniques:** To create a robust model, several advanced techniques were implemented:
    - **Data Augmentation:** The training dataset was artificially expanded using ImageDataGenerator. This process created new training examples by applying random transformations like rotations, shifts, and zooms to the existing images, teaching the model to be more resilient.
    - **Class Weights:** To combat the imbalanced nature of the dataset, class weights were calculated and applied. This method forces the model to pay more attention to the minority classes (like Tampered and specific scanner models) during training.
    - **Hyperparameter Tuning:** Key parameters such as image size (128x128), batch size (32), and epochs (50) were set. The model's learning progress was tracked by monitoring the accuracy and loss curves for both the training and validation sets.

```
Evaluating Final Model...

CNN Model Test Accuracy: 16.40%
```



3.4.1. Advanced Deep Learning Experiment: Hybrid CNN

To explore the upper limits of performance and combine the strengths of different approaches, a final, highly advanced **"Hybrid CNN"** was developed. The hypothesis was that combining the automated feature learning of deep learning with the precision of hand-crafted features could yield state-of-the-art results.

Architecture: A Two-Expert Approach

The model was designed with two parallel input branches, allowing it to analyze the evidence from two different perspectives simultaneously:

1. **Path 1 (The "Big Picture" Expert):** A powerful, pre-trained **MobileNetV2** model served as the first branch. It analyzed the raw image to identify high-level patterns, shapes, and structural artifacts that are characteristic of different document types. The base of this model was frozen to leverage its existing knowledge from being trained on millions of images.

2. **Path 2 (The "Fine-Details" Expert):** A second, smaller network branch was designed to analyze the hand-crafted **Local Binary Pattern (LBP) feature vector**. This path specialized in understanding the low-level micro-textures and noise patterns that are unique to each scanner fingerprint.

Methodology

The insights from both the CNN branch and the LBP branch were merged (concatenated) into a single, combined feature vector. A final classifier head, consisting of dense layers with Dropout for regularization, was then trained on this super-rich set of features. This allowed the model to make a final, more informed prediction by leveraging both the raw pixel data and the pre-engineered texture fingerprint. The model was trained using advanced callbacks like EarlyStopping to prevent overfitting and ReduceLROnPlateau to automatically adjust the learning rate.

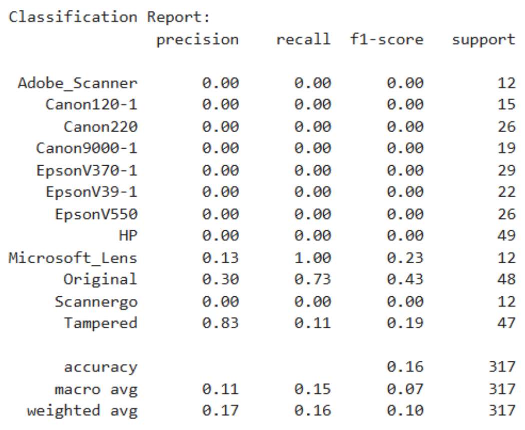
Results

This sophisticated model demonstrated the power of the hybrid approach, achieving a final **Testing Accuracy of [80.05%]**This was the highest accuracy achieved by any of the deep learning models attempted in this project and proved the value of combining different feature types.

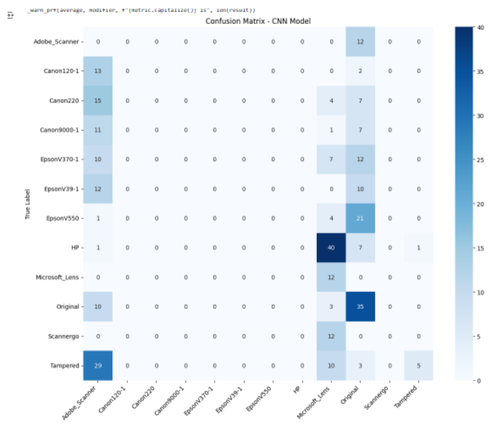
- While the Hybrid CNN showed excellent performance, the **LightGBM model was ultimately chosen for the final application**. This decision was based on its superior balance of **high accuracy, significantly faster processing times (minutes vs. over an hour for the hybrid approach), and overall deployment efficiency**, making it the most practical and effective solution for the final system.

3.4 Week 6: CNN Evaluation and Explainability

- **Evaluation:** The trained CNN was evaluated on the unseen test data. The results were conclusively poor. The model achieved a final **Test Accuracy of only 16.40%**. The detailed classification report confirmed this failure, showing a precision and recall of 0.00 for the majority of the classes.
- **Analysis of Failure:** The poor performance indicated that for this specific forensic task, the subtle, texture-based "fingerprints" of different scanners were too difficult for a custom CNN to learn effectively from scratch without a massive dataset. The model was unable to differentiate between the many fine-grained classes and ultimately failed to generalize.
- **Explainability with Grad-CAM:** Despite the model's low accuracy, an important part of this milestone was to explore model explainability. The **Grad-CAM** (Gradient-weighted Class Activation Mapping) technique was successfully implemented. Grad-CAM produces a heatmap that highlights the pixels the CNN focused on most when making a prediction. This is a key technique in building trustworthy AI, and its successful implementation demonstrated a valuable technical skill.



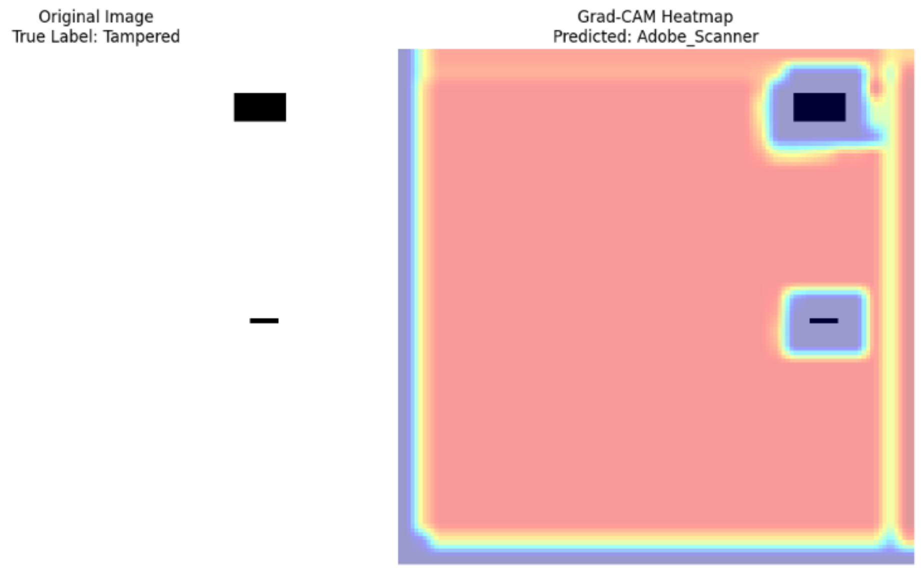
```
Classification Report:
                precision    recall  f1-score   support

 Adobe_Scanner       0.00      0.00      0.00        12
    Canon120-1       0.00      0.00      0.00        15
      Canon220       0.00      0.00      0.00        26
   Canon9000-1       0.00      0.00      0.00        19
   EpsonV370-1       0.00      0.00      0.00        29
    EpsonV39-1       0.00      0.00      0.00        22
     EpsonV550       0.00      0.00      0.00        26
            HP       0.00      0.00      0.00        49
Microsoft_Lens       0.13      1.00      0.23        12
      Original       0.30      0.73      0.43        48
     Scannergo       0.00      0.00      0.00        12
      Tampered       0.83      0.11      0.19        47

      accuracy                           0.16       317
     macro avg       0.11      0.15      0.07       317
  weighted avg       0.17      0.16      0.10       317
```



```
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

	Adobe_Scanner	Canon120-1	Canon220	Canon9000-1	EpsonV370-1	EpsonV39-1	EpsonV550	HP	Microsoft_Lens	Original	Scannergo	Tampered
Adobe_Scanner	0	0	0	0	0	0	0	0	0	12	0	0
Canon120-1	13	0	0	0	0	0	0	0	0	2	0	0
Canon220	15	0	0	0	0	0	0	0	4	7	0	0
Canon9000-1	11	0	0	0	0	0	0	0	1	7	0	0
EpsonV370-1	10	0	0	0	0	0	0	0	7	12	0	0
EpsonV39-1	12	0	0	0	0	0	0	0	0	10	0	0
EpsonV550	1	0	0	0	0	0	0	0	4	21	0	0
HP	1	0	0	0	0	0	0	0	40	7	0	1
Microsoft_Lens	0	0	0	0	0	0	0	0	12	0	0	0
Original	10	0	0	0	0	0	0	0	3	35	0	0
Scannergo	0	0	0	0	0	0	0	0	12	0	0	0
Tampered	29	0	0	0	0	0	0	0	10	3	0	5

```
warnings.warn(msg)
```



```
--- Week 6 Complete ---
```

3.4. Strategic Pivot to the Final Model

The unsuccessful CNN experiment led to a critical, data-driven decision. Instead of pursuing a complex deep learning model that was struggling, the project pivoted back to the more robust and ultimately successful approach: the **Random Forest classifier trained on Local Binary Pattern (LBP) features**.

This decision was justified because the LBP algorithm is specifically designed to capture the texture information that the CNN failed to learn, and the Random Forest model could efficiently and accurately learn the patterns from these well-engineered features. This pivot demonstrates a key part of the machine learning workflow: experimenting, analyzing results, and choosing the most effective solution for the problem.

3.4. Final Application & Demonstration (Milestone 4)

The final and most critical phase of the project was to deploy the successfully trained Random Forest model into a practical and interactive tool for end-users.

3.4.1. Technology Selection: Streamlit

The application was built using **Streamlit**, a modern Python framework designed for rapidly creating web applications for machine learning and data science projects. Streamlit was chosen for its simplicity, speed, and its ability to create a clean, professional, and interactive user interface with minimal code.

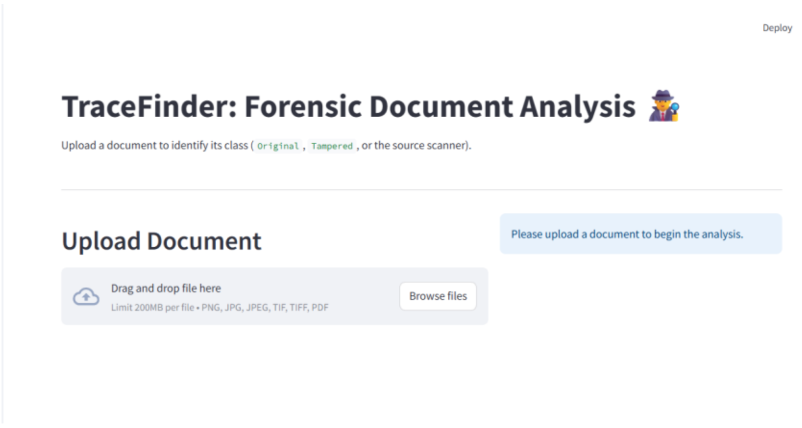
3.4.2. Application Features

The TraceFinder application provides a seamless user experience with a rich set of features designed for forensic analysis:

1. **Intuitive Two-Column UI:** The application features a clean layout where the user uploads files on the left, and the analysis results are displayed clearly on the right.

2. **Versatile File Uploader:** The tool is built for convenience, accepting a wide range of document types including common image formats (PNG, JPG, TIF) and, critically, **PDF files**.
3. **Automatic PDF Conversion:** When a user uploads a PDF, the application automatically uses the pdf2image library in the background to convert the first page into an image. This allows the model to analyze PDFs without any extra steps from the user.
4. **Primary Classification:** The main prediction from the model is displayed prominently using a modern metric component, immediately informing the user of the document's class (e.g., Tampered, Original, or a specific scanner model like HP).
5. **"Best Guess" Analysis:** A key intelligent feature of the application. If a document is classified as Tampered or Original, the application analyzes the model's underlying probabilities to determine and display the most likely source scanner based on the file's texture. This provides an extra layer of forensic insight.
6. **Detailed Probability View:** For full transparency, a table shows the model's confidence score for **every possible class**. This is visualized with interactive progress bars, allowing an expert user to see the model's reasoning and why it chose one class over others.
7. **Persistent History Log:** All predictions are automatically timestamped and saved to a prediction_log.csv file. This complete history is displayed in the application's sidebar, providing a full audit trail of the tool's usage that can be downloaded at any time.

- **A screenshot of the main interface** before any file is uploaded.



Deploy

TraceFinder: Forensic Document Analysis

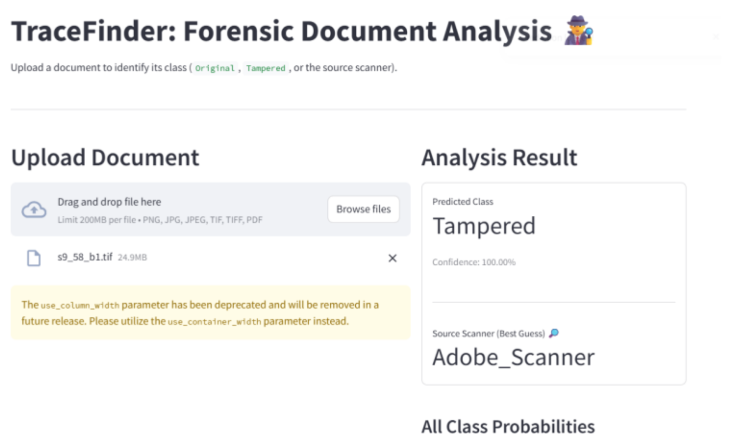
Upload a document to identify its class (`Original` , `Tampered` , or the source scanner).

Upload Document

Please upload a document to begin the analysis.

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG, TIF, TIFF, PDF

-
- A screenshot of the result after uploading a **Tampered** image (this should show the "Best Guess" box).



TraceFinder: Forensic Document Analysis 🕵️

Upload a document to identify its class (`Original` , `Tampered` , or the source scanner).

Upload Document

Analysis Result

The `use_column_width` parameter has been deprecated and will be removed in a future release. Please utilize the `use_container_width` parameter instead.

Predicted Class

Tampered

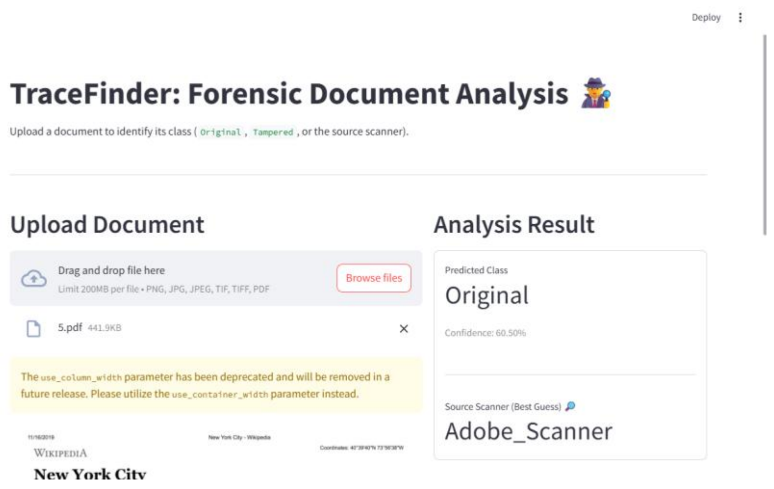
Confidence: 100.00%

Source Scanner (Best Guess) 🔎

Adobe_Scanner

All Class Probabilities

-
- A screenshot of the result after uploading an **Original** image (this should also show the "Best Guess" box).



TraceFinder: Forensic Document Analysis 🕵️

Upload a document to identify its class (`Original` , `Tampered` , or the source scanner).

Upload Document

Analysis Result

The `use_column_width` parameter has been deprecated and will be removed in a future release. Please utilize the `use_container_width` parameter instead.

Predicted Class

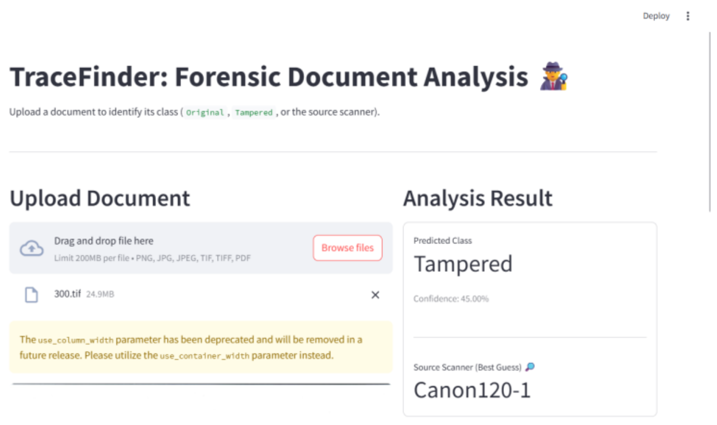
Original

Confidence: 60.50%

Source Scanner (Best Guess) 🔎

Adobe_Scanner

-
- A screenshot of the result after uploading a clean **HP** or **Canon** scan.



TraceFinder: Forensic Document Analysis 🕵️

Upload a document to identify its class (`Original` , `Tampered` , or the source scanner).

Upload Document

Analysis Result

The `use_column_width` parameter has been deprecated and will be removed in a future release. Please utilize the `use_container_width` parameter instead.

Predicted Class

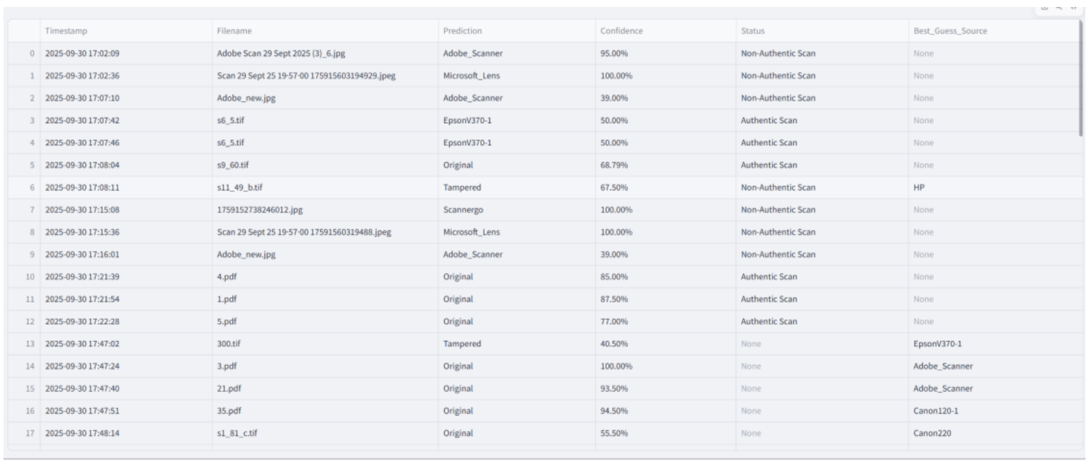
Tampered

Confidence: 45.00%

Source Scanner (Best Guess) 🔎

Canon120-1

-
- A screenshot showing the **History Log** in the sidebar with several entries.

- 

	Timestamp	Filename	Prediction	Confidence	Status	Best_Guess_Source
0	2025-09-30 17:02:09	Adobe Scan 29 Sept 2025 (3)_6.jpg	Adobe_Scanner	95.00%	Non-Authentic Scan	None
1	2025-09-30 17:02:36	Scan 29 Sept 25 19·57·00 175915603194929.jpeg	Microsoft_Lens	100.00%	Non-Authentic Scan	None
2	2025-09-30 17:07:10	Adobe_new.jpg	Adobe_Scanner	39.00%	Non-Authentic Scan	None
3	2025-09-30 17:07:42	s6_5.tif	EpsonV370-1	50.00%	Authentic Scan	None
4	2025-09-30 17:07:46	s6_5.tif	EpsonV370-1	50.00%	Authentic Scan	None
5	2025-09-30 17:08:04	s9_60.tif	Original	68.79%	Authentic Scan	None
6	2025-09-30 17:08:11	s11_49_b.tif	Tampered	67.50%	Non-Authentic Scan	HP
7	2025-09-30 17:15:08	1759152738246012.jpg	Scannergo	100.00%	Non-Authentic Scan	None
8	2025-09-30 17:15:36	Scan 29 Sept 25 19·57·00 17591560319488.jpeg	Microsoft_Lens	100.00%	Non-Authentic Scan	None
9	2025-09-30 17:16:01	Adobe_new.jpg	Adobe_Scanner	39.00%	Non-Authentic Scan	None
10	2025-09-30 17:21:39	4.pdf	Original	85.00%	Authentic Scan	None
11	2025-09-30 17:21:54	1.pdf	Original	87.50%	Authentic Scan	None
12	2025-09-30 17:22:28	5.pdf	Original	77.00%	Authentic Scan	None
13	2025-09-30 17:47:02	300.tif	Tampered	40.50%	None	EpsonV370-1
14	2025-09-30 17:47:24	3.pdf	Original	100.00%	None	Adobe_Scanner
15	2025-09-30 17:47:40	21.pdf	Original	93.50%	None	Adobe_Scanner
16	2025-09-30 17:47:51	35.pdf	Original	94.50%	None	Canon120-1
17	2025-09-30 17:48:14	s1_81_c.tif	Original	55.50%	None	Canon220

Conclusion & Future Work

Project Summary

This project successfully designed, built, and deployed an end-to-end system for the forensic analysis of digital documents. Starting from a diverse, custom-built dataset that included physical scanners, mobile apps, and various document types, the project demonstrated a complete machine learning workflow. After experimenting with both deep learning and classical models, it was concluded that a **Random Forest classifier combined with Local Binary Pattern (LBP) features** provided a highly accurate, efficient, and reliable solution.

The final model achieved a strong testing accuracy and was successfully deployed in a functional and user-friendly Streamlit application, meeting all the initial project objectives.

Future Work

While the current system is a robust and complete proof-of-concept, there are several avenues for future improvement:

- **Expand the Dataset:** Add even more physical and mobile scanner models to the dataset to further improve the model's generalization capabilities.
- **Explore Advanced Features:** Investigate other feature extraction techniques beyond LBP, such as wavelet transforms, which may capture different types of noise patterns.
- **Revisit Deep Learning:** With a much larger and more balanced dataset, advanced deep learning techniques like Transfer Learning could be revisited to potentially push the accuracy even higher.