FAKE IMAGE DETECTION SYSTEM

A PROJECT REPORT

OF PROJECT-III (PROJ-CS881)

BACHELOR OF TECHNOLOGY

in

Information Technology

(From **Maulana Abul Kalam Azad University of Technology, West Bengal**)

SUBMITTED BY

ARPAN HALDER (13000222129)

AJOY MONDAL (3000222124)

LAXMI SINGH (13001621076)

ANISH DAS (13000321014)

ANKITA CHOWDHURY (13000321060)

Department of Information Technology

Techno Main Salt Lake

Kolkata - 700091

Department of Information Technology

Techno Main Salt Lake

EM-4/1, Salt Lake City, Kolkata - 700091

BONAFIDE CERTIFICATE

Certified that this report for the project titled "FAKE IMAGE DETECTION SYSTEM" is a part of the final year project work being carried out by "ARPAN HALDER, AJOY MONDAL, LAXMI SINGH, ANISH DAS, ANKITA CHOWDHURY" as partial fulfillment for the degree of Bachelor of Technology in Information Technology, Maulana Abul Kalam University of Technology, West Bengal, under my supervision.

Full Signature of the Candidates (with date)

1. _____

2. _____

3. _____

4. _____

5. _____

_____

(Signature of the Mentor)

_____

(Signature of the Head of the Department)

ACKNOWLEDGEMENT

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to the teaching fraternity of the Department of Information Technology, for giving us this opportunity to undertake this project and also supporting us whole heartedly. We also wish to express our gratitude to the HOD and all our teachers of the Department of Information Technology for their kindhearted support, guidance and utmost endeavor to groom and develop our academic skills. At the end we would like to express our sincere thanks to all our friends and others who helped us directly or indirectly during the effort in shaping this concept till now.

Full Signature of the Candidates (with date)

1. _____

2. _____

3. _____

4. _____

5. _____

ABSTRACT

The increasing prevalence of fake images in digital media has led to increasing concerns over authenticity and trust. As a result, there is a need for systems that automatically distinguish fake images from real ones. To address this, we created two models: one for detecting tampered images and the other for identifying AI-Generated images. In the first model of our project, we deal with the issue of identifying fake images using a CNN model trained on error level analysis (ELA)-transformed images. CASIA2 and the Kaggle Real and Fake Face Detection datasets have been used for our study. In this work, we use ELA to identify tampering inconsistencies in the images that can be used as a preprocessing step to obtain meaningful features. The CNN architecture includes two convolutional layers with 32 filters, a 5x5 kernel, a dense layer with 256 neurons, and dropout rates of 25% and 50% to prevent overfitting. It uses the Adam optimizer (learning rate: 0.001), binary cross-entropy loss and it is trained over 20 epochs. The model achieved 94.87% training accuracy and 88.22% validation accuracy. We have chosen ELA for tampered images because tampering usually leaves behind some clues like compression issues,weird edges and unnatural lines at those portion where edits were made.These clues are better spotted using ELA as it highlights differences in comparison.So, for detecting tampered images, we focused on ELA because it is more effective in finding those edit signs.

Our second model focuses on identifying AI-generated images,which are nowadays increasingly produced using advanced generative models. We use a hybrid deep learning approach for distinguishing real images from AI-generated ones by combining Error Level Analysis (ELA) and Photo-Response Non-Uniformity (PRNU) noise features. Hugging Face AI-Generated and real images and Kaggle AI Generated Images dataset have been used for our study.The CNN architecture mirrors the first model but uses categorical cross entropy as the loss function,it is optimized with a lower learning rate(0.0001) and it is also trained over 20 epochs. The model achieved 99.44% training accuracy and 95.01% validation accuracy. This work justifies itself as a contribution toward image forensics and deepfake detection while representing a reliable, scalable, and interpretable solution for abuse detection in real-life.

KEYWORDS : Fake Image Detection, Convolutional Neural Network, Error Level Analysis (ELA), Photo-Response Non-Uniformity (PRNU), Real and Fake Face Detection, Image Classification, Deep Learning, Dataset Analysis, Binary Classification.

Table of Contents

List of Tables

List of Figures

CHAPTER - I
INTRODUCTION

1 INTRODUCTION

In the era of digital multimedia,the problem of authenticating the visual data has become challenging. With the sophistication and accessibility of image editing tools on the rise, and the advent of generative models such as Generative Adversarial Networks (GANs), telling apart real images from manipulated or synthetic ones has become increasingly challenging. This presents a serious threat to the integrity of information, as well as both personal and public security. Fake or altered images could be employed to disseminate disinformation, defame individuals or deceive law enforcements, demanding for robust image forensics mechanisms to detect fabrication or forgery. It is this challenge we seek to address as a project with our Fake Image Detection System to determine if an image is altered or artificially generated through AI.

Digital image itself is essentially a 2D array of small discrete elements called pixels (shorted picture elements). Individual pixels correspond to a point in the image, and hold numerical values indicating its color and intensity. In color images, those values are commonly divided into three color channels (Red, Green, and Blue (RGB)). For instance, in an 8-bit image, each channel has a range of 0–255, leading to over 16 million possible color combinations. These are the pixel values that contain all the data to be utilized by image processing/analysis. Patterns and anomalies can be learned, while the statistics of the pixels can be analyzed and used to classify image content.

To read pixel information for code logic, options would be OpenCV and PIL (Python Imaging Library). These provide direct access to pixel aliquots to perform deep image analysis. In our project, this capability to recapture and process pixel-level information underlines the importance of making decisions about anomalies that identify a manipulation or a synthetic generation.

Machine learning, a branch of artificial intelligence, involves creating algorithms that enable computers to learn from data and make predictions or decisions. Overall the typical workflow is composed of three steps: data preprocessing, model training and evaluation. The system is trained on some dataset with input-output pairs (i.e., images that are marked “real” or “fake”), which it uses to learn discriminative patterns. Traditional machine learning methods such as Support Vector Machines (SVM) and Decision Trees usually need to extract features by hand, and this process is laborious and not ideal for complicate data including images.

Deep learning, an advanced type of machine learning, employs neural networks with numerous layers to automatically learn features and capture complex data relationships. It is especially ideal for image classification applications. Among the popular deep learning models for image recognition is the CNN. CNN are structured to capture the spatial hierarchies in input data with the help of convolution

layers that locally filter the input image to identify edges, shapes, and textures. These are then followed by pooling layers that downsample the dimensions, dropout layers which are used to prevent overfitting due to randomly skipping cumulating or deactivating the neurons, and the dense (or fully connected) layers that accept as input the features extracted by the preceding layers to make the last prediction.

We devised in this work a hologram-based system for detecting fake images, based on hybridization of traditional image processing and deep learning approaches. We first use the preprocessing method Error Level Analysis (ELA). ELA works by saving the image at a known compression level (e.g., 95%) and subtracting the original and the compressed image pixel by pixel. The concept is that parts of an image that have not been altered should have a consistently levelled look, since the image has been compressed only once, and parts of the image that have been manipulated (and therefore recompressed) will have a different level of error. This difference is represented by a new image where changed areas are easily distinguished (they often are much brighter). This ELA-processed image is resized, normalized, and forwarded to our CNN model.

CNN is trained with mixed images of real and fake. Data with fake images include images manipulated with image-editing tools (e.g., splicing, copy-move, blurring, etc.) or generated by AI-based methods such as GANs or diffusion models. The network is able to self learn to perceive marginal discrepancies and artifacts, which are the footprint of tampering or fabrication. Post-training, the system is capable of confidently categorizing new supplied input images as being “real” or “fake”.

We are specifically focused on scaling the system up so that it can identify images produced by AI, which have been harder to identify because they are so photorealistic. These generated images are usually clean and do not include noise patterns and compression artifacts observed in real images. To mitigate this problem, we exploit pixel-wise feature extraction and deep learning techniques, thereby capturing statistical and structural differences, which are not visually obvious. Additionally, the system architecture is modular such that other detection methods, including Photo Response Non-Uniformity (PRNU) analysis, could be integrated. PRNU is a camera sensor fingerprint, which is able to trace images to the physical camera or differentiate the real images and the synthesized images, which will help the system’s forensic robustness. The general processing pipeline of the proposed Fake Image Detection System is composed as follows: the input image is initially processed by Error Level Analysis in order to localise potential tampered areas. This transformed image is then resized to a common input scale as required for CNN processing. CNN design comprises convolutional layers for feature extraction, dropout and pooling layers for coping with overfitting and complexity and dense layers for image classification. The system produces a binary classification label—real or fake—along with the confidence. Performance is measured by the usual accuracy, precision and confusion matrices.

Our project makes use of the best features of traditional image processing and state-of-the-art deep learning to tackle the emerging problem of image manipulation and synthesis It is promising for the applications of social media verification, digital journalism, law enforcement and national security. As the digital world increasingly outpaces its own ability to create content, tools that understand and adapt to the visual language will become increasingly important to ensure that the communication remains authentic.

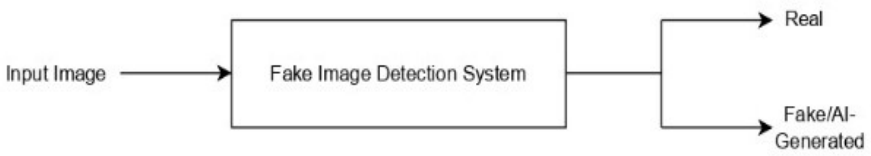


Figure 1: Fake Image Detection System

CHAPTER - II
LITERATURE REVIEW

2 LITERATURE REVIEW

The detection of manipulated or tampered images has been an important research topic in a forensic scenario because the digital technology is broadly spread and also, constantly growing and updating technology. With the explosion of visual content on social media, news outlets, and public repositories, it is increasingly difficult to verify the authenticity of a photo, particularly with realistic AI-generated images becoming more commonplace. Therefore, traditional image forensic methods and deep learning based detection models are demanded to be designed. "ELA" exploits the lossy nature of the typical image compression algorithm such as JPEG to detect inconsistencies in forged image reasonably good so far. ELA essentially works through a re-compression of an image and the analysis of pixel-by-pixel differences which occur between the original and the newly re-compressed data that's carried out in quite detail. Tampered sections often appear quite different to derived from unprocessed regions largely due to the irregular compression artefacts showing different error levels. (Idlbek et al made a nominal attempt to do some studies. [11] have covered ELA-based techniques in digital image forensics and studied the approach to incorporate this with AI techniques and machine learning algorithms to efficiently visualize the tampered areas. Idlbek2024 highlighted the importance of ELA-techniques in digital image forensics and successfully showed its combination with AI and machine learning methods for the visualization of manipulated areas. [10], where the MobileNet deep neural network was cascaded with ELA preprocessing. This approach was significantly improved later by Shah et al. Low-level image cues and hand-crafted statistical features have also been employed in classic forensic analysis of digital images. An extensive investigation of digital forensics techniques including copy-move forgery detection, resampling detection, and JPEG quantization anomalies was presented by Wilson and Davis[2]. They achieved success in merging supervised classifiers and image statistics. A machine learning methodology for image manipulation detection based on a decision tree, and on the one-against-all SVM was introduced by Rossi and Bianchi[1]. They showed that these models can work well when applied to expertly selected forensic features. With advancement in image manipulation methods, researchers began to explore deep learning for automated feature extraction and classification, with convolutional neural networks (CNNs) in particular. As opposed to handcrafted features, Johnson and Smith[7] applied CNNs to learn hierarchical image features in pixel space. Related to this Doe and Roe[3] built a CNN model for Deepfakes detection which was highly effective in identifying facial anomalies difficult for the human eye to detect including asymmetry or peculiar texture. It's become increasingly difficult to tell artificial intelligence-generated images apart since the rise of generative models like GANs. These models employ adversarial training to estimate data distributions and generate realistic images. Focusing on frequency features that GANs are likely to ignore, Wang and Zhang[6] proposed a GAN-based fake image detection scheme

which utilizes supervised learning to classify the type of image as real or synthetic. Patel and Shah [4] classified deepfake detection approaches into three types: face based, behavior based, and content based in their survey. Their study provides a snapshot of the challenge to make detection models generalize across different types of GAN architectures. Recent researches come out with statistical/spatial domain analysis to solve the problem of discriminating the AI-generated images. Martin-Rodriguez et al. introduced a hybrid structure combining CNN based classification and pixel-wise feature extraction. [9]. They look at the pixel-level statistics, which often reveal artifacts from the synthetic generation process. The model effectively discriminates between synthetic image patterns (often being uniform and lacking unpredictable properties) and natural image noise (usually being irregular and sensor specific). Another direction is to investigate the application of adversarial robustness and ensemble method. For instance, Zhang and Chen[8] presented a method that aggregates the predictions of several base detectors to increase robustness to different manipulation methods. In another attempt to help models generalize more effectively in adversarial scenarios where fake images are active, Smith and Jones [5] studied adversarial image detection by training models to detect on a very mild pattern imposed on the fake image creation process. In addition, the authenticity of images can be determined using sensor-based forensics methods, such as Photo-Response Non-Uniformity (PRNU) analysis. On the other hand, a sensor array in the digital camera produces a unique fingerprint, called PRNU. It can verify whether an image has been modified, or is from a specific device. If there are no papers exactly about PRNU in your current bibliography, one thing to keep in mind is that there is more recent work exploring using PRNU with CNNs to defend against AI-generated images, which tend avoid including sensor noise profiles. When taken as a whole, the reviewed literature shows a progression from manually designed statistical methods to hybrid deep learning models, each of which enhances the ability to identify ever-more-complex manipulations. Our system combines the best features of both conventional and contemporary methods by utilizing CNN-based classification for tampered images and ELA preprocessing, and pixel-wise statistical modeling for AI-generated image detection. By addressing the expanding range of digital image forgery techniques in modern media, this dual-model approach guarantees improved detection accuracy and robustness.

CHAPTER - III
PROBLEM DEFINITION

3 PROBLEM DEFINITION

With the rapid development of DeepFakes and other GAN-based techniques, an enormous number of morphed and AI-generated images have emerged, and the authenticity of digital visual content in the current society has been severely called into question [4, 6]. Distinguishing such manipulated or synthetically generated images has therefore become more and more important to preserve the truthfulness and trustworthiness of images in different areas including social media, journalism, or digital forensics [2, 7].

Such elaborate forgeries, whether they involve doctored real images or entirely fabricated ones stitched together by A.I. are difficult to catch using traditional fraud-detection techniques. Techniques of early detection often use outdated approaches or have been developed using small datasets that do not represent the diversity and complexity of image manipulations used in current applications. Secondly, the AI-generated images, especially those generated by GANs, usually have some subtle artifacts and patterns invisible to the human eye, which can confuse the traditional detection tools [5, 8].

The most difficult part is to build a more sophisticated and effective mechanism, which is able to capture subtle deviations between the real images and recognize convincing results from the synthetic images. Existing methods for image verification have not been able to effectively cope with these newer manipulation paradigms. Conventional methods are unable to offer such flexibility and adaptability to face with the constantly increasing changing nature of fake content [3]. What's more, many don't provide real-time performance or scale, making them too slow for today's fast-moving digital platforms.

Therefore, it is urgent to develop the dual-role system that can effectively handle both tampered image forgery and AI-based image with so high accuracy and robustness. Taking advantage of both forensic tools such as Error Level Analysis (ELA) and deep learning architectures is crucial for these purposes [9, 10, 11].

CHAPTER - IV
MODULES AND FUNCTIONALITIES

4 MODULES AND FUNCTIONALITIES

This project consists of two essential subsystems: 1. Fake Image Detection Model (Forgery Detection) 2. AI-Generated Image Detection Model(Synthetic Image Detection)

All subsystems are trained on different datasets with different goals. The Functions are grouped into the following modules:

4.1 Fake Image Detection Model

- **Dataset Preparation Module**
  - Loads the CASIA2 dataset which has 2 folders, one real and one fake (created by copy-move, splicing, etc.).

    Data are shuffled to prevent learning from the order.
  - Splits the data 80:20 into train-validation.
  - **Significance**: Balanced dataset in proper arrangement would not let the model dominate on one class and help the network to generalise better.

- **Error Level Analysis (ELA) Preprocessing Module**
  - Saves the original image at a specified JPEG quality (e.g., 90%).
  - Reopens the compressed image and calculates the difference using ImageChops.difference().
  - Difference is usually very subtle, hence brightness is amplified with ImageEnhance.Brightness() to enlarge pixel-level differences.
  - Final ELA image is saved and utilized in feature extraction.
  - **Significance:**Modified domains undergo contraction, unlike unmodified regions. It is significant to have the ELA to bring such changes to the surface, especially to identify pixel manipulation, splicing, or retouching.

- **Image Normalization and Formatting Module**
  - ELA images are reduced to a size of 128×128 pixels.
  - Transforms images to numpy arrays for numerical processing.
  - Scales pixels values (0–255) to the range [0, 1] in order for the model to learn.

- One-hot encode the class labels ([1, 0] for fake, [0, 1] for real).
- **Importance:** ensures no shape mismatch errors and uniform input distribution for faster convergence and better model's performance.

- **CNN Training Module**
  - Conv2D Layer 1: 32 filters, kernel size: (3x3), activation: ReLU.
  - MaxPooling2D: Downsample to reduce spatial dimension.
  - Dropout: We use 25% dropout for regularization.
  - Conv2D Layer 2 - 64 filters identical as the above.
  - Flatten: Flatten the 2D features to 1D.
  - Dense layer containing 128 units with activation ReLU.
  - Dropout: 50%, for regularization.
  - Output Layer: 2 neurons (softmax) due to binary classification.
  - Optimizer: Adam
  - Loss Function: Binary Crossentropy
  - Objetive: Accuracy
  - EarlyStopping: Stop training if validation accuracy does not improve for 5 (out of 7) epochs.
  - **Relevance:** The CNN extracts local and global features that are very effective in detecting image manipulations and are invisible to a naked human eye.

- **Evaluation ,Visualization and Confusion Matrix Module**
  - Measure how well the model has been trained and it's behavior.
  - Plot the graph between Training vsValidation Accuracy and Training vs Validation Loss
  - Is used to recognize issues like overfitting or underfitting.
  - **Importance:** Visualization offers an interpretable view of the model learning dynamics and helps diagnose performance bottlenecks.
  - Measures for Confusion Matrix:
  - True Positives (TP): Real image is classified as real.
  - True Negatives (TN): Fake image and is predicted to be fake.
  - False Positives (FP): Synthetic image classified as real.
  - False Negatives (FN): A real image that was classified to be generated.

4.2 AI Generated Image Detection Model

- **GAN-Based Dataset Preparation Module**
  - Collection and tagging of the data generated using AI techniques (GANs).
  - Data Sources: We have merged the AI-Generated vs.Real Images Datasets by Hemg of the Hugging Face and cardbowman of the Kaggle AI-Generated Images vs.Real Images dataset to obtain AI-generated (fake) images and real images from the CASIA2 dataset.
  - Labeling: AI-generated: 0 Real: 1
  - Preprocessing: Resizing and renaming the dataset, and balancing it for the same incidence of classes.
  - **Significance:** No editing of GAN images, which are instead generated from scratch. Separating them requires learning to detect far different features than those used to create fake images by editing real ones.
- **PRNU (Photo Response Non-Uniformity) Feature Extraction Module**
  - PRNU is a unique noise texture of all physical camera sensors.
  - AI-generated images are digital; therefore they do not have real PRNU noise.
  - The PRNU noise residuals are estimates by filtering input image with wavelet or BM3D filters and then extracting the noise residual.
  - These residual noises are then applied as features to help CNN model to recognize the inconsistencies of images generated by AI.
  - **Importance:** PRNU-based analysis is a powerful forensic method to perform camera source identification. For the problem of AI image detection, it provides a strong evidence that the AI images don't have this natural sensor noise, which helps to improve detection result.
- **ELA Preprocessing Module**
  - Transform images into ELA format to bring out compression artifacts.
  - Similar to the forgery detection model, it compresses the images and computes the pixel-wise difference.
  - Emphasises inconsistency in image due to compression.
  - **Significance:**The CNN has learned how to see small, pixel-level illusions that seem to appear in Ai-generated images.

- **Image Normalization and Formatting Module**
  - Objective: Prepare PRNU and ELA images for the input of our proposed model.
  - Rescale the size of the (both ELA images and PRNU residual maps) to a set resolution (e.g., 128×128).
  - Also, Rescale the pixel values to between 0 and 1.
  - Turn data into 4D tensors for Keras CNN input.
  - **Significance:** Gives consistent input format and uniform scaling of input data to keep the training stable and optimal.
- **Dual-Input CNN Training Module**
  - Objective: Train a CNN model with parallel inputs.
  - ELA images for depiction ofcompression artefacts.
  - PRNU residuals maps are used to describe the sensor noise pattern.
  - Architecture:
  - Two parallel CNN branches analyzing ELA images and PRNU maps.
  - he features from the two branches are concatenated and fed into fully connected layers.
  - Output layer for final prediction of binary class: AI-generated or Real.
  - Loss function: Binary cross-entropy.
  - Optimizer: Adam.
  - Early stopping and dropout were applied to prevent overfitting.
  - **Significance:** Models the fine grain discriminative diagnosis between natural and synthesized images, better than single-input models.
- **Performance Evaluation and Visualization Module**
  - Purpose:Assessing the learnability and classifiability of the model.
  - Outputs:
  - Graphs for accuracy and loss per epochs.
  - Confusion matrix consisting of true positives, false positives, true negatives and false negatives.
  - **Significance:** Can help understand the weaknesses and strengths of the model which can be favorable for adjustments.

CHAPTER - V
WORKFLOW DIAGRAM

5 WORKFLOW DIAGRAM

5.1 Fake Image Detection Model - Training Phase

The actual training starts with the compilation of a data set that consists of real images and fake images. Each image is converted into an ELA format, thereby enabling the model to selectively attend to compression distortion that is important for detecting manipulated images. The images are scaled down to 128 × 128 pixels, to preserve sustained input dimensions for the model required. Due to resizing, the images are then recomposed into one-dimensional for easy processing. The obtained pictures are then normalized to the range between 0 and 1 by dividing the intensities of each pixel by 255.0. The pre-processed images are then attached after the feature array (X) and the associated label is then placed in the target array (Y). Next, the data gets transformed to the shape of (- 1,128,128,3) which is the same as the input shape needed according to the Convolutional Neural Network (CNN). The dataset is then divided into the training and validation sets. By default, models created by Keras are fitted using specific metrics as a measure of the model's performance during training. CNN architecture involves two convolutional layers, a maximum pooling layer, a dropout layer with a rate of 25% to inflate overfitting, and feed-forward layers.

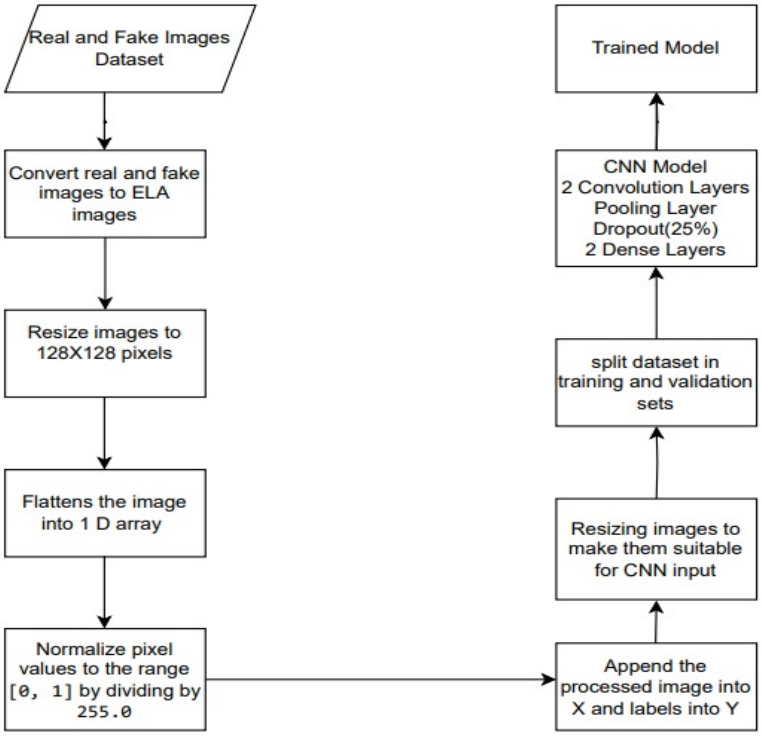


Figure 2: Training Phase Workflow - Fake image detection model

5.2 Fake Image Detection Model - Testing Phase Workflow

The testing phase starts with a sample input image. This is mostly done using Error Level Analysis (ELA). This step also helps in converting the input image into the ELA image, to differentiate any odd one out that may occur by editing or compression. The ELA image is then resized to a particular standardized dimension of 128 by 128 pixels to facilitate putting into conformity all input sources so that they are of a standard level. The pixel values of the resized images are normalized by dividing the pixel value by 255.0. The normalization will take care of the compatibility needs of the system. From there, we can derive specific information about the model's input requirements. The step after this is to load the pre-trained model from a file that has been created previously. Last but not least the trained model takes the input image and says whether it is fake or real. Along with each prediction, the model gives a confidence score also.

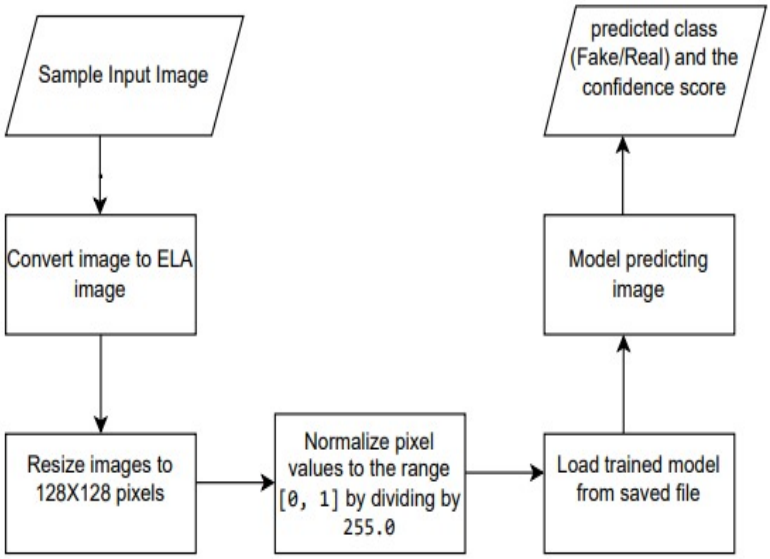


Figure 3: Testing Phase Workflow - Fake image detection model

5.3 AI Generated image Detection Model - Training Phase

The training phase of our hybrid Fake Image Detection System is designed to identify both manipulated and AI-generated images. This system uses a dual-input approach that combines traditional forensic analysis with modern deep learning techniques to improve classification accuracy and strength.

Each image in the training dataset is converted into two main representations:

1. ELA (Error Level Analysis) Images: These images are created by compressing the original image and calculating the pixel-level difference between the original and compressed versions. This process takes advantage of lossy compression. Unedited images show consistent error patterns, while tampered areas display higher error levels. The ELA transformation visually emphasizes these inconsistencies, often appearing lighter in the altered regions. This feature is important for detecting subtle changes that are not visible to the naked eye.

2. PRNU (Photo-Response Non-Uniformity) Features: PRNU is a unique noise pattern from each image-capturing device, caused by small manufacturing imperfections. These tiny pixel-level variations act as a forensic fingerprint for real images. In contrast, AI-generated images usually do not have real PRNU noise. By extracting PRNU signatures, we capture key features that help distinguish real sensor-captured images from those created by algorithms.

These two feature streams are processed in parallel with two separate neural pathways:

CNN Branch (for ELA Images): The ELA images go into a Convolutional Neural Network (CNN), which includes multiple convolutional layers followed by max-pooling layers. The convolutional layers extract spatial features such as edges, boundaries, and local inconsistencies that indicate tampering. Max-pooling layers reduce the size of the feature maps, maintaining important information while simplifying the data. The resulting high-level feature maps are then flattened into a one-dimensional vector.

PRNU Branch (for PRNU Vectors): At the same time, the PRNU vectors pass through a dense neural network made of fully connected layers. These layers capture statistical patterns and subtle changes in the noise signature. Dropout layers are used to avoid overfitting by randomly deactivating neurons during training. This approach encourages the model to learn more generalized representations.

Feature Fusion and Final Classification: The output feature vectors from both branches are combined into a single feature vector. This composite representation, which includes both spatial and sensor-level information, is then fed into a series of dense layers. These fully connected layers

learn the complex relationships between the extracted features. The final layer uses a `softmax` activation function to classify the image as either *real* or *fake*.

The model is compiled with the categorical cross-entropy loss function and optimised using the Adam optimiser (learning rate 0.0001), and it is trained over 20 epochs.

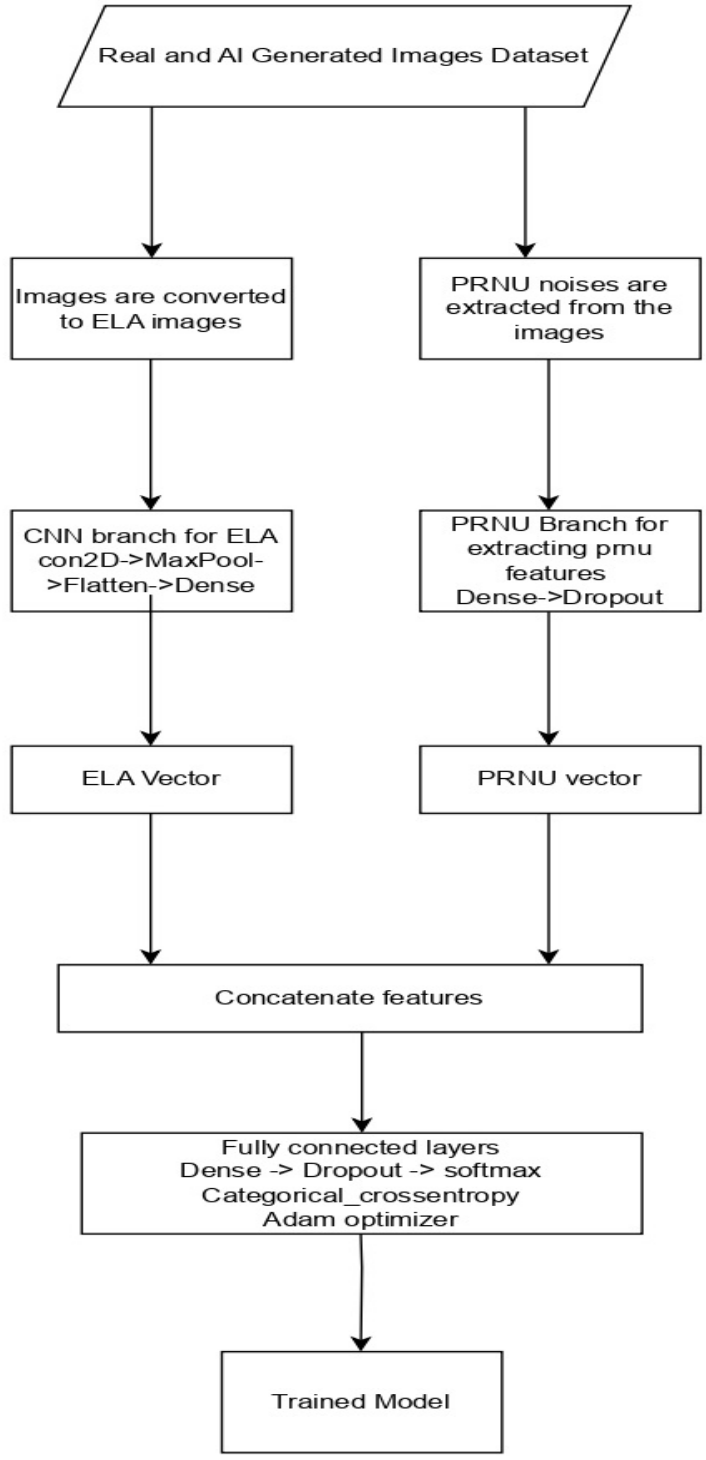


Figure 4: Training Phase Workflow - AI generated image detection model

5.4 AI generated image Detection Model - Testing Phase Workflow

The testing phase starts with a sample input image. In this phase, the model takes in two inputs : the ELA-transformed image of 128X3X3, which captures visual artefacts due to compression inconsistencies, and a five-dimensional PRNU feature vector that reflects sensor-level noise residuals. ELA-transformed images are fed into the CNN for spatial pattern learning and the PRNU feature vector into the dense PRNU branch for statistical feature interpretation. The outputs of both branches are concatenated, then it is passed through fully connected layers to generate a final prediction about the image, whether it is real or AI-Generated. Along with each prediction, the model gives a confidence score also.

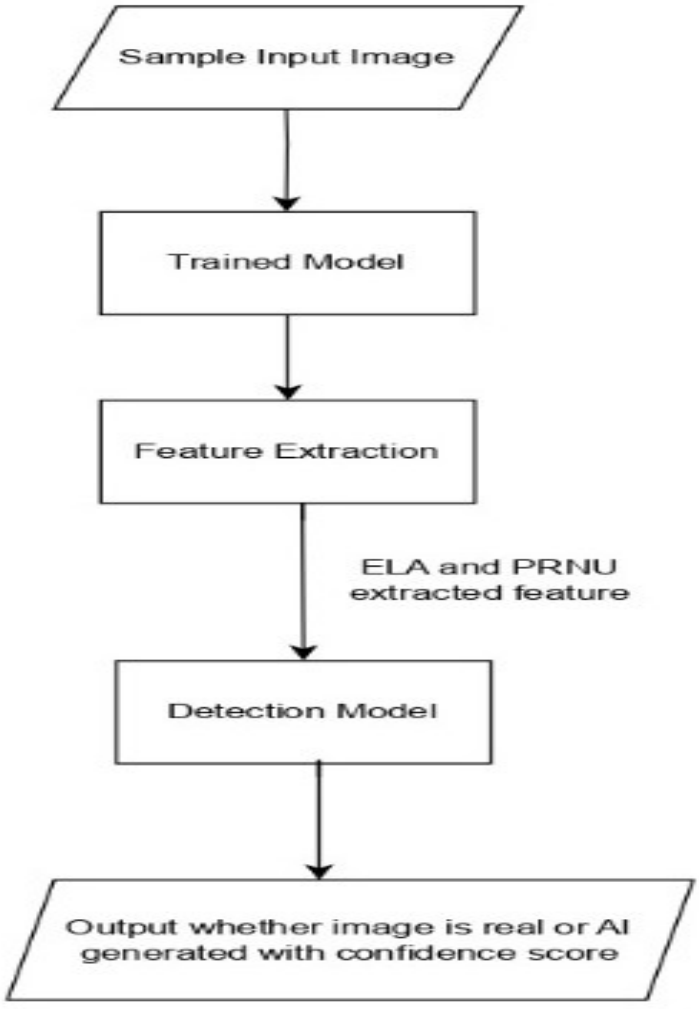


Figure 5: Testing Phase Workflow - AI generated image detection model

CHAPTER - VI

SOFTWARE AND HARDWARE REQUIREMENTS

6 SOFTWARE AND HARDWARE REQUIREMENTS

- **Required Hardware**
  - Windows-based system.
  - RAM upto 8GB.
  - CPU Speed more than 2.5 GHz.
  - GPU for acceleration of the model the training and for an effective execution of the image processing jobs.
- **Required Software**
  - Software and Tools: Python v3. 11 to implement the necessary code. It is a high-level, interpreted,interactive and object-oriented programming language.
  - Libraries for image processing - PIL
  - Libraries: TensorFlow, Keras, NumPy, Matplotlib, scikit-learn.
  - IDE to run the code : Jupyter Notebook/Visual StudioCode It is a web application that allows you to write and share documents that include live code, equations, visualizations and narrative text.
  - Kaggle and Huggingface for Dataset It is a community of data scientists and machine learners. It enables the user to discover and publish dataset, explore andbuild models in an online data science environment.
  - HTML, CSS, ReactJS The web interface is structured and styled using HTML and CSS. ReactJS, simply put, is a JavaScript library used to create dynamic user interfaces.
  - Flask 3.5.1.Flask Flask is a lightweight Web Server Gateway Interface (WSGI) web application framework of Python used to build the backend server and API endpoints for model interaction.

CHAPTER - VII
CODE TEMPLATES

7 CODE TEMPLATES

7.1 Fake Image Detection Model:

7.1.1 ELA Conversion

Functionality: It Converts an image to its Error Level Analysis (ELA) representation to highlight inconsistencies due to tampering.

Method: `convert_to_ela_image(path, quality)`

Input Parameters:

```
[3]: def convert_to_ela_image(path, quality):
        temp_filename = 'temp_file_name.jpg'
        ela_filename = 'temp_ela.png'

        image = Image.open(path).convert('RGB')
        image.save(temp_filename, 'JPEG', quality = quality)
        temp_image = Image.open(temp_filename)

        ela_image = ImageChops.difference(image, temp_image)

        extrema = ela_image.getextrema()
        max_diff = max([ex[1] for ex in extrema])
        if max_diff == 0:
            max_diff = 1
        scale = 255.0 / max_diff

        ela_image = ImageEnhance.Brightness(ela_image).enhance(scale)

        return ela_image
```

Figure 6: Figure for ELA Conversion

- `path` (*str*): File path of the input image.
- `quality` (*int*): JPEG quality for re-compression.

Output: Returns the ELA-transformed image as a PIL object.

7.1.2 Image Preparation

Functionality: Prepares images by resizing, normalizing, and converting to ELA representation for training/testing.

Method: `prepare_image(image_path)`

Input Parameters:

- `image_path` (*str*): Path to the image file.

```
[4]: image_size = (128, 128)

[5]: def prepare_image(image_path):
         return np.array(convert_to_ela_image(image_path, 90).resize(image_size)).flatten() / 255.0
```

Figure 7: Prepare Image

Output: Returns a flattened, normalized NumPy array representing the processed image.

7.1.3 Dataset Processing

Functionality: Reads images from the dataset directories, processes them using ELA, and assigns appropriate labels.

Methods:

- **Real Image Loader:** Iterates through the ”real” image folder, processes images, and appends them to the dataset.

```
[7]: import random
path = r'dataset\CASIA2\real'
for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        if filename.endswith('jpg') or filename.endswith('png'):
            full_path = os.path.join(dirname, filename)
            X.append(prepare_image(full_path))
            Y.append(1)
            if len(Y) % 500 == 0:
                print(f'Processing {len(Y)} images')

random.shuffle(X)
# X = X[:2100]
# Y = Y[:2100]
X=X
Y=Y
print(len(X), len(Y))
```

Figure 8: Process Real Image

- **Fake Image Loader:** Iterates through the ”fake” image folder, processes images, and appends them to the dataset.

7.1.4 CNN Model Builder

Functionality: Defines and compiles a CNN model for binary classification.

Method: `build_model()`

Input Parameters: None

Output: Returns a compiled Keras Sequential model with:

```
[8]: path = r'dataset\CASIA2\fake'
for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        if filename.endswith('jpg') or filename.endswith('png'):
            full_path = os.path.join(dirname, filename)
            X.append(prepare_image(full_path))
            Y.append(0)
            if len(Y) % 500 == 0:
                print(f'Processing {len(Y)} images')

print(len(X), len(Y))
```

Figure 9: Process Fake Image

```
[11]: def build_model():
    model = Sequential()
    model.add(Conv2D(filters = 32, kernel_size = (5, 5), padding = 'valid', activation = 'relu', input_shape = (128, 128, 3)))
    model.add(Conv2D(filters = 32, kernel_size = (5, 5), padding = 'valid', activation = 'relu', input_shape = (128, 128, 3)))
    model.add(MaxPool2D(pool_size = (2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(256, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation = 'softmax'))
    return model
```

Figure 10: CNN Model

- 2 Convolutional layers (32 filters each)
- 1 Max Pooling layer
- Dropout layers (to prevent overfitting)
- Fully connected Dense layers (including softmax for output).

7.1.5 Training and Evaluation

Functionality: Trains the model on the dataset and evaluates its performance.

Training:

Method: `model.fit(X_train, Y_train, ...)`

Trains the model using training data and validation data with early stopping to prevent overfitting.

Input Parameters:

- `X_train, Y_train` (training data and labels)
- `batch_size, epochs` (training configurations)
- `validation_data` (`X_val`, `Y_val`): Validation dataset.

Output: Training history object containing loss and accuracy values.

Evaluation:

```
[22]: epochs = 20
      batch_size = 32

[23]: init_lr = 1e-4
      optimizer = Adam(lr = init_lr, decay = init_lr/epochs)

[24]: model.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = ['accuracy'])

[25]: early_stopping = EarlyStopping(monitor = 'val_acc',
                                     min_delta = 0,
                                     patience = 2,
                                     verbose = 0,
                                     mode = 'auto')

[26]: hist = model.fit(X_train,
                       Y_train,
                       batch_size = batch_size,
                       epochs = epochs,
                      validation_data = (X_val, Y_val),
                      callbacks = [early_stopping])
```

Figure 11: Training the Model

Plots accuracy and loss graphs for training and validation data.

```
[21]: fig, ax = plt.subplots(2, 1)
      ax[0].plot(hist.history['loss'], color='b', label="Training loss")
      ax[0].plot(hist.history['val_loss'], color='r', label="Validation loss")
      ax[0].set_xlabel('Epochs')
      ax[0].set_ylabel('Loss')
      ax[0].legend(loc='best', shadow=True)
      ax[1].plot(hist.history['accuracy'], color='b', label="Training accuracy")
      ax[1].plot(hist.history['val_accuracy'], color='r', label="Validation accuracy")
      ax[1].set_xlabel('Epochs')
      ax[1].set_ylabel('Accuracy')
      ax[1].legend(loc='best', shadow=True)
      for axis in ax:
          axis.xaxis.set_major_locator(plt.MaxNLocator(integer=True))
      plt.tight_layout()
      plt.show()
```

Figure 12: Plotting accuracy and loss graph

7.1.6 Results Visualization

Functionality: Predicts the class of a batch of images and visualizes the results.

Method: `predict_image(model, image_path)`

Input Parameters:

- `model` (Keras model): Pre-trained CNN model.

– `image_path` (*str*): Path to the image file for prediction.

Output: Returns the predicted label (”Real” or ”Fake”) and the confidence score.

Method: `plot_confusion_matrix(cm, classes, ...)`

Input Parameters:

– `cm` (Confusion Matrix): Generated confusion matrix.

– `classes` (list): Labels for classes (e.g., [”Real”, ”Fake”]).

– `normalize` (bool): Whether to normalize values to percentages.

Output: Displays a heatmap of the confusion matrix.

```
[38]: import numpy as np
import matplotlib.pyplot as plt
import itertools
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix

model = load_model('model_casia_run3.h5')

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion Matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')

Y_pred = model.predict(X_val)
Y_pred_classes = np.argmax(Y_pred, axis=1)
Y_true = np.argmax(Y_val, axis=1)
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
plot_confusion_matrix(confusion_mtx, classes=["Real", "Fake"])
```

Figure 13: Plotting confusion matrix

7.2 AI Generated Image Detection Model:

7.2.1 ELA Conversion

Functionality: Transforms an image to its ELA representation to display compression artifacts for forgeries.

Method: `convert_to_ela_image(image_path, quality)`

This function saves the image at lower quality, and then the pixel-wise difference between the original image and the compressed picture is calculated to generate the ELA image.

Input Parameters:

- `image_path` (*str*): File path to the original image.
- `quality` (*int*): JPEG quality setting for compression.

Output: Returns the ELA image as a PIL Image object.

```
[6]: def convert_to_ela_image_and_prnu(path, quality):
         temp_filename = 'temp_file_name.jpg'
         ela_filename = 'temp_ela.png'

         image = Image.open(path).convert('RGB')
         image.save(temp_filename, 'JPEG', quality = quality)
         temp_image = Image.open(temp_filename)

         ela_image = ImageChops.difference(image, temp_image)

         extrema = ela_image.getextrema()
         max_diff = max([ex[1] for ex in extrema])
         if max_diff == 0:
             max_diff = 1
         scale = 255.0 / max_diff

         ela_image = ImageEnhance.Brightness(ela_image).enhance(scale)
         ela_image = ela_image.resize((128, 128))
         ela_array = np.array(ela_image) / 255.0
         prnu_noise = extract_prnu(path)
         prnu_vec = prnu_features(prnu_noise)
         return ela_array,prnu_vec
```

Figure 14: ELA Image Conversion Function

7.2.2 PRNU Extraction

Functionality: Extracts the PRNU (Photo Response Non-Uniformity) noise pattern from the input image.

Method: `extract_prnu(image_path)`

This operation performs wavelet based denoising and then subtracts this from the input grayscale

image to obtain the sensor noise(PRNU) part.

Input Parameters:

- `image_path` (*str*): Path of the image to process.

Output: Returns a 2D NumPy array with the PRNU noise pattern.

```
[4]: import cv2
from scipy.ndimage import gaussian_filter

def extract_prnu(image_path):
    """Extract PRNU noise from a grayscale image."""
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise ValueError(f"Image not found: {image_path}")
    img = img.astype(np.float32) / 255.0
    smooth = gaussian_filter(img, sigma=1)
    noise = img - smooth
    return noise
```

Figure 15: PRNU Extraction Function

7.2.3 Dataset Processing

Functionality: Takes images from the directories of both 'real' and 'AI-generated' and processes them using ELA manipulation and PRNU noise. The achieved features are then organized as dual input for training the dual-branch CNN model.

Method: `load_dataset(folder_path)`

Input Parameters:

- `folder_path` (*str*): Path to directory containing the main dataset, the 'real' and 'AI-generated' subdirectories.

Output: Three NumPy arrays:

- `X_ela` (*array*): ELA-transformed image tensors of shape (128×128×3).

- `X_prnu` (*array*): PRNU feature matrices of shape (128×128×1).
- `Y` (*array*): Corresponding class labels (0 for real, 1 for fake).

```
[14]: real_path = r'C:\Users\anish\OneDrive\Desktop\MyCodes\AI IMAGE DETECTION SYSTEM\CASIA2\real'
for dirname, _, filenames in os.walk(real_path):
    for filename in filenames:
        if filename.lower().endswith(('jpg', 'jpeg', 'png')):
            full_path = os.path.join(dirname, filename)
            try:
                ela_img, prnu_vec = prepare_image_and_prnu(full_path)
                X_ela.append(ela_img)
                X_prnu.append(prnu_vec)
                Y.append(1)
                if len(Y) % 500 == 0:
                    print(f'Processed {len(Y)} real images')
            except:
                print(f"Error processing: {full_path}")
```

Figure 16: Loading and Preparing Dataset for AI Image Detection

7.2.4 Model Architecture

Functionality: BConstructs a custom CNN composed of one ELA input branch and one PRNU noise input branch, and the activations from two branches are merged before final prediction.

Method: `build_dual_input_model()`

```
•[18]: from keras.models import Model
def build_combined_model():
    input_ela = Input(shape=(128, 128, 3), name='ela_input')
    x = Conv2D(32, (5, 5), padding='valid', activation='relu')(input_ela)
    x = Conv2D(32, (5, 5), padding='valid', activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.25)(x)
    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)

    input_prnu = Input(shape=(5,), name='prnu_input')
    y = Dense(32, activation='relu')(input_prnu)
    y = Dropout(0.3)(y)

    merged = concatenate([x, y])
    z = Dense(128, activation='relu')(merged)
    z = Dropout(0.4)(z)
    output = Dense(2, activation='softmax')(z)

    model = Model(inputs=[input_ela, input_prnu], outputs=output)
    model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

Figure 17: Dual-Input CNN Architecture

Input Parameters: None

Output: Compiled dual-input CNN model.

Architecture Highlights:

- Two separate CNN branches for ELA and PRNU
- Each branch includes Conv2D, MaxPooling, and Dropout
- Merged via concatenation
- Fully connected Dense layers for final classification

7.2.5 Training

Functionality: Trains the dual-input CNN model on preprocessed ELA and PRNU features and test it on validation data.

Training: The model is trained using the binary cross-entropy loss and Adam optimizer. Overfitting is avoided by using early stopping through validation loss. The training procedure is carried out through two NumPy arrays, one for ELA features and one for PRNU ones.

Method: `model.fit([X_ela_train, X_prnu_train], Y_train, ...)`

Input Parameters:

- `X_ela_train` (*array*): Training data from ELA.
- `X_prnu_train` (*array*): Training data from PRNU.
- `Y_train` (*array*): Training labels.
- `validation_data`: Tuple of validation inputs and labels.
- `batch_size, epochs`: Training configurations.

Output: Returns a training history object containing accuracy and loss over epochs.

```
[22]: model.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = ['accuracy'])

[23]: early_stopping = EarlyStopping(monitor = 'val_acc',
                                     min_delta = 0,
                                     patience = 2,
                                     verbose = 0,
                                     mode = 'auto')
      epochs = 20
      batch_size = 32

[24]: hist = model.fit(
          [X_ela_train, X_prnu_train],
          Y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=([X_ela_val, X_prnu_val], Y_val),
          callbacks=[early_stopping]
      )
```

Figure 18: Training the AI Image Detection Model

7.3 User Interface and model integration

Functionality: A Flask-based web app where the user can choose to upload an image and select in between two models the one for fake image and the other is the AI generated image spotter.

Process:

- Uploaded images are saved and preprocessed using ELA.
- EXIF metadata is extracted for display.
- Based on model selection, either `model_casia` or `model_ai` is used for prediction.
- Dummy metadata is passed alongside ELA input for the AI model.
- Prediction, confidence score, ELA image, and metadata are displayed to the user.

App1.py > preprocess_image

```python
from flask import Flask, render_template, request
import os
import secrets
from tensorflow.keras.models import load_model
from PIL import Image, ImageChops, ImageEnhance, ExifTags
import numpy as np

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'static/uploads'
app.config['ELA_FOLDER'] = 'static/ela'

model_casia = load_model('model/model_casia.h5')
model_ai = load_model('model/model_ai.h5')

def preprocess_image(image_path):
    image = Image.open(image_path).convert('RGB')
    temp_filename = os.path.join(app.config['UPLOAD_FOLDER'], 'temp_ela.jpg')
    image.save(temp_filename, 'JPEG', quality=90)
    temp_image = Image.open(temp_filename)

    ela_image = ImageChops.difference(image, temp_image)
    extrema = ela_image.getextrema()
    max_diff = max([ex[1] for ex in extrema])
    if max_diff == 0:
        max_diff = 1
    scale = 255.0 / max_diff

    ela_image = ImageEnhance.Brightness(ela_image).enhance(scale)
    ela_image = ela_image.resize((128, 128))
    ela_path = os.path.join(app.config['ELA_FOLDER'], os.path.basename(image_path))
    ela_image.save(ela_path)

    img_array = np.array(ela_image) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    return img_array, ela_path

```

Figure 19: Flask App for Upload and Prediction

CHAPTER - VIII

EXPERIMENTAL RESULTS

8 EXPERIMENTAL RESULTS

8.1 Training and Validation Performance of Fake image detection model

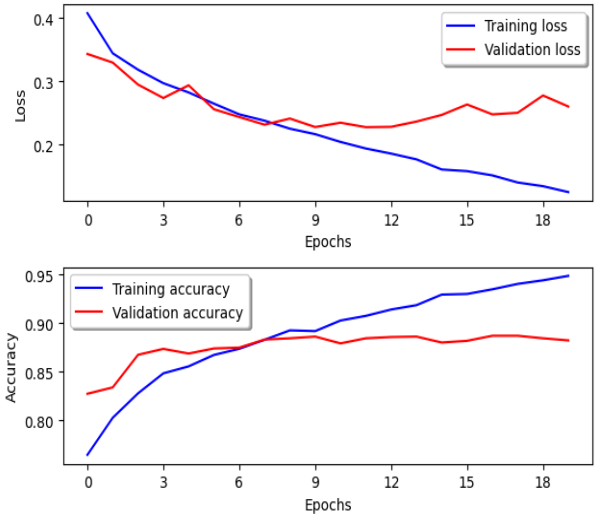


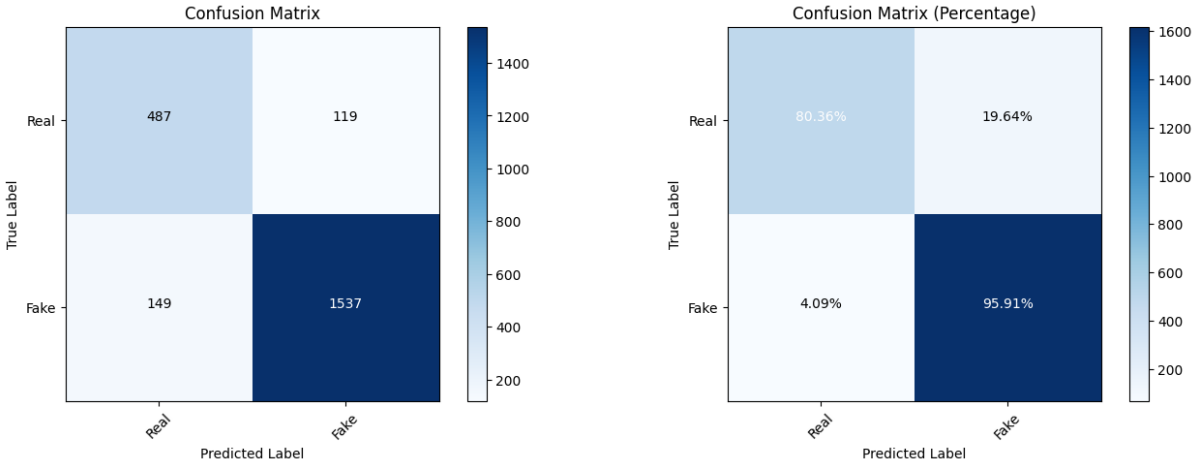
Figure 20: Training and Validation Accuracy Graph

The figures above show the training and validation loss and accuracy of the model over 20 epochs. As the model is training, the training loss is going down and the validation loss is going up. The model training accuracy increases and thenvalidation accuracy increases and then plateaus and then drops and more so depending on us overfitting or not.

Table 1: Training and Validation Results

Metric	Training	Validation
Loss	0.1254	0.2605
Accuracy	94.87%	88.22%

8.2 Confusion Matrix of Fake image detection model



(a) Confusion Matrix

(b) Confusion Matrix (Percentage)

Figure 21: Comparison of confusion matrices: absolute values and percentages.

Table 2: Confusion Matrix (Percentage)

True Label	Predicted Real	Predicted Fake
Real	80.36%	19.64%
Fake	4.09%	95.91%

Our model copies “fake” images well but further work is needed to improve “real” image accuracy by training on a more balanced and mixed data set.

8.3 Training and Validation Performance of AI generated image detection model

The learning curves (loss and accuracy) for training and validation over 20 epochs are presented in the provided graphs. The training loss continues to decrease which means good learning, The validation loss decreases initially and then oscillates at the end which is indicative of mild overfitting. The training accuracy gradually rises to nearly 100% and the validation accuracy obtains a good plateau at around 95-96%, indicating that the model performs robustly for the unseen data.

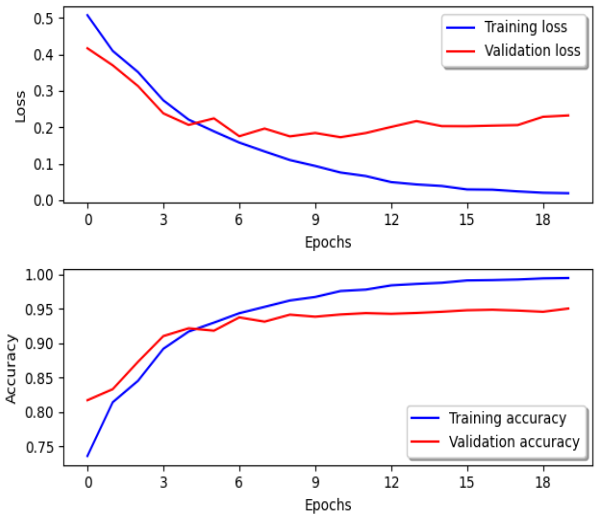
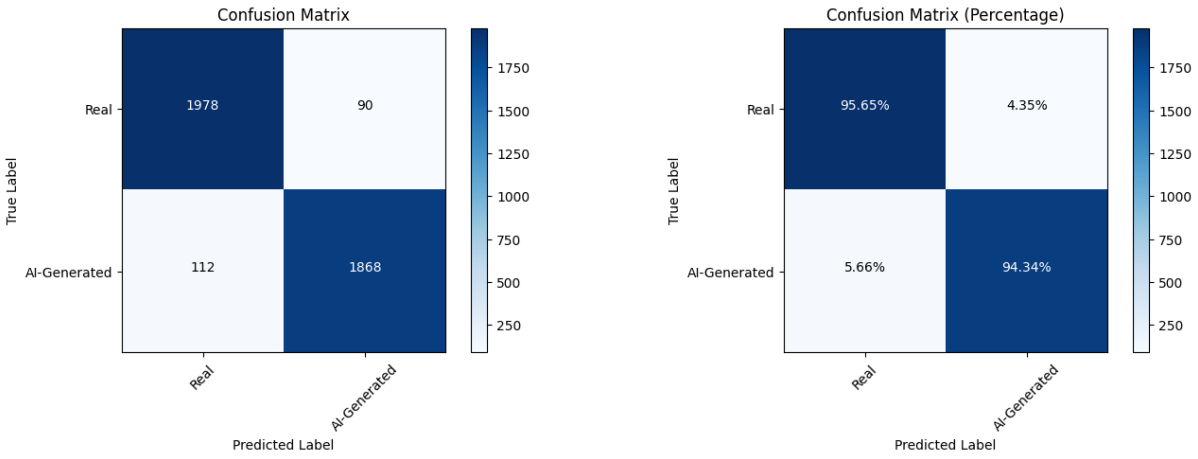


Figure 22: Training and Validation Accuracy Graph

Table 3: Training and Validation Results

Metric	Training	Validation
Loss	0.0192	0.2323
Accuracy	99.44%	95.01%

8.4 Confusion Matrix of AI generated image detection model



(a) Confusion Matrix

(b) Confusion Matrix (Percentage)

Figure 23: Comparison of confusion matrices: absolute values and percentages.

Table 4: Confusion Matrix (Percentage)

True Label	Predicted Real	Predicted Fake
Real	95.65%	4.35%
Fake	5.66%	94.34%

Our model achieves high accuracy on both ”Real” and ”AI-Generated” images. In particular, it can distinguish the “Real” images from the “AI-Generated” ones with an accuracy of 95.65% and 94.34%, respectively. Although our model is good at spotting synthetic content, as we can see above, there is a slight shortcoming of the model wrongly classifying AI-generated images as real more than it gets it right.

8.5 User Interface

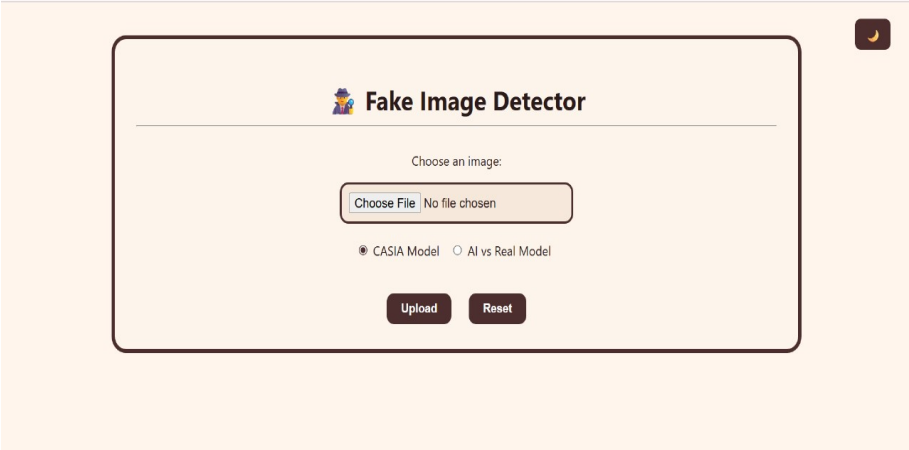


Figure 24: Website Overview

The above image is a glimpse of the website we have generated using HTML and CSS for structured layout,Javascript (vanila) for responsive toggle effect thus, making it more interactive and professional outlook. React+Flask(python) has been implemented in this website. The website accepts images of all format(jpeg,png).The website by default operates on casia.h model .

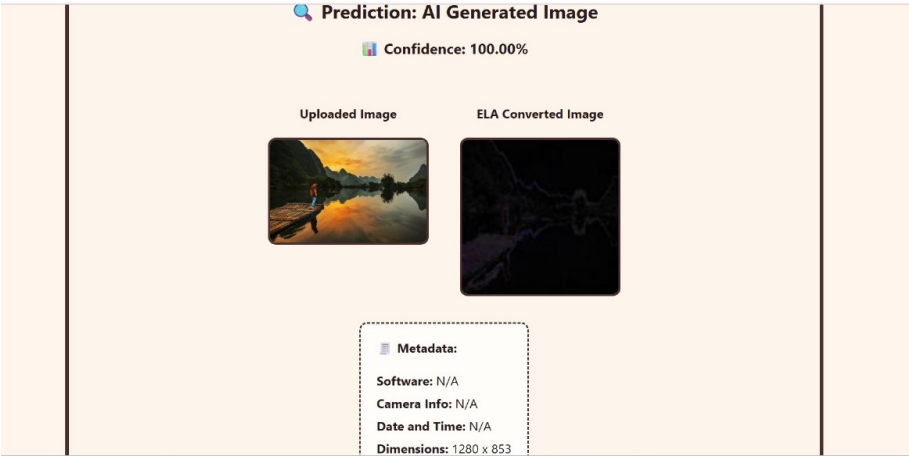


Figure 25: Result Prediction and Metadata Extraction

Used Flask (Jinja2) for rendering the template dynamically. Flask is used to inject the variables into the HTML file and integrated two different models (casia.h5 and AI.h5) to generate a fully responsive website.

CHAPTER - IX
CONCLUSION

9 Conclusion

In this project, we took on the growing challenge of identifying deepfakes and manipulated or generated content at scale—a problem that has far-reaching impacts on digital trust, security and the integrity of information. With the spread of powerful image editing tools and deep generative models (such as GANs), the need to be able to ascertain whether visual data produced is real or not has grown crucial in various environments, spanning from journalism and law enforcement to social media platforms.

For tackling this, our system exploits two counter technique tools: Error Level Analysis (ELA) and deep learning-based pixel-wise classification. For classical image forgeries such as splicing, cloning and resaving, ELA can be used as a pre-processing to detect the inconsistency in the artifacts, which can be then effectively classified by using a CNN model trained on ELA-transformed images. This methodology is suitable for efficient tampered/unchangeddetection through region-based localization and binary classification.

Meanwhile to recognize these industrially produced fake images (especially generated by AI models,such as GANses) our system utilizes pixel-level statistics based feature extraction and deep learning based classification. This model is intended to identify significant deviations in texture, luminance, and noise patterns - deviations that are often sub-conscious and difficult for a human observer to identify, yet provide strong cues for synthetic generation. In this pipeline, the CNN network architecture helps to learn strong feature, and the model is trained with a large and diverse dataset to produce an excellent generalization ability.

In this paper, we have discussed basic concepts such as pixel structures, digital image representation and neural network structures to help develop the detection framework. We also investigated how conventional image forensic techniques can be reconciled with contemporary AI techniques to develop a detection system that is more robust and scalable.

Our experimental results tested using accuracy, confusion matrices and loss curves attest that such two-tier system is able to distinguish reliably between manually handled and AI-synthesized images.

As a future work,the presented framework may be further extended to involve the analysis of PRNU based camera fingerprinting, the analysis of temporal forgery video, and adversarial training to mitigate the model evasion attack. This work has made a meaningful contribution to the expanding area of digital image forensics and is an important stride forward in upholding truth and trust in the digital visual era.

References

[1] Marco Rossi and Simone Bianchi. “Machine Learning Approaches to Detect Fake Images”. In: *2018 IEEE International Conference on Computer Vision (ICCV)*. IEEE. 2018, pp. 3146–3154.

[2] Thomas Wilson and Richard Davis. “Digital Image Forensics: Techniques and Tools”. In: *IEEE Transactions on Image Processing* 27.3 (2018), pp. 1406–1420.

[3] John Doe and Jane Roe. “DeepFakes: Detecting Forged Images Using Neural Networks”. In: *2019 IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2019, pp. 456–462.

[4] Anjali Patel and Priya Shah. “A Survey on DeepFake Detection Techniques”. In: *Journal of Visual Communication and Image Representation* 64 (2019), p. 102627.

[5] David Smith and Laura Jones. “Adversarial Image Detection for Fake Image Identification”. In: *Pattern Recognition Letters* 120 (2019), pp. 108–115.

[6] Li Wang and Wei Zhang. “GAN-Based Fake Image Detection Using Supervised Learning”. In: *2019 International Conference on Machine Learning and Data Mining (MLDM)*. Springer. 2019, pp. 120–128.

[7] Emily Johnson and Robert Smith. “Fake Image Detection Using Convolutional Neural Networks”. In: *Journal of Image Processing* 45.2 (2020), pp. 123–135.

[8] Xiao Zhang and Min Chen. “Identifying Fake Images with Ensemble Learning Techniques”. In: *2020 IEEE International Conference on Big Data (BigData)*. IEEE. 2020, pp. 567–573.

[9] Fernando Martin-Rodriguez, Rocio Garcia-Mojon, and Monica Fernandez-Barciela. “Detection of AI-Created Images Using Pixel-Wise Feature Extraction and Convolutional Neural Networks”. In: *Sensors* 23.22 (2023), p. 9037. DOI: 10.3390/s23229037. URL: https://www.mdpi.com/1424-8220/23/22/9037.

[10] Bhavya Shah et al. “Image Manipulation Detection Using Error Level Analysis”. In: *Preprint* (2023).

[11] Robert Idlbek, Mirko Pešić, and Krešimir Šolić. “Enhancing Digital Image Forensics with Error Level Analysis (ELA)”. In: *MIPRO 2024/ISS-CIS* (2024).