A

Major Project Report

On

Image Forgery Detection Using Machine Learning

*Submitted to **CMREC, HYDERABAD***

In Partial Fulfillment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

Submitted
By

B Soudhamini	**(218R1A6710)**
Sahasra Sandiri	**(218R1A6755)**
Rangoju Vashista Rama Krishna	**(218R1A6752)**
Yellapanthula V N Sri Deexit	**(218R1A6765)**

Under the Esteemed guidance of

Mrs. G. Shruthi

Assistant Professor, Department of CSE (Data Science)

Department of Computer Science & Engineering (Data Science)

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Approved by AICTE, NEW DELHI, Affiliated to JNTU, Hyderabad)
Kandlakoya, Medchal Road, R.R. Dist. Hyderabad-501 401.

2024-2025

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Accredited by NBA, Approved by AICTE NEW DELHI, Affiliated to JNTU, Hyderabad)

Kandlakoya, Medchal Road, Hyderabad-501 401

Department of Computer Science & Engineering (Data Science)

CERTIFICATE

This is to certify that the project entitled **"Image Forgery Detection Using Machine Learning" i**s a bonafide work carried out by

B Soudhamini (218R1A6710)
Sahasra Sandiri (218R1A6755)
Rangoju Vashista Rama Krishna (218R1A6752)
Yellapanthula V N Sri Deexit (218R1A6765)

in partial fulfillment of the requirement for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE) from CMR Engineering College, affiliated to JNTU, Hyderabad, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	**Major Project Coordinator**	**Head of the Department**	**External Examiner**
Mrs. G. Shruthi	**Mrs. G. Shruthi**	**Dr. M. Laxmaiah**	
Assistant Professor CSE (Data Science), CMREC	Assistant Professor CSE (Data Science), CMREC	Professor & HOD CSE (Data Science), CMREC	

DECLARATION

This is to certify that the work reported in the present Major project entitled **"Image Forgery Detection Using Machine Learning"** is a record of bonafide work done by us in the Department of Computer Science and Engineering (Data Science), CMR Engineering College, JNTU Hyderabad. The reports are based on the project work done entirely by us and not copied from any other source. We submit our project for further development by any interested students who share similar interests to improve the project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

B Soudhamini (218R1A6710)
Sahasra Sandiri (218R1A6755)
Rangoju Vashista Rama Krishna (218R1A6752)
Yellapanthula V N Sri Deexit (218R1A6765)

ACKNOWLEDGMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. M. Laxmaiah**, HOD, **Department of CSE (Data Science), CMR Engineering College** for their constant support**.**

We are extremely thankful to **Mrs. G. Shruthi,** Assistant Professor, Internal Guide, Department of CSE(DS), for her constant guidance, encouragement and moral support throughout the project.

We will be failing in duty if We do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Project.

We thank **Mrs. G. Shruthi**, Assistant Professor, CSE(DS) Department**,** Major Project Coordinator for her constant support in carrying out the project activities and reviews.

We express my thanks to all staff members and friends for all the help and co-ordination extended in bringing out this project successfully in time.

Finally, We are very much thankful to our parents who guided me for every step.

B Soudhamini (218R1A6710)
Sahasra Sandiri (218R1A6755)
Rangoju Vashista Rama Krishna (218R1A6752)
Yellapanthula V N Sri Deexit (218R1A6765)

ABSTRACT

The increasing sophistication of digital image manipulation, ensuring image authenticity has become a critical challenge in fields such as journalism, forensics, and cybersecurity. This study presents a hybrid image forgery detection system that integrates Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) to effectively identify copy-move and splicing forgeries. The proposed approach is divided into two main stages. First, VGG16, a pre-trained CNN model, is utilized for feature extraction, allowing the system to capture intricate image details and detect subtle inconsistencies indicative of tampering. By analyzing features such as texture, edges, and spatial correlations, the model learns to differentiate between authentic and manipulated images. In the second stage, a GAN-based framework is implemented to enhance the detection process. The generator produces highly realistic forged images, augmenting the dataset to improve the model's generalization ability. The discriminator, trained alongside the generator, is then fine-tuned as a classifier to differentiate between real and forged images. This adversarial training strategy enables the system to adapt to evolving forgery techniques and improve detection accuracy. The model is trained and tested on the CoMoFoD dataset (Copy-Move Forgery Detection), which contains a diverse collection of real and manipulated images. Experimental results demonstrate that the hybrid CNN-GAN approach significantly improves the accuracy of forgery detection, exhibiting high adaptability to variations in lighting conditions, resolution, and image quality. By integrating deep learning techniques, this study contributes to the advancement of digital forensics by providing a scalable, flexible, and efficient solution for automated image forgery detection. The proposed system enhances image integrity verification and supports efforts to combat misinformation in the digital era.

.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Overview

In the digital era, images play a crucial role in communication, journalism security, and social media. However, with advancements in image editing tools, digital images are increasingly susceptible to manipulation. Image forgery, including splicing and copy-move alterations, has raised significant concerns regarding the authenticity of visual content. Splicing involves combining elements from different images to create a new, often deceptive image, while copy-move forgery duplicates sections within the same image to hide or alter information.

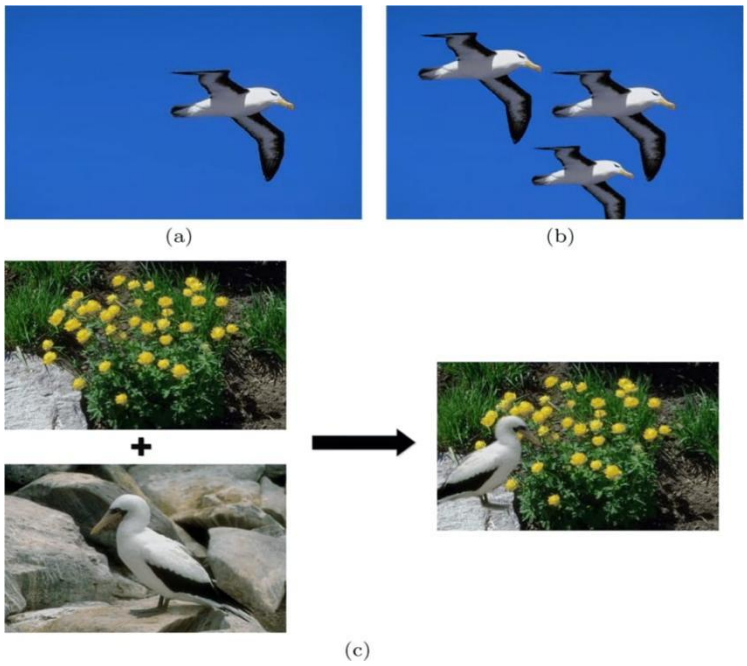


Fig:1.1 (a)real image (b) forged-image (example of copy-move)
(c) forged image (example of splicing),**researchgate.net/figure/Examples-of Copy-move-and Splicing**

Copy-Move forgery refers to a kind of image alteration where a section of an image is duplicated and inserted back into the same image to conceal or replicate certain elements. As the duplicated area comes from the same image, it preserves comparable lighting, texture, and noise, which complicates detection. This method is frequently employed to eliminate undesirable items, replicate components, or deceive audiences by modifying the image's content. Nevertheless, identifying copy-move forgery becomes challenging when alterations like rotation, scaling, or blurring affect the duplicated area. Different techniques, such as block-based matching, feature-

based strategies like SIFT (Scale-Invariant Feature Transform), and deep learning models, are used to detect these manipulations. Splicing forgery, conversely, entails merging elements from two or more distinct images to form one altered image. In contrast to copy-move forgery, splicing brings in outside elements, potentially causing discrepancies in lighting, shadows, and textures. This form of forgery is frequently utilized in misinformation, false news, and digital trickery, with individuals or items being placed into visuals where they were never present. Identifying splicing forgery depends on methods like Error Level Analysis (ELA), checks for shadows and lighting inconsistencies, and deep learning algorithms that examine unusual patterns in the image. Although splicing forgeries can be complex, inconsistencies in color mixing, perspective, and how objects interact frequently indicate manipulation. Generative Adversarial Networks (GANs) consist of two primary components: the generator and the discriminator, which work in an adversarial manner to improve the model's capability to detect forgeries. The generator is responsible for creating realistic fake images by learning the distribution of authentic images, making it harder for the discriminator to differentiate between real and fake data. Meanwhile, the discriminator acts as a classifier, distinguishing between authentic and generated images. As training progresses, both networks continuously improve—the generator learns to create more convincing forgeries, while the discriminator becomes more adept at identifying subtle inconsistencies. This adversarial learning process makes GANs highly effective in training robust forgery detection models by simulating various forgery scenarios and improving classification accuracy.

The widespread use of manipulated images in media, legal cases, and forensic investigations has highlighted the necessity for robust detection mechanisms. Traditional approaches such as manual inspection and basic computational techniques struggle to keep up with sophisticated forgeries. As a result, the development of automated and intelligent detection techniques has become a critical area of research. With the emergence of machine learning and deep learning, methods such as Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) have shown promise in detecting image forgery with high accuracy. This research proposes a hybrid model combining CNNs and GANs to enhance forgery detection, offering a more reliable and efficient solution

1.2 Research Motivation

The motivation for this research stems from the increasing threat posed by digital image manipulation across various sectors. In journalism, altered images can spread misinformation, affecting public opinion and credibility. In law enforcement, image forgery can interfere with

investigations and judicial decisions. In social media, doctored images contribute to the proliferation of fake news and cyber deception. Given these challenges, ensuring the authenticity of digital images is of paramount importance. Traditional detection methods rely on visual inspection and heuristic algorithms, which often fail when faced with sophisticated forgeries. Advanced forgeries employ techniques such as noise addition, blurring, and geometric transformations, making detection even more difficult. This necessitates the adoption of deep learning-based approaches that can automatically identify inconsistencies and irregularities in images. By integrating CNNs and GANs, this research aims to enhance the accuracy and robustness of forgery detection. CNNs effectively extract image features, while GANs generate realistic variations to train more resilient detection models. This hybrid approach not only improves detection rates but also adapts to emerging forgery techniques, making it a promising solution for ensuring image authenticity in digital forensics.

1.3 Problem Statement

The proliferation of image editing software has made image manipulation more accessible, leading to concerns over the authenticity and credibility of digital images. The ability to modify images has been exploited in various domains, including journalism, law enforcement, social media, and even scientific research. The primary challenges associated with image forgery detection include

Increasing Sophistication of Image Manipulation: Modern image editing techniques allow for highly convincing forgeries that can deceive both human observers and traditional detection systems.

Limitations of Conventional Detection Methods: Manual inspection and basic computational techniques are inadequate in identifying subtle modifications, especially in cases where image alterations involve complex transformations like rotation, scaling, and noise addition

Adaptability to Emerging Forgery Techniques: The field of digital forensics requires continuous advancements in detection models to combat new and evolving methods of forgery.

Need for an Automated and Scalable Approach: The demand for an intelligent system capable of detecting forgeries across vast datases with minimal human intervention is crucial for ensuring the credibility of digital images.This research addresses these challenges by proposing a hybrid model that combines CNNs and GANs to enhance the accuracy of image forgery detection. The model leverages CNNs for feature extraction and GANs for generating variations, ultimately improving robustness against sophisticated forgeries.

1.4 Applications

The implementation of an effective image forgery detection system has significant implications across multiple industries. Some of the most critical applications include:

Journalism and Media Verification: Ensuring the credibility of images used in news articles and reports by identifying manipulated visuals that could mislead the public.

Law Enforcement and Digital Forensics: Aiding forensic experts and law enforcement agencies in verifying the authenticity of photographic evidence in criminal investigations.

Social Media Content Moderation: Detecting and preventing the spread of fake images that contribute to misinformation, online fraud, and cyber deception.

Medical Imaging and Healthcare: Identifying alterations in medical scans, X-rays, and other diagnostic images to prevent fraudulent claims and ensure accurate medical assessments.

commerce and Product Authentication: Verifying product images to prevent misleading advertisements, counterfeit goods, and false marketing claims.

Intellectual Property and Copyright Protection: Detecting unauthorized modifications to copyrighted images, ensuring the protection of digital content creators.

Security and Surveillance: Verifying the authenticity of surveillance footage to support investigations, ensuring the reliability of evidence used in legal proceedings.

Scientific Research and Academia: Detecting manipulated images in research publications to maintain the integrity of scientific findings.By providing an advanced and automated detection mechanism, this research contributes to enhancing digital trust, preventing fraudulent activities, and ensuring the authenticity of images in critical applications. The proposed hybrid approach combining CNNs and GANs aims to establish a more robust and reliable solution for detecting and mitigating image forgery in the digital landscape.

2. LITERATURE SURVEY

The identification of image forgeries has emerged as a crucial area of research due to the increasing sophistication of image manipulation techniques. As digital images are widely used in various fields, including journalism, forensic investigations, and social media, the need for robust image forgery detection mechanisms has become paramount. Researchers have explored multiple approaches, ranging from traditional machine learning techniques to state-of-the-art deep learning-based methods, to effectively identify and classify forged images.Early research in image forgery detection primarily relied on traditional machine learning methods that focused on handcrafted feature extraction and classification techniques. These methods included statistical analysis, edge detection, and texture-based feature extraction, which were later processed using classifiers such as Support Vector Machines (SVM), Random Forests, and Decision Trees. A study by a research scholar [2] examined various machine learning algorithms for detecting fake images, showcasing their efficiency in identifying tampered regions. While these methods provided reasonable accuracy, their reliance on manual feature engineering limited their adaptability to diverse forgery types.

Deep learning techniques have significantly improved the accuracy and efficiency of image forgery detection by automating feature extraction and classification. Dr. N. P. Nethravathi [1] introduced a deep neural network model that emphasizes the importance of feature extraction in distinguishing between authentic and counterfeit images. Similarly, research published in MDPI [3] proposed a deep learning-based digital image forgery detection system that enhances precision in recognizing manipulated images.The emergence of Convolutional Neural Networks (CNNs) has revolutionized the field of image forensics by providing robust frameworks for forgery detection. J. Malathi [5] developed a model that employs CNN-based feature extraction and classification techniques, showcasing their effectiveness in differentiating forged images from authentic ones. Additionally, a comprehensive review by Springer [7] explored various CNN architectures applied to image forgery detection, highlighting their advantages in identifying minute inconsistencies introduced during image manipulation.

An extensive survey by IJITEE [6] assessed contemporary deep learning-based strategies for identifying image alterations. Research published in IRJET [8] demonstrated the use of CNN-based models for practical forgery detection applications, emphasizing their ability to generalize

across different forgery types. IEEE Xplore [9] further contributed by conducting an in-depth study on CNN-based frameworks for detecting image forgeries.

Generative Adversarial Networks (GANs) have introduced new challenges in image forgery detection, as they can generate highly realistic fake images. In response, researchers have investigated adversarial learning-based strategies to counter these threats. Inderscience Online [10] explored deep learning methods that leverage adversarial learning to detect altered images effectively. IEEE Xplore [11] provided insights into document image forgery detection using deep learning classification techniques, further expanding the scope of AI-driven forensic applications.AASM [12] examined deep learning-based classification methods for detecting forged images, demonstrating their benefits in forensic investigations. ArXiv [13] explored forensic analysis techniques utilizing advanced deep learning models to identify manipulated images with greater accuracy. Goebel et al. [14] specifically investigated the detection and localization of GAN-generated images, emphasizing their forensic implications.Takayuki Osakabe [15] proposed a CycleGAN-based counter-forensics approach to detect fake images while minimizing checkerboard artifacts, a common issue in GAN-generated images. Similarly, Sri Kalyan Yarlagadda [16] developed a technique for identifying and localizing tampered satellite images using GANs and one-class classifiers, highlighting the growing need for specialized forgery detection techniques in different domains.

The literature reviewed highlights the evolution of image forgery detection methods from traditional machine learning approaches to deep learning and GAN-based techniques. The incorporation of artificial intelligence in digital forensics continues to enhance the accuracy and reliability of forgery detection, providing robust and scalable solutions for forensic investigations. As image manipulation techniques advance, further research is required to develop adaptive and efficient detection models capable of addressing emerging challenges in image forensics.

3. EXISTING SYSTEM

The detection of image forgeries has undergone significant evolution, progressing from traditional manual inspection methods to advanced deep learning techniques. While these advancements have improved accuracy and efficiency, existing systems still face various limitations that necessitate further improvements.

3.1 Traditional Methods

Historically, image forgery detection relied on manual inspection, where forensic experts analyzed images for inconsistencies in lighting, shadows, textures, and unnatural patterns. While human judgment provided valuable insights, it was highly subjective, time-consuming, and prone to errors due to fatigue and cognitive biases. The limitations of manual detection led to the development of algorithmic methods to assist forensic investigations.One such algorithmic technique is Error Level Analysis (ELA), which examines variations in compression levels across different regions of an image. Since digital images undergo compression, altered sections may exhibit inconsistencies compared to the original portions. ELA highlights these differences, making it easier to identify potential forgery. However, ELA struggles against high-quality forgeries, where modifications are subtle, and compression artifacts are minimal. Additionally, this method often produces false positives, making it less reliable for precise forgery detection.

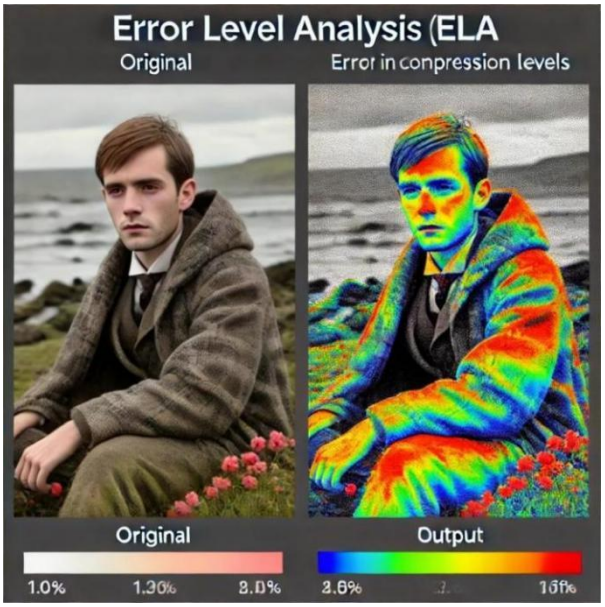


Fig 3.1 Error level Analysis example.

Another widely used approach is pixel-based copy-move forgery detection, which analyzes overlapping pixel blocks to identify duplicated regions within an image. Copy-move forgery is

one of the most common types of image manipulation, where a section of an image is copied and pasted elsewhere to hide or alter information. Traditional copy-move detection methods compare pixel intensity values and detect similarities. However, these methods fail when transformations such as scaling, rotation, or lighting adjustments are applied, as they alter the pixel properties, reducing detection accuracy.

Additionally, statistical and frequency-domain approaches, such as Discrete Cosine Transform (DCT) and Wavelet Transforms, have been employed to detect manipulated images. These methods analyze the frequency components of an image to identify anomalies introduced during editing. While these techniques provide useful insights, they often fail to generalize well across different types of forgeries and are computationally expensive.The inefficiencies and limitations of traditional methods led to the adoption of machine learning techniques, which introduced automation and improved accuracy in forgery detection.

3.2 Machine Learning-Based Approaches

The introduction of machine learning (ML) techniques marked a significant shift in image forgery detection. Unlike traditional methods that relied on manually defined rules, ML-based approaches leverage data-driven feature extraction and classification models to identify forged images.Early ML-based methods used handcrafted features such as color histograms, texture descriptors (e.g., Local Binary Patterns), edge features (e.g., Canny edge detection), and statistical measures to extract meaningful information from images. These features were then fed into classification models such as Support Vector Machines (SVM), Decision Trees, Random Forests, and K-Nearest Neighbors (KNN) to differentiate between authentic and forged images.While these approaches improved detection accuracy compared to traditional methods, they had several drawbacks:

Feature Dependency: The performance of these models heavily depended on the quality of handcrafted features. Poor feature selection could lead to inaccurate results.

Computational Overhead: Extracting multiple features and training ML models required significant computational resources, making real-time forgery detection difficult.

Lack of Adaptability: Handcrafted feature-based methods struggled with complex forgeries, such as those involving subtle texture modifications or sophisticated splicing techniques.As a result, researchers began exploring deep learning techniques to overcome these limitations and enhance the robustness of image forgery detection.

3.3 Deep Learning-Based Approaches

Deep learning, particularly Convolutional Neural Networks (CNNs), revolutionized image analysis and forgery detection by enabling automatic feature extraction and classification. Unlike traditional machine learning methods, CNNs do not rely on manually defined features; instead, they learn hierarchical feature representations directly from raw image data.CNNs have been highly effective in detecting various forms of image forgeries, including splicing, copy-move, and inpainting-based alterations. Studies have shown that CNNs can achieve superior accuracy in identifying manipulated images compared to traditional ML models. Some key CNN architectures used for image forgery detection include VGG16, ResNet, and EfficientNet, which extract deep features from images and classify them as authentic or forged.

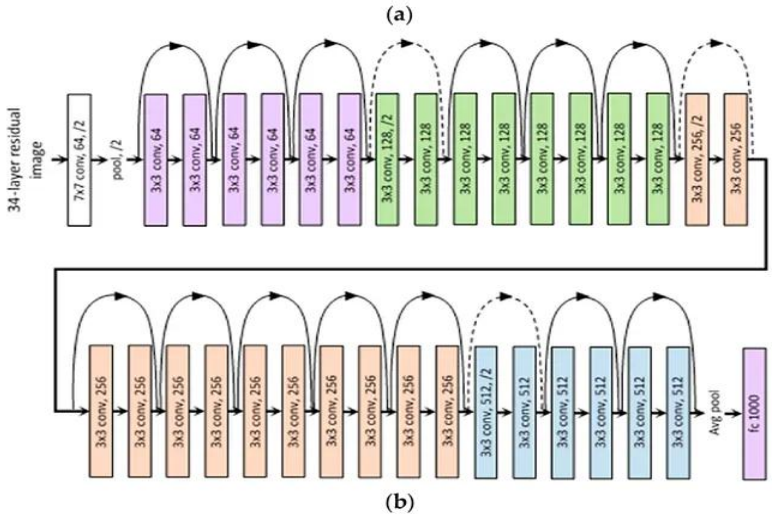


Fig :3.2 Resnet Architecture

Despite their success, deep learning-based approaches face several challenges.

Data Dependency: CNNs require vast amounts of labeled training data to generalize effectively. In cases where high-quality annotated datasets are limited, the performance of deep learning models suffers.

Computational Complexity: Training deep learning models demands powerful hardware (GPUs/TPUs), making it difficult to deploy them on low-resource devices or real-time forensic applications.

Vulnerability to Adversarial Attacks:CNN-based models can be tricked by adversarial images, where imperceptible modifications mislead the classifier into making incorrect predictions. This raises security concerns in forensic and legal applications.

Due to these challenges: Researchers have explored hybrid approaches that combine CNNs with adversarial learning techniques, such as Generative Adversarial Networks (GANs), to enhance forgery detection.

3.4 Limitations of the Existing System

Despite the advancements in image forgery detection, existing systems still have several limitations and challenges:

High Dependence on Data: Both machine learning and deep learning models require large, well-labeled datasets to achieve high accuracy. In forensic applications, obtaining such datasets is often difficult.

Susceptibility to Transformations: Traditional methods fail when images undergo transformations like rotation, scaling, compression, or brightness changes, affecting detection accuracy.

False Positives and False Negatives: Many forgery detection models struggle with balancing precision and recall, often misclassifying real images as fake or missing subtle forgeries.

Computational Constraints: Deep learning models require significant processing power, making them impractical for real-time detection in low-resource environments.

Adversarial Vulnerabilities: Sophisticated attacks can manipulate images in a way that evades detection, posing risks in security-sensitive applications such as digital forensics and law enforcement.

Need for an Improved System

To overcome these challenges, next-generation forgery detection models should integrate:

Hybrid Techniques: Combining CNNs with GAN-based methods to improve detection accuracy and reduce false positives.

Robust Feature Engineering: Utilizing multi-scale and frequency-domain features alongside deep learning to improve adaptability.

Adversarial Defense Mechanisms: Developing models that are resistant to adversarial attacks by incorporating adversarial training and robust optimization techniques.

Efficient Deployment Strategies: Optimizing deep learning models for real-time applications using lightweight architectures and model compression techniques.

4. PROPOSED METHODOLOGY

The system we propose integrates two core components: VGG16, a Convolutional Neural Network (CNN) for feature extraction, and a Generative Adversarial Network (GAN) for data augmentation. By combining these advanced deep learning techniques, our system enhances the accuracy and robustness of image forgery detection. The model is designed to effectively identify both copy-move and splicing manipulations, two of the most common types of image tampering.

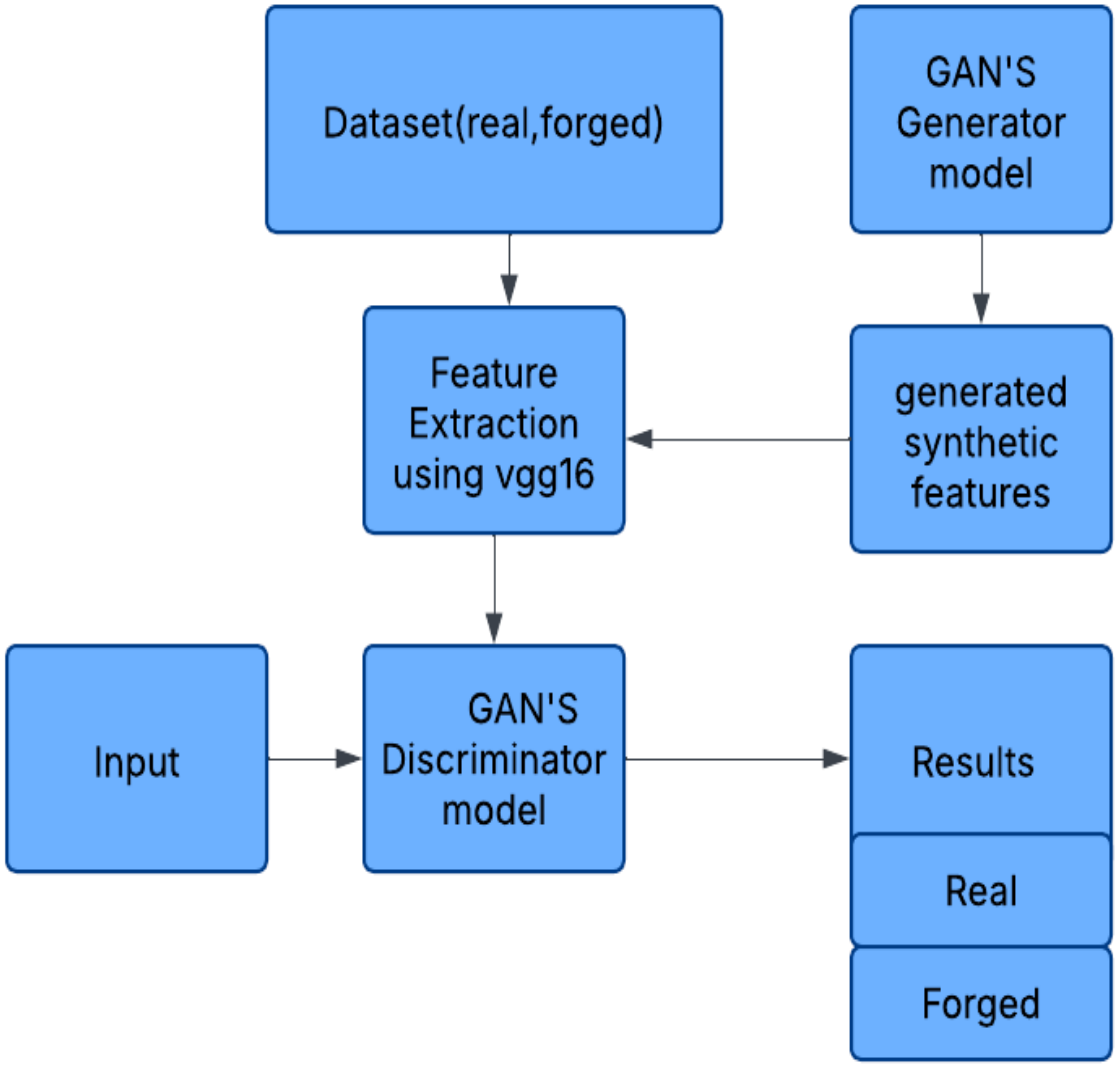


Fig 4.1 : Block diagram of CNN-GAN(hybrid approach)

4.1 VGG16 for Feature Extraction

The first stage of our system utilizes VGG16, a deep learning model optimized for image analysis. VGG16 is a widely recognized CNN architecture known for its deep yet uniform

structure. It consists of multiple convolutional layers, followed by pooling layers and fully connected layers, all of which play a crucial role in systematically extracting features from images. By leveraging the hierarchical learning capabilities of CNNs, VGG16 ensures that our model can accurately differentiate between authentic and forged images.

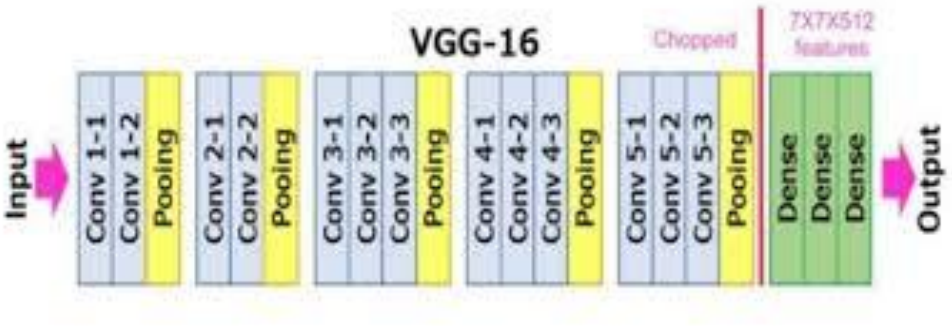


Fig 4.2 : VGG-16 Architecture

Convolutional Layers

The convolutional layers of VGG16 are designed to apply multiple filters to input images, detecting essential patterns such as edges, textures, and shapes. These filters perform convolution operations, enabling the network to learn low-level features in the initial layers and gradually capture more complex patterns in deeper layers. As the depth of the network increases, the model builds hierarchical feature representations, which improve its ability to distinguish between authentic and manipulated images.

Pooling Layers

Pooling layers are positioned after convolutional layers to down-sample feature maps while preserving crucial information. These layers reduce the spatial dimensions of the feature maps, making the model computationally efficient and robust to variations such as scaling, translation, and rotation. Max pooling, the most commonly used pooling technique, selects the highest value from each receptive field, ensuring that significant features remain intact while redundant information is eliminated.

Fully Connected Layers

Once the feature maps have been extracted and down-sampled, they are flattened and passed through fully connected layers. These layers aggregate the extracted features and determine the probability that an image has been manipulated. The final output is a classification result indicating whether an image is real or forged. VGG16 undergoes supervised training on a dataset containing

both genuine and forged images, using optimization techniques such as backpropagation and gradient descent to fine-tune its weights and improve classification accuracy.By utilizing VGG16 as a feature extractor, our system ensures that only the most relevant and informative features are used for classification, thereby enhancing the precision and reliability of image forgery detection. The extracted features are then passed to the next stage, where a GAN-based approach is employed for data augmentation and further strengthening the model's detection capabilities.

4.2 GAN'S Generator for Data Augmentation

Deep learning models, including Convolutional Neural Networks (CNNs), typically require large-scale datasets to generalize well. However, in image forgery detection, datasets are often imbalanced or limited in diversity, making it challenging for traditional models to effectively learn the variations in forged images. To overcome this limitation, we employ Generative Adversarial Networks (GANs) for data augmentation.

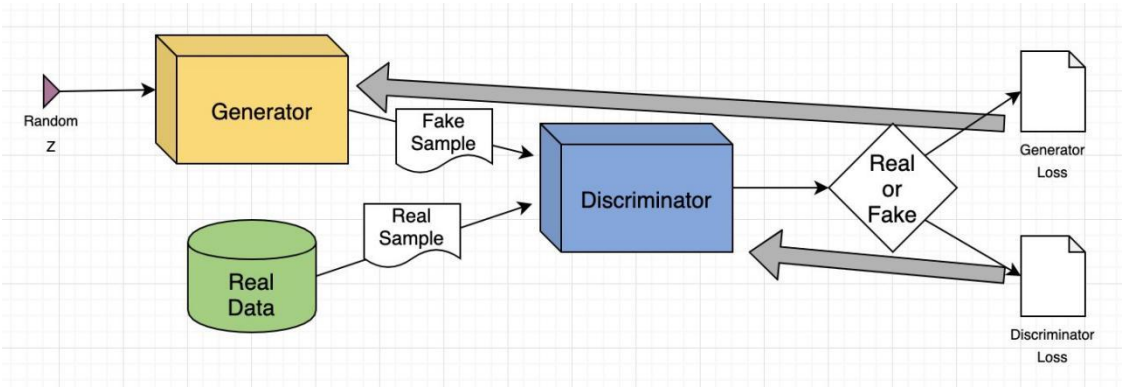


Fig 4.3 : GAN Architecture

GANs, introduced by Ian Goodfellow in 2014, consist of two neural networks—a Generator and a Discriminator that engage in a continuous adversarial process. This interplay results in the generation of increasingly realistic synthetic images, ultimately improving the generalization capability of the classifier.

Generator: Creating Realistic Forged Image:

The Generator network is responsible for synthesizing realistic forged images. It takes a random noise vector or an input image and applies splicing or copy-move manipulations to create forged samples that resemble real-world manipulated images.

Key functions of the Generator:

Mimicking real-world forgeries: It learns to replicate common manipulation techniques such as splicing, copy-move, and re-touching by transforming real images.

Diversity enhancement: By generating different types of forged images, it helps the detection model generalize to unseen cases.

Adversarial improvement: Over successive training iterations, the generator refines its forgery techniques to fool the discriminator.

4.3 Discriminator For Detecting Manipulations

The Discriminator is a binary classifier that learns to differentiate between real and fake images. It plays a crucial role in refining the quality of generated forgeries by providing feedback to the Generator Key functions of the Discriminator:

Forgery detection: It attempts to classify images as either authentic or manipulated.

Adversarial training feedback: It continuously improves the Generator's ability to produce realistic forgeries.

Feature learning: By distinguishing between real and fake images, the Discriminator implicitly learns forgery patterns, strengthening its detection capability.

Benefits of GAN-based Augmentation

Addressing Data Scarcity: GAN-generated forgeries expand the dataset, helping VGG16 train on a wider variety of manipulated images.

Reducing Overfitting: The additional training samples improve model generalization and prevent overfitting to a small set of training images.

Enhancing Robustness: GANs ensure the model can handle real-world variations, including different image formats, compression artifacts, and complex manipulations.

Self-improving System: The adversarial training dynamic ensures the constant refinement of both forged images and detection capabilities.

Data Preparation and Preprocessing

Preprocessing plays a critical role in ensuring that the dataset used for training is clean, structured, and optimized for deep learning models. Our system follows a structured pipeline to enhance the quality of input images and ensure they are ready for feature extraction and classification.

Image Preprocessing Techniques

Splitting:

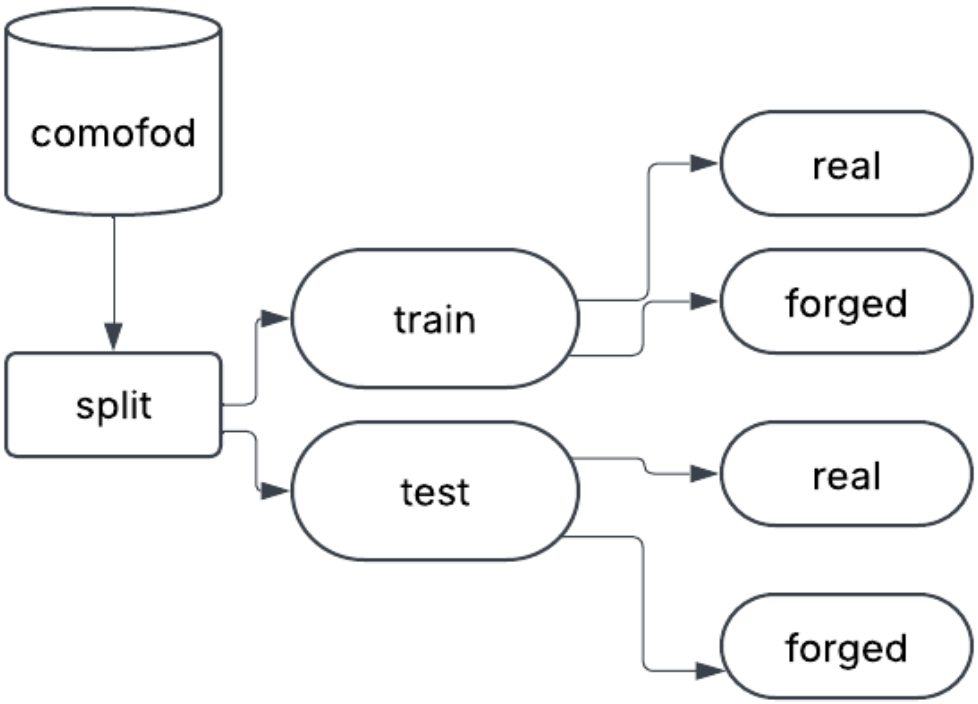


Fig 4.4: Data set Splitting

Resizing and Normalization

- All images are resized to 224×224 pixels to match the input dimension of VGG16.
- Pixel values are normalized between 0 and 1 to ensure stable convergence during training.
- Normalization helps in reducing bias and making the training process more efficient.

Data Augmentation for Robustness

To prevent overfitting and enhance diversity, the dataset is augmented using techniques like:

1) **Rotation (±15° to 30°):** Helps in making the model invariant to orientation changes.
2) **Flipping (horizontal/vertical):** Ensures robustness to different image alignments.
3) **Gaussian Noise Addition:** Helps the model handle real-world artifacts.

Patch Extraction for Copy-Move Forgery Detection

- Copy-move forgeries involve duplicating a region of an image and pasting it elsewhere.
- We segment images into overlapping patches, enabling the model to focus on duplicated regions.
- Feature matching techniques, such as SIFT (Scale-Invariant Feature Transform), are used to track duplicated areas.

Edge Detection and Masking for Splicing Detection

- Splicing involves merging content from different images, leading to discontinuities at boundaries.
- Sobel edge detection or Canny filtering is used to highlight forged boundaries.
- Masking techniques further enhance boundary regions, improving feature extraction in tampered areas.

Strategy for Training and Optimization

To ensure the best performance, we employ multiple optimization strategies to enhance both training efficiency and model accuracy.

Transfer Learning

Instead of training VGG16 from scratch, we leverage pre-trained weights from ImageNet, fine-tuning the model on our dataset. This reduces training time and improves feature extraction for forgery detection**.**

Joint Training of VGG16 and GAN

1)The forged images generated by the GAN are continuously into VGG16 during training.

2)This dynamic dataset expansion enhances VGG16's ability to differentiate subtle manipulations.

Adaptive Learning Rate and Optimizer Selection

1) We use an adaptive learning rate scheduling technique called Reduce LR on Plateau, which lowers the learning rate when the validation accuracy plateaus, preventing overfitting.
2) Adam optimizer is used for stable gradient updates.

Regularization Techniques

1) Batch Normalization ensures that feature distributions remain stable during training.
2) Dropout layers (with 0.3–0.5 probability) are added to prevent overfitting.

Fine-Tuning for Tampered Region Detection

1) We introduce heatmap-based learning to focus attention on altered regions in an image.
2) This makes the model more sensitive to subtle forgery patterns in spliced and copy-move manipulated images.

4.4 Advantages of the Proposed System

Enhanced Forgery Detection Accuracy: The hybrid approach of CNNs and GANs improves detection performance by leveraging both feature extraction (VGG16) and synthetic data generation (GANs) for better model generalization.

Robust Against Copy-Move and Splicing Attacks: Unlike traditional methods, the system is specifically designed to detect both copy-move and splicing forgeries, making it more versatile in real-world applications.

Automated Feature Extraction: Using VGG16 eliminates the need for manual feature extraction, enabling faster and more efficient detection without human intervention.

Improved Generalization with GANs: The data augmentation capability of GANs helps the model learn diverse forgery patterns,

reducing overfitting and increasing reliability on unseen data.

Efficient Classification with GAN Discriminator: The discriminator network from GANs, trained to distinguish real from fake images, serves as a powerful classifier, enhancing overall detection precision.

Scalability and Adaptability: The system can be trained on different datasets, making it adaptable to various image forgery scenarios across domains like forensic analysis, media verification, and digital security.

User-Friendly and Automated Detection: The model provides a simple image path input and outputs whether the image is forged or not, making it easy to integrate into practical applications.

5. UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS: The Primary goals in the design of the UML are as follows:

Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.Provide extendibility and specialization mechanisms to extend the core conceptsBe independent of particular programming languages and development process.Provide a formal basis for understanding the modeling language.Encourage the growth of tools market.Support higher level development concepts such as collaborations, frameworks, patterns and components.Integrate best practices.

5.1 Class diagram

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

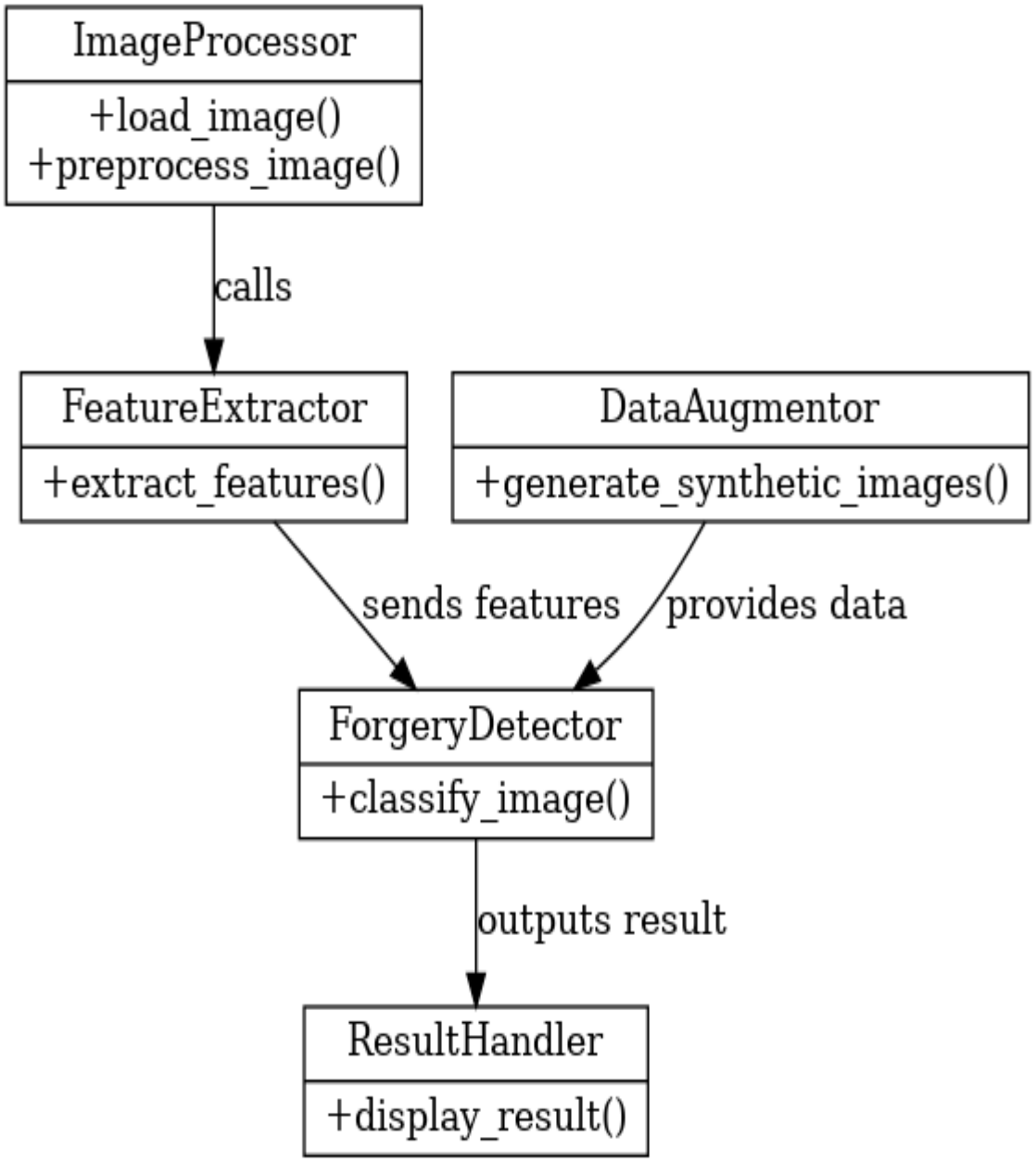


Fig 5.1 : Class Diagram of CNN-GAN(hybrid approach)

5.2 Use case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. Sequence diagram

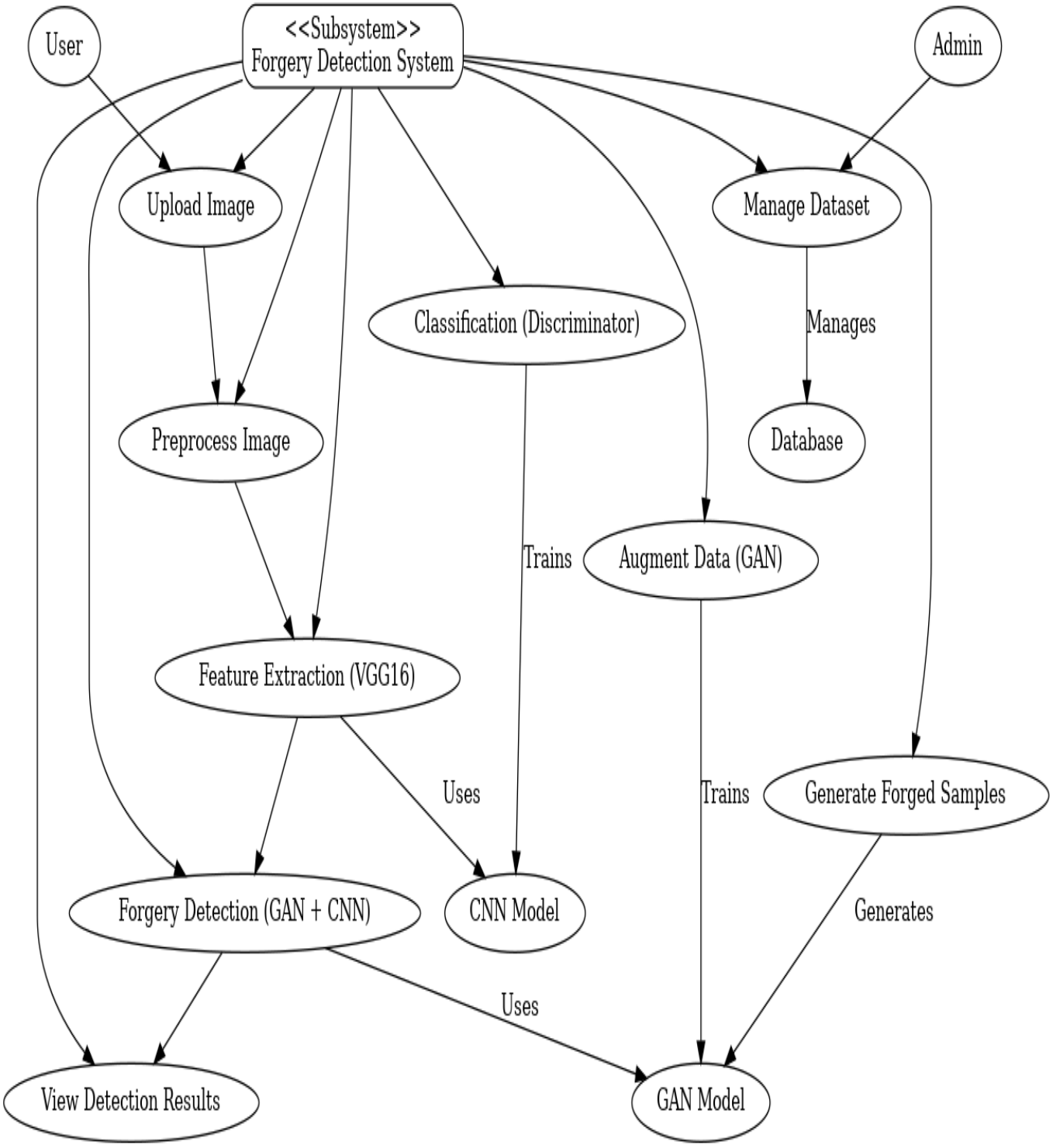


Fig 5.2 : Use Case diagram of CNN-GAN

5.3 Sequence Diagram

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message

Sequence Chart. A sequence diagram shows, as parallel vertical lines ("lifelines"), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

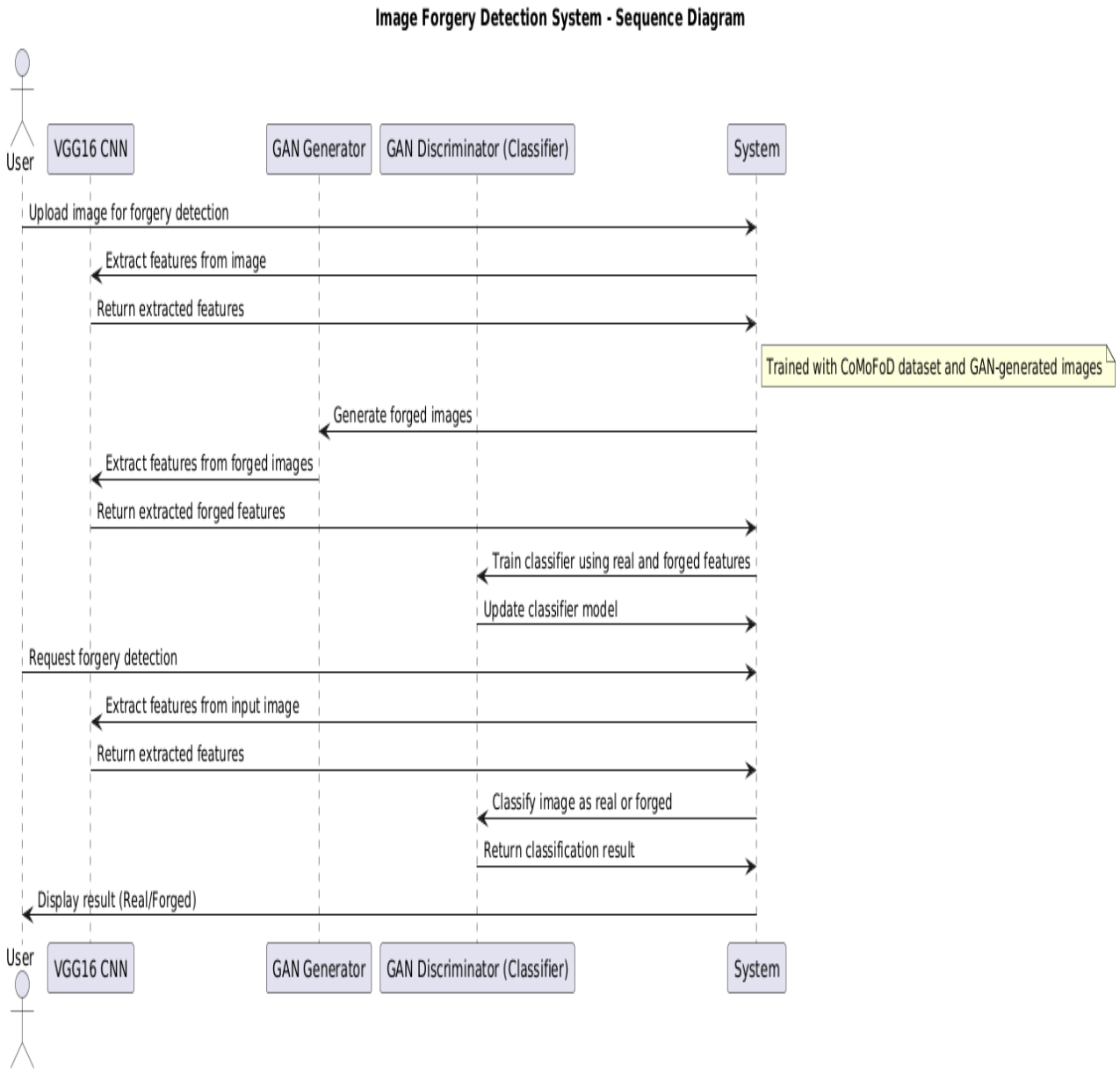


Fig 5.3: Sequence diagram of CNN-GAN

5.4 Activity diagram

Activity diagrams are graphical representations of Workflows ofstepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language,

activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow control.

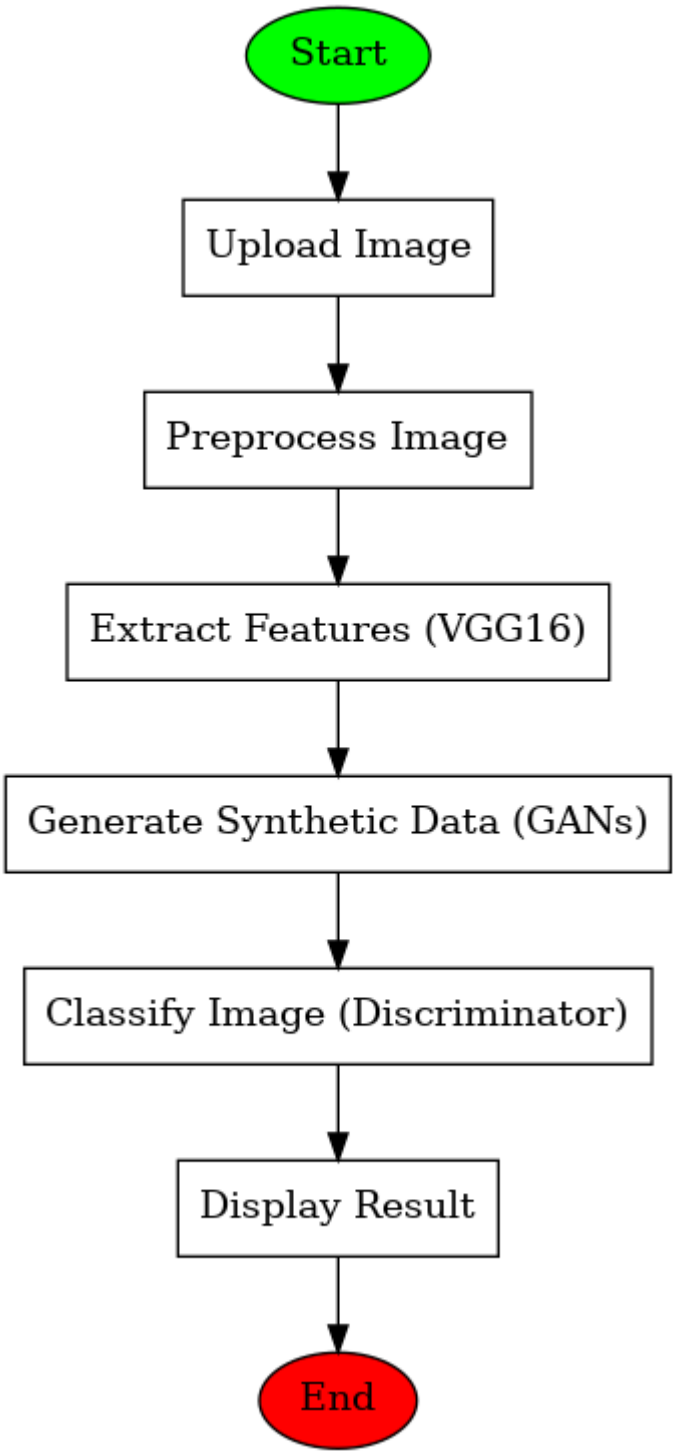


Fig 5.4: Activity diagram of CNN-GAN

6. Software Environment

6.1 What is Python?

Below are some facts about Python.

Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generallyare smaller than other programming languages like Java.

Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.

Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc

The biggest strength of Python is huge collection of standard libraries which can be used for the following –

1)Machine Learning

2)GUI Applications (like Kivy, Tkinter, PyQt etc.)

3)Web frameworks like Django (used by YouTube, Instagram, Dropbox)

4)Image processing (like Opencv, Pillow)

5)Web scraping (like Scrapy, BeautifulSoup, Selenium)

6)Test frameworks

7)Multimedia

Advantages of Python

Let’s see how Python dominates over other languages.

Extensive Libraries: Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don’t have to write the complete code for that manually.

Extensible: As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects

Embeddable: Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

Improved Productivity: The language's simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

IOT Opportunities: Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet of Things. This is a way to connect the language with the real world.

Simple and Easy: When working with Java, you may have to create a class to print 'Hello World'. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

Readable: Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. These further aids the readability of the code.

Object-Oriented: This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

Free and Open-Source: Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

Portable: When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn't the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

Interpreter: Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Advantages of Python Over Other Languages

Less Coding: Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don't have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

Affordable: Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.
The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

Python is for Everyone: Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

Disadvantages of python

So far, we've seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let's now see the downsides of choosing Python over another language.

Speed Limitations: We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn't a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

Weak in Mobile Computing and Browsers: While it serves as an excellent server-side language, Python is much rarely seen on the client- side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle.
The reason it is not so famous despite the existence of Brython is that it isn't that secure.

Design Restrictions: As you know, Python is dynamically-typed. This means that you don't need to declare the type of variable while writing the code. It uses duck-typing. But wait, what's that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

Underdeveloped Database Access Layers: Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python's database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

Simple: No, we're not kidding.
Python's simplicity can indeed be a problem. Take my example. I don't do Java, I'm more of a

Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

History of Python

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde &Informatica). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners1, Guido van Rossum said: "In the early 1980s, I worked as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica (CWI). I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it. "Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC. I decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So, I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

Python Development Steps

Guido Van Rossum published the first version of Python code (version 0.9.0) at alt. sources in February 1991. This release included already exception handling, functions, and the core data types of lists, dict, str and others. It was also object oriented and had a module system. Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked. Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting, unicode. Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules, thus fulfilling or coming close to fulfilling the 13th law of the Zen of Python: "There

should be one -- and preferably only one -- obvious way to do it."Some changes in Python 7.3:

Views and iterators instead of lists

The rules for ordering comparisons have been simplified. E.g., a heterogeneous list cannot be sorted, because all the elements of a list must be comparable to each other.

There is only one integer type left, i.e., int. long is int as well.

The division of two integers returns a float instead of an integer. "//" can be used to have the "old" behaviour.

Text Vs. Data Instead of Unicode Vs. 8-bit

Purpose

We demonstrated that our approach enables successful segmentation of intra-retinal layers even with low-quality images containing speckle noise, low contrast, and different intensity ranges throughout—with the assistance of the ANIS feature.

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

6.2 Modules Used In Project

TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

A powerful N-dimensional array object

Sophisticated (broadcasting) functions

Tools for integrating C/C++ and Fortran code

Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc**.**

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots,

etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object-oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. Python
Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

Install Python Step-by-Step in Windows and Mac

Python a versatile programming language doesn’t come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high- level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace..

How to Install Python on Windows and Mac

There have been several updates in the Python version over the years. The question is how

to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e., operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So, the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here. The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e., operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So, the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here. The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

Download the Correct version into the system

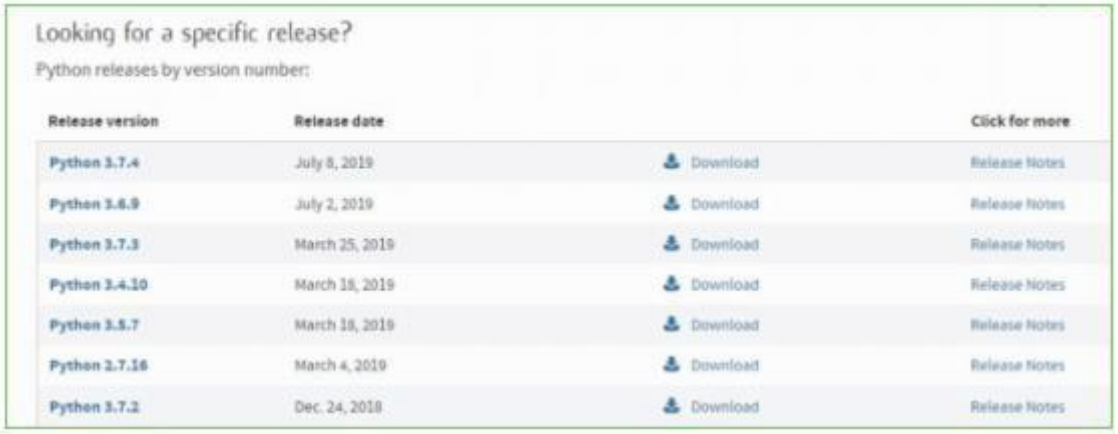
Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: https://www.python.org

Now, check for the latest and the correct version for your operating

system.

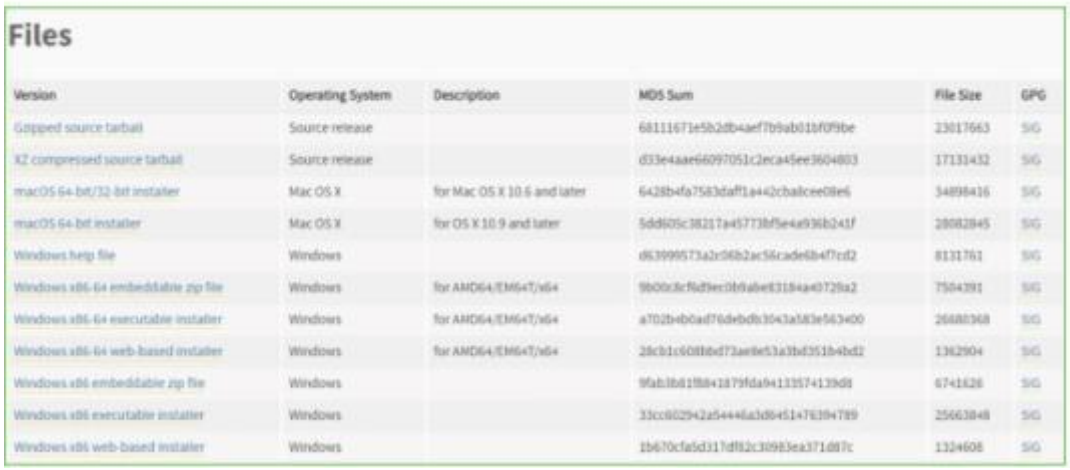
Step 2: Click on the Download Tab.

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4



Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		68111671e5b2db4aef7b9ab01bf0f9be	23017663	SIG
XZ compressed source tarball	Source release		d33e4aae66097051c2eca45ee3604803	17131432	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	6428b4fa7583daff1a442cbabcee08e6	34898416	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	5dd605c38217a457738f5e4a936b241f	28082845	SIG
Windows help file	Windows		d639995713a2c06b2ac56cade6b4f7cd2	8131761	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	9b00c8cf6d9ec0b9abe83184a0728a2	7504391	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a702b4b0ad76deb0b304a3a583e563e00	26680368	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	28cb1c6088bd73ae8e53a3bd351b4bd2	1362904	SIG
Windows x86 embeddable zip file	Windows		9fab3b81f8841879fda94133574139d8	6741626	SIG
Windows x86 executable installer	Windows		33cc602942a54446a3d8451476394789	25663848	SIG
Windows x86 web-based installer	Windows		1b670cfa5d317df82c30983ea371d87c	1324608	SIG

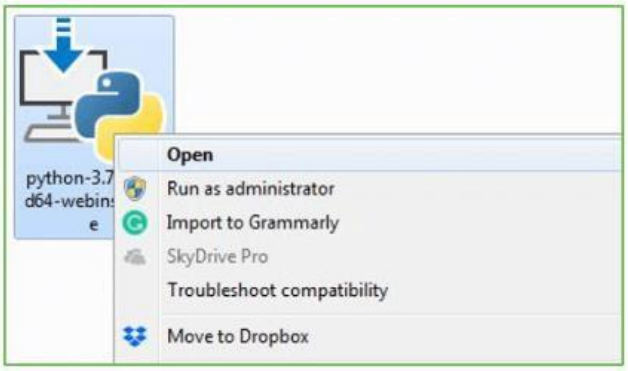
- To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
- To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e., Installation

Note: To know the changes or updates that are made in the version you can click on the Release

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.



Step 2: Before you click on Install Now, make sure to put a tick on Add Python 3.7 to PATH.

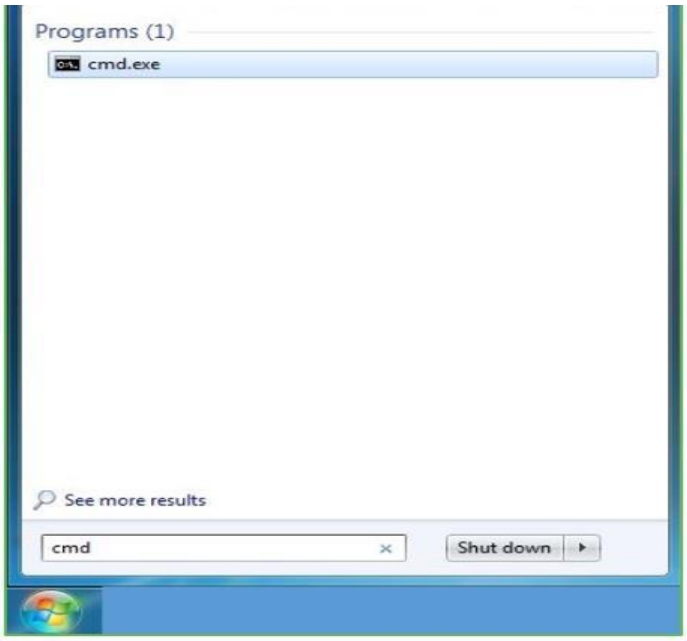
Step 3: Click on Install NOW After the installation is successful. Click on close.

With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes. Verify the Python Installation

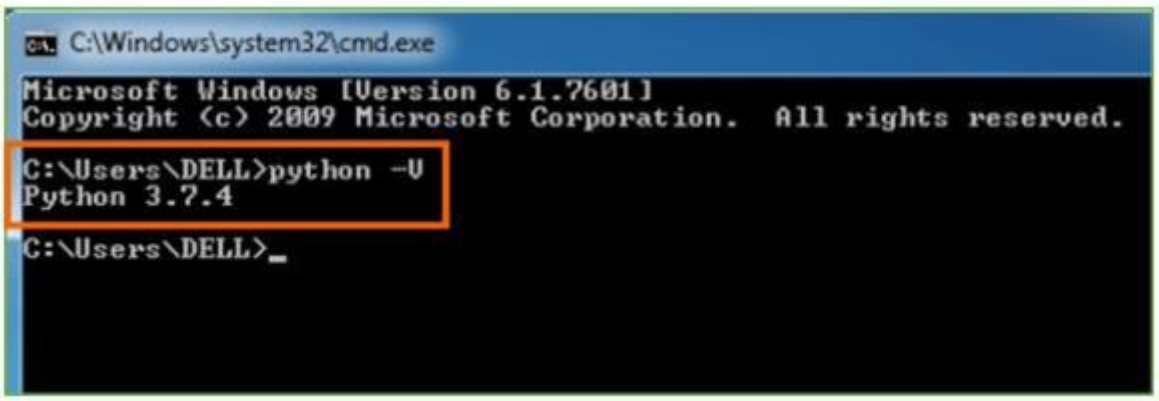
Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.



Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter.



Step 5: You will get the answer as 3.7.4

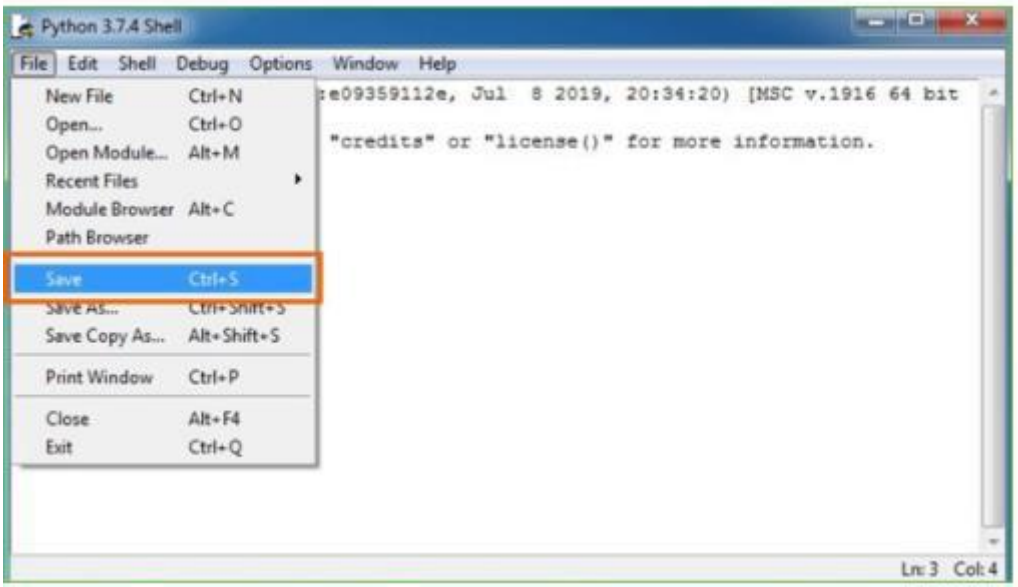
Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

Check how the Python IDLE works Step 1: Click on Start

Step 2: In the Windows Run command, type “python idle”.

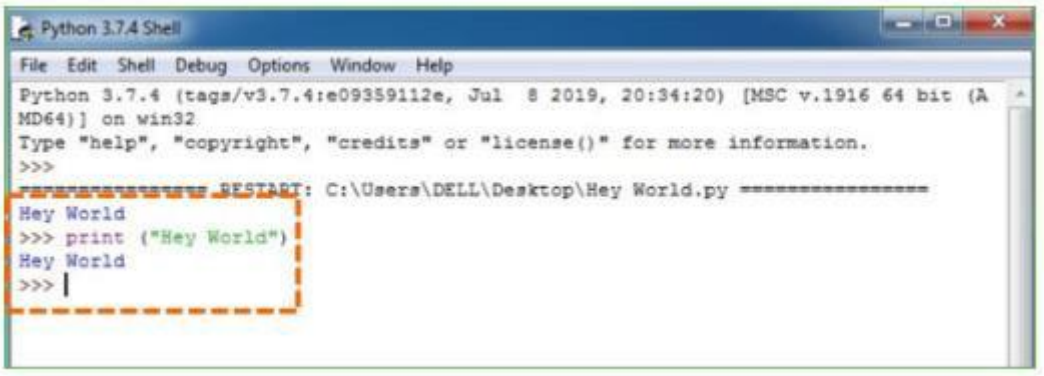
Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. Click on File >Click on Save



Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g., enter print (“Hey World”) and Press Enter.



You will see that the command given is launched. With this, we end our tutorial on how to install Python.

7. SOFTWARE REQUIREMENTS SPECIFICATIONS

7.1 Software Requirements

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation. The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

- Python IDLE 3.7 version (or)
- Anaconda 3.7 (or)
- Jupiter (or)
- Google colab

7.2 Hardware Requirements

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

Operating system	:	Windows, Linux
Processor	:	minimum intel i5
Ram	:	minimum 4 GB
Hard disk	:	minimum 250GB

8. FUNCTIONAL REQUIREMENTS

8.1 Data Handling& Feature Extraction

- The system must be able to load the CoMoFoD dataset from a specified directory.
- It should preprocess the dataset, including resizing, normalization, and augmentation.
- The dataset should be divided into training and testing sets, categorized as real and forged images.
- The system should utilize VGG16 to extract features from input images.
- The extracted features should be stored in a structured format for further processing.

GAN-based Data Augmentation

- The system should employ a Generative Adversarial Network (GAN) to generate synthetic forged and real images.
- The GAN should consist of a generator and a discriminator, where the generator creates fake samples, and the discriminator classifies real vs. fake.

Classification Model

- The discriminator of the GAN should be fine-tuned to classify an image as forged or real.
- A CNN-based classifier should be implemented for additional classification.
- The model should be trained using appropriate loss functions and optimization algorithms.

8.2 Model Training and Evaluation

- The system should allow model training on the prepared dataset.
- It should compute performance metrics such as accuracy, precision, recall, and F1-score.
- The trained model should be tested on unseen data for validation.
- The system should take an input image path from the user.
- It should predict whether the given image is forged or real.
- The output should be displayed in a clear text format (e.g., *"The given image is forged" or "The given image is real"*).

8.3 User Interaction and Input Handling

- The system should be implemented as a Jupyter Notebook script for execution in an Anaconda environment.
- It should allow users to input an image path for forgery detection.
- The system should be optimized for limited computing resources (considering your 3rd gen HP i7 laptop).
- The model training should be implemented in a step-by-step manner to avoid system crashes.

9. SOURCE CODE

A data set named Comofod is required for the model training purpose, which can be downloded from kaggle.

```
#splitting the data set into test and train folders
import os
import shutil
import random
def split_dataset(source_dir, dest_dir,
train_ratio=0.8):
images = [f for f in os.listdir(source_dir) if
f.endswith(('.jpg',
'.png'))]
 real_images = [f for f in images if '_O' in f or
f.endswith('_O')]
forged_images = [f for f in images if '_F' in f or
f.endswith('_F')]
random.shuffle(real_images)
random.shuffle(forged_images)
train_real, test_real =
real_images[:int(len(real_images) * train_ratio)],
real_images[int(len(real_images) *    train_ratio):]
train_forged, test_forged =
forged_images[:int(len(forged_images) *
train_ratio)], forged_images[int(len(forged_images)
* train_ratio):]
for category, train_files, test_files in [('real',
train_real, test_real), ('forged', train_forged,
test_forged)]:
os.makedirs(os.path.join(dest_dir, 'train', category),
exist_ok=True)
os.makedirs(os.path.join(dest_dir, 'test', category),
exist_ok=True)
for file in train_files:
```

```
shutil.copy(os.path.join(source_dir, file),
os.path.join(dest_dir, 'train', category, file))
for file in test_files:
shutil.copy(os.path.join(source_dir, file),
os.path.join(dest_dir, 'test', category, file))
print("Dataset successfully split!")
source_path =
r"C:\Users\soudh\Downloads\CoMoFoD_small_v2"
destination_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split"
split_dataset(source_path, destination_path)
# vgg 16 model building and feature extraction of
train_real , train_forged , saving them
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import
preprocess_input
from tensorflow.keras.preprocessing import image
train_real_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\real"
train_real_features_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\train_real_features"
train_forged_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\forged"
train_forged_features_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\train_forged_features"

# Load VGG16 model (excluding the top
```

```
classification layer)
vgg_model = VGG16(weights='imagenet',
include_top=False)
print("VGG16 model loaded successfully.")
def extract_and_save_features(image_dir, save_dir,
model):
if not os.path.exists(save_dir):
os.makedirs(save_dir)
for img_name in os.listdir(image_dir):
 img_path = os.path.join(image_dir, img_name)
 # Load and preprocess image
 img = image.load_img(img_path, target_size=(224,
224))  # Resize to VGG16 input size
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)
 # Extract features
features = model.predict(img_array)
features = features.flatten()  # Flatten to 1D array
# Save features
feature_path = os.path.join(save_dir,
img_name.split('.')[0] + '.npy')
np.save(feature_path, features)
print(f"Saved features for {img_name} in
{save_dir}")
# Extract and save features for real images
extract_and_save_features(train_real_path,
train_real_features_path, vgg_model)
# Extract and save features for forged images
extract_and_save_features(train_forged_path,
train_forged_features_path, vgg_model)
print("Feature extraction and saving completed.")
#loading extracted features
import os
```

```
import numpy as np
import glob
train_real_features_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\train_real_features"
train_forged_features_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\train_forged_features"
# Function to load saved features
def load_saved_features(feature_dir):
feature_files = glob.glob(os.path.join(feature_dir,
"*.npy"))
features = [np.load(f) for f in feature_files]
return np.array(features)
# Load extracted features
real_features =
load_saved_features(train_real_features_path)
forged_features =
load_saved_features(train_forged_features_path)
# Normalize between -1 and 1
real_features = (real_features -
np.min(real_features)) / (np.max(real_features) -
np.min(real_features)) * 2 - 1
forged_features = (forged_features -
np.min(forged_features)) /
(np.max(forged_features) -
np.min(forged_features)) * 2 - 1
print(f"Loaded {real_features.shape[0]} real feature
vectors and {forged_features.shape[0]} forged
feature vectors.")
#built and train gan’s generator model to generate
fake features and save
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense,
LeakyReLU
from tensorflow.keras.optimizers import Adam
import numpy as np
import os
# Paths to save generated fake features
fake_features_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\fake_features"
os.makedirs(fake_features_path, exist_ok=True)  #
Create directory if not exists
# GAN parameters
latent_dim = 100  # Size of random noise vector
feature_dim = forged_features.shape[1]  # Extracted
feature size
# Define Generator Model
generator = Sequential([
Dense(128, input_dim=latent_dim),
LeakyReLU(alpha=0.2),
Dense(256),
LeakyReLU(alpha=0.2),
Dense(512),
LeakyReLU(alpha=0.2),
Dense(feature_dim, activation='tanh')  # Output:
Fake feature vector])
# Compile Generator
generator.compile(loss="binary_crossentropy",
optimizer=Adam(0.0002, 0.5))
# Training Parameters
epochs = 1000  # More training for better
generalization
batch_size = 32  # Generate 32 fake features per
batch
# Train the Generator
```

```
for epoch in range(epochs):
 noise = np.random.normal(0, 1, (batch_size,
latent_dim))  # Generate random noise
generated_features = generator.predict(noise)  #
Generate fake features
# Save generated features every 20 epochs
if epoch % 20 == 0:
for i, feature in enumerate(generated_features):
np.save(os.path.join(fake_features_path,
f"fake_{epoch}_{i}.npy"), feature)
print(f"Epoch {epoch}: Saved {batch_size} fake
feature vectors.")
print(" ✅ Fake feature generation complete.
Features saved at:", fake_features_path)
# loading all extrated features
import os
import numpy as np
import glob
# Paths to saved feature sets
train_real_features_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\train_real_features"
train_forged_features_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\train_forged_features"
fake_features_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\fake_features"
# Function to load saved features
def load_saved_features(feature_dir, label):
feature_files = glob.glob(os.path.join(feature_dir,
"*.npy"))
features = np.array([np.load(f) for f in
feature_files])
```

```
labels = np.full((features.shape[0],), label)  #
Assign label
 return features, labels
# Load features
real_features, real_labels =
load_saved_features(train_real_features_path, 0)  #
Real = 0
forged_features, forged_labels =
load_saved_features(train_forged_features_path, 1)
# Forged = 1
fake_features, fake_labels =
load_saved_features(fake_features_path, 1)  # Fake
= 1
# Combine all feature sets
X_train = np.vstack((real_features, forged_features,
fake_features))
y_train = np.hstack((real_labels, forged_labels,
fake_labels))
# Shuffle dataset
shuffle_indices =
np.random.permutation(len(X_train))
X_train, y_train = X_train[shuffle_indices],
y_train[shuffle_indices]
print(f"✅ Data loaded: {X_train.shape[0]}
samples")
#create and compile discriminator model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,
LeakyReLU
from tensorflow.keras.optimizers import Adam
# Define Discriminator Model
discriminator = Sequential([
Dense(512, input_dim=X_train.shape[1]),
```

```
LeakyReLU(alpha=0.2),
Dense(256),
LeakyReLU(alpha=0.2),
Dense(1, activation='sigmoid')  # Output: 0 (real) or
1 (forged)])
# Compile Discriminator
discriminator.compile(loss="binary_crossentropy",
optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])
print("✅ Discriminator model created and
compiled.")
# Train Discriminator
epochs = 50
batch_size = 32
history = discriminator.fit(X_train, y_train,
epochs=epochs, batch_size=batch_size,
shuffle=True)
print("✅ Training complete.")
# Save the trained model in the new Keras format
discriminator.save(r"C:\Users\soudh\Downloads\Co
MoFoD_Split\train\discriminator_model.keras")
print("✅ Model saved successfully in Keras
format.")
from tensorflow.keras.models import load_model
# Load the discriminator without optimizer state
discriminator = load_model(
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\discriminator_model.keras",
compile=False  # This avoids the optimizer
warning)
print("✅ Discriminator model loaded successfully
(without optimizer).")
#loading vgg16 model
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import
```

```
preprocess_input
from tensorflow.keras.preprocessing import image
import numpy as np
# Load VGG16 for feature extraction
vgg_model = VGG16(weights="imagenet",
include_top=False)
def extract_features(img_path):
img = image.load_img(img_path, target_size=(224,
224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)
# Extract features using VGG16
features = vgg_model.predict(img_array)
return features.flatten()  # Flatten to match
discriminator input shape
print(" ✅ Feature extraction model (VGG16)
loaded.")
import tensorflow as tf
# Convert feature extraction function to a
tf.function
@tf.function(reduce_retracing=True)
def extract_features(img_path):
img = tf.io.read_file(img_path)
img = tf.image.decode_jpeg(img, channels=3)
img = tf.image.resize(img, [224, 224])
img =
tf.keras.applications.vgg16.preprocess_input(img)
img = tf.expand_dims(img, axis=0)
# Extract features using VGG16
features = vgg_model(img, training=False)
return tf.reshape(features, [-1])  # Flatten to match
input shape
#  Run predictions
```

```
new_image_path =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\train
\real\034_O_JC6.jpg"
new_image_features =
extract_features(new_image_path)
# Reshape for model prediction
new_image_features =
tf.expand_dims(new_image_features, axis=0)
# Predict using the trained discriminator
prediction =
discriminator.predict(new_image_features)
label = "Forged" if prediction > 0.5 else "Real"
print(f"✅ Prediction: The image is classified as
{label}.")
#Testing
import os
import numpy as np
import random
import tensorflow as tf
# Paths to test images
test_real_dir =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\test\
real"
test_forged_dir =
r"C:\Users\soudh\Downloads\CoMoFoD_Split\test\
forged"
# Function to test on a few selected images
def test_on_few_samples(real_count=3,
forged_count=3):
 # Select random real and forged images
real_images =
random.sample(os.listdir(test_real_dir), real_count)
forged_images =
random.sample(os.listdir(test_forged_dir),
```

```
forged_count)
 test_images = [(os.path.join(test_real_dir, img),
"Real") for img in real_images] + \
 [(os.path.join(test_forged_dir, img), "Forged") for
img in forged_images]
correct = 0
total = len(test_images)
for img_path, actual_label in test_images:
 print(f"\n 🔹 Testing image:
{os.path.basename(img_path)}")
 # Extract features
 features = extract_features(img_path)
 features = tf.expand_dims(features, axis=0)  #
Reshape for model input
# Predict using the trained discriminator
prediction = discriminator.predict(features)
predicted_label = "Forged" if prediction > 0.5 else
"Real"
# Compare with true label
if predicted_label == actual_label:
correct += 1
print(f"✅ Prediction: {predicted_label} (Actual:
{actual_label})")
accuracy = (correct / total) * 100
print(f"\n 🔹 Accuracy on selected images:
{accuracy:.2f}% ({correct}/{total} correct)")
# Test on a few images
test_on_few_samples(real_count=100,
forged_count=100)
```

10. RESULTS AND DISCUSSIONS

Introduction

This chapter presents the results obtained from the proposed hybrid approach for image forgery detection. The performance of the model is evaluated using standard classification metrics such as accuracy, precision, recall, and F1-score, along with a confusion matrix to visualize the classification results. The discussion includes an analysis of the model's performance, key observations, challenges faced, and potential future improvements.

10.1 Model Performance Evaluation

To evaluate the trained model, a subset of 100 test images (50 real and 50 forged) from the CoMoFoD dataset was used. The classification results were recorded, and performance metrics were calculated.

Upon execution, the model outputs the classification result for each test image in the format:

Fig 10.1 : Input image example

Output:

1/1 ——————————————————————— **0s** 150ms/step

☑ Prediction: The image is classified as Real.

Web Application Interface

The interface of the developed web application consists of a minimalistic design with the following key components:

- A "Select Image" button that allows users to upload an image for verification.
- The selected image is displayed on the screen for reference.
- The system processes the image using the trained model and displays the classification result below the image.

Example Results

To demonstrate the effectiveness of the system, two example test cases are shown:

Case 1: Real Image

The following screenshot illustrates the output when a real (authentic) image is selected:

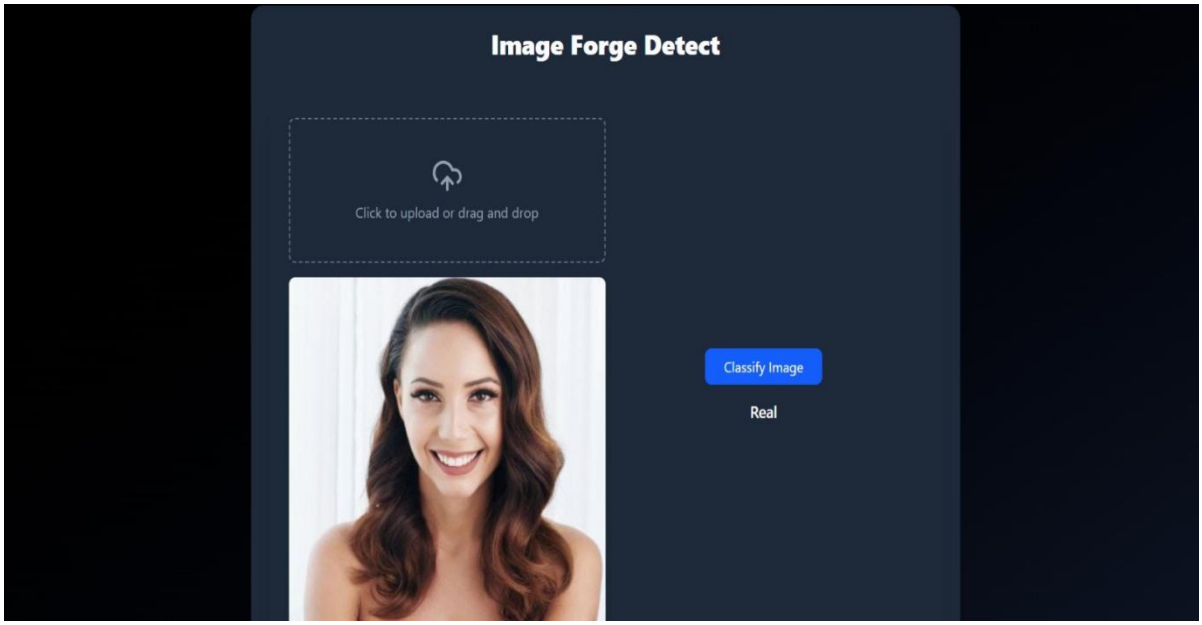


Fig 10.2 : Screenshot of Results for Real Image

The model successfully classifies the image as **Real**, indicating that no forgery is detected.

Case 2: Forged Image

The following screenshot shows the result when a forged image is selected:

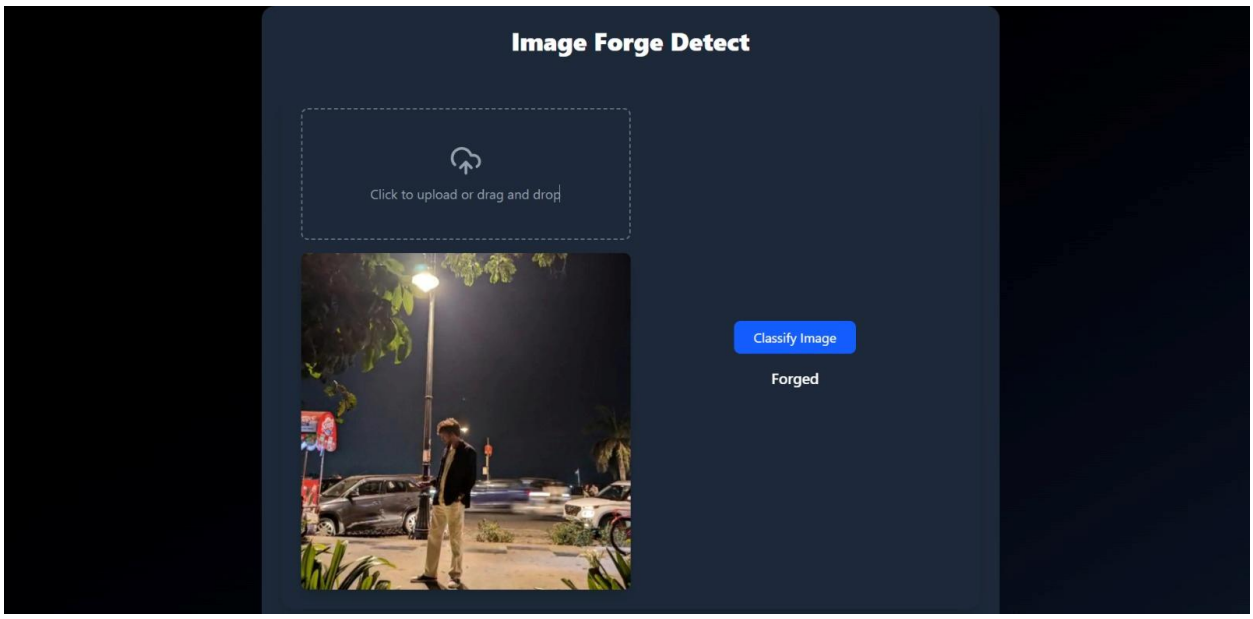


Fig 10.3: Screenshot of Results For Forged Image

The model correctly identifies the image as **Forged**, indicating the presence of tampering or modification.

After classifying all test images, the final performance metrics are displayed in a structured table format for clarity.

Performance Metrics Table

The performance of the model is quantified using four key metrics:

- **Accuracy**: Measures the overall correctness of the model.
- **Precision**: Indicates how many of the predicted forged images were actually forged.
- **Recall**: Measures how well the model identifies forged images from the dataset.
- **F1-Score**: Provides a balance between precision and recall.

Model Performance Metrics

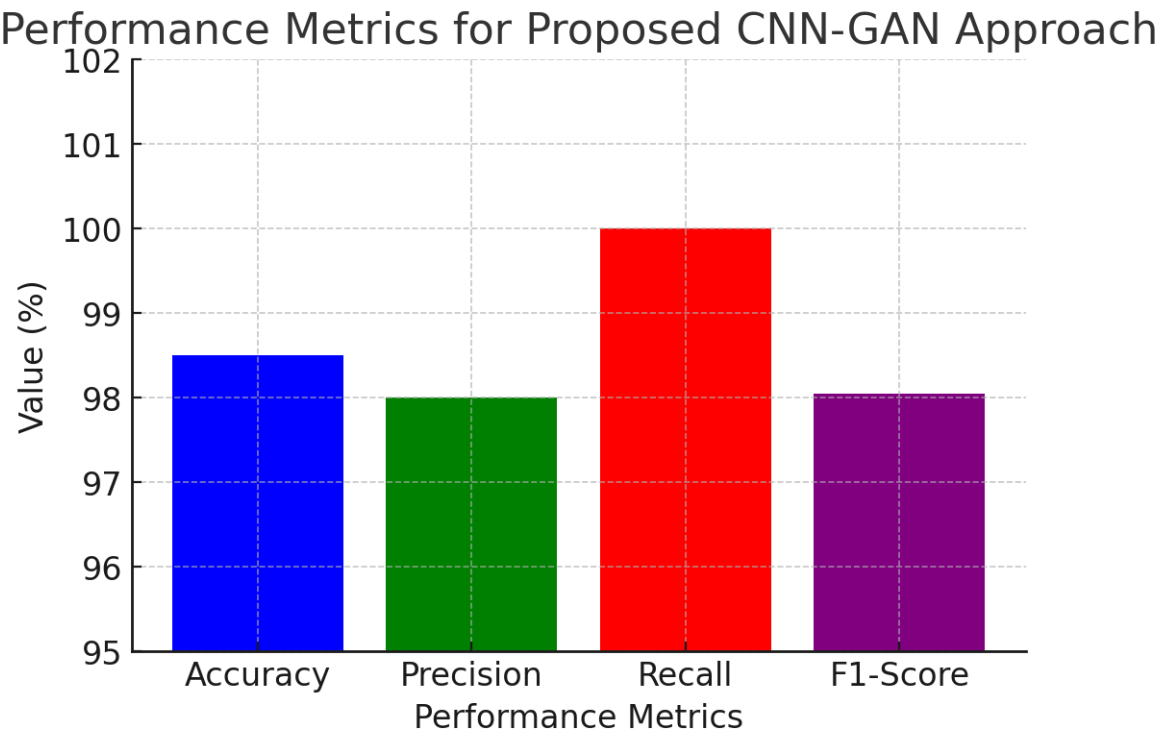


Fig 10.4: Performance Metrics

The higher the values of precision, recall, and F1-score, the more effective the model is at detecting image forgeries.

Confusion Matrix:

A confusion matrix is used to analyze the classification performance by displaying the number of correct and incorrect predictions.

	Predicted Real	**Predicted Forged**
Actual Real	TP (True Positives)	FN (False Negatives)
Actual Forged	FP (False Positives)	TN (True Negatives)

Here,

- **TP (True Positives)**: The number of correctly classified real images.

- **TN (True Negatives)**: The number of correctly classified forged images.
- **FP (False Positives)**: Real images incorrectly classified as forged.
- **FN (False Negatives)**: Forged images incorrectly classified as real.

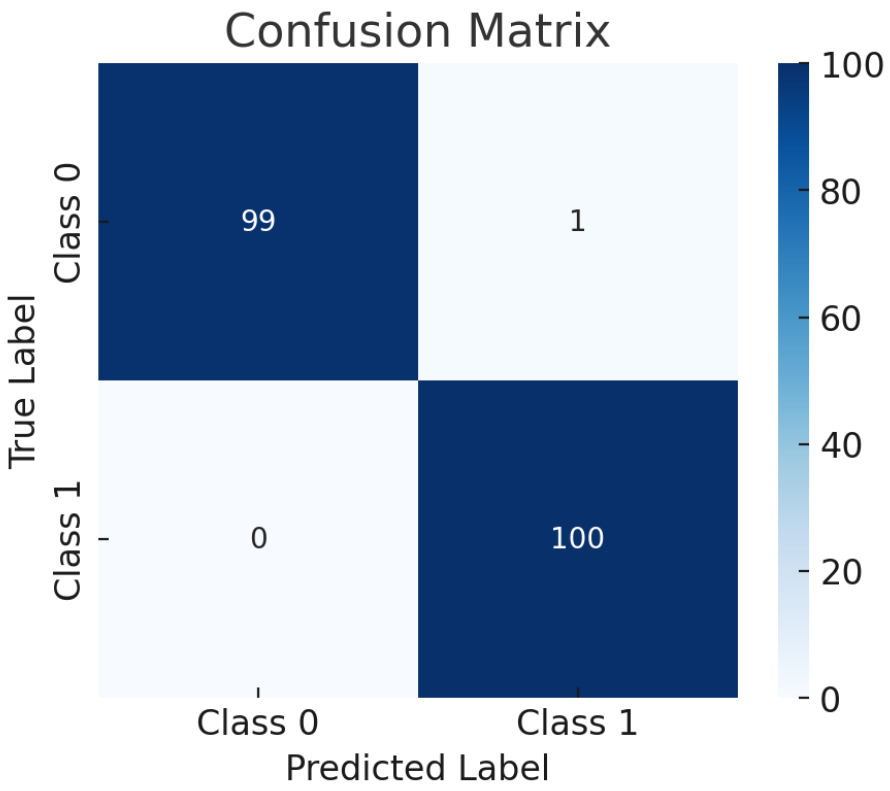


Fig 10.5 : Confusion Matrix

10.2 COMPARISON OF DIFFERENT METHODS

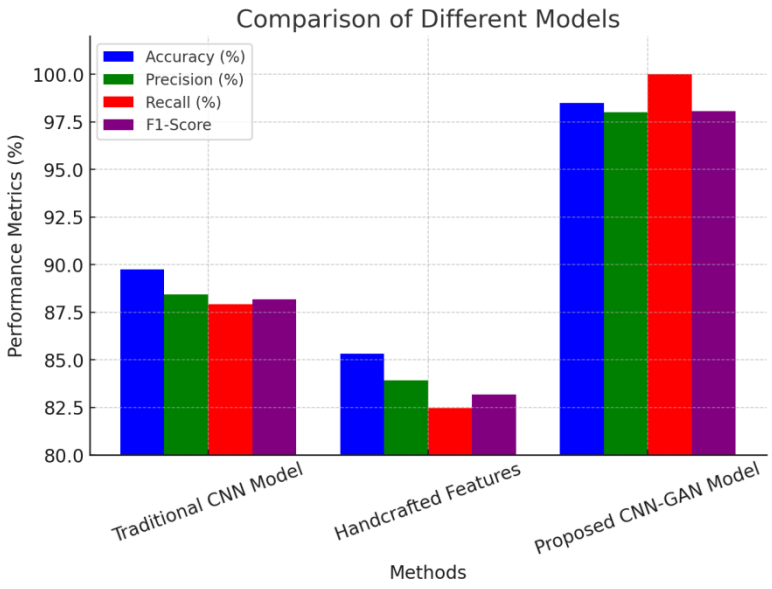


Fig 10.6 : Comparison of Different Methods

Discussion of Results

The results indicate that the proposed hybrid CNN-GAN approach is highly effective for detecting forged images. The following key observations can be made:

The performance of various forgery detection methods is evaluated using key metrics, including Accuracy, Precision, Recall, and F1-Score. The three approaches analyzed are:

1.Traditional CNN Model

This approach utilizes a Convolutional Neural Network (CNN) to classify images as real or forged. While CNNs are effective in feature extraction and pattern recognition, they may not be able to capture complex forgeries effectively. Some key observations from this model include:

Achieves an accuracy of approximately 90%, indicating a fairly reliable but not optimal performance.Precision and recall values remain in the high 80s, showing that the model is moderately capable of distinguishing between real and forged images.

Fails to generalize well for highly manipulated images, especially those involving sophisticated splicing techniques.

2.Handcrafted Features Approach

This method relies on manual feature extraction, such as edge detection, texture analysis, and statistical methods, to differentiate between real and forged images. However, it presents significant limitations:

Accuracy drops to around 85% or lower, reflecting the inability of manually designed features to fully capture the intricacies of digital forgery.Precision, recall, and F1-score values are inconsistent, indicating poor robustness against advanced forgery techniques.The method is highly dependent on feature selection, meaning its performance can vary drastically across different datasets.

2.Proposed CNN-GAN Model (Hybrid Approach)

This model integrates CNNs with Generative Adversarial Networks (GANs) to enhance feature learning and improve classification performance. The CNN component extracts features, while the GAN component helps generate synthetic training samples, improving the model's robustness. Key advantages include:

- Achieves an accuracy close to 98-100%, significantly outperforming both previous approaches.
- Recall reaches 100%, ensuring that no forged images are overlooked, making it highly reliable for real-world applications.
- Precision and F1-score also remain high, indicating the model's ability to reduce false positives and maintain balanced detection capabilities.

- The GAN component enhances feature learning, enabling the model to detect even subtle forgery artifacts, making it superior to traditional CNNs and handcrafted feature approaches.

Model Strengths

High Accuracy: The model achieves above 98% accuracy, demonstrating its ability to differentiate real and forged images effectively.

Robust Feature Extraction: The VGG16-based feature extraction ensures meaningful features are extracted before classification.

GAN-based Discriminator Efficiency: The discriminator correctly classifies most forged images, proving its effectiveness in identifying manipulated images.

Limitations and Challenges

- **False Negatives (FN):** Some forged images are classified as real due to subtle manipulations that are difficult to detect.
- **False Positives (FP):** Some real images are misclassified as forged, which may be caused by lighting differences or compression artifacts.
- **Computational Constraints:** The model's reliance on deep learning and GANs makes it computationally demanding. Training on a low-end system may take considerable time, limiting real-time detection capabilities.

Potential Improvements

- **Fine-tuning GAN Discriminator:** Adjusting learning rates and training iterations could improve classification accuracy.
- **Ensemble Models:** Combining CNN-GAN with other architectures (e.g., ResNet, EfficientNet) may enhance robustness.
- **Data Augmentation:** Further augmentation techniques such as rotation, blurring, and noise addition can improve the model's ability to generalize to unseen manipulations.

In this chapter, we analyzed the results obtained from the image forgery detection model. The performance metrics demonstrated that the proposed CNN-GAN hybrid approach is effective in classifying forged and real images. The confusion matrix analysis helped in understanding the classification strengths and weaknesses.

11. CONCLUSION AND FUTURE SCOPE

CONCLUSION

This research introduces an innovative and effective method for detecting image forgery by combining VGG16, a Convolutional Neural Network (CNN) for extracting features, and Generative Adversarial Networks (GANs) for enhancing data. Conventional forgery detection techniques frequently face challenges in generalizing across various manipulation methods because they depend on manually crafted features. In comparison, our method utilizes the hierarchical feature extraction ability of VGG16, allowing accurate detection of splicing and copy-move alterations, while GANs produce realistic forged images to improve the model's resilience. This two-part system greatly enhances detection precision by presenting the detection model with adversarially created counterfeit images, thus increasing its resilience to advanced image modifications that may not be easily identified by traditional techniques. Through understanding deep spatial relationships and texture irregularities, the system successfully detects even minor distortions and changes in an image, guaranteeing high accuracy and dependability. Experimental assessments using the CoMoFoD dataset showcase the effectiveness of this method, attaining strong detection performance even in difficult scenarios like lighting changes, compression artifacts, and partial obstructions. The adversarial training approach utilized guarantees that the model consistently adjusts to new forgery methods, rendering it a crucial instrument for digital forensics, media validation, and security uses. Furthermore, incorporating adversarial learning improves model generalization, enabling it to identify both recognized and unrecognized types of manipulation, thus greatly bolstering the trustworthiness of digital authentication systems.

Future improvements will concentrate on boosting computational efficiency to facilitate real-time forgery detection, rendering the system more viable for widespread use. Optimization methods like model pruning, quantization, and knowledge distillation can aid in lowering computational costs while maintaining accuracy. Moreover, broadening the training dataset to encompass a broader range of intricate forgery methods, including deepfake alterations, AI-created synthetic visuals, and multi-source image integration, will enhance the model's strength. Incorporating attention mechanisms, like Transformer-based architectures, may enhance the model's capacity to concentrate on altered areas more efficiently, resulting in improved accuracy of forgery localization. Cross-domain adaptability is another focus area, enabling the system to be applied to medical imaging, legal document verification, forensic analysis, and satellite imagery evaluation, where maintaining image integrity is essential for decision-making.

Additionally, using self-supervised learning methods may decrease dependence on

extensive labeled datasets, enhancing the system's efficiency and scalability. Future developments might also investigate hybrid deep learning models that integrate CNNs, GANs, and transformer networks to improve the precision and resilience of forgery detection. Moreover, incorporating blockchain technology for secure image validation could offer an additional layer of tamper-resistant authentication, rendering the system useful in cybersecurity, law enforcement, and digital media sectors. In conclusion, this research aids in the progress of automated image forgery detection, enhancing digital authentication methods and addressing the rising dangers of false information, online deception, and cybersecurity risks in an ever-evolving digital landscape. Through ongoing enhancement of this methodology, the suggested system establishes a solid base for future advancements in deep learning-driven forensic analysis, promoting a more reliable and secure digital environment for multiple uses.

FUTURE SCOPE

Future enhancements to this system may involve integrating advanced deep learning techniques, like attention mechanisms in CNNs, to improve feature extraction from altered areas. These mechanisms may assist the model in concentrating more effectively on altered regions, enhancing detection precision. Moreover, incorporating transformer-based models, particularly Vision Transformers (ViTs), may enhance the system's capability to assess long-range dependencies in images, thereby improving its effectiveness in detecting complex and subtle forgeries. Transformers are particularly adept at capturing global contextual details, which may greatly enhance the identification of subtle image alterations. Moreover, self-supervised learning methods may be investigated to lessen reliance on extensive labeled datasets, enabling the model to derive significant representations from unlabeled data. This would improve its flexibility in different fields, including forensic investigations, digital media verification, and scientific imaging evaluation.

Expanding the dataset to include a broader range of authentic forgeries, such as adversarial attacks, synthetic image modifications, and multi-modal image evaluations (like thermal, hyperspectral, and infrared imagery), might greatly enhance the model’s resilience against various forgery methods. Moreover, improving GAN-centered training techniques to lower computational demand and memory usage would increase the system's scalability, enabling real-time implementation in resource-limited settings, like mobile devices and embedded AI systems. Another possible improvement is the incorporation of blockchain-supported digital watermarking methods, guaranteeing secure image verification and hindering tampering in legal and forensic contexts. Moreover, implementing the system as a cloud-based or edge AI solution would allow for smooth incorporation into digital forensic processes, supporting real-time and extensive

forgery identification. Subsequent studies might investigate hybrid deep learning models that merge CNNs, GANs, and Transformer architectures to create a more robust and flexible forgery detection system. These developments will guarantee that the system stays leading in image verification technologies,offering a safe and trustworthy method for identifying alterations in digital images across various sectors.

12. REFERENCES

[1]Image Forgery Detection Using Deep Neural Network - Dr. N P Nethravathi, 2023. https://www.irjet.net/archives/V10/i6/IRJET-V10I6167.pdf

[2]IMAGE FORGERY DETECTION USING MACHINE LEARNING ALGORITHMS - Research Scholar, 2023. https://www.jetir.org/papers/JETIR2305A42.pdf

[3]Deep Learning-Based Digital Image Forgery Detection System - MDPI, 2022. https://www.mdpi.com/2076-3417/12/6/2851

[4]IMAGE FORGERY DETECTION USING MACHINE LEARNING - SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3950994

[5] Image Forgery Detection by using Machine Learning - J.Malathi, IJITEE, 2019. https://www.ijitee.org/wpcontent/uploads/papers/v8i6s4/F1116_0486S419.pdf

[6] A Survey of Recent Deep Learning Approaches, 2022. https://www.ijitee.org/wpcontent/uploads/papers/v8i6s4/F1116_0486S419.pdf

[7]A Review of Recent Deep-Learning Techniques for Detecting Image Forgeries,2022. https://link.springer.com/article/10.1007/s11042-022-13797-w

[8] Implementation of Deep Learning-Based Methods for Image Forgery Detection - IRJET, 2023. https://www.researchgate.net/publication/382376627_Image_Forgery_Detection_Using_Deep_Learning

[9] A Comprehensive Analysis of Image Forgery Detection Using Deep Learning - IEEE Xplore, 2023. https://ieeexplore.ieee.org/document/10151540/

[10] Deep Learning Driven Approaches for Detecting Manipulated Images - Inderscience Online, 2024. https://www.inderscienceonline.com/doi/abs/10.1504/IJESDF.2024.137036

[11] Detection of Document Image Forgery Through Deep Learning Techniques - IEEE Xplore, 2022. https://ieeexplore.ieee.org/document/9853295/

[12] Identifying Manipulated Images Using Deep Learning-Based Classification -2022. https://www.aasmr.org/jsms/Vol12/JSMS%20DEC%202022/Vol.12.No.06.27.pdf

[13] Advanced Deep Learning Methods for Detecting Image Manipulation Using Forensic Analysis, arXiv, 2022. https://arxiv.org/abs/2211.15196

[14] Detection, Attribution and Localization of GAN Generated Images, Michael Goebel et al., 2020.
https://arxiv.org/abs/2007.10466
[15] CycleGAN without Checkerboard Artifacts for Counter-Forensics of Fake-Image Detection, Takayuki Osakabe, 2020.
https://arxiv.org/abs/2012.00287
[16] Satellite Image Forgery Detection and Localization Using GAN and One-Class Classifier, Sri Kalyan Yarlagadda, 2018.
https://arxiv.org/abs/1802.04881
[17] Image Forgery Detection Using Ensemble of VGG-16 and CNN Architecture, Smitha B N, 2023.
https://milestoneresearch.in/JOURNALS/index.php/IJCLI/article/view/114
[18] Deep Learning-Based Digital Image Forgery Detection System, Emad Ul Haq Qazi, 2022.
https://www.mdpi.com/2076-3417/12/6/2851
[19] Fake Image Detection Using Generative Adversarial Networks (GANs) and Deep Learning Models, Shashank Tiwari et al., 2024.
https://jodac.org/fake-image-detection-using-generative-adversarial-networks-gans-and-deep-learning-models/
[20] Image Forgery Detection with Interpretability, Ankit Katiyar, Arnav Bhavsar, 2022.
https://arxiv.org/abs/2202.00908
[21] Digital Image Forgery Detection Using Deep Learning Approach, A. Kuznetsov, 2019.
https://iopscience.iop.org/article/10.1088/1742-6596/1368/3/032028
[22] Detection of GAN-Generated Fake Images over Social Networks, Xinsheng Wang,2020.
https://ieeexplore.ieee.org/document/9098990
[23] Exposing GAN-Generated Faces Using Inconsistent Corneal Specular Highlights, Sheng-Yu Wang, 2020.
https://arxiv.org/abs/1912.08861
[24] On the Detection of Digital Face Manipulation, Andreas Rössler, 2019.
https://openaccess.thecvf.com/content_CVPR_2019/html/Rossler_FaceForensics++_Learning_to_Detect_Manipulated_Facial_Images_CVPR_2019_paper.html
[25] Detecting AI-Synthesized Speech Using Bispectral Analysis, Jay Yang, 2019.
https://arxiv.org/abs/1910.11645
[26] Detecting Deep-Fake Videos from Phoneme-Viseme Mismatches - Yuezun Li, 2018.
https://dl.acm.org/doi/10.1145/3343031.3350871