Text Encryption and Decryption using Gen AI

Project Report Submitted

To

Gujarat University

In partial fulfilment of the requirements for

the award to the Degree of

5 YEAR INTEGRATED MASTER OF SCIENCE (COMPUTER SCIENCE)

SEMESTER – VIII

GUIDED BY:

Ms. Saloni Sah

SUBMITTED BY:

Ansh Yadav (80021)

Trupal Lathiya (80007)

DEPARTMENT OF COMPUTER SCIENCE

GUJARAT UNIVERSITY, AHMEDABAD

YEAR: 2024-25

Department Of Computer Science

Gujarat University

Certificate

Roll No : 14 *Seat No :* 80007

This is to certify that Mr. /Ms. Lathiya Trupal v *student of Eigthth Semester of 5 years Integrated M.Sc (Computer Science) has duly completed his/her project titled* Text Encryption and Decryption using Gen AI *for the semester ending in June 2025, towards partial fulfillment of degree of 5 years Integrated M.Sc (Computer Science).*

Date of Submission

Internal Project Guide

Ms. Saloni Sah

Course Coordinator

Head of Department

Department Of Computer Science

Gujarat University

Certificate

Roll No : 36 *Seat No :* 80021

This is to certify that Mr. /Ms. Yadav Ansh J *student of Eigthth Semester of 5 years Integrated M.Sc (Computer Science) has duly completed his/her project titled* Text Encryption and Decryption using Gen AI *for the semester ending in June 2025, towards partial fulfillment of degree of 5 years Integrated M.Sc (Computer Science).*

Date of Submission

Internal Project Guide
Ms. Saloni Sah

Course Coordinator

Head of Department

ACKNOWLEDGEMENT

We extend our heartfelt gratitude to **Dr. Hiren Joshi**, Head of the Department of Computer Science, for his support and leadership. We are especially thankful to our Course Coordinator, **Dr. Jyoti Pareek**, for her unwavering encouragement and continuous guidance throughout the course of this project. Her mentorship and generous sponsorship laid the foundation for this research. Dr. Jyoti Pareek's insightful feedback, strategic direction, and active involvement were instrumental in shaping the project's scope and connecting us with key resources that greatly enhanced the overall quality and impact of our work.

We would also like to extend our heartfelt thanks to **Ms. Saloni Sah** for her expert guidance and constructive feedback at every critical stage of the project. Her clarity of thought and deep understanding of the subject matter enabled us to refine our research questions and maintain focus on our objectives. Her meticulous review of the methodology ensured the scientific rigor and validity of our approach. She consistently offered practical insights and thoughtful critique, which were instrumental in shaping the final outcome of this study.

Their encouragement, patience, and belief in our capabilities have been a source of constant motivation. We deeply appreciate the time, energy, and expertise they invested in our project. It is through their support that we were able to overcome challenges and successfully complete this research.

We are sincerely indebted to **Dr.Hiren Joshi, Dr. Jyoti Pareek** and **Ms. Saloni Sah** for their invaluable contributions and mentorship, which have had a lasting impact on our academic and professional growth.

INDEX

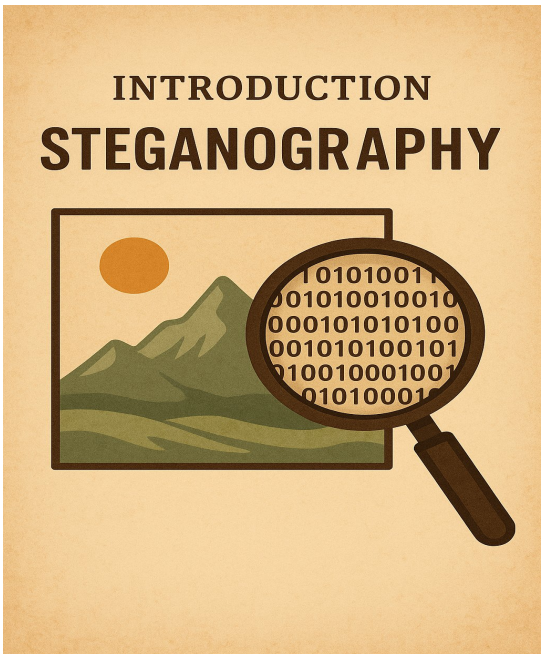
ABSTRACT

In the modern landscape of digital communication, ensuring both the confidentiality and undetectability of sensitive information has become paramount. Traditional encryption techniques, while effective in protecting content, often fail to conceal the presence of a message—leaving communication vulnerable to detection and analysis. This project introduces a comprehensive, multi-layered framework that combines steganography, artificial intelligence, and cryptographic protocols to achieve secure, covert, and intelligent data transmission.

At its core, the system employs **Least Significant Bit (LSB) steganography** for subtle and efficient data embedding, enhanced through the use of **Generative Adversarial Networks (GANs)**, which optimize concealment and minimize detectability. To ensure robust content protection, the embedded data is encrypted using lightweight yet powerful **cryptographic algorithms** such as AES. The framework further incorporates **pose estimation** to intelligently identify key visual features for context-aware embedding, and **Multi-View Stereo (MVS)** to enable depth-aware 3D data embedding that remains resilient across varying perspectives and image transformations.

By synergizing these advanced technologies, the proposed system achieves a high-capacity, imperceptible, and secure method of embedding and retrieving sensitive data. Its practical applications span military communication, digital rights management, medical imaging, surveillance, and identity authentication—offering a versatile solution for modern secure communication challenges.

1. INTRODUCTION



In the modern era of digital communication, the secure transmission of sensitive data has become increasingly critical. Whether for personal privacy, corporate confidentiality, or national security, the need for secure and covert communication channels is paramount. Traditional encryption methods protect the content of a message, but they often fail to conceal the very presence of that message. This makes them susceptible to traffic analysis, interception, and targeted attacks.

To address this issue, the field of **steganography**—the art and science of hiding information in plain sight—has gained significant attention. Among the various steganographic methods, **Least Significant Bit (LSB) steganography** is one of the most popular due to its simplicity and efficiency. LSB steganography works by embedding hidden data into the least significant bits of image pixels, altering them in a way that is imperceptible to the human eye. However, traditional LSB techniques are vulnerable to detection through steganalysis, especially when applied indiscriminately or without optimization.

The integration of **Artificial Intelligence (AI)**, particularly **Generative Adversarial Networks (GANs)**, offers a new horizon in the evolution of steganographic methods. GANs, through their adversarial learning architecture, can learn to generate images that not only look realistic but also carry embedded information in a way that is nearly undetectable—even by sophisticated forensic tools. By optimizing the data embedding process, GANs enhance the stealth and robustness of hidden communication.

Furthermore, the inclusion of **cryptographic techniques** within the steganographic pipeline ensures that even if the embedded message is extracted, it remains encrypted and inaccessible to unauthorized entities. This dual-layer security approach—combining steganography and cryptography—significantly boosts the overall strength of the system.

1.2 Project Overview

This project presents an advanced and comprehensive approach for secure data transmission by embedding sensitive information within digital images using a fusion of steganography, AI techniques, and cryptographic protocols. The core of this framework relies on:

1. **Least Significant Bit (LSB) Steganography** for efficient and minimal-distortion data embedding.
2. **Generative Adversarial Networks (GANs)** for optimizing concealment and resisting detection.
3. **Cryptographic Techniques** for robust encryption of the hidden data.
4. **Pose Estimation** to identify key visual features for strategic and intelligent embedding.
5. **Multi-View Stereo (MVS)** for constructing depth-aware 3D models, ensuring robust embedding across perspectives.

By synergizing these technologies, the system aims to achieve not only high-capacity and undetectable data embedding but also secure, intelligent, and context-aware data retrieval.

1.3 Least Significant Bit (LSB) Steganography

LSB steganography involves manipulating the least significant bits of pixel values in an image. Since changes in these bits do not significantly affect the overall visual appearance of the image, this method allows for data to be embedded subtly and invisibly.

Consider an 8-bit grayscale image where each pixel value ranges from 0 to 255. Changing the least significant bit of a pixel alters its value by a maximum of ±1, which is imperceptible to human vision. If done carefully, LSB steganography can embed large amounts of data without degrading image quality.

1.4 Generative Adversarial Networks (GANs) for Enhanced Steganography

GANs consist of two neural networks—a **generator** and a **discriminator**—competing against each other. The generator attempts to create realistic images (or embed data imperceptibly), while the discriminator tries to distinguish between genuine and modified images. Through this adversarial training, the generator learns how to embed data more effectively while maintaining the realism of the host image.

GANs are employed to:

- Learn optimal locations within the image for data embedding.

- Reduce the statistical artifacts that commonly result from traditional steganographic methods.
- Generate stego-images that are visually and statistically indistinguishable from natural images.

This AI-driven optimization significantly enhances the stealth of the hidden data, making the system resilient against common detection techniques.

1.5 Cryptographic Layer for Data Security

While steganography hides the existence of a message, it does not encrypt its content. Therefore, embedding raw sensitive data—even if undetectable—poses a risk if an adversary somehow retrieves the hidden bits. To mitigate this, our system employs **standard cryptographic algorithms**, such as AES (Advanced Encryption Standard), to encrypt the data before embedding.

This ensures a second layer of protection, meaning that even if the stego-content is intercepted or partially extracted, the information remains protected from unauthorized access. The cryptographic layer is lightweight yet highly effective, balancing performance and security.

1.6 Pose Estimation for Intelligent Feature Extraction

Pose estimation is a computer vision technique used to detect and track the key points of the human body or objects in an image or video. In this project, pose estimation enables **intelligent and context-aware embedding** of hidden data.

Rather than embedding information uniformly across the image, the system identifies significant visual features—such as limbs, joints, or facial landmarks—and uses these regions for embedding. These areas are more complex and variable, making embedded data less susceptible to detection. Additionally, by focusing on regions that naturally contain higher levels of visual information, the changes introduced by embedding become even less perceptible.

Pose estimation also assists in maintaining consistency during data retrieval by acting as a reference for locating embedded data across different perspectives or frames.

1.7 Multi-View Stereo (MVS) for 3D Reconstruction and Robustness

Multi-View Stereo (MVS) is a technique used to reconstruct detailed 3D models of a scene or object from multiple 2D images taken from different viewpoints. In the context of this project, MVS plays a vital role in enabling **depth-aware embedding**.

By understanding the 3D structure of a scene, MVS allows for data to be embedded in a manner that respects spatial consistency and depth. This is particularly useful in scenarios where the stego-image might be viewed from different angles or subjected to transformations. MVS ensures that the embedded data can be accurately retrieved even when the image is distorted, cropped, or rotated.

This depth-aware approach not only improves resilience against tampering and data loss but also opens the door to embedding data across a **sequence of images or video frames**, further enhancing the system's capacity and flexibility.

1.8 Applications

The proposed system can be applied in various domains, including:

- **Military and Intelligence Communication**: Secure and covert data exchange in high-risk environments.
- **Digital Rights Management (DRM)**: Embedding ownership or licensing information in media content.
- **Medical Imaging**: Securing patient data embedded within diagnostic images.
- **Surveillance and Security**: Embedding metadata in visual streams without compromising visual clarity.
- **Authentication Systems**: Embedding verifiable data in ID images for authentication.

2. LITERATURE SURVEY

2.1. Title

DreamBooth: Fine-Tuning Text-to-Image Diffusion Models for Subject-Driven Generation (https://arxiv.org/pdf/2208.12242)

2.1.1. Year

25 Aug 2022

2.1.3. Description

DreamBooth presents a novel method for customizing powerful text-to-image diffusion models, such as Imagen or Stable Diffusion, to generate images of specific real-world subjects with high fidelity and flexibility. The core idea is to fine-tune a pre-trained diffusion model using just a few images (as few as 3–5) of a target subject. This fine-tuning teaches the model to associate a rare, user-defined identifier (e.g., "a photo of *[identifier]* dog") with the visual characteristics of that subject. Once trained, the model can generate photorealistic images of the subject in various contexts, poses, scenes, and lighting conditions, all guided by text prompts.

To achieve this personalization while maintaining generalization capabilities, DreamBooth introduces a **class-specific prior preservation loss**. This loss function ensures that during fine-tuning, the model doesn't forget how to generate generic members of the class (e.g., other dogs or people), thereby preserving its expressive power and avoiding overfitting to the limited subject images. This is crucial for generating new images that maintain the subject's identity but still reflect prompt-guided variety in style, environment, and composition.

The technique unlocks a wide range of capabilities in subject-driven image generation. For example, users can recontextualize a pet, a person, or a product into different environments (e.g., "a photo of [identifier] in a medieval village"), apply artistic styles ("a painting of [identifier] by Van Gogh"), or synthesize novel views and angles ("a side view of [identifier] under sunset light"). These outputs preserve key visual features such as facial structure, fur texture, or distinctive markings—attributes learned from the limited subject images.

DreamBooth is impactful not just as a personalization tool, but as a foundational technique in generative AI. It serves as a critical component in systems like DreamCraft3D, where subject-specific knowledge must be embedded into diffusion priors for view-consistent 3D texture generation. By enabling powerful and flexible subject-aware image synthesis, DreamBooth bridges the gap between generic generative models and individualized creative tools.

2.1.4. Result

Method	DINO ↑	CLIP-I ↑	CLIP-T ↑
Real Images	0.774	0.885	N/A
DreamBooth (Imagen)	**0.696**	**0.812**	**0.306**
DreamBooth (Stable Diffusion)	0.668	0.803	0.305
Textual Inversion (Stable Diffusion)	0.569	0.780	0.255

Table 1. Subject fidelity (DINO, CLIP-I) and prompt fidelity (CLIP-T, CLIP-T-L) quantitative metric comparison.

Method	Subject Fidelity ↑	Prompt Fidelity ↑
DreamBooth (Stable Diffusion)	**68%**	**81%**
Textual Inversion (Stable Diffusion)	22%	12%
Undecided	10%	7%

Table 2. Subject fidelity and prompt fidelity user preference.

2.2. Title

DreamCraft3D: Hierarchical 3D Generation with Bootstrapped Diffusion Prior
(https://arxiv.org/pdf/2310.16818)

2.2.1. Year

25 Oct 2023

2.2.3. Description

DreamCraft3D is a novel hierarchical framework for generating high-fidelity and coherent 3D objects from a single 2D reference image. Its design tackles a central challenge in 3D content creation: achieving both geometric coherence and photorealistic texture quality. Traditional methods often suffer from issues like inconsistent object shapes across views or textures that appear blurry and misaligned. DreamCraft3D addresses these shortcomings through a two-stage pipeline that separates geometry generation from texture refinement, while introducing mechanisms for the two to improve one another over time.

In the first stage, DreamCraft3D constructs a consistent 3D geometry using a process called Score-Distillation Sampling (SDS), guided by a view-dependent diffusion model. This model provides strong supervision by using a pre-trained 2D image generator to estimate what the object should look like from different camera viewpoints. By optimizing a 3D neural representation (such as a neural radiance field or mesh), the system shapes the object in a way that remains coherent across multiple perspectives. However, this stage prioritizes geometric consistency over texture detail, which can result in oversimplified or low-fidelity surface appearances.

To overcome the limitations of texture quality, DreamCraft3D introduces Bootstrapped Score Distillation in the second stage. This technique trains a personalized, 3D-aware diffusion model based on DreamBooth, which is fine-tuned on renderings of the generated 3D object. These renderings are produced from diverse viewpoints and undergo augmentation to simulate various lighting, orientation, and visual conditions. As a result, the DreamBooth model becomes intimately familiar with the object's shape and appearance, allowing it to act as a powerful prior that enhances texture realism while preserving geometric integrity.

The process is iterative: as the 3D scene improves, new renderings are created and used to further fine-tune the DreamBooth model. This in turn provides better gradients during score distillation, which helps refine the texture on the 3D model even more. This bootstrapped learning loop leads to mutual improvement between the diffusion model and the 3D representation, ensuring that texture fidelity and geometric coherence evolve together. By continually refining both components, DreamCraft3D ensures that the final 3D object looks realistic and remains consistent from all angles.

This system represents a significant step forward in 3D content generation. By combining techniques from neural rendering, diffusion modeling, and image-based supervision, DreamCraft3D

delivers photorealistic 3D models that align closely with the visual characteristics of the input image. Its approach balances shape accuracy and texture detail, solving a long-standing problem in the field. Compared to prior methods like DreamFusion, which often suffer from artifacts and inconsistency, DreamCraft3D produces cleaner, more realistic, and more usable 3D assets. These qualities make it highly suitable for applications in virtual reality, gaming, digital art, and e-commerce, where visual fidelity and spatial coherence are critical.

Overall, DreamCraft3D not only advances the state of the art in single-image 3D reconstruction but also establishes a flexible framework that can be extended to more complex scenes or integrated with interactive design tools. Its modular design and innovative training loop position it as a powerful solution for the future of automated 3D content creation.

2.2.4. Result

	CLIP ↑	Contextual ↓	PSNR ↑	LPIPS ↓
Make-it-3D	0.872	1.609	18.937	0.054
Magic123	0.843	1.628	22.838	0.053
DreamCraft3D	**0.896**	**1.579**	**31.801**	**0.005**

3.3. Title
Alias-Free Generative Adversarial Networks (https://arxiv.org/abs/2106.12423)

3.3.1. Year
2021

3.3.3. Description

This paper introduces a novel architectural approach to address a persistent issue in traditional GANs—aliasing due to improper signal processing. Conventional GANs rely heavily on absolute pixel coordinates, resulting in image artifacts where details appear fixed to the image grid rather than the depicted objects. This weakens spatial generalization and can produce unrealistic outputs, especially when applied to dynamic scenes like video.

In recent years, **Generative Adversarial Networks (GANs)** have dominated the field of high-fidelity image synthesis, with models such as StyleGAN2 setting benchmarks in terms of image quality and realism. However, GANs are often plagued by training instabilities and issues like mode collapse. To address these shortcomings, researchers have explored alternative generative modeling approaches, among which **diffusion probabilistic models** have emerged as a promising and robust direction. These models offer a principled likelihood-based framework, enabling better theoretical understanding and improved sample diversity.

The foundational concept behind diffusion models involves a two-stage process: a **forward diffusion process** that gradually adds Gaussian noise to the data, and a **reverse denoising process** that reconstructs the original data from noise. While initially considered less practical due to high sampling costs, recent advances—particularly by **Ho et al. (2020)**—demonstrated that simplified objective functions could lead to competitive results. Dhariwal and Nichol (2021) build upon this work by proposing architectural and training enhancements that significantly improve sample quality. In their work, the authors introduce several innovations that elevate the performance of diffusion models. Notably, they employ **classifier-free guidance**, which allows the model to leverage label information without depending on an external classifier. This not only improves sample fidelity but also simplifies the training pipeline. They also adopt **U-Net architectures** with attention mechanisms, which enhance spatial reasoning in high-resolution images. These changes lead to state-of-the-art performance on datasets like ImageNet, achieving better FID (Fréchet Inception Distance) scores than GANs at various resolutions.

The success of this work challenges the prevailing belief that GANs are superior for realistic image synthesis. It also suggests that **likelihood-based models**, once considered inferior in visual quality, can match or even exceed GANs when scaled and tuned appropriately. Furthermore, the interpretability, stability, and sampling diversity of diffusion models make them highly attractive for a broader range of applications, including conditional generation, image editing, and data augmentation. Overall, Dhariwal and Nichol's paper marks a **turning point in generative modeling**, demonstrating that diffusion models are not only viable but also state-of-the-art contenders in image synthesis. Their work has inspired a growing body of follow-up research aimed at reducing sampling time, improving efficiency, and expanding to modalities beyond vision, such as audio and text. This progress highlights the importance of continued exploration in diffusion-based approaches for next-generation generative AI systems.

3.3.4. Results :

Dataset	Config	FID↓	EQ-T↑	EQ-R↑
FFHQ-U 70000 img, 1024^2 Train from scratch	StyleGAN2	3.79	15.89	10.79
	StyleGAN3-T (ours)	3.67	61.69	13.95
	StyleGAN3-R (ours)	**3.66**	**64.78**	**47.64**
FFHQ 70000 img, 1024^2 Train from scratch	StyleGAN2	**2.70**	13.58	10.22
	StyleGAN3-T (ours)	2.79	61.21	13.82
	StyleGAN3-R (ours)	3.07	**64.76**	**46.62**
METFACES-U 1336 img, 1024^2 ADA, from FFHQ-U	StyleGAN2	18.98	18.77	13.19
	StyleGAN3-T (ours)	**18.75**	64.11	16.63
	StyleGAN3-R (ours)	**18.75**	**66.34**	**48.57**
METFACES 1336 img, 1024^2 ADA, from FFHQ	StyleGAN2	15.22	16.39	12.89
	StyleGAN3-T (ours)	**15.11**	**65.23**	16.82
	StyleGAN3-R (ours)	15.33	64.86	**46.81**
AFHQV2 15803 img, 512^2 ADA, from scratch	StyleGAN2	4.62	13.83	11.50
	StyleGAN3-T (ours)	**4.04**	60.15	13.51
	StyleGAN3-R (ours)	4.40	**64.89**	**40.34**
BEACHES 20155 img, 512^2 ADA, from scratch	StyleGAN2	5.03	15.73	12.69
	StyleGAN3-T (ours)	**4.32**	59.33	15.88
	StyleGAN3-R (ours)	4.57	**63.66**	**37.42**

Ablation	Translation eq.		+ Rotation eq.		
	FID↓	EQ-T↑	FID↓	EQ-T↑	EQ-R↑
* Main configuration	4.62	63.01	**4.50**	66.65	40.48
With mixing reg.	**4.60**	63.48	4.67	63.59	40.90
With noise inputs	4.96	24.46	5.79	26.71	26.80
Without flexible layers	4.64	45.20	4.65	44.74	22.52
Fixed Fourier features	5.93	64.57	6.48	66.20	41.77
With path length reg.	5.00	**68.36**	5.98	**71.64**	**42.18**
0.5× capacity	7.43	63.14	6.52	63.08	39.89
* 1.0× capacity	4.62	63.01	4.50	66.65	40.48
2.0× capacity	**3.80**	**66.61**	**4.18**	**70.06**	**42.51**
* Kaiser filter, $n = 6$	**4.62**	**63.01**	4.50	**66.65**	**40.48**
Lanczos filter, $a = 2$	4.69	51.93	**4.44**	57.70	25.25
Gaussian filter, $\sigma = 0.4$	5.91	56.89	5.73	59.53	39.43

G-CNN comparison	FID↓	EQ-T↑	EQ-R↑	Params	Time
* StyleGAN3-T (ours)	4.62	63.01	13.12	23.3M	**1.00**×
+ $p4$ symmetry [16]	4.69	61.90	17.07	21.8M	2.48×
* StyleGAN3-R (ours)	**4.50**	**66.65**	**40.48**	15.8M	1.37×

3. METHODOLOGY

3.1 : LSB (Least Significant Bit) Steganography

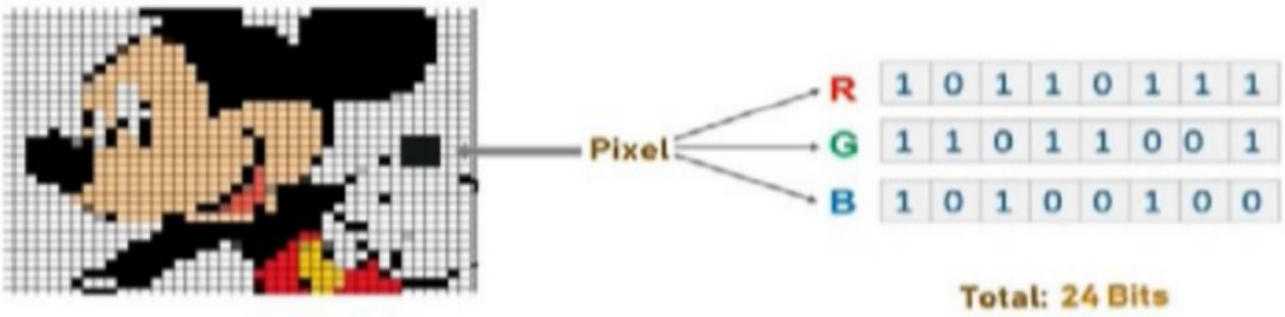


Photo credits to Edureka Steganography tutorial

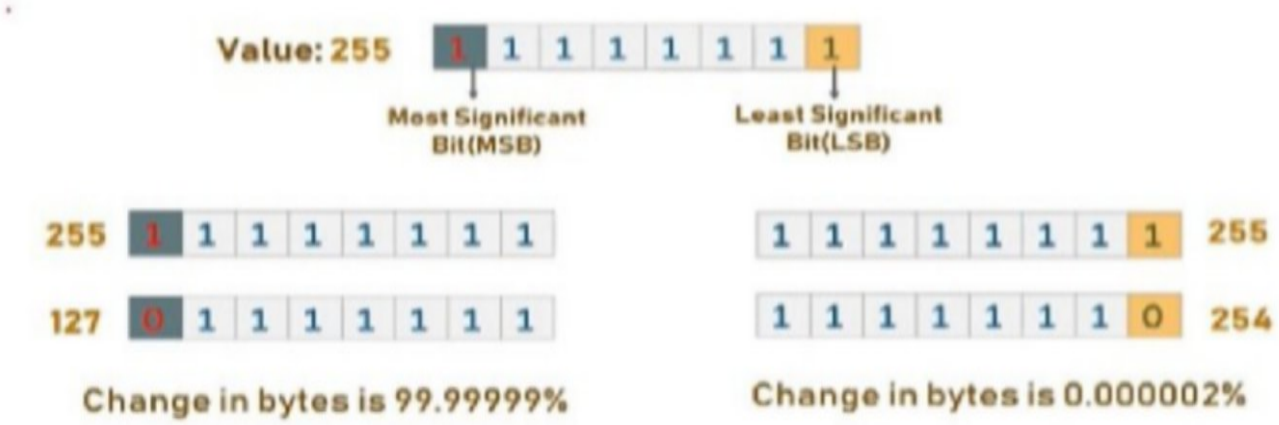


Photo by Edureka Steganography tutorial

LSB (Least Significant Bit) steganography is one of the most widely used and straightforward techniques for hiding secret information within digital images. The core idea of LSB steganography is to embed the message bits into the least significant bits of the pixel values in an image. Since these bits contribute the least to the overall color or intensity of a pixel, modifying them causes only minor changes that are visually imperceptible to the human eye. This subtlety makes LSB steganography an effective method for covert communication.

A digital image, particularly an RGB image, is composed of pixels, each containing three color components: Red, Green, and Blue. Each of these components is typically represented using 8 bits, allowing for values ranging from 0 to 255. In LSB steganography, the rightmost bit (i.e., the least significant bit) of each 8-bit color value is replaced with a bit from the secret message. Since altering just the LSB changes the value by at most 1, the impact on the image is minimal and generally undetectable to the naked eye.

Before the embedding process begins, the secret message (which may be text, binary data, or another file) is first converted into a binary stream. To facilitate accurate extraction later, metadata such as the length of the message or a special delimiter (e.g., a unique bit pattern) can be appended to the binary stream. This ensures the extraction process knows where the embedded message ends.

The embedding process involves scanning the image pixel by pixel. For each pixel, the LSBs of the red, green, and blue channels are accessed and replaced with bits from the binary message. This process is continued sequentially across pixels until the entire message is embedded. The resulting image, referred to as the **stego-image**, looks virtually identical to the original image when viewed normally. However, it now contains the secret data hidden within its pixel structure.

To extract the hidden message, the stego-image is scanned in the same sequential order used during embedding. The LSBs of each color channel are read and combined to reconstruct the original binary message stream. Once the complete stream is collected, it can be decoded—using the length metadata or delimiter if present—back into the original message format, whether text, a file, or other data.

LSB steganography is highly effective when used with lossless image formats such as PNG or BMP. These formats preserve pixel values exactly, ensuring that the hidden data remains intact and retrievable. In contrast, lossy formats like JPEG apply compression algorithms that can alter pixel values and potentially destroy or corrupt the embedded message.

While simple and efficient, LSB steganography is not without limitations. It is vulnerable to image processing operations such as compression, resizing, or filtering, which may alter the LSBs and damage the hidden data. Moreover, because the method is relatively easy to detect with steganalysis tools, it is best used in scenarios where low-risk concealment is acceptable or when combined with encryption for added security.

3.2 : Image-based Steganography

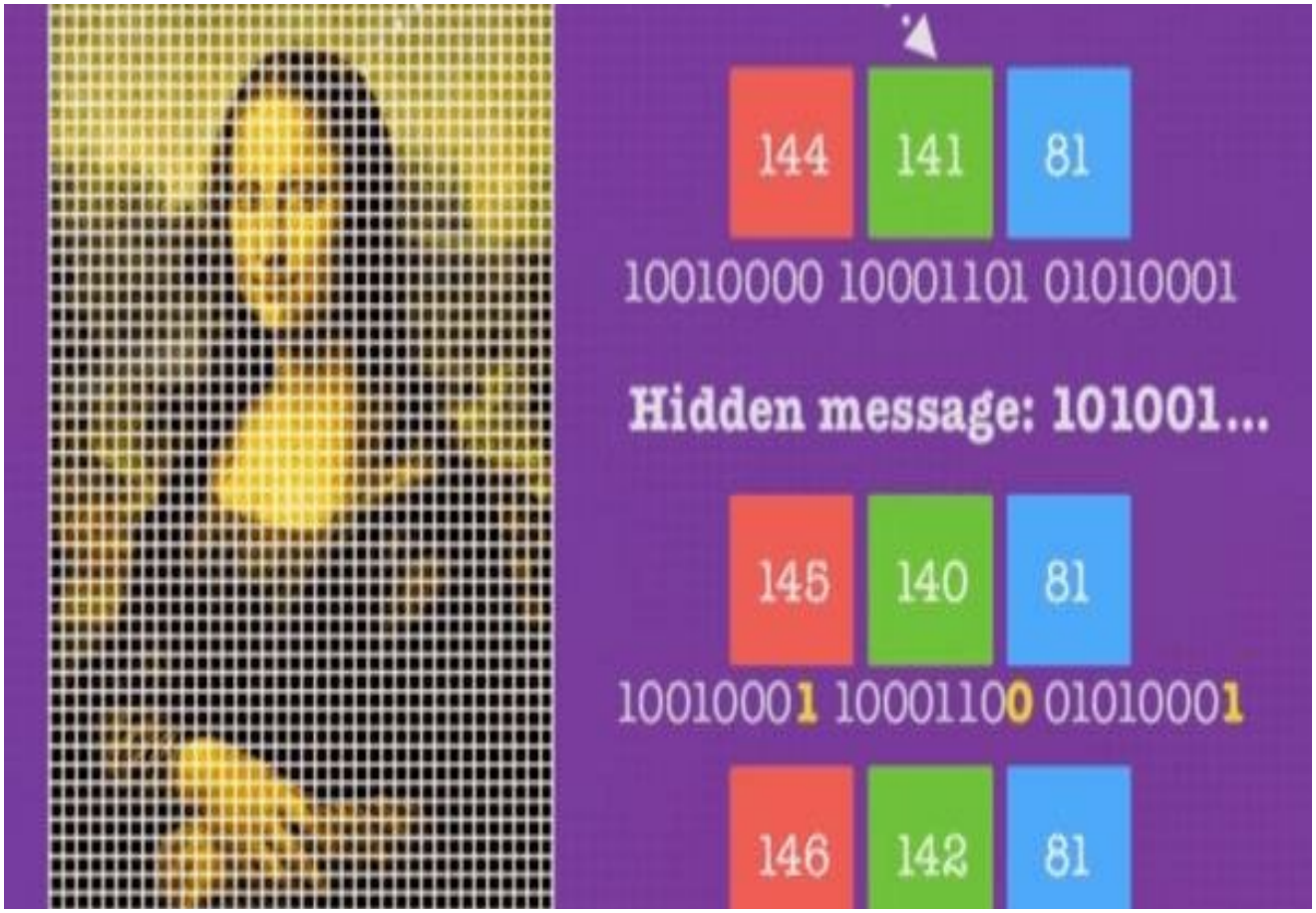


Image-based steganography is a technique used to conceal secret information within a digital image in such a way that the presence of the data remains hidden from plain sight. Unlike encryption, which scrambles the content to make it unreadable to outsiders, steganography hides the very existence of a message by embedding it within a seemingly innocent file—in this case, an image. This method is highly effective for covert communication and is widely used in areas such as digital watermarking, copyright protection, and secure information exchange.

Digital images are made up of tiny elements called pixels. In a standard RGB image, each pixel consists of three color components—Red, Green, and Blue—each typically represented by 8 bits. These binary values determine the color intensity of each channel. Because of the human eye's limited sensitivity to minor color changes, it is possible to alter some of these bits—particularly the least significant bits—without causing noticeable visual differences. This makes digital images an excellent medium for steganographic embedding.

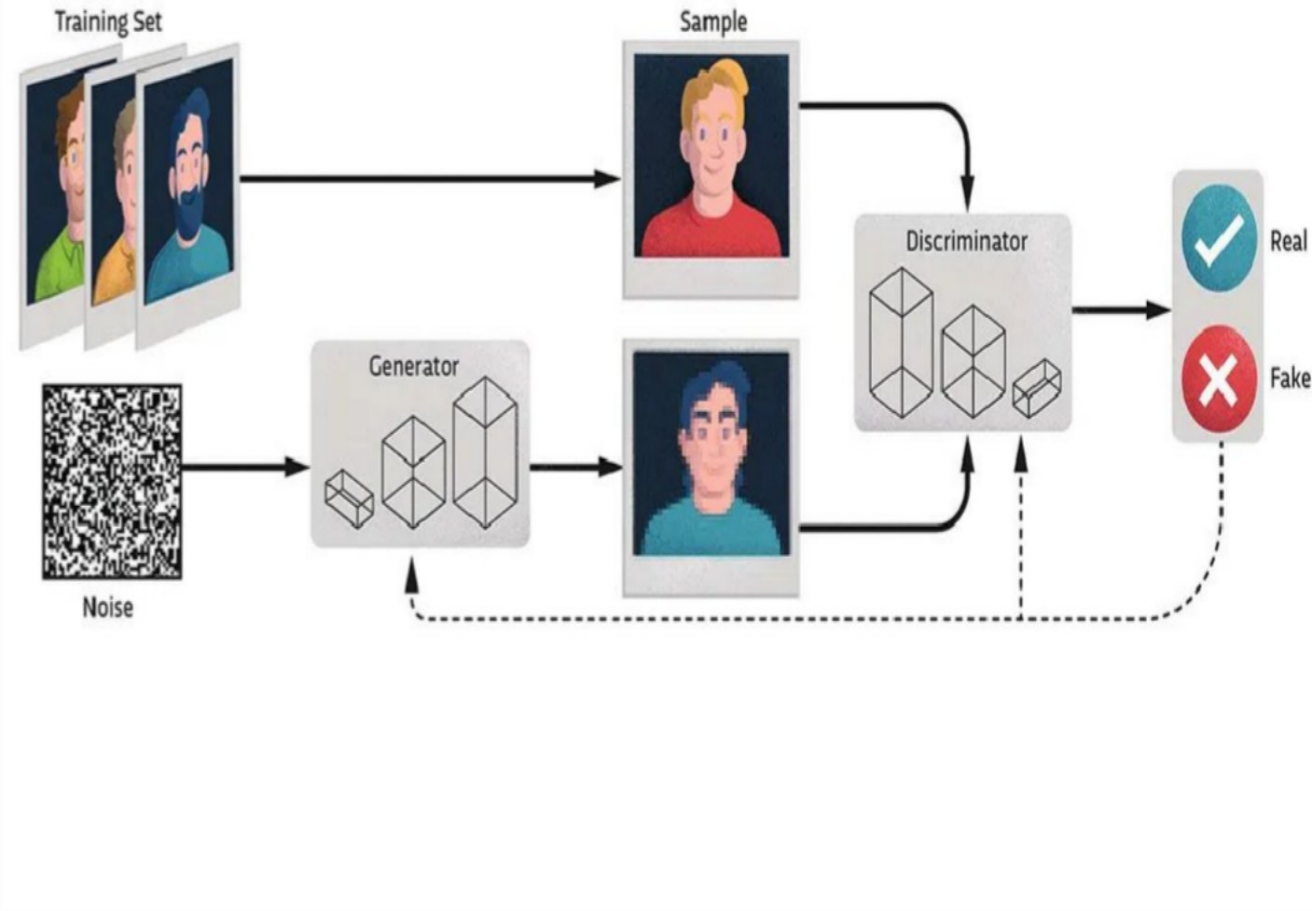
The process begins by converting the secret data, such as a text message or binary file, into a stream of binary digits (0s and 1s). Once in binary form, the image is scanned pixel by pixel. Within each pixel, selected bits—most commonly the least significant bits of the RGB channels—are replaced with bits from the message. The Least Significant Bit (LSB) substitution method is the most widely used approach due to its simplicity and minimal visual impact. For example, changing the last bit of a color value from 0 to 1 results in such a small difference in color that it is imperceptible to the human eye.

After the entire message is embedded, the modified image—now referred to as the **stego-image**—looks virtually identical to the original image, despite carrying hidden data. This ensures that observers, or even basic digital analysis tools, cannot easily detect the presence of the embedded information.

To retrieve the hidden message, the extraction process involves reading the stego-image in the same pixel order and accessing the bits that were altered during embedding. These bits are then reassembled into a binary stream, which is finally decoded back into its original form, such as readable text or a usable file.

A key advantage of image-based steganography is that it does not raise suspicion, as the image appears normal and can be shared or transmitted freely. However, for the method to work reliably, it is essential to use **lossless image formats** like PNG or BMP. Lossy formats such as JPEG apply compression algorithms that can alter or remove the least significant bits, thereby corrupting or destroying the hidden data.

3.3 : Generative Adversarial Networks (GANs)



Generative Adversarial Networks (GANs) are a powerful class of machine learning models designed for generating synthetic data that closely resembles real-world data. Introduced by Ian Goodfellow in 2014, GANs have revolutionized various fields, especially computer vision, by enabling machines to create highly realistic images, videos, and other forms of data from scratch. The core architecture of GANs involves two neural networks—**a Generator** and **a Discriminator**—that are trained simultaneously in a competitive, adversarial setup.

The **Generator** is responsible for producing synthetic data. It starts with random noise as input and learns to transform this noise into data that mimic the characteristics of the real dataset. Initially, the Generator's outputs are far from realistic, but it improves over time by learning from its mistakes, guided by the feedback it receives from the Discriminator.

The **Discriminator**, on the other hand, acts as a classifier. It takes both real data (from the training dataset) and fake data (produced by the Generator) and tries to distinguish between them. Its goal is to correctly label whether a given sample is “real” or “fake.” The Discriminator is trained to maximize its accuracy in identifying the origin of each input sample, while the Generator is trained to minimize the Discriminator’s ability to make that distinction.

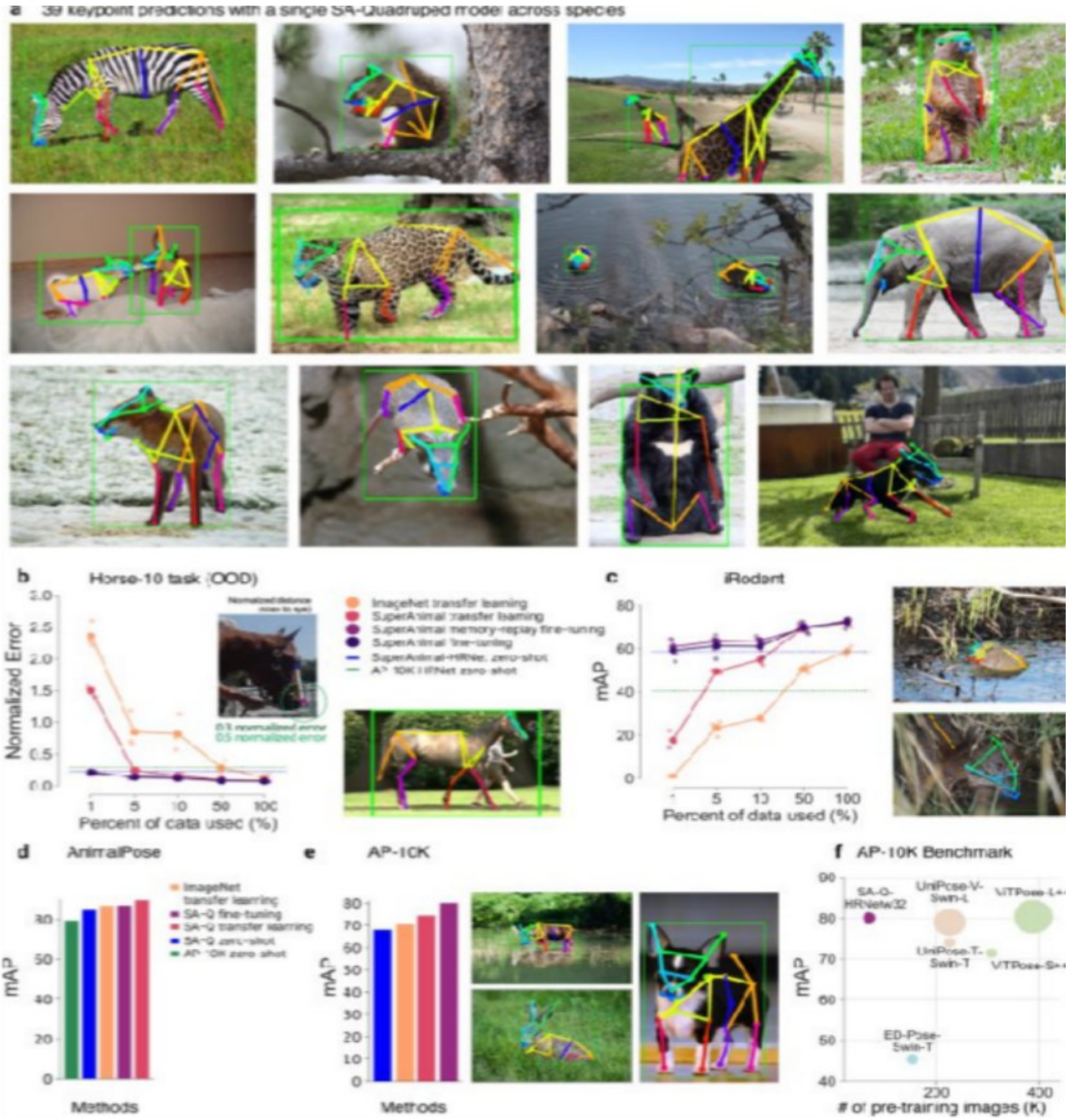
The training process is modeled as a zero-sum game, where one model’s gain is the other’s loss. As the Generator improves its ability to fool the Discriminator, the Discriminator must also become better at catching these fakes. This adversarial interaction drives both networks to improve continuously. The Generator gets feedback from the Discriminator in the form of gradients that help it adjust its parameters to produce more convincing data.

Over many iterations, this competitive training leads to a point where the Generator becomes so effective that the Discriminator can no longer reliably distinguish real data from generated data. At this stage, the GAN is said to have reached equilibrium, and the synthetic data produced by the Generator is nearly indistinguishable from real data to both the model and human observers.

GANs have found widespread applications in areas such as image synthesis (e.g., generating faces, objects, or scenes), video generation, super-resolution, style transfer, and data augmentation for machine learning. They are also used in creative industries for generating art, music, and deepfake videos. Beyond content creation, GANs play a role in scientific simulations and anomaly detection.

The strength of GANs lies in their ability to learn through **interaction, competition, and feedback**—mirroring human learning in certain ways. This dynamic, adversarial training process enables the Generator to refine its outputs to high levels of realism without direct supervision on what makes the data “real.” As such, GANs represent one of the most influential advancements in deep learning, pushing the boundaries of what artificial intelligence can generate and understand.

3.4 : Pose Estimation



Pose estimation is a key technique in computer vision that focuses on determining the spatial positions of human body joints from images or videos. The objective is to identify the coordinates of anatomical keypoints—such as elbows, wrists, shoulders, hips, knees, and ankles—and connect them to represent the human body as a simplified skeletal model. This skeletal structure consists of joints (depicted as dots) and limbs (represented as lines connecting related joints), forming a visual representation of the person's pose.

Pose estimation can be performed in either **2D or 3D**. In 2D pose estimation, the system identifies the (x, y) coordinates of each joint on the image plane. In contrast, 3D pose estimation includes depth information, adding a z-coordinate to model the position of joints in three-dimensional space. While 2D estimation is common and effective for many use cases, 3D estimation provides a more complete understanding of body orientation and spatial relationships, which is valuable in advanced applications like motion capture and virtual reality.

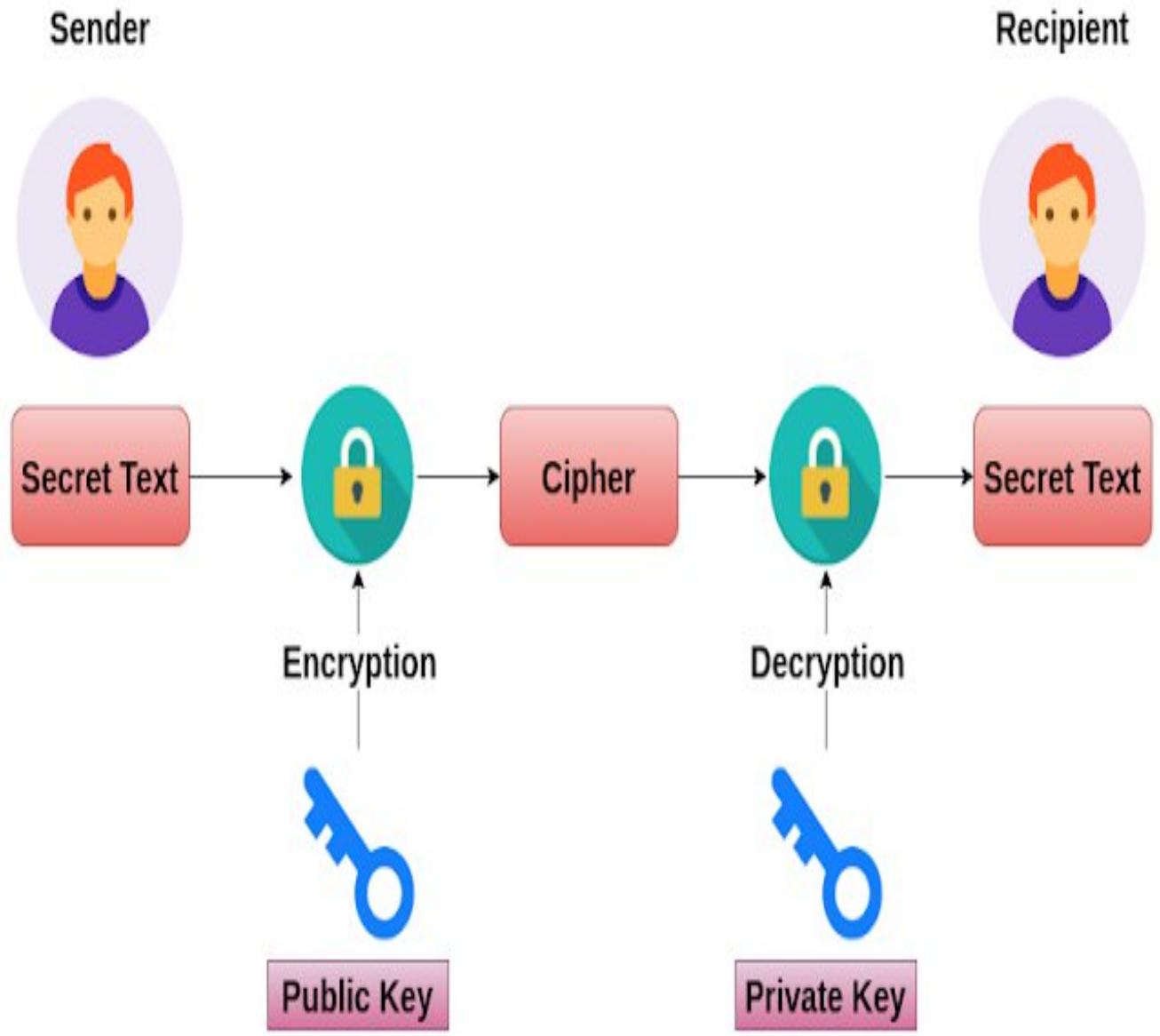
Modern pose estimation systems are typically based on **deep learning**, especially convolutional neural networks (CNNs), which are trained on large annotated datasets such as COCO, MPII, or Human3.6M. These networks process input images and output **heatmaps** for each joint. A heatmap is a probability distribution over the image space, indicating the likelihood of a joint being located at each pixel. The peak of the heatmap corresponds to the most likely position of a particular joint.

Each predicted keypoint is associated with a **confidence score**, which reflects the model's certainty about the joint’s position. During post-processing, keypoints with high confidence values are selected, and corresponding limbs are connected to form a coherent **pose skeleton**. This skeleton captures the structure and orientation of the body and can be visualized as an overlay on the original image or video.

Pose estimation systems can be designed for **single-person** or **multi-person** scenarios. In single-person pose estimation, the model assumes that only one person is present and focuses on that individual. In multi-person pose estimation, additional mechanisms like object detection or instance segmentation are used to identify and separate individuals before estimating their poses.

The versatility of pose estimation enables a wide range of practical applications. In **fitness tracking**, it helps monitor and analyze exercise form. In **animation and motion capture**, it drives realistic character movements. In **gesture recognition** and **human-computer interaction**, it interprets body movements to control devices or software interfaces. Pose estimation is also useful in **surveillance**, **sports analytics**, and **rehabilitation monitoring**.

3.5 : Cryptography



Cryptography is a fundamental aspect of information security that involves the transformation of data into an unreadable or encoded format to protect it from unauthorized access or tampering. It plays a crucial role in ensuring the **confidentiality**, **integrity**, **authentication**, and **non-repudiation** of information across digital communication and storage systems. By converting readable data, known as plaintext, into an encrypted format called ciphertext, cryptography ensures that sensitive information remains secure even if intercepted by malicious actors.

The transformation of plaintext into ciphertext is achieved through **encryption algorithms** and the use of cryptographic **keys**. Only individuals or systems possessing the correct decryption key can reverse the process and recover the original plaintext. Cryptography can be broadly classified into two main types: **symmetric** and **asymmetric** cryptography. In symmetric cryptography, the same key is

used for both encryption and decryption. This method is fast and efficient, making it suitable for encrypting large volumes of data. A widely used symmetric algorithm is **AES (Advanced Encryption Standard)**.

In contrast, **asymmetric cryptography** uses a pair of keys: a **public key** for encryption and a **private key** for decryption. This method is more secure for key exchange and digital signatures, as the private key never needs to be shared. A common asymmetric algorithm is **RSA (Rivest–Shamir–Adleman)**, which is often used in secure communications like SSL/TLS and digital certificates.

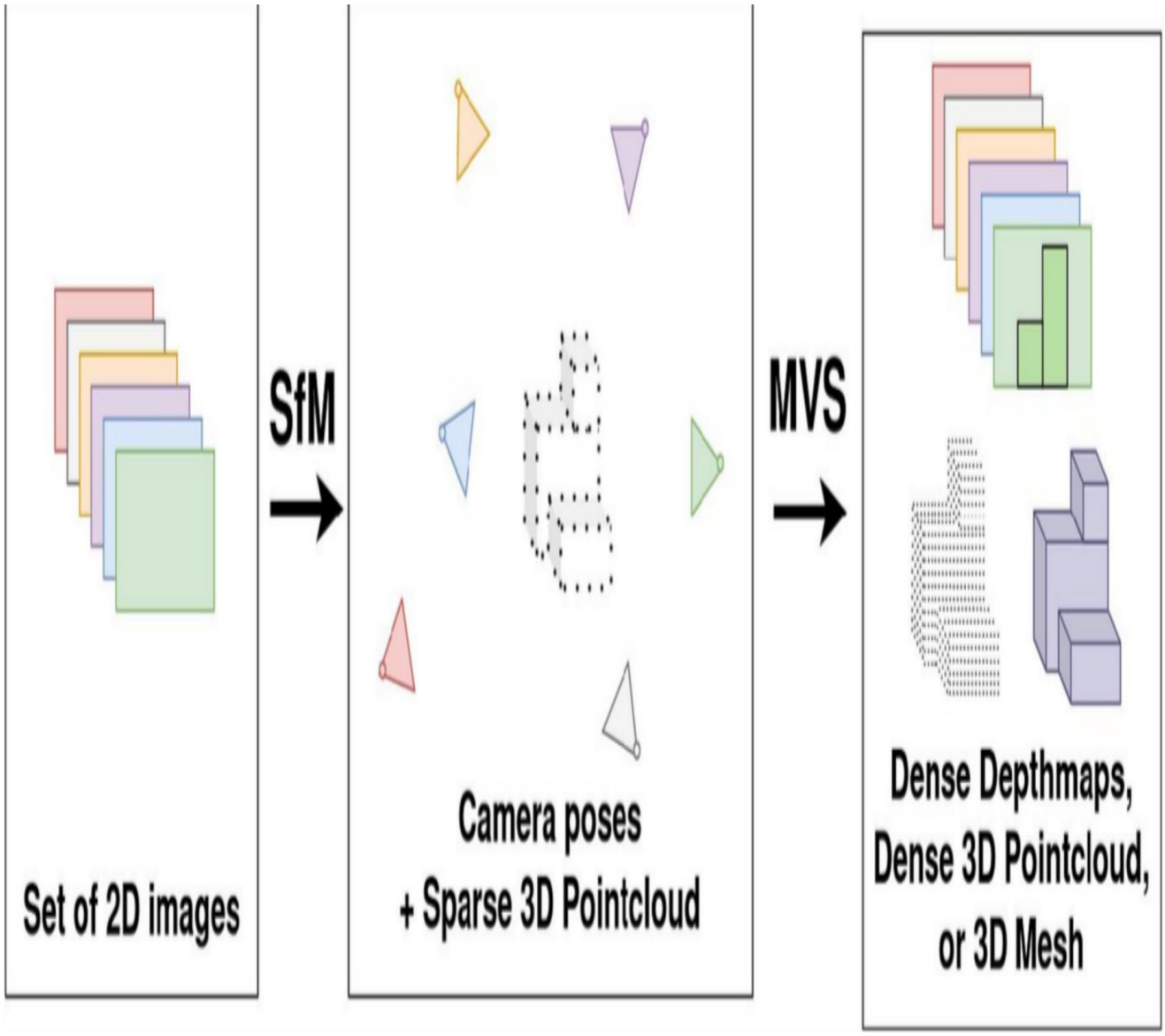
The typical cryptographic process follows a cycle: plaintext is encrypted using a specific algorithm and key to generate ciphertext. This ciphertext can then be transmitted or stored securely. When needed, the authorized party decrypts it using the appropriate key, restoring the original plaintext. This cycle ensures that data remains confidential and accessible only to intended recipients.

Besides encryption and decryption, cryptography also involves the use of **hash functions**, which generate a fixed-size string (or fingerprint) from input data. These hashes are used to verify data integrity; any small change in the original data results in a completely different hash value, making tampering detectable. Common hashing algorithms include SHA-256 and MD5.

In diagrams and illustrations, cryptography is often symbolized using **binary code**, **padlocks**, and **mathematical operations**, reflecting its technical and protective nature. A crucial aspect of any cryptographic system is **key management**—the generation, distribution, storage, and disposal of cryptographic keys. Poor key management can compromise even the strongest encryption algorithms.

As computational capabilities and cyber threats evolve, cryptographic standards continue to advance. Ongoing research into **post-quantum cryptography** aims to develop encryption methods resistant to the power of quantum computing, which could potentially break current encryption techniques.

3.6 Multi-View Stereo (MVS)



Multi-View Stereo (MVS) is an advanced computer vision technique used to reconstruct detailed three-dimensional (3D) models from a collection of two-dimensional (2D) images taken from multiple, overlapping viewpoints. The central goal of MVS is to derive accurate 3D geometry of an object or scene by analyzing how it appears from different perspectives. This approach enables the creation of high-fidelity, photorealistic 3D reconstructions that are essential in fields like photogrammetry, robotics, digital preservation, and 3D scanning.

The MVS process begins with **camera calibration**, which determines the internal parameters of the cameras as well as their positions and orientations relative to one another. This calibration ensures that the geometry of the image capture setup is well understood, forming the basis for accurate 3D reconstruction. Once calibration is complete, the system proceeds to **feature matching**, a step where corresponding visual features—such as corners or textures—are identified across different images. These shared points allow the system to establish relationships between the views.

Using the matched features, MVS applies **triangulation** to compute the 3D coordinates of each point by determining where lines of sight from different viewpoints intersect. While early steps focus on sparse reconstruction, the strength of MVS lies in its ability to perform **dense reconstruction**—estimating depth values for most pixels in the overlapping regions of the images. The output is a **dense point cloud**, which is a detailed set of 3D points representing the shape and structure of the observed object or environment.

From the dense point cloud, **surface reconstruction** algorithms generate a continuous 3D surface or mesh, which connects the points to form a coherent geometric model. To enhance realism, **texture mapping** is applied by projecting the original 2D images onto the 3D surface, allowing the reconstructed model to retain accurate color and texture information.

One of the defining features of MVS is its use of **more than two images** (in contrast to traditional stereo vision), which allows for greater detail, depth accuracy, and robustness to occlusion or noise. The **overlapping fields of view** provided by multiple images improve coverage and ensure that hidden or partially visible surfaces can still be reconstructed when seen from alternative angles.

Visual representations of MVS often depict a target object surrounded by multiple camera positions, with lines indicating the viewing direction of each camera. These diagrams highlight how images converge to form a unified and comprehensive 3D model.

The final outputs of an MVS pipeline typically include **dense 3D point clouds**, **depth maps**, **triangulated meshes**, and **textured models**. These outputs are widely used in industries such as **cultural heritage digitization**, where ancient artifacts are captured in digital form; **autonomous navigation**, where robots use 3D maps for spatial awareness; and **virtual reality**, where accurate models enhance immersive environments.

4. Challenges

4.1. Noise and Distortion Control

In steganographic systems, embedding hidden data within images must be done with extreme care to avoid noticeable visual degradation. Any alteration to pixel values, even slight, can introduce noise or artifacts that compromise the stealth of the hidden message. To ensure visual fidelity, adaptive embedding techniques are used, placing data in complex image regions like edges or textured areas where changes are less perceptible. Objective image quality metrics such as PSNR and SSIM are often employed to measure and limit distortion. Using lossless image formats (e.g., PNG or BMP) prevents additional compression noise. Moreover, the embedding algorithm should minimize redundancy and avoid excessive modification of smooth areas. Advanced methods use perceptual models to make visually minimal changes. The goal is to achieve a high data payload while ensuring the host image remains indistinguishable from the original, both to human observers and digital filters.

4.2. Steganalysis Resistance

One of the greatest challenges in steganography is resisting detection by steganalysis—tools and techniques designed to uncover hidden data. Modern steganalysis often uses machine learning and AI to detect statistical anomalies in image features. Therefore, embedding strategies must be sophisticated, incorporating randomness and content-awareness to avoid detectable patterns. Encrypting the payload before embedding further increases entropy and masks structure that could aid detection. Varying payload size, position, and encoding strategy helps obfuscate the data presence. Training steganographic models with adversarial networks can also enhance robustness by simulating real-world attack scenarios. A resilient system avoids predictable embedding behaviors and spreads the payload subtly across the image. Ultimately, success in this area means the data remains hidden not only from the eye but also from forensic analysis tools

4.3. Cryptographic Integration

Integrating cryptography with steganography ensures dual-layer security: encryption protects the content, while steganography conceals its existence. However, this must be done without introducing excessive computational overhead. Efficient integration typically involves using symmetric encryption algorithms like AES, known for their strong security and speed. To avoid payload bloat, data can be compressed before encryption. The steganographic algorithm must be

capable of embedding encrypted binary data without distortion or data loss. Additionally, secure key management must be in place to ensure authorized access during decryption. Extraction and decryption processes should align smoothly for reliable data retrieval. A well-integrated system safeguards both the content and the communication channel without sacrificing performance or visual quality. When properly combined, these techniques offer a robust framework for confidential and covert data exchange.

4.4. Pose Estimation Accuracy

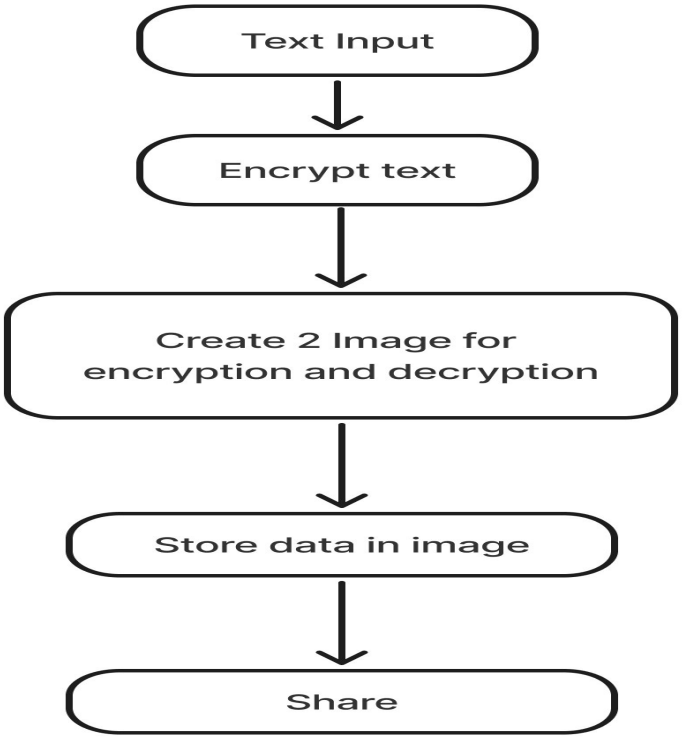
Pose estimation plays a key role in intelligent steganographic systems, especially those using human body features for context-aware embedding. Accurate detection of keypoints—such as elbows, shoulders, and knees—is essential for reliable data placement and retrieval. Deep learning models like OpenPose are typically used, but their accuracy can be affected by variations in lighting, background clutter, occlusion, and body orientation. Errors in pose estimation can misplace the embedded data or break synchronization in dynamic systems like gesture-based communication. High-quality image input and domain-specific training datasets significantly improve accuracy. Additionally, temporal consistency is vital in video-based pose tracking to ensure stable and coherent embedding over time. Overall, the precision of pose estimation directly impacts the security, usability, and effectiveness of any pose-driven steganographic application.

4.5. MVS Complexity

Multi-View Stereo (MVS) is essential for reconstructing high-fidelity 3D models from multiple overlapping images, but it comes with significant computational challenges. The process involves camera calibration, feature matching, depth estimation, and surface reconstruction—all of which are resource-intensive. Misalignment, occlusions, and lighting variations between views can introduce errors or gaps in the 3D model. Generating dense point clouds and converting them into usable 3D meshes requires considerable memory and processing time. Optimizations such as GPU acceleration, multi-threading, and automated pipeline tuning are often necessary to make MVS practical at scale. Furthermore, ensuring both geometric accuracy and visual detail across all views remains a technical hurdle. Despite its complexity, MVS is a cornerstone technology in photogrammetry, augmented reality, and digital reconstruction, enabling highly detailed spatial representations of real-world scenes.

5. Process

5.1 Process flow



5.1.1. Text Input

- The user begins by entering plaintext (e.g., a message, secret note, or data).
- This is the raw information that will undergo encryption before it is securely stored and shared.

5.1.2. Encrypt Text

- The entered text is encrypted using a cryptographic algorithm.
- This involve:

    - **Symmetric encryption** (e.g., AES), where the same key is used for encryption and decryption.
    - **Asymmetric encryption** (e.g., RSA), where a public key encrypts the data and a private key decrypts it.
- The result is ciphertext—scrambled data that cannot be read without the proper key.

5.1.3. Create 2 Images for Encryption and Decryption

- Two images are generated:
    - **Encryption image (or share 1)**: Contains part of the encrypted data or acts as a key.
    - **Decryption image (or share 2)**: Complements the first image to retrieve the original message.
- This could be based on:
    - **Visual cryptography**, where both images must be overlaid to decode the message.
    - **Steganography**, where encrypted data is hidden inside images.

5.1.4. Store Data in Image

- The encrypted text is embedded in one or both of the images.
- This is often done using **steganography**, where binary data is hidden within the pixel values (e.g., in the least significant bits).
- The data is now concealed and indistinguishable from a regular image to the human eye.

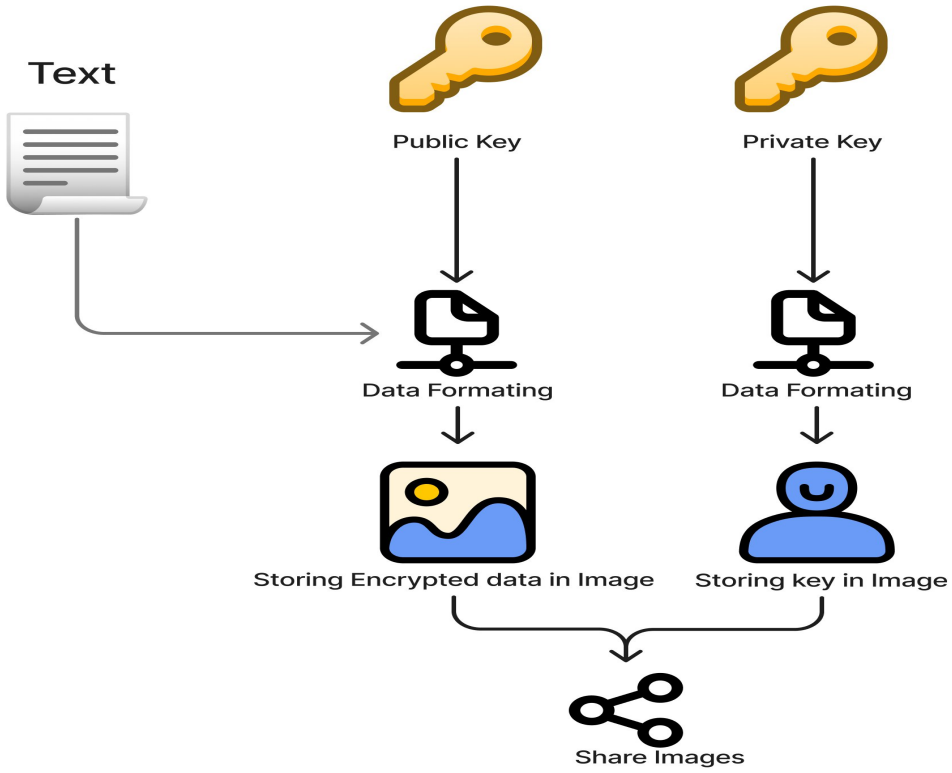
5.1.5. Share

- The resulting image(s) are shared through a secure or public channel.
- Only recipients with the appropriate second image or decryption method can extract and decrypt the original text.

Use Cases:

- **Secure communication** where traditional text encryption might raise suspicion.
- **Two-factor secured data sharing**, requiring both an image and a key.
- **Confidential information exchange** in hostile or surveillance-heavy environments.

5.2 Image storing diagram



5.2.1. Public Key and Private Key

- You have two keys:
  - **Public Key**: Used to encrypt data.
  - **Private Key**: Used to decrypt data.
- These keys are part of **asymmetric cryptography** (e.g., RSA, ECC).
- The public key can be shared with anyone, but the private key must be kept secure.

5.2.2. Data Formatting

Both keys are formatted for embedding:

- They are likely converted to a suitable binary or encoded format (like Base64).
- This prepares the keys for hiding in an image.

5.2.3. Storing Keys in Images (Steganography)

Steganography is used to embed the keys in images:

- Public key is stored in one image (e.g., a general image).
- Private key is stored in another, likely more secure image.

This involve:

- Least Significant Bit (LSB) manipulation.
- Encoding the binary key into pixel color values subtly.

The user (represented by the person icon) may hold the private key image securely.

5.2.4. Share Image

Once the keys are embedded:

- Images can be shared, for example over public networks or to a second party.
- Only the receiver who has access to the correct image (and knows how to decode it) can extract the key.

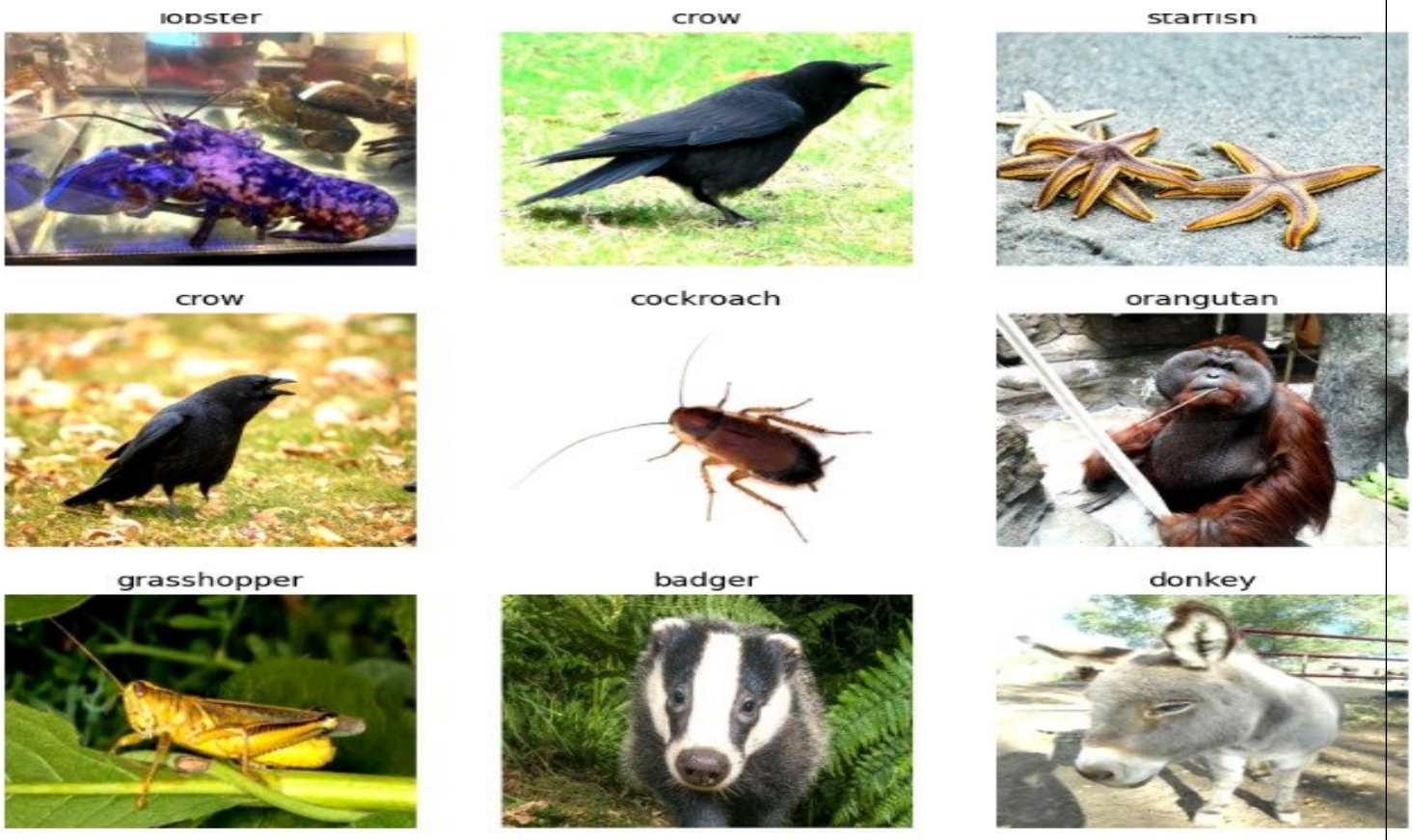
If the private key is securely stored, only the intended receiver can decrypt the message or data.

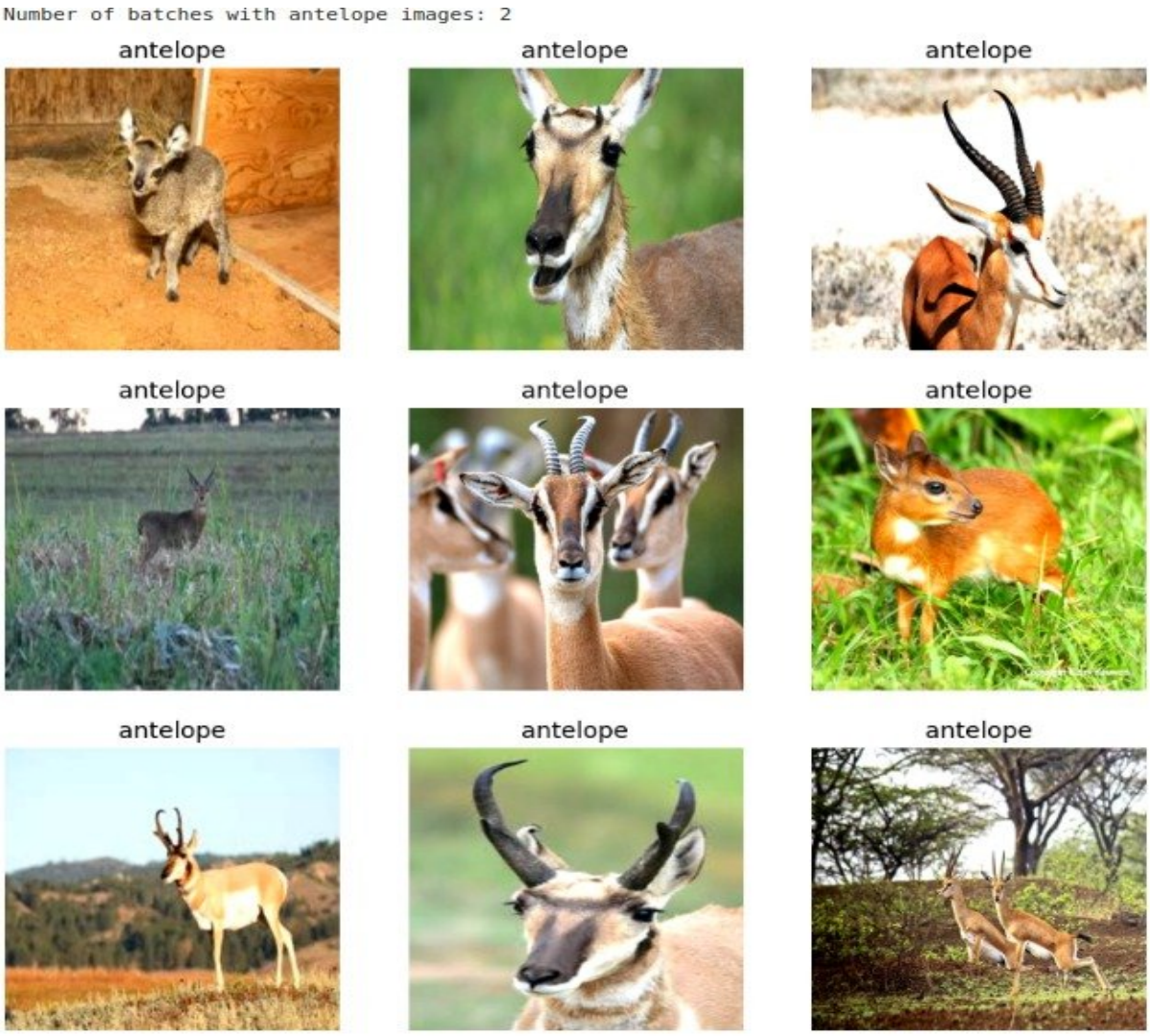
This process enables secure key sharing and storage by:

- Hiding sensitive cryptographic keys in plain-looking images.
- Allowing key distribution in a way that avoids direct textual transmission (which is more detectable).
- Supporting secure communication or decryption later, when the images are used as key sources.

6. DATASET

- Found **5400 files** belonging to **90 classes.**
- Using **1080 files** for validation.
- **Classes** : ['antelope', 'badger', 'bat', 'bear', 'bee', 'beetle', 'bison', 'boar', 'butterfly', 'cat', 'caterpillar', 'chimpanzee', 'cockroach', 'cow', 'coyote', 'crab', 'crow', 'deer', 'dog', 'dolphin', 'donkey', 'dragonfly', 'duck', 'eagle', 'elephant', 'flamingo', 'fly', 'fox', 'goat', 'goldfish', 'goose', 'gorilla', 'grasshopper', 'hamster', 'hare', 'hedgehog', 'hippopotamus', 'hornbill', 'horse', 'hummingbird', 'hyena', 'jellyfish', 'kangaroo', 'koala', 'ladybugs', 'leopard', 'lion', 'lizard', 'lobster', 'mosquito', 'moth', 'mouse', 'octopus', 'okapi', 'orangutan', 'otter', 'owl', 'ox', 'oyster', 'panda', 'parrot', 'pelecaniformes', 'penguin', 'pig', 'pigeon', 'porcupine', 'possum', 'raccoon', 'rat', 'reindeer', 'rhinoceros', 'sandpiper', 'seahorse', 'seal', 'shark', 'sheep', 'snake', 'sparrow', 'squid', 'squirrel', 'starfish', 'swan', 'tiger', 'turkey', 'turtle', 'whale', 'wolf', 'wombat', 'woodpecker', 'zebra']
- **Training size :** 135
- **Validation size :** 34



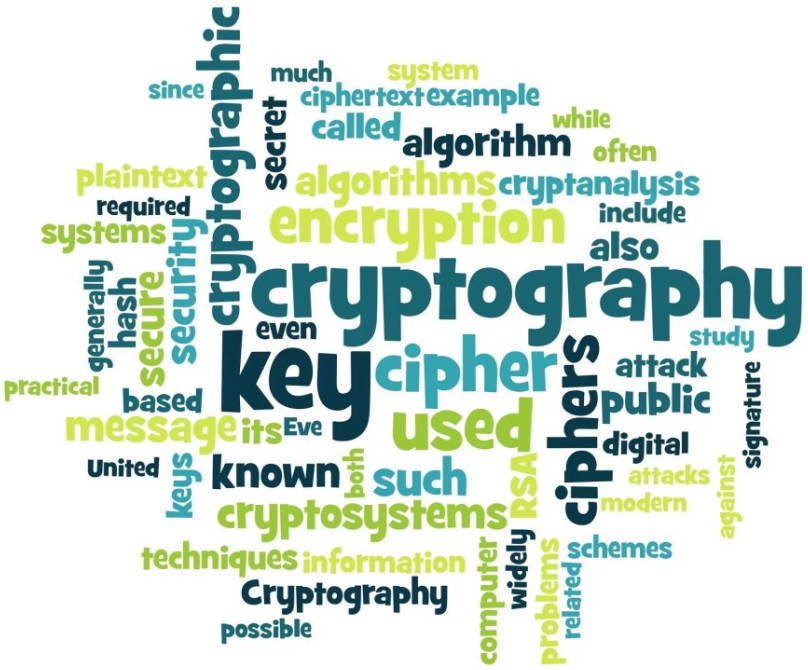


The dataset used in this project consists of 5400 high-quality images categorized into 90 distinct animal classes, covering a wide range of biological diversity. These classes include terrestrial animals (e.g., lion, zebra, koala), aquatic species (e.g., shark, dolphin, octopus), birds (e.g., eagle, owl, woodpecker), insects (e.g., bee, butterfly, mosquito), and other unique creatures like starfish and seahorse. The dataset is evenly split with 135 training images per class and 34 validation images per class, resulting in a well-balanced training set and a validation set of 1080 images in total. This class balance helps prevent bias toward any specific category during model training and supports stable performance evaluation.

Each image is pre-labeled, allowing for supervised learning tasks such as image classification using convolutional neural networks (CNNs) or transfer learning with pre-trained models. The diverse and balanced class representation makes this dataset an excellent choice for developing and testing robust computer vision models, especially in tasks that require generalization across varying animal appearances, environments, and anatomical structures. For a beginner developer, this dataset presents an ideal opportunity to explore data preprocessing, model training, validation techniques, and performance evaluation in a real-world context. From a business perspective, such a dataset could be foundational in building wildlife recognition tools, biodiversity monitoring systems, or educational applications powered by AI.

7. Tools and Technology

7.1. Cryptography



Cryptography is the practice of securing data by converting it into unreadable formats to protect it from unauthorized access. It uses encryption algorithms to transform plaintext into ciphertext, which can only be decrypted by those with the correct key. There are two main types: symmetric cryptography, using the same key for encryption and decryption, and asymmetric cryptography, using a public-private key pair. Common algorithms include AES, RSA, and ECC. Cryptography ensures confidentiality, integrity, authentication, and non-repudiation of data. It is widely used in secure communications, data storage, and identity verification. Python libraries like cryptography, PyCrypto, and Fernet provide easy-to-use APIs for implementing these methods. Fernet offers strong symmetric encryption with built-in authentication, making it suitable for secure message transmission. Effective cryptography also involves proper key management and protecting against attacks. It plays a critical role in securing online transactions, messaging apps, and digital signatures. As technology evolves, cryptographic techniques adapt to counter new threats and ensure ongoing data protection.

7.2. Pandas

Pandas is a powerful Python library used for data analysis and manipulation of structured data. It introduces two key data structures: Series (1D) and DataFrame (2D), which make handling and analyzing data easier and more efficient. Pandas supports importing and exporting data from various formats like CSV, Excel, SQL, and JSON. It provides tools for filtering, grouping, sorting, and aggregating data, as well as handling missing values. Its intuitive syntax enables users to perform complex data transformations with minimal code. Pandas integrates well with other libraries such as NumPy and Matplotlib for seamless data processing and visualization. The library supports time series data and offers advanced features like pivot tables and hierarchical indexing. Pandas is widely used in data science, machine learning preprocessing, and exploratory data analysis. It offers optimized performance for large datasets, making data handling fast and scalable. The active development community ensures continual improvements and new feature additions.

7.3. NumPy

NumPy is the fundamental package for numerical computing in Python, providing fast and efficient operations on large multi-dimensional arrays and matrices. It offers a comprehensive set of mathematical functions, including linear algebra, statistics, and Fourier transforms. NumPy arrays are more efficient than Python lists due to their fixed data types and contiguous memory allocation. The library supports broadcasting, enabling arithmetic operations on arrays of different shapes without explicit loops. NumPy serves as the backbone for many scientific computing libraries, including Pandas and TensorFlow. Its functionality includes random number generation and array manipulation

tools like reshaping, slicing, and indexing. NumPy improves performance and simplifies code for numerical algorithms and simulations. It is essential for data preprocessing, mathematical modeling, and machine learning workflows. The library is well-documented and widely used in both academia and industry for high-performance computing tasks.

7.4. Stepic

Stepic is a Python library designed for image steganography using the Least Significant Bit (LSB) technique. It allows embedding secret messages within PNG images by modifying the least significant bits of pixel values, which minimally affects the visual appearance. This method is ideal for hiding encrypted or sensitive data securely without obvious image distortion. Stepic supports both encoding and decoding of hidden information, making it easy to retrieve secret messages. Because it works on PNG images, it avoids lossy compression artifacts that could destroy the hidden data. The library is simple to use and integrates well with Python projects involving covert communication or watermarking. Stepic enables covert data transfer and digital watermarking with minimal computational overhead. It is suitable for research and applications requiring lightweight image-based steganography. However, care must be taken as LSB techniques may be vulnerable to advanced steganalysis if not combined with encryption or randomization. Despite this, stepic remains a practical tool for basic image hiding needs.

7.5. TensorFlow

TensorFlow is an open-source machine learning framework developed by Google, widely used for building and training deep learning models. It provides flexible tools for designing neural networks, including layers, optimizers, and loss functions. TensorFlow supports automatic differentiation and GPU acceleration, enabling efficient training of large models. It offers high-level APIs like Keras for easier model development and prototyping. TensorFlow's computational graph architecture allows scalable deployment across different platforms, from mobile devices to cloud servers. It supports a wide range of applications including image recognition, natural language processing, and reinforcement learning. TensorFlow integrates with other Python libraries and tools, making it versatile for research and production. Its ecosystem includes TensorBoard for visualization and TensorFlow Lite for mobile inference. The library has extensive documentation and community support, making it accessible to beginners and experts alike. TensorFlow continues to evolve, adding features like distributed training and improved performance optimizations.

7.6. Matplotlib

Matplotlib is a comprehensive Python library for creating static, animated, and interactive visualizations. It provides a flexible interface for generating plots, histograms, bar charts, scatter plots, and more. Matplotlib integrates seamlessly with NumPy and Pandas, enabling visualization of

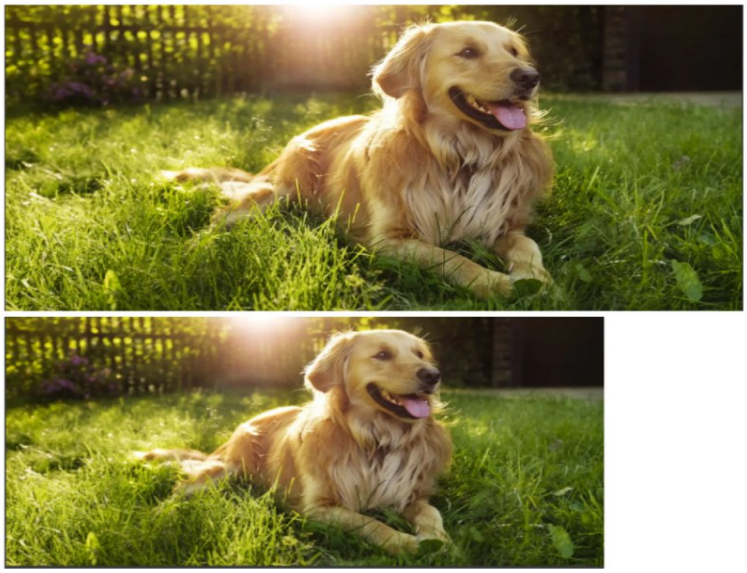
data arrays and DataFrames. It supports customization of plot styles, colors, labels, and axes to create publication-quality graphics. The library includes tools for interactive features like zooming and panning, useful in data exploration. Matplotlib is highly extensible, allowing users to build complex multi-plot figures and embed visualizations in applications. It is widely used in scientific computing, data analysis, and machine learning projects to interpret results visually. With extensive documentation and examples, Matplotlib remains the go-to visualization library in the Python ecosystem. It supports exporting plots to various formats, including PNG, PDF, and SVG, facilitating sharing and reporting. Its community continues to expand its capabilities with new backends and tools.

7.7. Pytorch

PyTorch is an open-source deep learning framework developed by Meta, known for its dynamic computation graph and Pythonic design, making it intuitive for model development and easy to debug. It's widely used in both research and production for tasks like computer vision, NLP, and reinforcement learning. It offers essential tools like modular layers, optimizers, and loss functions, with Autograd for automatic differentiation and GPU acceleration for fast training. Its "define-by-run" approach enables flexible and dynamic model building, ideal for prototyping and advanced architectures. For production, TorchScript allows exporting models to run on C++ or mobile. PyTorch also supports TensorBoard, Weights & Biases, and distributed training. Its strong community and clear documentation make it beginner-friendly and powerful for scaling real-world ML solutions.

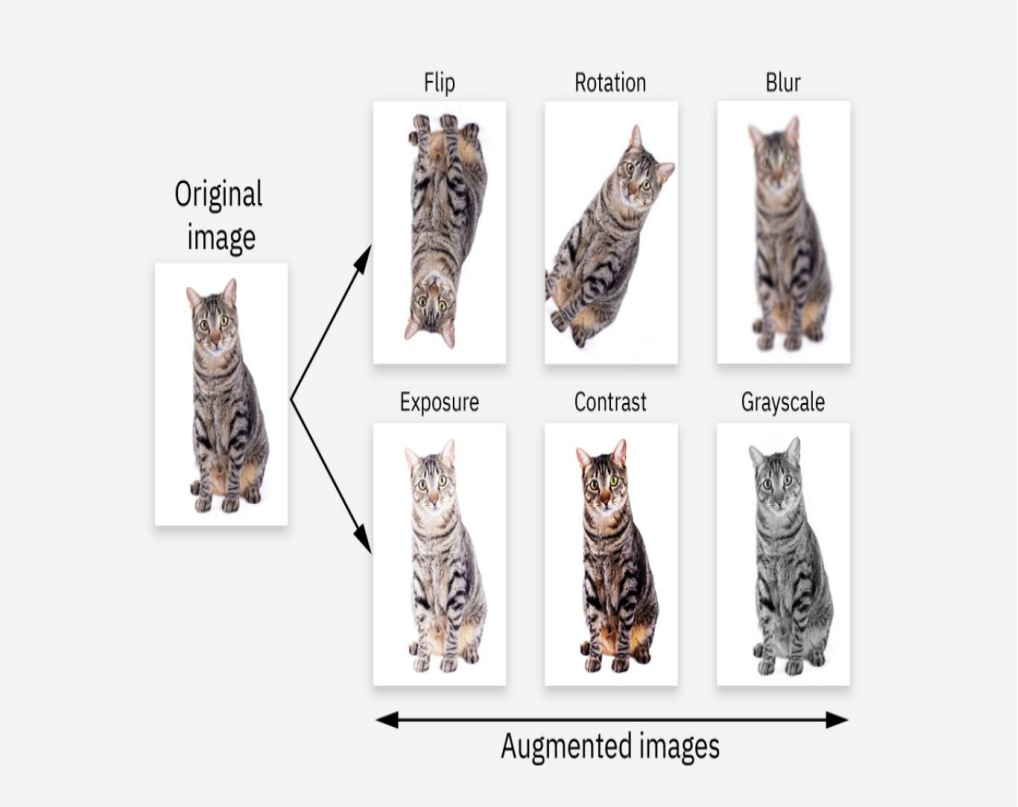
8. PRE-PROCESSING

8.1. Resizing



Resizing is the process of changing the dimensions of an image to meet specific requirements for analysis or modeling. It ensures uniformity across datasets by standardizing the input size, which is crucial for deep learning models that expect fixed-size inputs. Resizing also improves computational efficiency by reducing the amount of data to process, speeding up training and inference. Various interpolation methods are used to calculate pixel values when resizing, including nearest-neighbor, bilinear, and bicubic interpolation. Nearest-neighbor is the simplest and fastest but may produce blocky images. Bilinear interpolation considers the closest 2x2 neighborhood of pixels, providing smoother results. Bicubic interpolation uses a 4x4 pixel neighborhood and usually produces the best quality with smoother edges and fewer artifacts. Choosing the right interpolation depends on the application and the desired balance between quality and speed. Resizing must be performed carefully to preserve the aspect ratio to avoid distortion. Consistent resizing improves model performance by providing uniform input data. It is a common preprocessing step in computer vision pipelines before further processing or feature extraction.

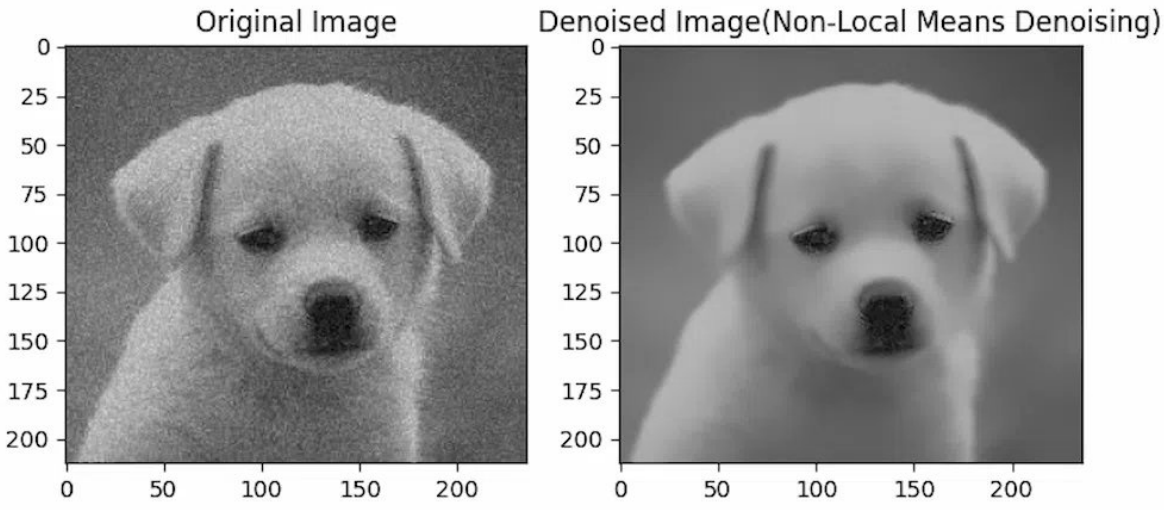
8.2. Augmentation




```
# Data augmentation (for training only)
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255),  # Normalize pixel values (0-1)
    tf.keras.layers.RandomFlip("horizontal"),
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomZoom(0.2)
])

# Apply preprocessing to datasets
def preprocess_dataset(dataset, augment=False):
    def process_image(image, label):
        image = tf.image.resize(image, img_size)  # Ensure correct size
        if augment:
            image = data_augmentation(image)  # Apply augmentation to training data
        else:
            image = tf.keras.layers.Rescaling(1./255)(image)  # Only normalize for validation
        return image, label

    dataset = dataset.map(process_image, num_parallel_calls=AUTOTUNE)
    return dataset.prefetch(buffer_size=AUTOTUNE)  # Optimize performance
```

Augmentation refers to artificially increasing the diversity and size of a dataset by applying various transformations to the original images. This technique helps deep learning models generalize better by exposing them to different variations of the data, reducing overfitting. Common augmentation methods include geometric transformations such as rotation, flipping (horizontal and vertical), and scaling, which simulate different viewing angles and object sizes. Color adjustments, like brightness and contrast modification, mimic different lighting conditions. Adding Gaussian blur can help models become robust to image focus variations. Other techniques include cropping, translation, and adding noise. Augmentation is often applied randomly during training to continuously present new variations, making the model more resilient. It effectively multiplies the amount of training data without needing additional data collection. Data augmentation is essential for improving the robustness and accuracy of models in tasks such as image classification, object detection, and segmentation.

8.3. Noise Removal



Noise removal is the process of eliminating unwanted random variations in image intensity, known as noise, which can degrade the quality of images and hinder accurate analysis. Common types of noise include salt-and-pepper noise, where random pixels are either very bright or very dark, and Gaussian noise, characterized by variations following a normal distribution. Removing noise enhances edge clarity and improves the accuracy of feature extraction, which is critical for image

processing tasks. Techniques for noise removal include median filtering, which replaces each pixel with the median of its neighborhood and is particularly effective against salt-and-pepper noise. Gaussian blur smooths the image by averaging pixels weighted by a Gaussian function, reducing Gaussian noise while preserving overall structures. Bilateral filtering smooths images while maintaining sharp edges by considering both spatial proximity and pixel intensity differences. Effective noise removal ensures cleaner inputs for downstream tasks like segmentation or classification, ultimately improving model performance. Choosing the right noise removal technique depends on the type and level of noise present in the images.

9. Implementation

9.1. Data encryption and storing in image

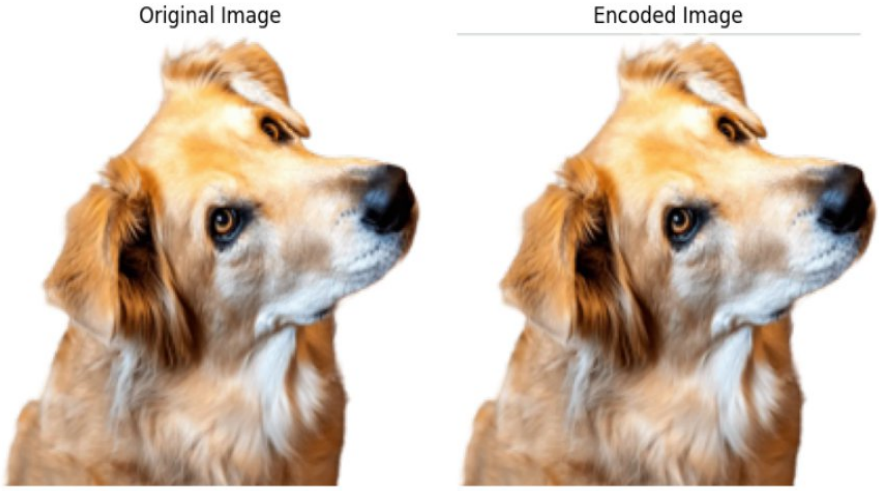
Enter the message you want to encrypt and store in image: hello, This is a Secret Message

Original Message: hello, This is a Secret Message

Encrypted Message Bytes:

b'\x1e\xc5e\xc7a\xad\xd8,\x8c\xdd\xea}$\x13\xa3\xcdl\xb2<\x98\xf9\xaf\x0c\x0cH\xcf\xcf\xc0i\x14C\
xf5-
\x90;"w\x06~_N\xadb\x88+Q\x1d8YKs\x93\x97S\xa7\xbe\x15{\xb1\x16\xdd\x052q\xb9\xecx\x83A\x
00\xe5\xdb\x91\xb2\x9fql\xae\xff8l*U\xa4K\x11C\xe4\xd2\x87\x12\xcaM\x1e$\x93T\xe4\xa84\xae\xc
d\xc6<n\x1c\xcfA\x08*\xd95\xe5\x93\xec\x8fsl\xef\x13q\xbf\xdd\x8c\xadt\xe6*\x9eH\'\x06\x88T\xd0I
\x07\x8e\x89\x8cq\xa0OM\xca\xb9\xc4\xf7\x0c\\\x9c\reGW\xf9\xbd\x88:M\xe9\xb9\x0b\xd1
\xa3\x9c\xf7,\x08\xae\x18\r\xfeV\xde\x9a\x0c(DuH\x0f!\xa2\x93\x1fh\x0c\xe0\xcf\x93b6ke&\xd8\n\x0
3\x92\x8d\x96=\x88\xb4\xb3Y0M\xad\xff\x06/7\xfa\xc1+\x8f\x9e$O,w\xce\x16"\xe3\x96\xaf\x0b\x8f\
x14\x9f\xc1=7|\x95w\xa1\xa0*5Y\x9e{\xc5E3\xb2@\xcfE-('

🖼 Encrypted message hidden and image saved as: kaggle/working/encoded_image.png



The input message, "hello, This is a Secret Message," is first converted into a byte format and then encrypted using a secure cryptographic algorithm. This encryption transforms the readable message into a complex sequence of bytes, making it unreadable to anyone without the decryption key. Encrypting the message ensures that the data remains confidential and protected from unauthorized access during storage or transmission.

Next, the encrypted byte data is embedded within an image using steganographic techniques, specifically designed to hide information without visibly altering the image. This combination of encryption and steganography provides a dual layer of security by both concealing the presence of the message and protecting its content. The embedded data remains imperceptible to the naked eye and resilient against basic detection methods.

Finally, when needed, the hidden data is extracted from the image and decrypted back into its original readable form. This decryption confirms that the embedded message has been securely stored and accurately recovered without loss or corruption. This process guarantees both the confidentiality and integrity of the message, enabling secure and covert communication.

9.2. Background Removal

1

1

1

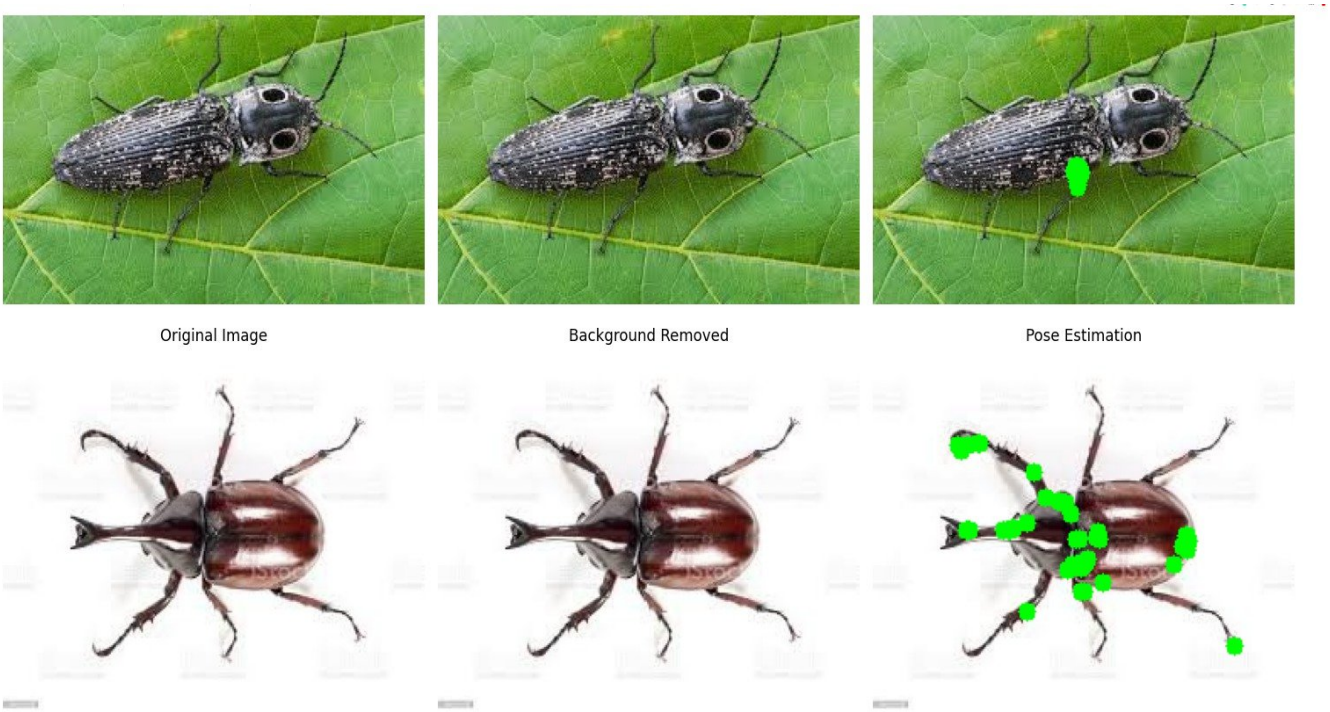
]:

```
from rembg import remove
for images, labels in antelope_dataset.take(1):  # Take one batch
    for i in range(min(9, len(images))):
        single_image = images[i].numpy().astype("uint8")  # Extract the first image
        single_label = labels[i].numpy()  # Extract the corresponding label
        # break
        print("1")

        img = Image.fromarray(single_image).convert("RGB")
        img_no_bg = remove(img)
        img_bright = ImageEnhance.Brightness(img_no_bg).enhance(1.2)
        img_contrast = ImageEnhance.Contrast(img_bright).enhance(1.1)
        enhanced_image_no_bg = np.array(img_contrast)

        # Visualize the image with background removed
        plt.figure(figsize=(5, 5))
        plt.imshow(enhanced_image_no_bg)
        plt.title(f"Class: {class_names[single_label]} (Background Removed)")
        plt.axis("off")
        plt.show()
```

```
Downloading data from 'https://github.com/danielgatis/rembg/releases/download/v0.0.0/u2net.onnx' to file
'/root/.u2net/u2net.onnx'.
1
100%|██████████████████████████████████████| 176M/176M [00:00<00:00, 316GB/s]
```

Background removal isolates the main subject of an image by eliminating unwanted surrounding areas. It enhances focus on the object, improving clarity for further analysis or processing. Common techniques include thresholding, segmentation, and deep learning-based methods. Accurate background removal is essential for applications like object recognition, photo editing, and augmented reality. It helps in reducing noise and improving the performance of downstream tasks.

9.3. Pose Estimation



Pose estimation identifies and locates key body joints and limbs in images or videos. It maps these points to form a skeletal structure representing human posture and movement. This technique is widely used in fields like animation, sports analysis, and human-computer interaction. Accurate pose estimation enables better understanding of actions and gestures. It often relies on deep learning models trained on large annotated datasets.

9.4. Fine tuning using StyleGan3

```
!git clone https://github.com/NVlabs/stylegan3.git
%cd stylegan3
```

```
Cloning into 'stylegan3'...
remote: Enumerating objects: 212, done.
remote: Counting objects: 100% (163/163), done.
remote: Compressing objects: 100% (73/73), done.
remote: Total 212 (delta 99), reused 90 (delta 90), pack-reused 49 (from 1)
Receiving objects: 100% (212/212), 4.16 MiB | 14.16 MiB/s, done.
Resolving deltas: 100% (108/108), done.
/kaggle/working/stylegan3
```

```
# For StyleGAN2-ADA / StyleGAN3 (latest version):
!python /kaggle/working/stylegan3/dataset_tool.py --source=/kaggle/working/animal_dataset --dest=/kaggle/working/animal_dataset-512x512.zip --resolution=512x512
```

```
100%|██████████| 5000/5000 [01:18<00:00, 63.39it/s]
```

```
!wget https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan3/versions/1/files/stylegan3-r-afhqv2-512x512.pkl -O stylegan3-pretrained-512x512.pkl
!mv stylegan3-pretrained-512x512.pkl /kaggle/working/
!ls -lh /kaggle/working/stylegan3-pretrained-512x512.pkl
```

```
--2025-06-11 08:58:26--  https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan3/versions/1/files/stylegan3-r-afhqv2-512x512.pkl
Resolving api.ngc.nvidia.com (api.ngc.nvidia.com)... 54.68.249.82, 52.25.204.74
Connecting to api.ngc.nvidia.com (api.ngc.nvidia.com)|54.68.249.82|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://xfiles.ngc.nvidia.com/org/nvidia/team/research/models/stylegan3/versions/1/files/stylegan3-r-afhqv2-512x512.pkl?ssec-algo=AES256&versionId=TsHJSzIRVUwRkXNjCC6J7cCR7nmREwJv&ssec-key=c89IwM5JotbIqGKN1zk00l4m4o%2Fm%2Fp3b05bqa%2FhR59SUXGEwESweunnjTBe7gv9mAo40c%2B0b72i2YheM9YrLxqXtmYxh1g%2BylgWstnYvX5ltE2OJXOnnHgJ2Bsk0Fyi6E6x%2BlGlaRvQOwaQM1ZsXmmFJz8uoCB4k4qYWsapSgS10wuvcOgQfvJNQvgdz1mjyVRm3LIpW7Q1%2BDmCTF4ihvR4Gxjr%2BqZaTyortRRNWrS%2FigA2PfR%2FMarwJxBI7IoTl7MZxCvtjU%2FlqOESqSkaasfN82Y6xxld9Pnfr2gh9I7%2B5ROqGp0dn6Y%2BwQPRzxHBo%2BguIng9IH4tc6Ke%2BC3WQgh2U0S6ojiEvQNXl5B3Db3fTp90T9UQPvX9CHcovTRvBmXWB4lLll%2FM1fVj7GR6Bn5EwwEdCZnF%2FdS7dCUU2RG4cu9pbKME2D%2BU224Zv0ueZRkQppokaTi8e6oXldeHXg%2BuVCPABLiNrC5wj6g0CR9n9MtfF4VSPETjIxDSMkYn6&Signature=WnysOGb1ZZyjQEps~EsS~I~yanJNVlPa7y67EnHmRxKo7bPx0lwL~o97OCqMDtugeWNXSrwaj67Bkd6Ek7gHSkyJa3rT9mXnzaIa3k2GZ5cDXRDGFlo6VF18Qn2jjEy4f7GO4e6X4aQP1Fen7CZK8xCEkn4fRJfXqDhPK1vuyIY92eq~XhHiI2h5C1xsALICF~t18Jo80YbQSCG18PXV-PJzZw6e8rI9mEuLmVI7D~133xi-S1IxJfViRbae~9fkFEOsviu6~NpC-78DEauBBg4Al5CDS38Pxxz2wG4u69wj-P769DIXMtisFWBnb16NUu~35GOCZTINg4x87P7q6w__&kid=bXJrLWU3OGM1M2FhZjE4YzRiNmJiNjlkYmRhZjcxNjA3YWEw&Expires=1749718706&Key-Pair-Id=KCX06E8E9L60W&ssec-enabled=true [following]
--2025-06-11 08:58:26--  https://xfiles.ngc.nvidia.com/org/nvidia/team/research/models/stylegan3/versions/1/files/stylegan3-r-afhqv2-512x512.pkl?ssec-algo=AES256&versionId=TsHJSzIRVUwRkXNjCC6J7cCR7nmREwJv&ssec-key=c89IwM5JotbIqGKN1zk00l4m4o%2Fm%2Fp3b05bqa%2FhR59SUXGEwESweunnjTBe7gv9mAo40c%2B0b72i2YheM9YrLxqXtmYxh1g%2BylgWstnYvX5ltE2OJXOnnHgJ2Bsk0Fyi6E6x%2BlGlaRvQOwaQM1ZsXmmFJz8uoCB4k4qYWsapSgS10wuvcOgQfvJNQvgdz1mjyVRm3LIpW7Q1%2BDmCTF4ihvR4Gxjr%2BqZaTyortRRNWrS%2FigA2PfR%2FMarwJxBI7IoTl7MZxCvtjU%2FlqOESqSkaasfN82Y6xxld9Pnfr2gh9I7%2B5ROqGp0dn6Y%2BwQPRzxHBo%2BguIng9IH4tc6Ke%2BC3WQgh2U0S6ojiEvQNXl5B3Db3fTp90T9UQPvX9CHcovTRvBmXWB4lLll%2FM1fVj7GR6Bn5EwwEdCZnF%2FdS7dCUU2RG4cu9pbKME2D%2BU224Zv0ueZRkQppokaTi8e6oXldeHXg%2BuVCPABLiNrC5wj6g0CR9n9MtfF4VSPETjIxDSMkYn6&Signature=WnysOGb1ZZyjQEps~EsS~I~yanJNVlPa7y67EnHmRxKo7bPx0lwL~o97OCqMDtugeWNXSrwaj67Bkd6Ek7gHSkyJa3rT9mXnzaIa3k2GZ5cDXRDGFlo6VF18Qn2jjEy4f7GO4e6X4aQP1Fen7CZK8xCEkn4fRJfXqDhPK1vuyIY92eq~XhHiI2h5C1xsALICF~t18Jo80YbQSCG18PXV-PJzZw6e8rI9mEuLmVI7D~133xi-S1IxJfViRbae~9fkFEOsviu6~NpC-78DEauBBg4Al5CDS38Pxxz2wG4u69wj-P769DIXMtisFWBnb16NUu~35GOCZTINg4x87P7q6w__&kid=bXJrLWU3OGM1M2FhZjE4YzRiNmJiNjlkYmRhZjcxNjA3YWEw&Expires=1749718706&Key-Pair-Id=KCX06E8E9L60W&ssec-enabled=true
Resolving xfiles.ngc.nvidia.com (xfiles.ngc.nvidia.com)... 13.35.166.57, 13.35.166.124, 13.35.166.27, ...
Connecting to xfiles.ngc.nvidia.com (xfiles.ngc.nvidia.com)|13.35.166.57|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 249525556 (238M) [application/x-zerosize]
Saving to: ‘stylegan3-pretrained-512x512.pkl’
```

9.4.1. Training Configurations

```
!python /kaggle/working/stylegan3/train.py \
  --outdir=/kaggle/working/animal-stylegan-results-512x512 \
  --cfg=stylegan3-r \
  --data=/kaggle/working/animal_dataset-512x512.zip \
  --gpus=1 \
  --batch=8 \
  --mirror=1 \
  --gamma=8 \
  --snap=1 \
  --resume=/kaggle/working/stylegan3-pretrained-512x512.pkl \
  --kimg=5000 \
  --metrics=none \
  --glr=0.002 \
  --dlr=0.002 \
  --aug=ada \
  --freezed=0 \
  --workers=4 \
  --target=0.6
```

9.4.2. Training options:


```
{
 "G_kwargs": {
  "class_name": "training.networks_stylegan3.Generator",
  "z_dim": 512,
  "w_dim": 512,
  "mapping_kwargs": {
   "num_layers": 2
  },
  "channel_base": 65536,
  "channel_max": 1024,
  "magnitude_ema_beta": 0.9997227795604651,
  "conv_kernel": 1,
  "use_radial_filters": true
 },
 "D_kwargs": {
  "class_name": "training.networks_stylegan2.Discriminator",
  "block_kwargs": {
   "freeze_layers": 0
  },
  "mapping_kwargs": {},
  "epilogue_kwargs": {
   "mbstd_group_size": 4
  },
  "channel_base": 32768,
  "channel_max": 512
 },
 "G_opt_kwargs": {
  "class_name": "torch.optim.Adam",
  "betas": [
   0.0,
   0.99
  ],
  "eps": 1e-08,
  "lr": 0.002
 },
 "D_opt_kwargs": {
  "class_name": "torch.optim.Adam",
  "betas": [
   0.0,
   0.99
  ],
  "eps": 1e-08,
  "lr": 0.002
 },
 "loss_kwargs": {
  "class_name": "training.loss.StyleGAN2Loss",
  "r1_gamma": 8.0,
  "blur_init_sigma": 0,
  "blur_fade_kimg": 50.0
 },
 "data_loader_kwargs": {
  "pin_memory": true,
```

```
  "prefetch_factor": 2,
  "num_workers": 4
 },
 "training_set_kwargs": {
  "class_name": "training.dataset.ImageFolderDataset",
  "path": "/kaggle/working/animal_dataset-512x512.zip",
  "use_labels": false,
  "max_size": 5000,
  "xflip": true,
  "resolution": 512,
  "random_seed": 0
 },
 "num_gpus": 1,
 "batch_size": 8,
 "batch_gpu": 8,
 "metrics": [],
 "total_kimg": 5000,
 "kimg_per_tick": 4,
 "image_snapshot_ticks": 1,
 "network_snapshot_ticks": 1,
 "random_seed": 0,
 "ema_kimg": 2.5,
 "augment_kwargs": {
  "class_name": "training.augment.AugmentPipe",
  "xflip": 1,
  "rotate90": 1,
  "xint": 1,
  "scale": 1,
  "rotate": 1,
  "aniso": 1,
  "xfrac": 1,
  "brightness": 1,
  "contrast": 1,
  "lumaflip": 1,
  "hue": 1,
  "saturation": 1
 },
 "ada_target": 0.6,
 "resume_pkl": "/kaggle/working/stylegan3-pretrained-512x512.pkl",
 "ada_kimg": 100,
 "ema_rampup": null,
 "run_dir": "/kaggle/working/animal-stylegan-results-512x512/00001-stylegan3-r-animal_dataset-512x512-gpus1-
batch8-gamma8"
}

Output directory:    /kaggle/working/animal-stylegan-results-512x512/00001-stylegan3-r-animal_dataset-512x512-
gpus1-batch8-gamma8
Number of GPUs:      1
Batch size:          8 images
Training duration:   5000 kimg
Dataset path:        /kaggle/working/animal_dataset-512x512.zip
Dataset size:        5000 images
Dataset resolution:  512
```

9.4.3. Generator parameters

Generator	Parameters	Buffers	Output shape	Datatype
mapping.fc0	262656	-	[8, 512]	float32
mapping.fc1	262656	-	[8, 512]	float32
mapping	-	512	[8, 16, 512]	float32
synthesis.input.affine	2052	-	[8, 4]	float32
synthesis.input	1048576	3081	[8, 1024, 36, 36]	float32
synthesis.L0_36_1024.affine	525312	-	[8, 1024]	float32
synthesis.L0_36_1024	1049600	157	[8, 1024, 36, 36]	float32
synthesis.L1_36_1024.affine	525312	-	[8, 1024]	float32
synthesis.L1_36_1024	1049600	157	[8, 1024, 36, 36]	float32
synthesis.L2_52_1024.affine	525312	-	[8, 1024]	float32
synthesis.L2_52_1024	1049600	169	[8, 1024, 52, 52]	float32
synthesis.L3_52_1024.affine	525312	-	[8, 1024]	float32
synthesis.L3_52_1024	1049600	157	[8, 1024, 52, 52]	float32
synthesis.L4_84_1024.affine	525312	-	[8, 1024]	float32
synthesis.L4_84_1024	1049600	169	[8, 1024, 84, 84]	float16
synthesis.L5_84_1024.affine	525312	-	[8, 1024]	float32
synthesis.L5_84_1024	1049600	157	[8, 1024, 84, 84]	float16
synthesis.L6_148_1024.affine	525312	-	[8, 1024]	float32
synthesis.L6_148_1024	1049600	169	[8, 1024, 148, 148]	float16
synthesis.L7_148_967.affine	525312	-	[8, 1024]	float32
synthesis.L7_148_967	991175	157	[8, 967, 148, 148]	float16
synthesis.L8_276_645.affine	496071	-	[8, 967]	float32
synthesis.L8_276_645	624360	169	[8, 645, 276, 276]	float16
synthesis.L9_276_431.affine	330885	-	[8, 645]	float32
synthesis.L9_276_431	278426	157	[8, 431, 276, 276]	float16
synthesis.L10_532_287.affine	221103	-	[8, 431]	float32
synthesis.L10_532_287	123984	169	[8, 287, 532, 532]	float16
synthesis.L11_532_192.affine	147231	-	[8, 287]	float32
synthesis.L11_532_192	55296	157	[8, 192, 532, 532]	float16
synthesis.L12_532_128.affine	98496	-	[8, 192]	float32
synthesis.L12_532_128	24704	25	[8, 128, 532, 532]	float16
synthesis.L13_512_128.affine	65664	-	[8, 128]	float32
synthesis.L13_512_128	16512	25	[8, 128, 512, 512]	float16
synthesis.L14_512_3.affine	65664	-	[8, 128]	float32
synthesis.L14_512_3	387	1	[8, 3, 512, 512]	float16
synthesis	-	-	[8, 3, 512, 512]	float32
---	---	---	---	---
Total	16665594	5588	-	-

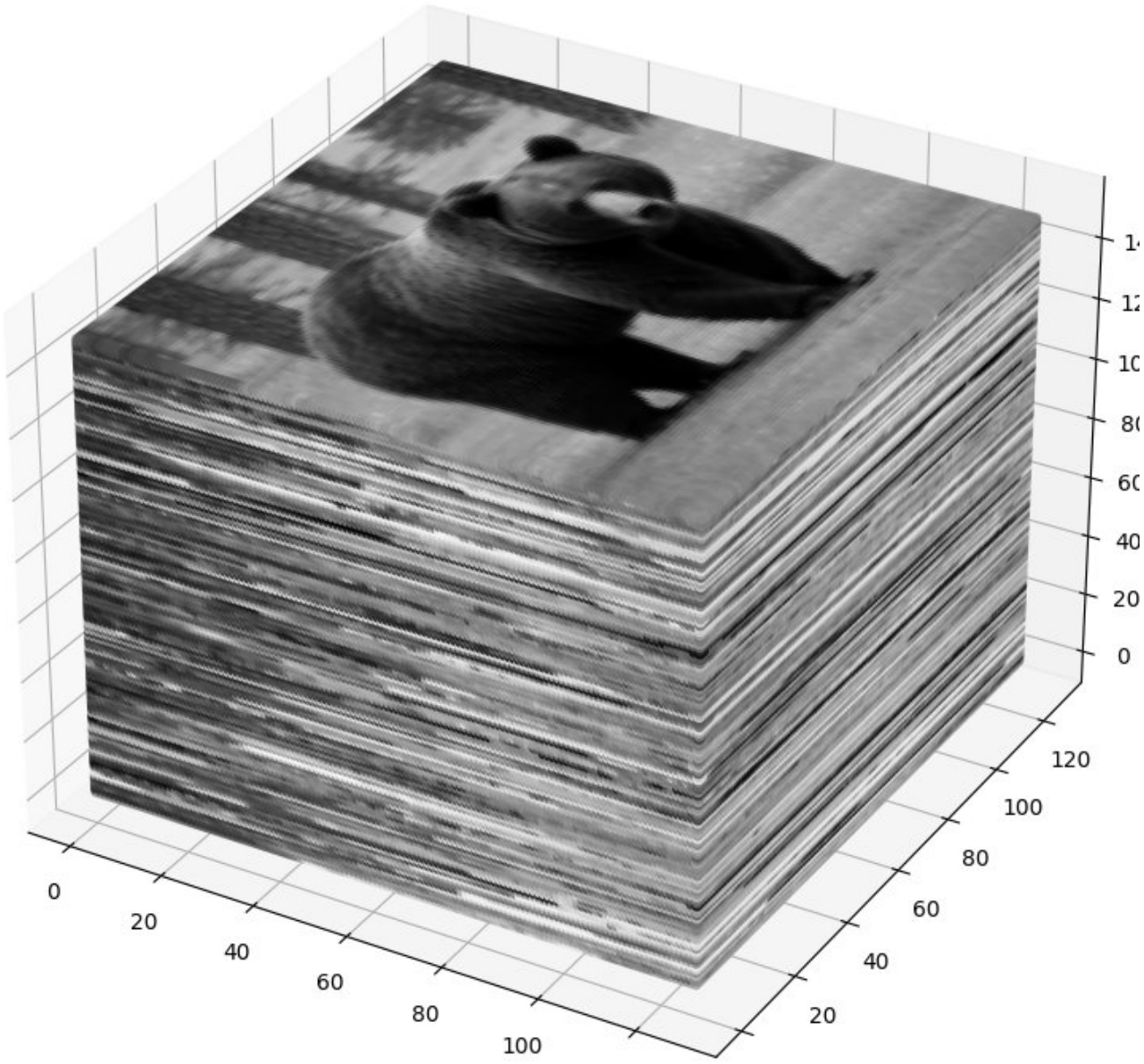
9.4.4 Discriminator Parameter

Done.

Discriminator	Parameters	Buffers	Output shape	Datatype
b512.fromrgb	256	16	[8, 64, 512, 512]	float16
b512.skip	8192	16	[8, 128, 256, 256]	float16
b512.conv0	36928	16	[8, 64, 512, 512]	float16
b512.conv1	73856	16	[8, 128, 256, 256]	float16
b512	-	16	[8, 128, 256, 256]	float16
b256.skip	32768	16	[8, 256, 128, 128]	float16
b256.conv0	147584	16	[8, 128, 256, 256]	float16
b256.conv1	295168	16	[8, 256, 128, 128]	float16
b256	-	16	[8, 256, 128, 128]	float16
b128.skip	131072	16	[8, 512, 64, 64]	float16
b128.conv0	590080	16	[8, 256, 128, 128]	float16
b128.conv1	1180160	16	[8, 512, 64, 64]	float16
b128	-	16	[8, 512, 64, 64]	float16
b64.skip	262144	16	[8, 512, 32, 32]	float16
b64.conv0	2359808	16	[8, 512, 64, 64]	float16
b64.conv1	2359808	16	[8, 512, 32, 32]	float16
b64	-	16	[8, 512, 32, 32]	float16
b32.skip	262144	16	[8, 512, 16, 16]	float32
b32.conv0	2359808	16	[8, 512, 32, 32]	float32
b32.conv1	2359808	16	[8, 512, 16, 16]	float32
b32	-	16	[8, 512, 16, 16]	float32
b16.skip	262144	16	[8, 512, 8, 8]	float32
b16.conv0	2359808	16	[8, 512, 16, 16]	float32
b16.conv1	2359808	16	[8, 512, 8, 8]	float32
b16	-	16	[8, 512, 8, 8]	float32
b8.skip	262144	16	[8, 512, 4, 4]	float32
b8.conv0	2359808	16	[8, 512, 8, 8]	float32
b8.conv1	2359808	16	[8, 512, 4, 4]	float32
b8	-	16	[8, 512, 4, 4]	float32
b4.mbstd	-	-	[8, 513, 4, 4]	float32
b4.conv	2364416	16	[8, 512, 4, 4]	float32
b4.fc	4194816	-	[8, 512]	float32
b4.out	513	-	[8, 1]	float32
---	---	---	---	---
Total	28982849	480	-	-

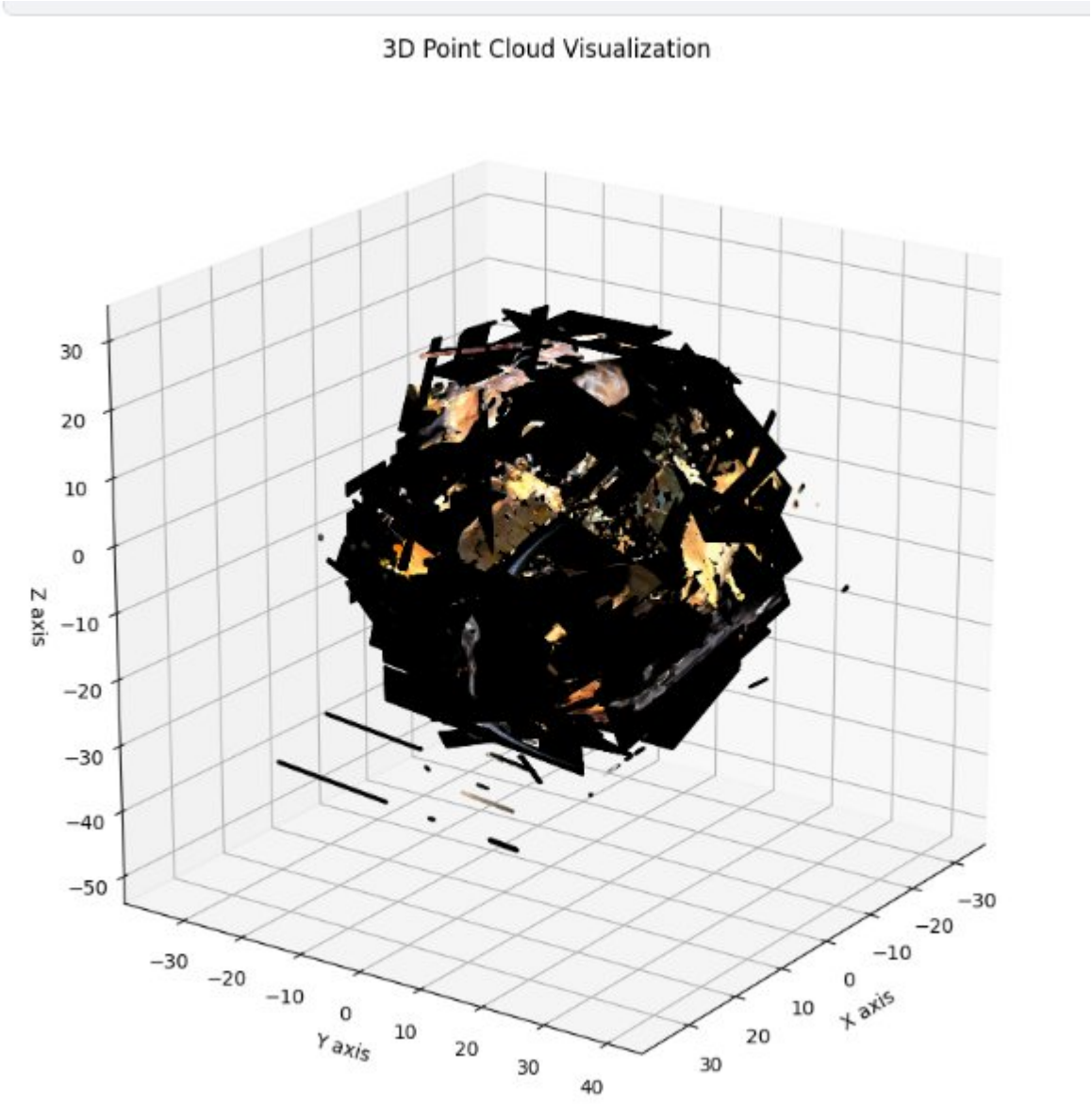
10. RESULTS

10.1. Image creation (without pose estimation)



We are leveraging TensorFlow's convolutional neural network (CNN) layers to process a dataset containing over 100 images of the same object category. The goal is to extract deep spatial features across multiple viewpoints, which can then be aggregated and interpreted to assist in the generation or reconstruction of a 3D representation of the object. This workflow involves feeding the image data through a series of convolutional layers to capture low-level to high-level visual features, which serve as the foundation for depth estimation or volumetric modeling necessary for 3D image creation.

10.2. Image creation using MVS

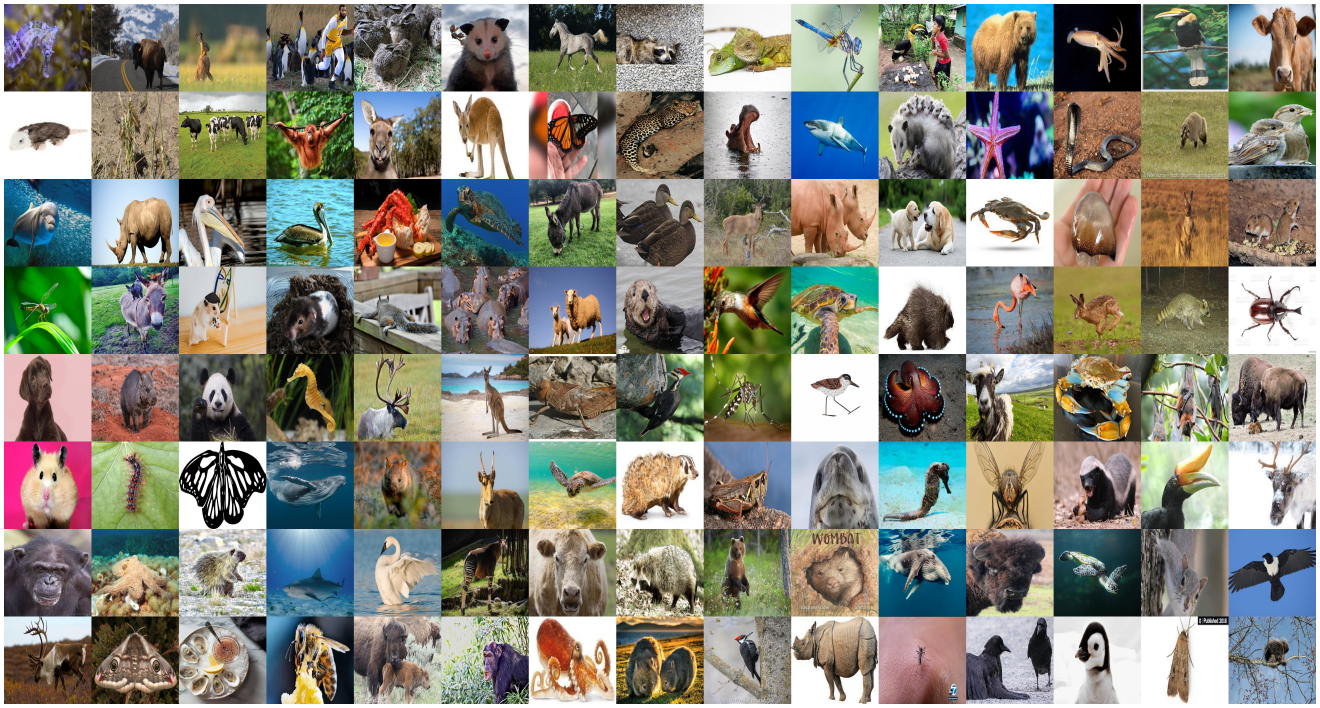


We are applying Multi-View Stereo (MVS) techniques for 3D image reconstruction and visualization. The goal is to generate a detailed 3D representation of an object or scene using multiple 2D images captured from different viewpoints. Once the 3D model is reconstructed, we can render or extract a novel 2D view from any desired angle or position within the 3D space. This allows for the generation of new 2D images by simulating how the object or scene would appear from that specific viewpoint, enabling flexible image synthesis and interactive visualization.

10.3. StyleGAN3 Model fine tuning

10.3.1 Fake Images by GAN3 Model

10.3.2 Real Images by GAN3 Model



10.3.3 Generated Images

We are fine-tuning the pre-trained StyleGAN3-R model—originally trained on animal images—using our custom dataset consisting of 90 high-quality images representing different animals. The goal of this fine-tuning process is to adapt the model's generative capabilities to better capture the specific characteristics and visual styles of the animals in our dataset. This involves customizing the generator network through transfer learning techniques to ensure it can synthesize new, high-fidelity animal images that align closely with our unique data distribution.

11. CONCLUSION

The project introduces a comprehensive and intelligent framework for generating high-quality, realistic animal images by fine-tuning a pre-trained Generative AI model. Leveraging techniques such as GANs, the system learns intricate features and patterns within the animal dataset to produce visually coherent and diverse images. This approach not only enhances the fidelity and variety of generated images but also demonstrates the potential of generative AI in fields like wildlife research, education, digital content creation, and data augmentation. By combining domain-specific training with advanced generative techniques, the project contributes to the growing capabilities of AI-driven image synthesis.

12. REFERENCES

1. Alias-Free Generative Adversarial Networks (https://arxiv.org/abs/2106.12423)
2. DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation (https://arxiv.org/pdf/2208.12242)
3. SurfaceNet+: An End-to-end 3D Neural Network for Very Sparse Multi-View Stereopsis (https://openaccess.thecvf.com/content_ICCV_2017/papers/Ji_SurfaceNet_An_End-To-End_ICCV_2017_paper.pdf)
4. Animal Image Dataset (90 Different Animals) (https://www.kaggle.com/api/v1/datasets/download/iamsouravbanerjee/animal-image-dataset-90-different-animals)
5. Stepic (https://pypi.org/project/stepic/)
6. Tensorflow (https://www.tensorflow.org/)