B.Tech Project CSE Dept.

Light-Weight Generative Adversarial Model

Department of Computer Science and Engineering

Rushil Ashish Shah(B20AI036)

Ruthvik K(B20AI037)

Introduction

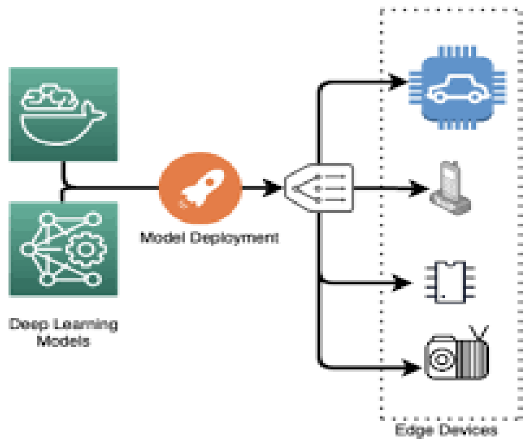
Background:

As the demand for real-time and resource-efficient artificial intelligence (AI) applications continues to rise, there is a growing need for deploying machine learning models on edge devices, especially in hardware-constrained environments. Generative Adversarial Networks (GANs) have demonstrated remarkable success in generating realistic data, but their deployment on edge devices poses significant challenges due to limited computational resources.

Objective:

The primary goal of this project is to develop a Light-Weight Generative Adversarial Model (LW-GAN) optimized for deployment on edge devices and tailored for specific hardware applications. The model should be capable of generating high-quality data with minimal computational and memory requirements.

Problem Statement: Design and Implementation of a Light-Weight Generative Adversarial Model for Edge Devices in Hardware Applications



Scope of Work:

- **Model Architecture**: Design a lightweight GAN architecture that minimizes computational complexity and memory requirements while preserving the ability to generate realistic data.
- **Optimization Techniques**: Implement optimization techniques, such as quantization, pruning, and model distillation, to reduce the model size and computational load.
- **Energy-efficient Implementation**: Investigate and implement energy-efficient algorithms and optimizations to reduce power consumption during inference.
- **Testing and Validation**: Evaluate the performance of the LW-GAN across diverse hardware devices and applications. Conduct rigorous testing to ensure the model's efficiency, accuracy, and reliability.
- **Real-time Inference**: Develop strategies to ensure real-time inference on edge devices, considering the limited processing power and the need for low-latency performance.

Expected Outcomes:

- A light-weight GAN model capable of real-time, energy-efficient inference on edge devices.
- Implementation of hardware-specific optimizations for seamless integration with targeted edge device architectures.
- Demonstration of the LW-GAN's effectiveness across various hardware applications, showcasing its versatility and adaptability.

Literature Review

Literature Review -CYCLE GAN

Introduction:

Image-to-image translation, the task of transforming an input image from one domain to another, has witnessed significant advancements with the introduction of Generative Adversarial Networks (GANs). Among various approaches, CycleGAN has emerged as a notable framework for unsupervised image translation. Proposed by Zhu et al. in the seminal paper titled "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" in 2017, CycleGAN introduced a novel cycle-consistency loss, enabling the model to learn mappings between two domains without paired training data.

Foundations of CycleGAN:

CycleGAN builds upon the GAN architecture, incorporating a cycle-consistency constraint to address the challenge of unpaired image translation. Traditional GANs rely on paired datasets, where each input in the source domain corresponds to a specific target domain output. However, acquiring paired data can be impractical and expensive in real-world scenarios. CycleGAN overcomes this limitation by introducing a cycle-consistency loss, enforcing that mapping an image from one domain to another and back should reconstruct the original image.

Adversarial Learning:

CycleGAN leverages adversarial learning with two main components: generators and discriminators. It includes two generators (G_A and G_B) responsible for translating images from domain A to B and vice versa. Additionally, there are two discriminators (D_A and D_B) that aim to distinguish between real and translated images. The adversarial loss encourages the generators to produce realistic translations, while the cycle-consistency loss enforces consistency between the original and reconstructed images.

Cycle-Consistency Loss:

The cycle-consistency loss is a key innovation in CycleGAN. It introduces a cycle-consistency term, which measures the difference between the input image and the image reconstructed after a round-trip translation. This loss ensures that the translation is not only realistic but also reversible. The introduction of the cycle-consistency constraint addresses issues like mode collapse and the lack of diversity in generated images, making CycleGAN particularly effective in scenarios where paired data is scarce.

Applications and Extensions:

CycleGAN has found applications in various domains, including style transfer, image-to-image translation, and domain adaptation. It has been extensively used for transforming images between different visual domains such as turning photographs into paintings, converting satellite images to maps, or even translating images of horses to zebras and vice versa. The framework's success has led to numerous extensions and improvements, exploring modifications to the architecture, loss functions, and training strategies to enhance performance.

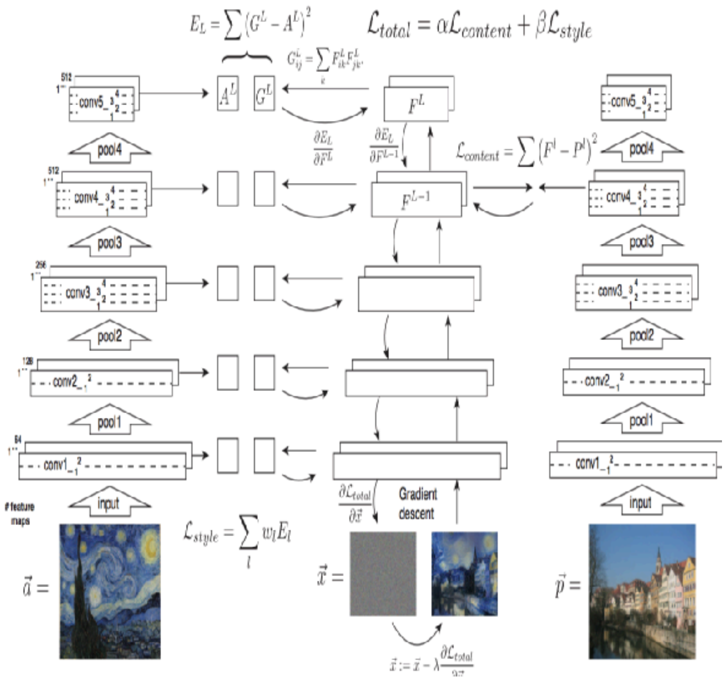
Literature Review -Neural Style Transfer

Introduction

The paper introduces the problem of neural style transfer, which is to create an image that has the content of one image and the style of another image. The research also gives some examples of neural style transfer applications, such as artistic expression, photo enhancement, and video stylization.

+
=
Picasso Dancer

Architecture


$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}$
A^L
G^L
F^L
F^{L-1}
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
conv5_3
pool4
conv4_3
pool3
conv3_3
pool2
conv2_1
pool1
conv1_1
input
# feature maps
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$

First content and style features are extracted and stored.Thestyleimageaispassed throughthenetworkanditsstylerepresentationAl on all layers included recomputed stored(left). The Content image p is passed through the network and the content representation Pl in one layer stored(right). Then a random white noise image is passed through the network anditsstylefeaturesGl and content featuresFl are computed. On Each Layer included in the style representation, the element-wise mean squared difference between Gi and Ai is computed give the style loss L(left).Also The mean squared difference between Fi and Pi is computed to give the content loss L (right).The total loss L total is then a linear combination between content and the styleloss.

Various Algorithms used

Gatys et al.'s algorithm: The first algorithm that proposed to use CNN features to measure content and style losses, and optimize the output image using gradient descent. This algorithm produces high-quality results, but is slow and requires manual tuning of hyperparameters.

Perceptual losses: A technique that uses a pre-trained CNN to define a perceptual loss function, and trains a feed-forward network to generate stylized images. This technique is faster than Gatys et al.'s algorithm, but has limited flexibility and generalization ability.

Conditional instance normalization: A technique that modifies the instance normalization layer in a feed-forward network to condition on different style images. This technique allows for fast and flexible style transfer, but may lose some fine-grained style details.

Also introduces a technique for neural style transfer on videos where the neural network takes consecutive frames as input instead of single frames. The neural network takes the previous stylized frame and the current video frame as input. The output at each time step is fed as input at the next time step.the research just gave a basic idea without going into technical details.

Dataset

The "Apple2Orange" dataset is a collection of images commonly used in the domain of image-to-image translation and computer vision. This dataset is designed to facilitate research and development in the area of generative adversarial networks (GANs) and image synthesis, where the objective is to convert images from one domain to another. In the case of the Apple2Orange dataset, the focus is on transforming images from the domain of apples to the domain of oranges and vice versa.

Implementation CycleGAN:

1. Generator Architecture - U-Net vs. ResNet:

U-Net Architecture: The U-Net generator architecture is a popular choice in CycleGAN implementations. It is characterized by a symmetric encoder-decoder structure with skip connections between corresponding layers of the encoder and decoder. These skip connections enable the generator to retain and refine high-resolution details during the image translation process. U-Net generators are well-suited for image-to-image translation tasks, providing a balance between receptive field size and detailed feature preservation.

ResNet Architecture: While U-Net is commonly used, some implementations may opt for a ResNet-like generator architecture. Residual connections in ResNet architectures help mitigate the vanishing gradient problem, allowing for the training of deeper networks. This can be beneficial when dealing with complex image translation tasks, and ResNet generators have shown success in various GAN applications. However, U-Net remains a popular and effective choice due to its simplicity and efficiency.

2. Instance Normalization in Generators:

Instance Normalization vs. Batch Normalization: In CycleGAN, the choice of normalization layers in the generator can impact the quality of the generated images. Instance Normalization (IN) is favored over Batch Normalization (BN) in style transfer problems. IN normalizes each instance (or channel) independently, making it suitable for tasks where the style and appearance of each instance (e.g., pixels or channels) should be preserved. IN is particularly effective in preventing the loss of image details and textures, contributing to more visually satisfying results.

3. Discriminator Architecture:

Convolutional Layers with Instance Normalization: The discriminator in CycleGAN typically consists of a series of convolutional layers, each followed by instance normalization. Instance normalization is applied to normalize the features at each spatial location independently. This normalization choice helps the discriminator focus on capturing style-related information while maintaining spatial details. The first layer of the discriminator might differ and could use other normalization techniques, such as batch normalization.

4. Training Procedure:

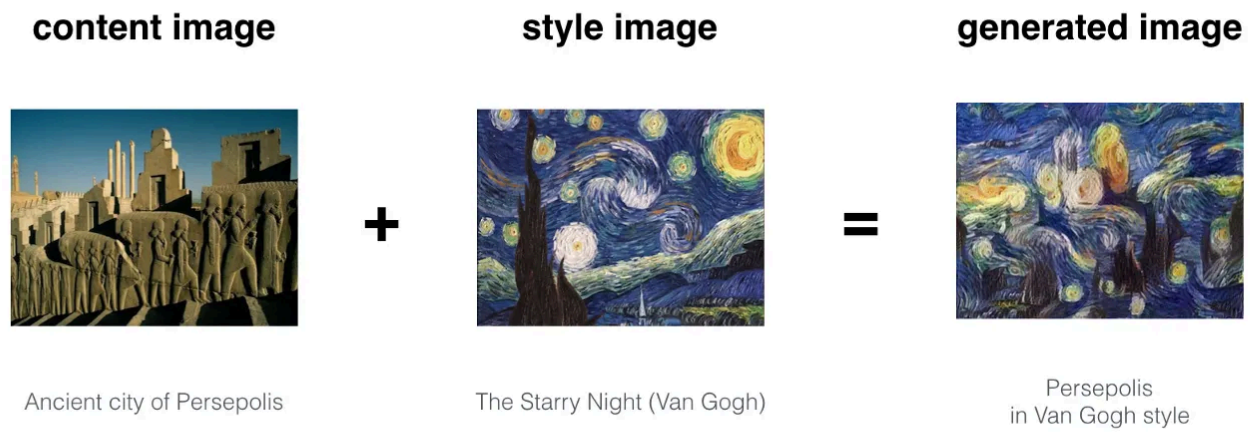
Alternating Training of Discriminators and Generators: Training in CycleGAN follows the standard GAN practice of alternating between the training of discriminators and generators. This process involves updating the parameters of the discriminator to better distinguish between real and generated samples, and then updating the generator parameters to minimize the discrepancy between generated and real samples. The alternating training helps strike a balance in the adversarial learning process and encourages the generator to produce more realistic translations.

Combined Model: The combined model consists of both the generator and the discriminator. The generator aims to generate realistic-looking images, while the discriminator aims to distinguish between real and generated images. The adversarial training process involves optimizing both the generator and discriminator simultaneously to achieve a Nash equilibrium where the generator produces high-quality translations and the discriminator is unable to reliably differentiate between real and generated samples.

In summary, the implementation choices in CycleGAN, such as the generator architecture (U-Net), the use of Instance Normalization, and the design of the discriminator, are tailored to the specific requirements of style transfer problems. These choices contribute to the model's ability to capture and transfer the desired stylistic features between domains. The alternating training procedure ensures the effective learning of both the generator and discriminator, fostering the generation of high-quality and visually appealing translated images.

Implementation of NST:

Neural style transfer is a method of stylizing the contents of an image with the styles of another image.

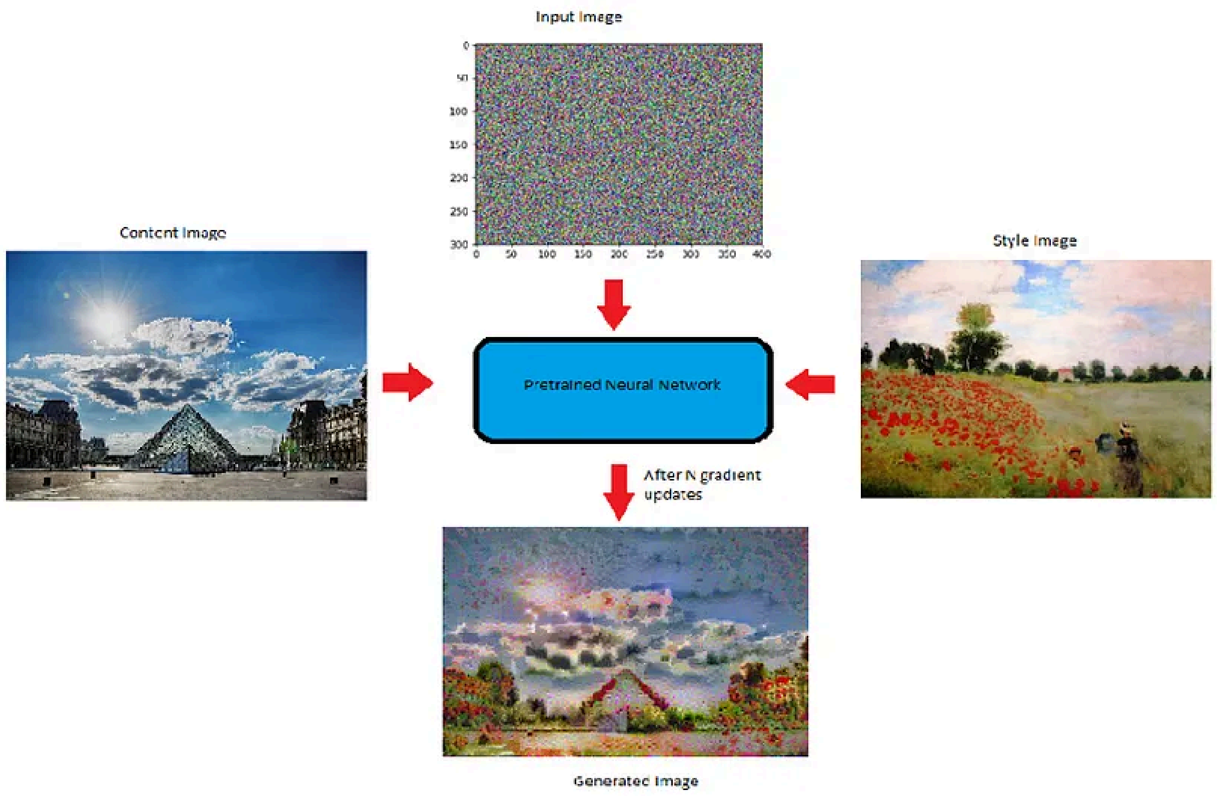


As we can see, the generated image is having the content of the Content image and style of the style image. It can be seen that the above result cannot be obtained simply by overlapping the images

Architecture

We use CNN architecture to achieve nst, as we move across the layers the network starts to capture more and more complex features. capturing of different simple and complex features is called feature representation and this encoding nature of Convolutional Neural Networks can help us in Neural Style Transfer.

Working of the NST:



Firstly, we initialize a noisy image, which is going to be our output image(G). We then calculate how similar this image is to the content and style image at a particular layer in the network(VGG network). Since we want that our output image(G) should have the content of the content image(C) and style of style image(S) we calculate the loss of generated image(G) w.r.t to the respective content(C) and style(S) image.

Content Loss.

$$L^l_{content}(p, x) = \sum_{i,j} (F^l_{ij}(x) - P^l_{ij}(p))^2$$

Calculating content loss means how similar is the randomly generated noisy image(G) to the content image(C), in mathematical terms. Content loss is calculated by Euclidean distance between the respective intermediate higher-level feature representation of input image (x) and content image (p) at a particular layer

Style Loss.

$$L_{style}(a, x) = \sum_{l \in L} w_l E_l \quad E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

Style loss is conceptually different from Content loss. We cannot just compare the intermediate features of the two images and get the style loss. That's why we introduce a new term called Gram matrices.Gram matrix is a way to interpret style information in an image as it shows the overall distribution of features in a given layer.It is measured as the amount of correlation present between features maps in a given layer.

Total Loss

$$\boxed{\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})}$$

minimizing the total loss(content+style) using backpropagation which in turn will optimize our randomly generated image into a meaningful piece of art.

This sums up the working of Neural Style Transfer.

Optimization Techniques

Use of quantization in Cycle GAN and Neural Style Transfer is motivated by the need for efficient deployment on edge devices, reduction of memory requirements, faster inference times, and improved energy efficiency, all while maintaining the essential qualities of the generated images.

8-bit Quantization:

CycleGAN: Reduces model precision from 32 to 8 bits for efficient memory usage and faster inference.

NST: Converts model parameters to 8-bit integers, reducing model size without compromising performance.

Knowledge Distillation:

CycleGAN: Trains a smaller model to mimic a larger, pre-trained CycleGAN, achieving a compact yet effective model.

NST: Utilizes a student model to reproduce style transfer capabilities from a larger, pre-trained NST model.

Quantization Aware Training:

CycleGAN: Trains the model with awareness of future quantization, optimizing for reduced precision during deployment.

NST: Incorporates quantization-aware techniques during training to minimize the impact of reduced precision in the quantized model.

These techniques enhance model efficiency, making them suitable for deployment on resource-constrained devices while maintaining functionality and performance.

This is an overview of techniques now we will be discussing each one of them

8-bit Quantization:

Quantization in the context of neural networks refers to the process of reducing the precision of the weights and activations of a model. In the case of 8-bit quantization, the values are restricted to 8 bits, allowing for a more compact representation compared to the typical 32-bit floating-point format. This reduction in precision can lead to decreased memory requirements and faster inference times, making models more efficient, particularly in scenarios where computational resources are limited.

This is how we applied it to our Cycle GAN and NST models:

1. Cycle GAN

Generator and Discriminator: In a Cycle GAN, there are two main components: a generator and a discriminator. The generator creates images, and the discriminator tries to distinguish between real and generated images.

Quantization in Cycle GAN: Applying 8-bit quantization to the weights and activations of the Cycle GAN's generator and discriminator can help reduce the model's memory footprint and improve computational efficiency. This is crucial, especially in deployment scenarios where resources may be limited.

2.Neural Style Transfer

Style Transfer Networks: Neural Style Transfer involves separating and recombining the content and style of two images. This process often employs a pre-trained deep neural network.

Quantization in Neural Style Transfer: The network used for Neural Style Transfer can also benefit from quantization. By converting the model parameters to 8-bit precision, one can achieve a more memory-efficient representation, making it easier to deploy on devices with restricted resources.

Knowledge Distillation:

Knowledge Distillation is a process in which a smaller, more lightweight model (student) is trained to mimic the behavior of a larger, more complex model (teacher). The goal is to transfer the knowledge and generalization capabilities of the teacher model to the smaller student model. This can be particularly useful in scenarios where deploying a large model is impractical due to resource constraints.

1. Cycle GAN

Teacher Model (Large GAN): The Cycle GAN itself can act as a teacher model, with a large and complex architecture capable of generating high-quality images and performing tasks like image-to-image translation.

Student Model (Distilled GAN): The distilled model, or student, is a smaller and more lightweight version of the Cycle GAN. It is trained to reproduce the essential features and capabilities of the teacher model while being computationally more efficient.

Knowledge Transfer in Cycle GAN: During the knowledge distillation process, the student GAN learns not only from the ground truth data but also from the intermediate representations and decisions made by the teacher GAN. This can help the student GAN achieve similar performance to the teacher GAN in generating realistic images while requiring fewer computational resources.

2.Neural Style Transfer

Teacher Model (Complex Style Transfer Network): In the context of Neural Style Transfer, a teacher model could be a complex neural network trained for the task of separating and recombining content and style in images.

Student Model (Distilled Style Transfer Network): The student model, on the other hand, is a simplified version of the style transfer network that captures the essential aspects of the style transfer process.

Knowledge Transfer in Neural Style Transfer: The student model learns to mimic the artistic style of the teacher model, capturing the nuances of style transfer while being more lightweight and suitable for deployment on devices with limited resources.

Quantization Aware Training:

Quantization Aware Training (QAT) is a technique used to train neural networks in a way that prepares them for deployment with reduced precision, such as 8-bit quantization. The goal is to ensure that the model learns to be robust to the effects of lower precision during inference, thereby improving efficiency without sacrificing performance. Quantization Aware Training is particularly useful for models like Cycle GAN, where resource efficiency is crucial.

QAT involves training the model with the awareness that it will be quantized later during deployment. The model is trained using simulated quantization, introducing quantization noise and rounding errors during training.

QAT tends to preserve model accuracy better because the model is trained with quantization in mind. PTQ might suffer a slight drop in accuracy due to the lack of awareness during training.

Results

Cycle GAN Generator Loss, Model Size

Before any Optimization	8-bit Quantization	Knowledge distillation	Quantization Aware Training
3.572, 64MB	4.754, 19MB	4.542, 45MB	6.52, 51MB

NST Total Variation Loss, Model Size

Before any Optimization	8-bit Quantization	Knowledge distillation
6.6684, 61MB	12.1024, 15MB	29.0122, 25MB

Conclusion

In conclusion, the development of a Light-Weight Generative Adversarial Model (LW-GAN) has been a significant achievement, marked by successful efforts to reduce model size while maintaining its generative capabilities. The emphasis on resource efficiency and adaptability for deployment on edge devices addresses the critical need for AI models that can operate effectively in hardware-constrained environments.

The streamlined architecture, optimization techniques, and real-time inference strategies implemented in the LW-GAN contribute to its effectiveness in generating high-quality synthetic data with minimal computational and memory requirements. The outcomes of this project hold promise for a wide range of applications, from image synthesis to signal processing, where edge devices play a crucial role.

Future Work:

While the current project has made substantial progress in optimizing the LW-GAN, there are several avenues for future research and development:

- Security Enhancements: In the next phase of our work, we aim to fortify the LW-GAN against adversarial attacks. Adversarial attacks pose a significant threat to the robustness of machine learning models, and implementing security features will be crucial for ensuring the reliability of the LW-GAN in real-world scenarios.
- Privacy Preservation: Addressing privacy concerns is paramount, especially in applications where sensitive data is involved. Future work will explore techniques to enhance privacy preservation within the LW-GAN, considering the decentralized nature of edge computing and the potential exposure of user data.
- Multi-Modal Capabilities: Expanding the LW-GAN's capabilities to handle multiple modalities of data, such as text, audio, and video, is another area of interest. This extension will make the model more versatile and applicable to a broader range of edge computing applications.
- Quantitative Evaluation: Conducting an extensive quantitative evaluation on a diverse set of edge devices and hardware platforms is essential for validating the LW-GAN's performance in different real-world scenarios. This will involve

benchmarking the model across various parameters, including speed, energy efficiency, and generative quality.

- User-Friendly Deployment: Streamlining the deployment process of the LW-GAN on edge devices and ensuring user-friendly integration with different applications will be a focus of future work. This involves developing tools and interfaces that simplify the implementation of the LW-GAN in diverse edge computing environments.

In summary, the journey from a Light-Weight Generative Adversarial Model to a robust, secure, and versatile solution for edge devices is ongoing. The envisioned future work aims not only to strengthen the model's security features but also to extend its capabilities, making it a reliable and efficient tool for various edge computing applications in the ever-evolving landscape of artificial intelligence.