# Synthetic Image Data Generation using GAN with Statistical Similarity

**BHAVIN NAGINBHAI NIRMAL**

A Project

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the

Requirements for the Degree

**Master of Science in Information Technology**

**Concordia University of Edmonton**

**Department of Mathematical and Physical Sciences**

**Faculty of Graduate Studies**

Edmonton, Alberta

August, 2023

CONCORDIA UNIVERSITY OF EDMONTON

© Copyright 2023 by Bhavin Naginbhai Nirmal

# Synthetic Image Data Generation using GAN with Statistical Similarity

Bhavin Naginbhai Nirmal

**Approved:**

Dr. Nasim Hajari, Ph.D.

| | |
|---|---|
| Supervisor | Date: August 12, 2023 |
| Committee Member | Date: |

Dr. Patrick Kamau, Ph.D.

| | |
|---|---|
| Dean of Graduate Studies | Date: August 12, 2023 |

## Abstract

Synthetic image generation using Generative Adversarial Networks (GANs) has emerged as a promising technique to address the challenge of limited datasets in the field of garbage classification. With supervised machine learning algorithms relying on labeled data and larger training examples, the small size and scarcity of annotated samples in the medical imaging domain pose significant obstacles. Traditional data augmentation techniques offer limited diversity, motivating the exploration of synthetic data examples to introduce more variability and enhance the training process. GANs, known for their ability to generate high-quality and realistic images, have gained popularity in computer vision tasks and have been successfully applied to medical imaging applications. In this study, we investigate the application of GANs for synthetic image generation in the garbage classification dataset. By training the GAN model with a large number of images, we aim to generate synthetic images that closely resemble real images, thereby expanding the dataset and improving the robustness of garbage classification algorithms. We also evaluate the similarity between the real and generated images using metrics such as the Inception Score and Frechet Inception Distance. Through our research, we seek to demonstrate the efficacy of GAN-based synthetic image generation as a means to enhance the garbage classification dataset and improve the accuracy of classification algorithms for efficient waste management.

**Keywords**: Synthetic Image Data Generation, GAN, Inception Score, Frechet Inception Distance, Machine Learning.

# Acknowledgments

I would like to take this opportunity to express my deepest gratitude and appreciation to all the individuals and resources that have contributed to the successful completion of this project. Without their support and guidance, this endeavor would not have been possible.

First and foremost, I extend my heartfelt thanks to the authors and creators of the referenced works. Their pioneering research and valuable insights have served as the cornerstone of our project, providing us with a strong foundation to build upon.

I am also immensely grateful for the assistance provided by ChatGPT and other language models. These powerful AI tools have played a significant role in shaping our ideas, refining the concepts, and enhancing the overall quality of my project.

Furthermore, I would like to extend my appreciation to my mentor whose expertise, constructive feedback, and encouragement have been instrumental in guiding me through the challenges and complexities of the project. Her unwavering support has motivated me to push the boundaries of my capabilities and strive for excellence.

## Contents

## List of Figures

# 1 Introduction

For resolving data availability concerns in a variety of fields, including computer vision, natural language processing, and medicine, synthetic dataset generation has emerged as a promising approach [1]. In recent years, the application of Generative Adversarial Networks (GANs) for synthetic image data generation has gained significant attention and has proven to be highly effective. GANs are deep learning models that consist of a generator network and a discriminator network, working together in a competitive fashion to generate realistic and high-quality images.

Numerous recent papers have contributed to the advancement of synthetic image data generation using GANs, proposing innovative architectures and training strategies to improve the quality and diversity of generated images. One prominent research direction focuses on the development of conditional GANs, which allow the generation of images conditioned on specific attributes or class labels [6]. By incorporating additional information into the GAN framework, researchers have achieved impressive results in tasks such as image synthesis, image-to-image translation, and style transfer.

Synthetic image data generation plays a pivotal role in various computer vision applications, enabling researchers and developers to overcome limitations associated with scarce or expensive real-world data [2]. In the context of Garbage Classification Dataset, generating synthetic images using a Generative Adversarial Network (GAN) has become an increasingly important technique. By training the GAN model on a large number of images, the aim is to generate synthetic images that closely resemble real images, thus expanding the dataset and enhancing the robustness of garbage classification algorithms [1].

Recent research has highlighted the significance of assessing the similarity between real and generated images in order to evaluate the effectiveness of the GAN model. Two widely used metrics for this purpose are the Inception Score and the Frechet Inception Distance. The Inception Score measures the quality and diversity of the generated images by evaluating the conditional class probabilities assigned by an Inception model [4]. On the other hand, the Frechet Inception Distance quantifies the dissimilarity between the feature distributions of real and generated images, providing a more comprehensive measure of similarity.

To achieve the goal of generating high-quality synthetic images for the Garbage Classification Dataset, researchers have explored various techniques and architectures within the GAN framework. Conditional GANs have proven to be effective by incorporating specific attributes or class labels into the generation process, allowing for the synthesis of images tailored to different garbage classes [6]. Additionally, advancements in unsupervised learning with GANs have enabled the development of meaningful representations and improved the overall quality and diversity of generated images [8].

This report aims to present a comprehensive implementation of synthetic image data generation for the Garbage Classification Dataset. By training a GAN model with a large number of images, we focus on generating synthetic images that closely resemble real images in order to enhance the dataset and improve the performance of garbage classification algorithms. Moreover, we evaluate the similarity between the real and generated images using the Inception Score and Frechet Inception Distance, providing quantitative measures of the GAN model's effectiveness.

## 2 Literature review

The field of synthetic image data generation has proven to be crucial in various domains, particularly when dealing with limited datasets and a scarcity of annotated samples. In the context of medical imaging, where annotations are often time-consuming and require expert knowledge, the challenge becomes even more pronounced. While public medical datasets and challenges have provided some relief, they still have limitations in terms of size and applicability to specific medical problems. Collecting medical data is an intricate and expensive process that necessitates collaboration between researchers and radiologists [1].

To address these challenges, researchers have explored data augmentation techniques, which typically involve simple modifications of dataset images, such as translation, rotation, flip, and scale. While these techniques have become standard practice in computer vision tasks and provide some diversity, the variability they introduce remains relatively limited [1]. Consequently, the use of synthetic data examples has emerged as a promising approach to enhance dataset variability and improve the training process of systems.

Generative Adversarial Networks (GANs) have gained significant popularity in the computer vision community and have shown promise in generating high-quality and realistic natural images. GANs consist of a generator network and a discriminator network that compete against each other during training. Recently, the GAN framework has also been applied to various medical imaging applications [1]. These studies have employed GANs for tasks such as label-to-segmentation translation, segmentation-to-image translation, cross-modality translations, and image inpainting [1]. By leveraging the GAN framework, the study seeks to create synthetic medical images that can be used for data augmentation. This data augmentation technique is expected to boost the performance of liver lesion classification models by increasing the diversity and size of the training dataset [1].

In the specific domain of liver lesion classification, there is a pressing need to develop automated diagnostic tools to assist radiologists in diagnosing liver lesions, as liver cancer is a significant cause of mortality worldwide. Prior studies have proposed methods for automatic classification of focal liver lesions in CT images [1]. By maximizing the use of the GAN's learned parameters, the authors seek to overcome the limitations of previous methods and provide a better way to train deep CNNs using synthetic data [10]. This strategy is expected to improve the realism and diversity of the generated examples, ultimately enhancing the performance of deep CNNs trained on the synthetic data [10].

Large amounts of synthetic data have been produced using Generative Adversarial Networks (GANs). This data is used to train deep convolutional neural networks (CNNs) by supplementing with real-world examples. Studies have demonstrated that the produced instances are deficient in variety and don't have enough realism to train deep CNNs [11]. We propose our straightforward, practical, and easy-to-implement synthetic data generation method to train the model more effectively and precisely, in contrast to prior research that randomly augmented the synthetic data with actual data [11].

Generative Adversarial Networks (GAN) provide training samples that replicate the distribution of the actual dataset. These GAN-synthesized picture samples are utilised extensively in medical imaging for dataset augmentation; further uses for the images include image translation, registration, super-resolution, denoising, motion correction, segmentation, reconstruction, and contrast enhancement [8].A GAN is made up of two concurrently trained image-based networks: the discriminator and the generator. The discriminator network compares and evaluates the false pictures against the real ones after the generator creates copies of the original images [8]. The discriminator gains knowledge of the original input's characteristics by training on both kinds of inputs. Discriminator loss, or the difference between the two inputs, is sent to the generator network to change the settings for better performance [8].

GAN's work through an adversarial process between a generator and a discriminator. The generator's objective is to produce increasingly realistic images, while the discriminator's goal is to distinguish between the generated images and real images [12]. This ongoing competition between the two components enables GANs to create high-quality and lifelike images either from existing images or random noise [12].

## 3 Implementation Plan

1. Data Collection

   Collected dataset for Garbage Classification with images for 12 classes and storing them appropriately for processing the data.

2. Data Preprocessing

   Applied necessary preprocessing steps to the images to ensure they are in a consistent format and size. This includes resizing the images, normalizing pixel values, and converting them to a suitable format (eg. numpy array).

3. GAN Architecture Design

   The GAN consists of two main components: the generator and the discriminator. The generator creates synthetic images, while the discriminator distinguishes between real and synthetic images. The generator generates images of size 128,128 and discriminator distinguishes between the images of same size.

4. Training the GAN

   Trained the GAN by alternating between updating the discriminator and the generator. The generator constantly gets trained on the data that is fed to the model while the discriminator becomes more accurate in distinguishing between real and fake images.

5. Generating Synthetic Data

   Generating synthetic data based on the parameters defined such as Latent Dim and Input Shape.

6. Calculating Evaluation Metrics

   The Inception Score measures the quality and diversity of generated images. It uses the Inception-v3 neural network to evaluate the generated images based on their class probabilities and the entropy of the distribution. Higher Inception Score values indicate better image quality and diversity.

   The FID is another popular metric that uses the Inception-v3 network to measure the similarity between the real data distribution and the generated data distribution in the feature space. Lower FID values indicate better similarity and thus better quality of generated samples.

7. Statistical Similarity To evaluate the statistical similarity between the real data distribution and the generated data distribution in a GAN, one of the commonly used metrics is the Fréchet Inception Distance (FID). FID uses a pre-trained Inception-v3 model to extract feature representations from both real and generated samples. Then, it calculates the Fréchet distance between the two multivariate Gaussian distributions formed by the feature representations.

8. Iterative Improvement In order to improve results, it is necessary to regularly analyse the data collected, collect feedback, and modify the GAN implementation.

## 4 Project Design

Fig. 1 illustrates the GAN architecture. The generator is supplied with random noise as input, while the discriminator receives real images. The generator utilizes feedback from the discriminator to produce synthetic images [13]. Both components of the GAN work together in tandem to enhance overall performance.

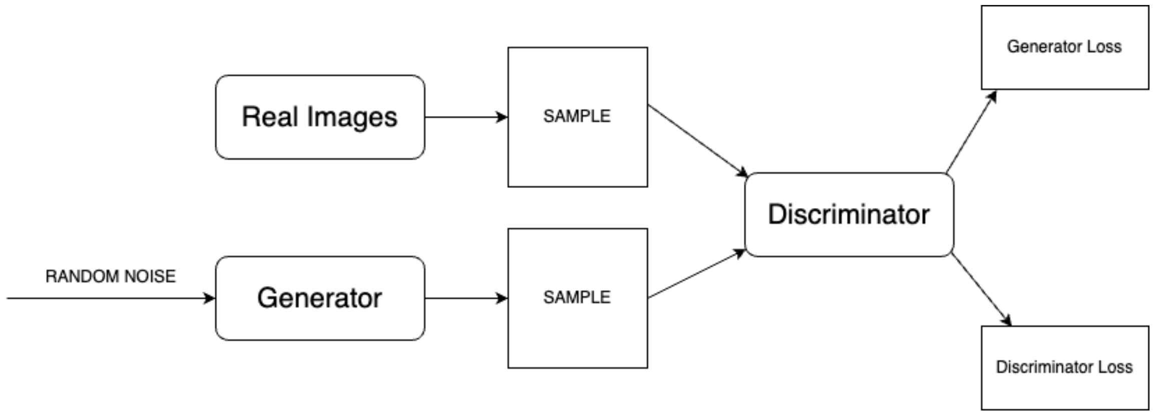

Figure 1: GAN Architecture [13]

The primary task of the discriminator is to accurately distinguish between real and fake images. If the discriminator fails to keep pace with the generator, the model will not effectively train, resulting in poorly generated images[13].

Ensuring accurate Generator and Discriminator losses is crucial for the GAN model's success [13]. Inaccurate losses can lead to false results, undermining the model's ability to produce high-quality and realistic synthetic images.

## 5 Generator Model

Fig. 2 and Fig. 3 shows the layout of the Generator model used for implementing the GAN architecture.

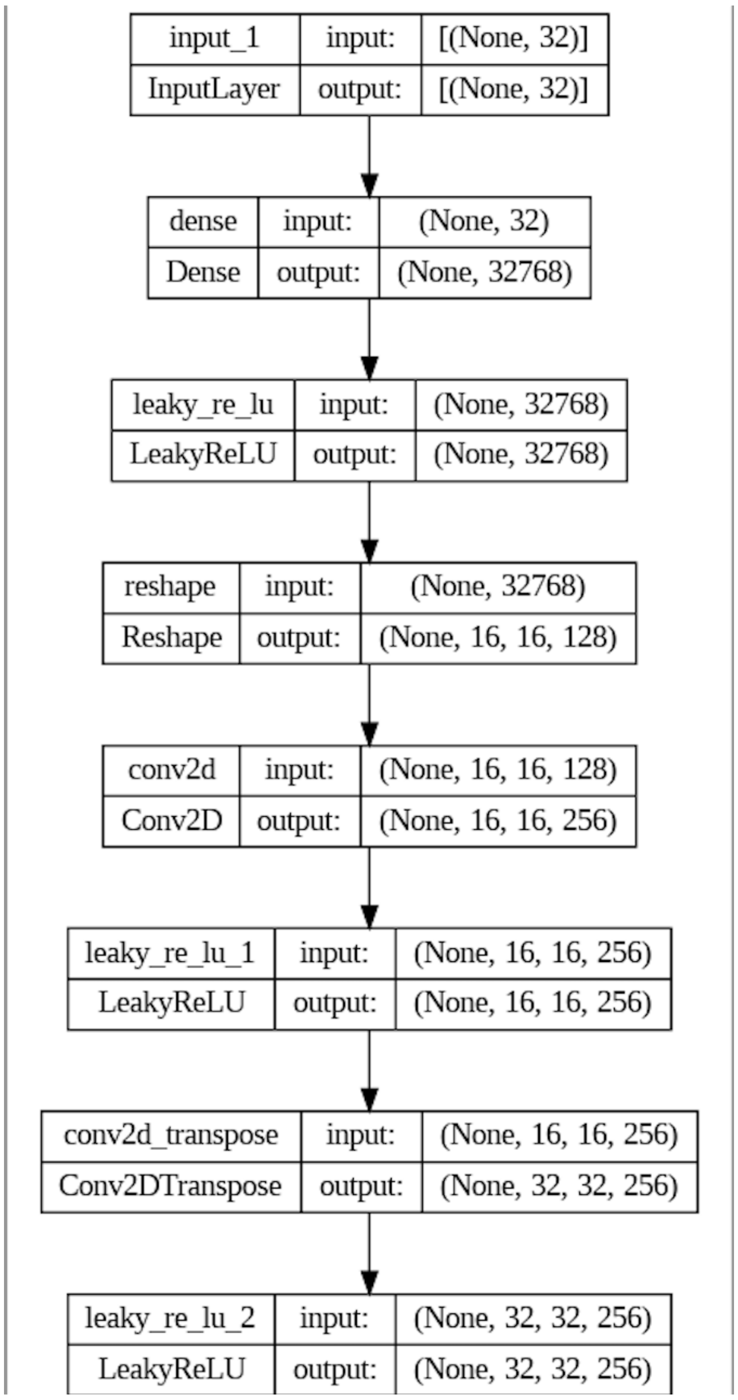
input_1 | input: | [(None, 32)]
InputLayer | output: | [(None, 32)]
dense | input: | (None, 32)
Dense | output: | (None, 32768)
leaky_re_lu | input: | (None, 32768)
LeakyReLU | output: | (None, 32768)
reshape | input: | (None, 32768)
Reshape | output: | (None, 16, 16, 128)
conv2d | input: | (None, 16, 16, 128)
Conv2D | output: | (None, 16, 16, 256)
leaky_re_lu_1 | input: | (None, 16, 16, 256)
LeakyReLU | output: | (None, 16, 16, 256)
conv2d_transpose | input: | (None, 16, 16, 256)
Conv2DTranspose | output: | (None, 32, 32, 256)
leaky_re_lu_2 | input: | (None, 32, 32, 256)
LeakyReLU | output: | (None, 32, 32, 256)

Figure 2: Generator Architecture

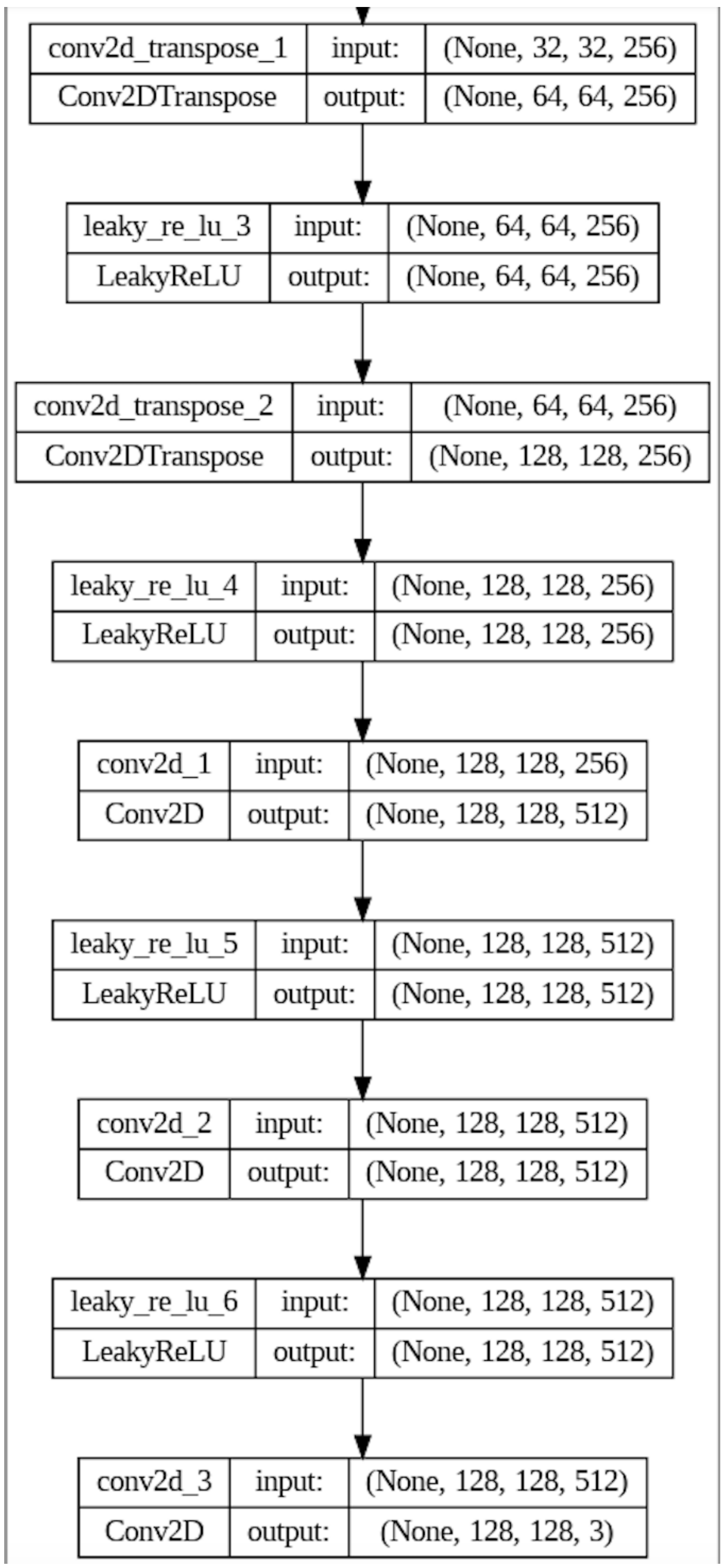
conv2d_transpose_1
Conv2DTranspose
input: (None, 32, 32, 256)
output: (None, 64, 64, 256)
leaky_re_lu_3
LeakyReLU
input: (None, 64, 64, 256)
output: (None, 64, 64, 256)
conv2d_transpose_2
Conv2DTranspose
input: (None, 64, 64, 256)
output: (None, 128, 128, 256)
leaky_re_lu_4
LeakyReLU
input: (None, 128, 128, 256)
output: (None, 128, 128, 256)
conv2d_1
Conv2D
input: (None, 128, 128, 256)
output: (None, 128, 128, 512)
leaky_re_lu_5
LeakyReLU
input: (None, 128, 128, 512)
output: (None, 128, 128, 512)
conv2d_2
Conv2D
input: (None, 128, 128, 512)
output: (None, 128, 128, 512)
leaky_re_lu_6
LeakyReLU
input: (None, 128, 128, 512)
output: (None, 128, 128, 512)
conv2d_3
Conv2D
input: (None, 128, 128, 512)
output: (None, 128, 128, 3)

Figure 3: Generator Architecture (Cont.)

- **Input(shape=(LATENT_DIM, )):** This defines the input layer for the generator. The generator takes a random noise vector as input, which is usually drawn from a normal distribution with a fixed dimension LATENT_DIM. The noise vector acts as a seed for generating the synthetic data.
- **Dense(128 * 16 * 16):** This dense layer takes the input noise vector and maps it to a higher-dimensional space. It transforms the 1D noise vector into a higher-dimensional representation (128 * 16 * 16), which will later be reshaped to a 3D tensor.
- **LeakyReLU:** This activation function is used throughout the model. Leaky ReLU allows small negative values, preventing the vanishing gradient problem during training and promoting better learning.
- **Reshape((16, 16, 128))** This reshapes the higher-dimensional tensor into a 3D tensor with dimensions (16, 16, 128). This 3D tensor acts as the initial feature map for the subsequent convolutional layers.
- **Conv2D:** The first set of convolutional layers aims to upscale the initial feature map gradually. This layer applies 256 filters of size 5x5 to the input, preserving the spatial dimensions while increasing the depth to 256. The 'same' padding ensures the output spatial dimensions remain unchanged.
- **Conv2DTranspose:** The transpose convolutional layers are used to upsample the feature map. This layer applies 256 filters of size 4x4, with a stride of 2, which effectively doubles the spatial dimensions while maintaining the depth. The generator repeats the Conv2DTranspose layer three times. With each repetition, the spatial dimensions are doubled while preserving the depth.
- **Conv2D(512, 5, padding='same'):** This layer further increases the depth of the feature map to 512 while preserving spatial dimensions.
- **Conv2D(CHANNELS, 7, activation='tanh', padding='same'):** The final convolutional layer reduces the depth to the desired number of channels (e.g., 3 for RGB images) and applies a hyperbolic tangent (tanh) activation function to scale the output values between -1 and 1. This ensures the generated images are within a valid pixel range.

The output of this generator is a synthetic image that is intended to resemble a real image. The generator's training involves minimizing the difference between the generated images and real images, while the discriminator (another component of the GAN) aims to distinguish between real and fake images. Through adversarial training, the generator learns to produce increasingly realistic synthetic images over time.

## 6 Discriminator Model

Fig. 4 and Fig. 5 shows the layout of the Discriminator model used for implementing the GAN architecture.

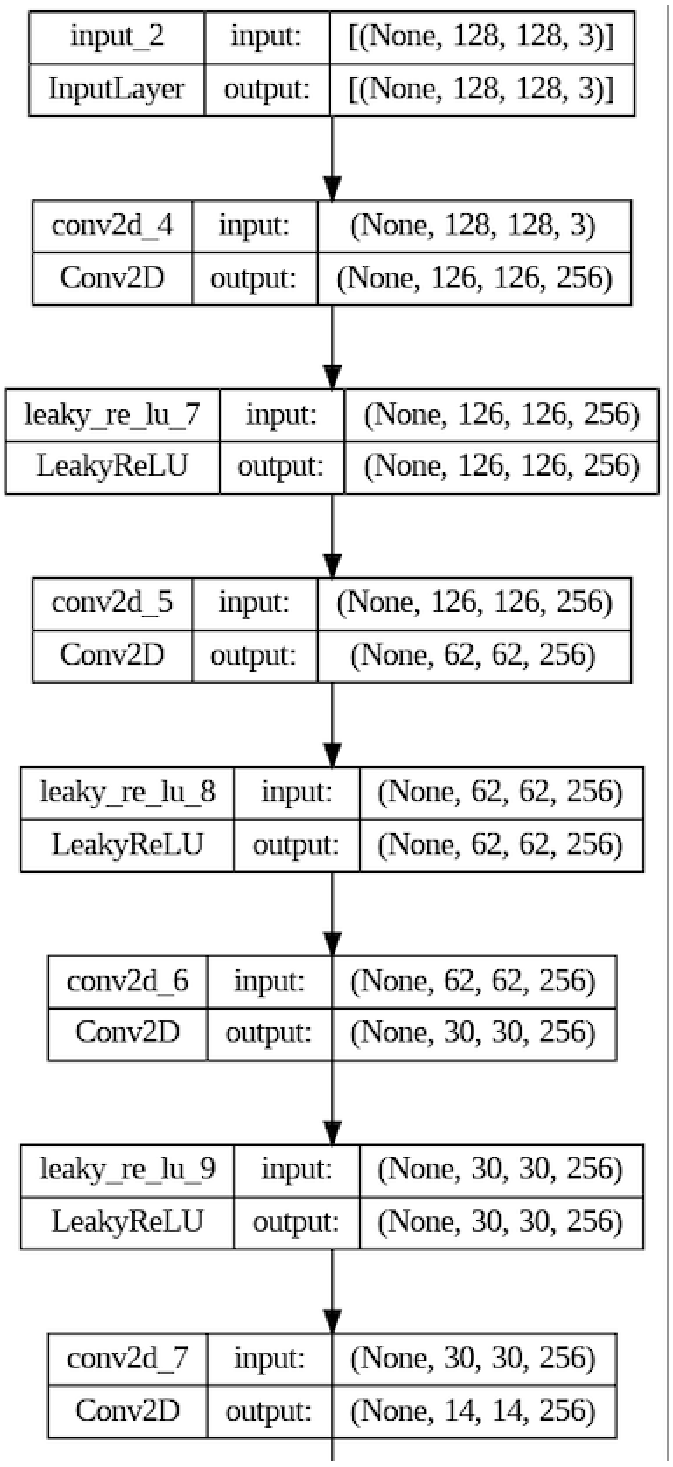
input_2 | input: | [(None, 128, 128, 3)]
InputLayer | output: | [(None, 128, 128, 3)]
conv2d_4 | input: | (None, 128, 128, 3)
Conv2D | output: | (None, 126, 126, 256)
leaky_re_lu_7 | input: | (None, 126, 126, 256)
LeakyReLU | output: | (None, 126, 126, 256)
conv2d_5 | input: | (None, 126, 126, 256)
Conv2D | output: | (None, 62, 62, 256)
leaky_re_lu_8 | input: | (None, 62, 62, 256)
LeakyReLU | output: | (None, 62, 62, 256)
conv2d_6 | input: | (None, 62, 62, 256)
Conv2D | output: | (None, 30, 30, 256)
leaky_re_lu_9 | input: | (None, 30, 30, 256)
LeakyReLU | output: | (None, 30, 30, 256)
conv2d_7 | input: | (None, 30, 30, 256)
Conv2D | output: | (None, 14, 14, 256)

Figure 4: Discriminator Architecture

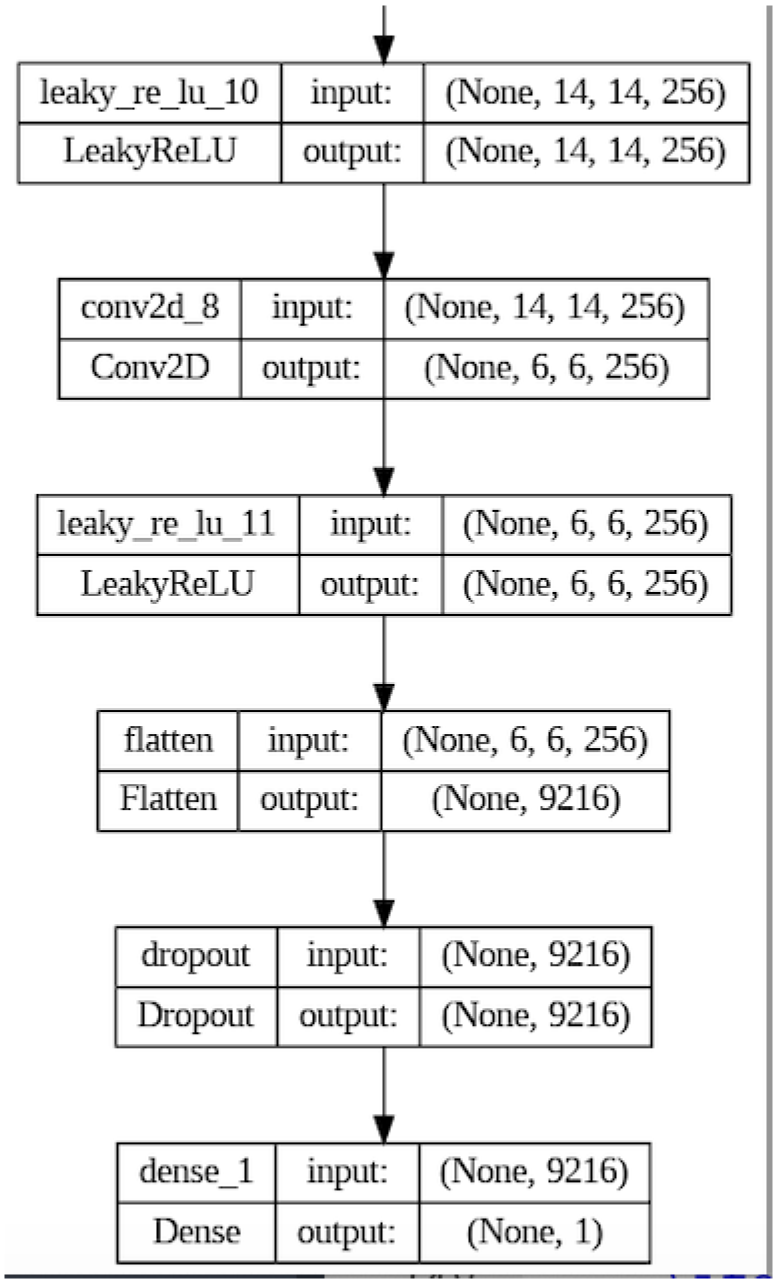

Figure 5: Discriminator Architecture (Cont.)

- **Input(shape=(HEIGHT, WIDTH, CHANNELS)):** This defines the input layer for the discriminator. The discriminator takes an image tensor as input, where HEIGHT, WIDTH, and CHANNELS represent the dimensions of the input image.
- **Conv2D:** The discriminator uses a series of convolutional layers to extract features from the input images. This layer applies 256 filters of size 3x3 to the input image, capturing different patterns and features in the data. The output from this layer is a feature map.

- **LeakyReLU:** This activation function is used throughout the model. Leaky ReLU allows small negative values, preventing the vanishing gradient problem during training and promoting better learning.
  The discriminator repeats the Conv2D layer four times, each time with a stride of 2. This progressively reduces the spatial dimensions of the feature map while increasing its depth, resulting in a compressed representation of the input image.
- **Flatten** The flatten layer converts the 2D feature map into a 1D vector, preparing it for input to the dense layers.
- **Dropout:** To avoid overfitting, a dropout layer is applied, randomly setting a fraction (40%) of the inputs to 0 during training.
- **Dense:** The dense layer is a fully connected layer with 1 neuron, responsible for the final classification. It performs a binary classification, with the activation function set to sigmoid. The output value is a probability score, indicating the likelihood that the input image is real (1) or fake (0).

  The discriminator is trained to minimize the binary cross-entropy loss, which measures the difference between its predictions and the ground truth labels (real or fake). The GAN's training involves a continuous interplay between the generator and the discriminator. The generator tries to produce more realistic images to deceive the discriminator, while the discriminator continuously improves its ability to distinguish between real and fake images. This adversarial process leads to the generation of increasingly realistic synthetic images by the GAN.

## 7 Evaluation Metrics

- **Inception Score:** The Inception Score measures the quality and diversity of generated images. It uses the Inception-v3 neural network to evaluate the generated images based on their class probabilities and the entropy of the distribution. Higher Inception Score values indicate better image quality and diversity [14]. The first step is to use the preprocess_input function for the generated images. Inception model then predicts the pixel values of the generated images and computes the score. The below mentioned Inception Score is calculated after executing the GAN model to predict synthetic images for Garbage Classification Dataset [14].

  ```
  Inception Distance: 41.576263427734375
  ```

- **Frechet Inception Distance:** The Fréchet Inception Distance (FID) is a commonly used metric for assessing the quality of generated images produced by a Generative Adversarial Network (GAN) or other generative models [14]. FID quantifies the similarity between the distribution of real images and the distribution of generated images in a high-dimensional feature space.

```
Fréchet Inception Distance (FID): 361.72460972948346
```

To evaluate the statistical similarity between the real data distribution and the generated data distribution in a GAN, one of the commonly used metrics is the Fréchet Inception Distance (FID). FID uses a pre-trained Inception-v3 model to extract feature representations from both real and generated samples. [14] Then, it calculates the Fréchet distance between the two multivariate Gaussian distributions formed by the feature representations [14].

## 8 Results

Fig. 6 indicates the plot for Generator and Discriminator loss.

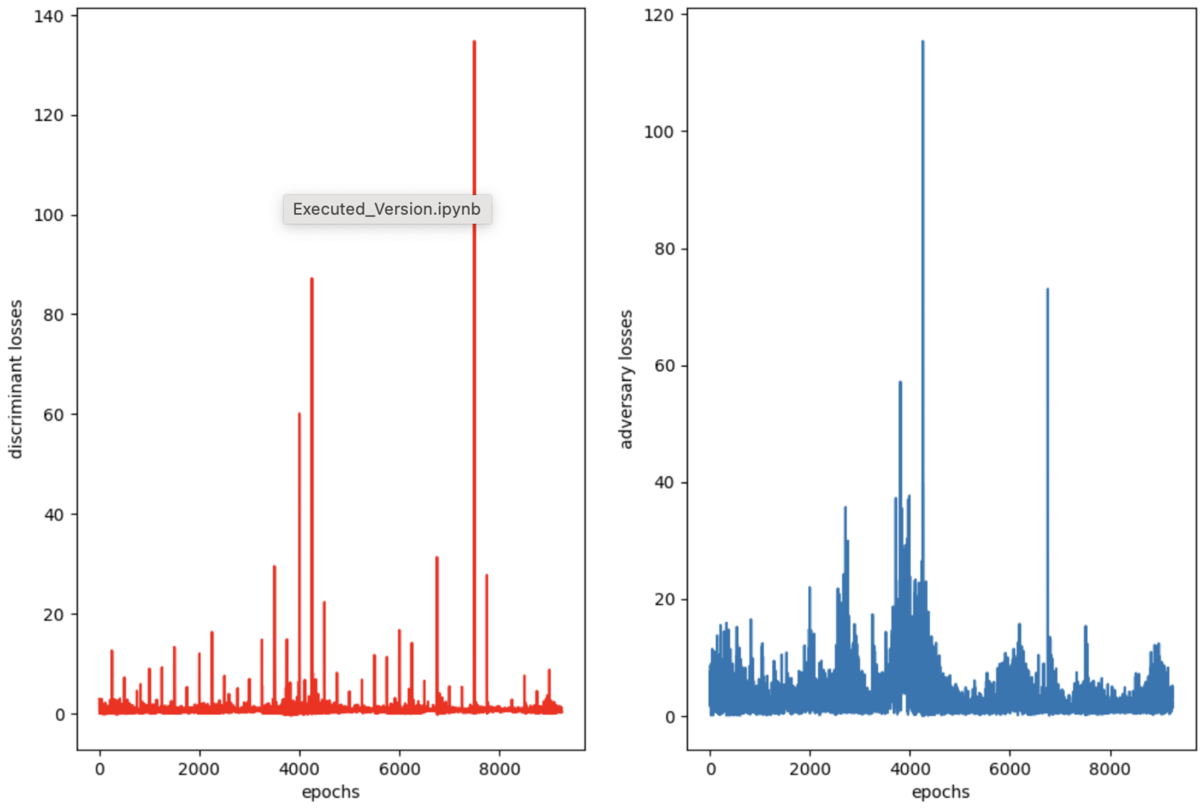

Figure 6: Generator and Discriminator Loss

The loss function for the Generator measures how well it performs in generating realistic data. he most common loss function for the Generator in GANs is the binary cross-entropy loss, which encourages the Generator to maximize the probability that the Discriminator classifies its generated samples as real (outputting a label close to 1).

The Discriminator's objective is to distinguish between real data and data generated by the Generator. Its loss function measures how well it performs in correctly classifying real and generated samples. Similar to the Generator, the Discriminator often uses binary cross-entropy loss to optimize its training. The loss function encourages the Discriminator to maximize the probability of correctly classifying real samples as real (outputting a label close to 1) and generated samples as fake (outputting a label close to 0).

Fig. 7 indicates the series of images from the initial stage to the final result.

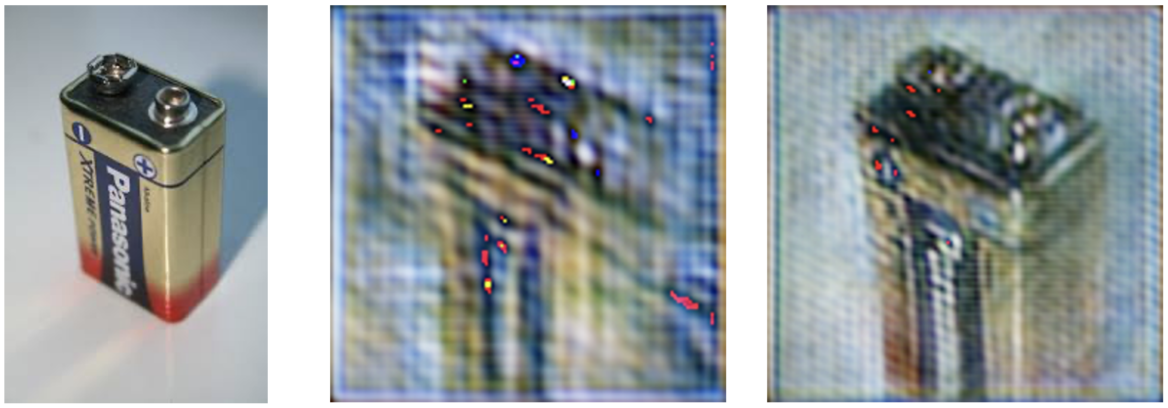

Figure 7: Real Image to Generated Image

The initial image displayed serves as the authentic input for training the Generative Adversarial Network (GAN) model.

Prior to training, this image is resized to a consistent dimension of 128 x 128 pixels and converted into a numpy array to facilitate further processing. Subsequently, the numpy array undergoes normalization, transforming the pixel values into a standardized range between 0 and 1, which ensures stability during training.

The initial image displayed serves as the authentic input for training the Generative Adversarial Network (GAN) model. Prior to training, this image is resized to a consistent dimension of 128 x 128 pixels and converted into a numpy array to facilitate further processing. Subsequently, the numpy array undergoes normalization, transforming the pixel values into a standardized range between 0 and 1, which ensures stability during training.

During compilation, the GAN model is configured with the Adam optimizer, utilizing a learning rate of 0.0001. This optimizer is a popular choice for GANs due to its adaptive nature, which enhances training efficiency and convergence. With the model prepared for training, the numpy array representing the real image is fed into the GAN, accompanied by pre-defined hyperparameters. These hyperparameters play a vital role in shaping the GAN's learning process and influence the generated results.

Throughout the training procedure, the GAN generates images at various stages of learning. After every 50 steps, an intermediate image is generated, demonstrating the model's progression in capturing features and patterns from the training data. At the 150th step, the second image reflects the model's improved ability to generate more realistic samples. Finally, after 350 steps, the third image showcases the GAN's proficiency in generating high-quality synthetic images.

## 8.1 Learning Rate: 0.0001

Fig. 8 illustrates the plot for Generator and Discriminator Loss with learning rate set to 0.0001.

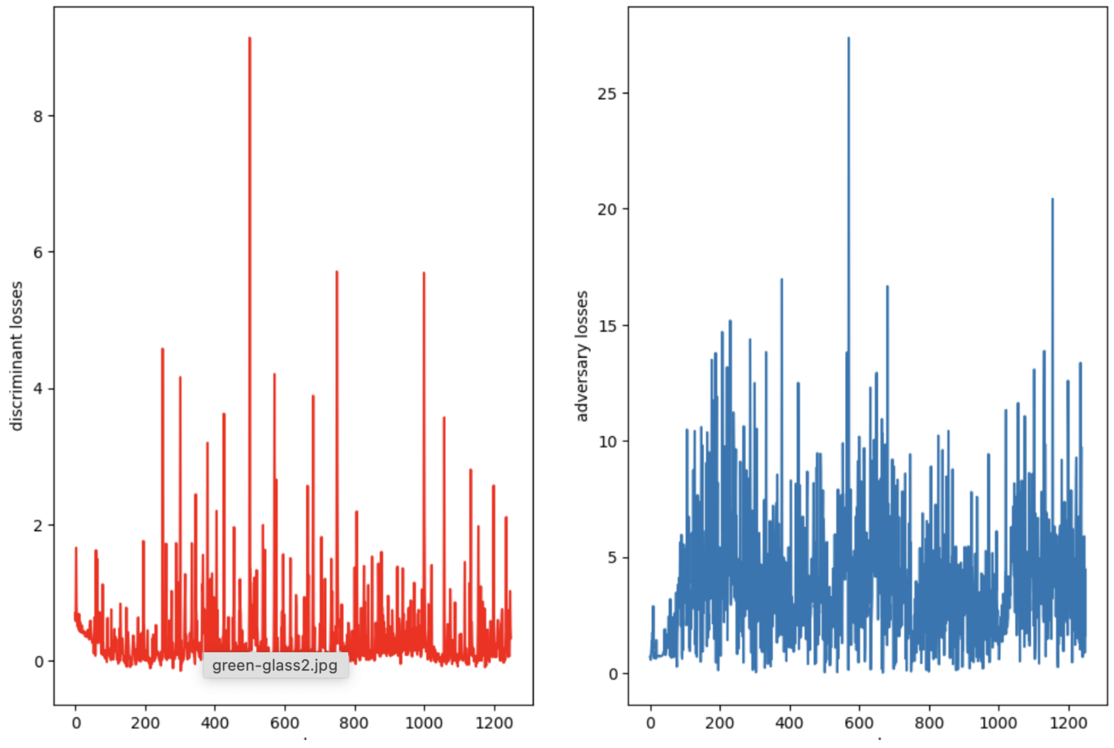

Figure 8: Generator and Discriminator Plot(LR:0.0001)

Setting an appropriate learning rate is essential to ensure the stability and convergence of the GAN during training. If the learning rate is too high, it may cause training instability, leading to oscillations or divergence in the loss functions. On the other hand, if the learning rate is too low, the training process may become slow, and the model may struggle to learn meaningful features. Therefore, as per the experiments and results, the generator and discriminator loss plot for learning rate 0.0001 is observed to be more efficient.

## 8.2 Learning Rate: 0.0002

Fig. 9 illustrates the plot for Generator and Discriminator Loss with learning rate set to 0.0002.

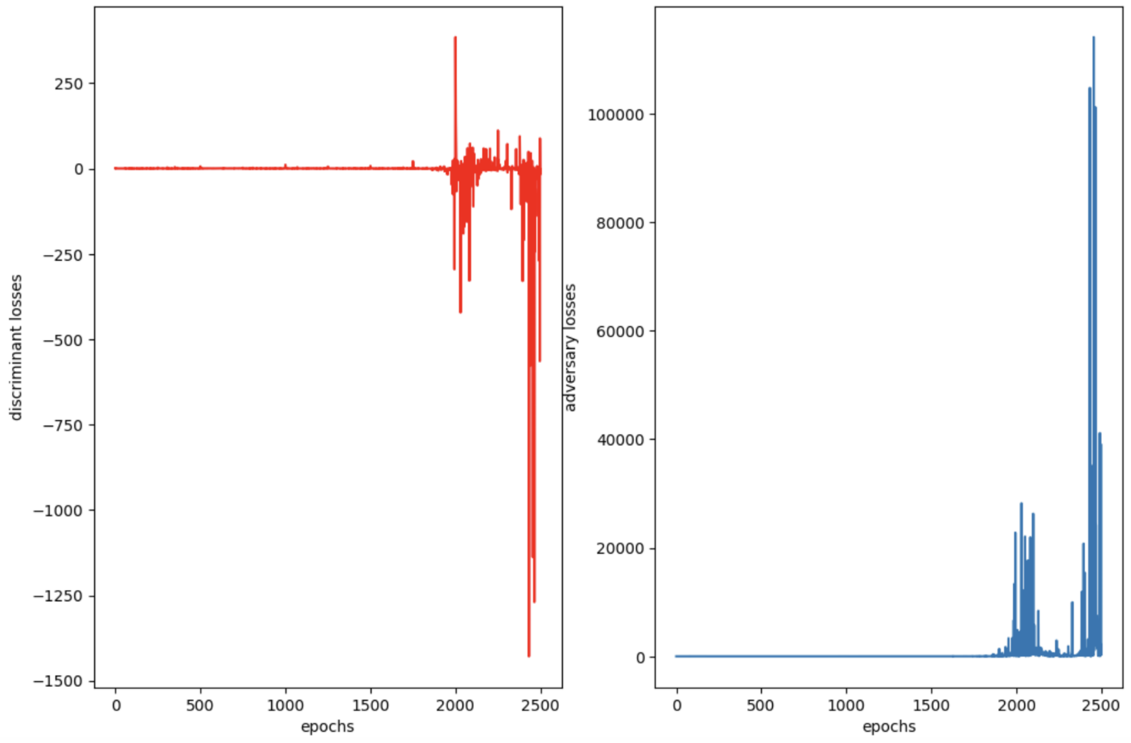

Figure 9: Generator and Discriminator Plot(LR:0.0002)

Upon careful observation, it becomes evident that using a learning rate of 0.0002 is causing significant oscillations in the loss plot, leading to unstable results during GAN training. Consequently, this instability can negatively impact the model's accuracy and overall performance on image generation tasks. To improve the stability and quality of the training process, it is highly recommended to revert to a learning rate of 0.0001.

By setting the learning rate to 0.0001, the GAN's training process is expected to be more controlled and smoother. A lower learning rate ensures more gradual updates to the discriminator and generator's parameters, which can lead to a more stable convergence of the GAN. This stability helps prevent oscillations and divergence in the loss functions, resulting in a more reliable and accurate GAN model for image synthesis.

In conclusion, reverting to a learning rate of 0.0001 is a prudent decision to enhance the stability, convergence, and accuracy of the GAN during training. This adjustment is expected to yield improved image generation results and contribute to the overall success of the GAN-based image synthesis model. As always, fine-tuning the learning rate and other hyperparameters may still be necessary to achieve the best possible results, but starting with a lower learning rate is a step in the right direction.

## 9 Future Scope

1. **Enhanced Realism:** Future research will focus on improving the realism of generated images. GANs will be equipped with more sophisticated architectures, attention mechanisms, and advanced loss functions to create images that are even more indistinguishable from real-world data.

2. **Conducting Qualitative Analysis:** One of the main future goals for qualitative analysis in GANs is to enhance the visual quality of generated images. Researchers can explore new architectural modifications, loss functions, or training techniques to produce more realistic and visually appealing images.

3. **Fusion of Generated Image and Noise:** Synthetic Images can be further generated using a Fusion technique of Noise and Image instead of random vector noise. To achieve synthesis realism in both geometry and appearance spaces, Spatial Fusion GAN combines a geometry synthesiser with an appearance synthesiser. The appearance synthesiser harmoniously integrates foreground items into background pictures by adjusting their colour, brightness, and styles [12].

4. **GAN Generated Image Detection:** Generative Adversarial Networks (GAN), which have been used in phoney social media profiles and other forms of deception that have the potential to have substantial effects, have produced very realistic face photos [9]. As a result, techniques for GAN-face identification that can look at and spot these fake faces are now being developed [9].

## 10 Advantages of Synthetic Image Data Generation

Fig. 10 indicates a diagrammatic representation of Advantages of Synthetic Image Data generator.

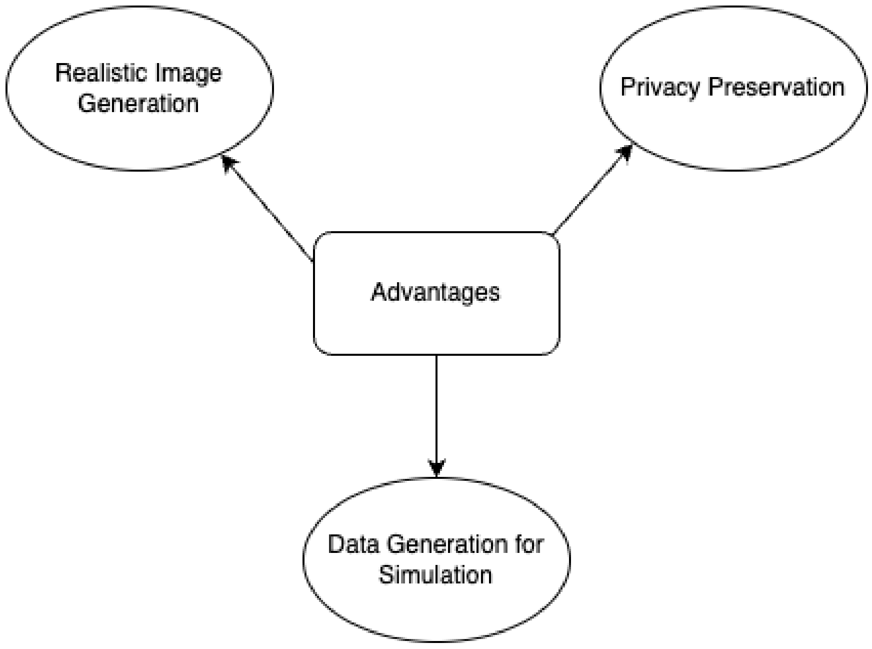

Figure 10: Advantages of GAN

1. **Realistic Image Generation**: GANs are capable of producing high-quality, realistic images that closely resemble the distribution of real-world data. The generator and discriminator adversarial setup forces the generator to continually improve its image generation capabilities, leading to more visually convincing and accurate results.
2. **Privacy Preservation**: GANs allow data generation without direct access to the original data, making them valuable for applications in privacy-sensitive domains. Synthetic data can be used for testing and development without exposing sensitive information.
3. **Data Generation for Simulation**: GANs can produce synthetic data for training models in simulation environments, making it easier to test and refine algorithms in various scenarios without relying solely on real-world data.

## 11 Conclusions

In conclusion, synthetic image data generation using Generative Adversarial Networks (GANs) for garbage classification offers a promising solution to address the challenges of limited and imbalanced real-world datasets in this domain. GANs have proven to be effective in generating realistic and diverse images that can supplement the existing data and enhance the performance of garbage classification models.

By leveraging GANs, we can create synthetic images that closely resemble real-world garbage items across various classes. These synthetic images can effectively increase the size of the dataset, enabling the garbage classification model to generalize better and improve its accuracy and robustness.

However, it is essential to acknowledge some challenges and considerations when using synthetic data. The quality and diversity of the generated images heavily depend on the GAN's architecture, training parameters, and the size of the original dataset. Ensuring the GAN produces high-quality and diverse images requires careful tuning and validation.

In summary, synthetic image data generation using GANs presents a valuable tool for augmenting garbage classification datasets, mitigating class imbalance issues, and improving the performance of garbage classification models. By combining real and synthetic data judiciously, we can enhance the accuracy and robustness of garbage classification systems, contributing to more effective waste management and environmental conservation efforts.

## 12 Appendix

### 12.1 Code

```
from tqdm import tqdm
from keras.optimizers import RMSprop
from matplotlib import pyplot as plt

from tqdm import tqdm
from PIL import Image
from keras import Input
from keras.layers import Dense, Reshape, LeakyReLU, Conv2D, Conv2DTranspose, Flatten,
from keras.models import Model
from keras.optimizers import RMSprop
import os
import numpy as np
from IPython.display import Image
from keras.utils.vis_utils import model_to_dot

PIC_DIRS = [
    '/content/drive/MyDrive/Capstone/Test Data/Battery',
    '/content/drive/MyDrive/Capstone/Test Data/Biological',
    '/content/drive/MyDrive/Capstone/Test Data/Brown-Glass',
    '/content/drive/MyDrive/Capstone/Test Data/Cardboard',
    '/content/drive/MyDrive/Capstone/Test Data/Clothes',
    '/content/drive/MyDrive/Capstone/Test Data/Green-glass',
    '/content/drive/MyDrive/Capstone/Test Data/Metal',
    '/content/drive/MyDrive/Capstone/Test Data/Paper',
    '/content/drive/MyDrive/Capstone/Test Data/Plastic',
    '/content/drive/MyDrive/Capstone/Test Data/Shoes',
    '/content/drive/MyDrive/Capstone/Test Data/Trash',
    '/content/drive/MyDrive/Capstone/Test Data/White-Glass'

    # Add paths for the remaining classes
]

IMAGES_COUNT = 10000

ORIG_WIDTH = 178
ORIG_HEIGHT = 208
diff = (ORIG_HEIGHT - ORIG_WIDTH) // 2

WIDTH = 128
HEIGHT = 128

crop_rect = (0, diff, ORIG_WIDTH, ORIG_HEIGHT - diff)

images = []
class_labels = []

for class_dir in PIC_DIRS:
    class_name = os.path.basename(class_dir)  # Get the class name from the directory
    for pic_file in tqdm(os.listdir(class_dir)[:IMAGES_COUNT]):
        pic_path = os.path.join(class_dir, pic_file)
        pic = img.open(pic_path).crop(crop_rect)
        pic.thumbnail((WIDTH, HEIGHT), img.ANTIALIAS)
        images.append(np.uint8(pic))
        class_labels.append(class_name)

images = np.array(images)
class_labels = np.array(class_labels)

print("Images shape:", images.shape)
print("Labels shape:", class_labels.shape)

# Now 'class_labels' contains class names instead of class indices
# Use 'class_labels' as required for your further processing.
```

```
#Code That Runs
#Display Some Images
plt.figure(1, figsize=(10, 10))
for i in range(len(images)):
    plt.subplot(10, 10, i+1)
    plt.imshow(images[i])
    plt.axis('off')
plt.show()

#Image shape
images = np.array(images) / 255
print(images.shape)

#Generator Function
LATENT_DIM = 32
CHANNELS = 3

def create_generator():
    gen_input = Input(shape=(LATENT_DIM, ))

    x = Dense(128 * 16 * 16)(gen_input)
    x = LeakyReLU()(x)
    x = Reshape((16, 16, 128))(x)

    x = Conv2D(256, 5, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)
    x = LeakyReLU()(x)

    x = Conv2D(512, 5, padding='same')(x)
    x = LeakyReLU()(x)
    x = Conv2D(512, 5, padding='same')(x)
    x = LeakyReLU()(x)
    x = Conv2D(CHANNELS, 7, activation='tanh', padding='same')(x)

    generator = Model(gen_input, x)
    return generator

#Discriminator Function

def create_discriminator():
    disc_input = Input(shape=(HEIGHT, WIDTH, CHANNELS))

    x = Conv2D(256, 3)(disc_input)
    x = LeakyReLU()(x)

    x = Conv2D(256, 4, strides=2)(x)
    x = LeakyReLU()(x)

    x = Conv2D(256, 4, strides=2)(x)
    x = LeakyReLU()(x)

    x = Conv2D(256, 4, strides=2)(x)
    x = LeakyReLU()(x)

    x = Conv2D(256, 4, strides=2)(x)
    x = LeakyReLU()(x)

    x = Flatten()(x)
    x = Dropout(0.4)(x)

    x = Dense(1, activation='sigmoid')(x)
    discriminator = Model(disc_input, x)

    optimizer = RMSprop(
        lr=.0001,
        clipvalue=1.0,
        decay=1e-8
    )

    discriminator.compile(
        optimizer=optimizer,
        loss='binary_crossentropy'
    )

    return discriminator

#Declare Generator
generator = create_generator()
generator.summary()
```

```
#Display Generator Model as Image
Image(model_to_dot(generator, show_shapes=True).create_png())

#Declare Discriminator
discriminator = create_discriminator()
discriminator.trainable = False
discriminator.summary()

#Define Gan Model
gan_input = Input(shape=(LATENT_DIM, ))
gan_output = discriminator(generator(gan_input))
gan = Model(gan_input, gan_output)

#Optimizer Function
optimizer = RMSprop(lr=.0001, clipvalue=1.0, decay=1e-8)
gan.compile(optimizer=optimizer, loss='binary_crossentropy')

#Print GAN Summary
gan.summary()

#Training Function
import time
from PIL import Image as Img
iters = 250
batch_size = 1

RES_DIR = '/content/drive/MyDrive/CAPSTONE/Data/Test Result Data/Synthetic_Data_v5/Sy
FILE_PATH = '%s/generated_%d.png'
if not os.path.isdir(RES_DIR):
    os.mkdir(RES_DIR)

CONTROL_SIZE_SQRT = 1
control_vectors = np.random.normal(size=(CONTROL_SIZE_SQRT**2, LATENT_DIM)) / 2

start = 0
d_losses = []
a_losses = []
images_saved = 0
for i in range(len(images)):
  plt.imshow(images[i])
  plt.axis('off')
  for step in range(iters):
      print(step)
```

```
        start_time = time.time()
        latent_vectors = np.random.normal(size=(batch_size, LATENT_DIM))
        generated = generator.predict(latent_vectors)

        # real = images[start:start + batch_size]
        real = np.expand_dims(images[i], axis=0)
        combined_images = np.concatenate([generated, real])

        labels = np.concatenate([np.ones((batch_size, 1)), np.zeros((batch_size, 1))])
        labels += .05 * np.random.random(labels.shape)

        d_loss = discriminator.train_on_batch(combined_images, labels)
        d_losses.append(d_loss)

        latent_vectors = np.random.normal(size=(batch_size, LATENT_DIM))
        misleading_targets = np.zeros((batch_size, 1))

        a_loss = gan.train_on_batch(latent_vectors, misleading_targets)
        a_losses.append(a_loss)

        start += batch_size
        if start > images.shape[0] - batch_size:
            start = 0

        if step % 50 == 49:
            gan.save_weights('/gan.h5')

            #print('%d/%d: d_loss: %.4f,  a_loss: %.4f.  (%.1f sec)' % (step + 1, iters
            print('%d/%d:   a_loss: %.4f.  (%.1f sec)' % (step + 1, iters, a_loss, time

            control_image = np.zeros((WIDTH * CONTROL_SIZE_SQRT, HEIGHT * CONTROL_SIZE_
            control_generated = generator.predict(control_vectors)

            for j in range(CONTROL_SIZE_SQRT ** 2):
                x_off = j % CONTROL_SIZE_SQRT
                y_off = j // CONTROL_SIZE_SQRT
                control_image[x_off * WIDTH:(x_off + 1) * WIDTH, y_off * HEIGHT:(y_off
            im = Img.fromarray(np.uint8(control_image * 255))#.save(StringIO(), 'jpeg')
            im.save(FILE_PATH % (RES_DIR, images_saved))
            images_saved += 1

#Plotting the Generator and Discriminator Loss
plt.figure(1, figsize=(12, 8))
plt.subplot(121)
plt.plot(d_losses, color='red')
plt.xlabel('epochs')
plt.ylabel('discriminant losses')
plt.subplot(122)
plt.plot(a_losses)
plt.xlabel('epochs')
plt.ylabel('adversary losses')
plt.show()
```

## References

[1] Zhang, C., Kuppannagari, S. R., Kannan, R., & Prasanna, V. K. (2018, October). Generative adversarial network for synthetic time series data generation in smart grids. In 2018 IEEE international conference on communications, control, and computing technologies for smart grids (SmartGridComm) (pp. 1-6). IEEE.

[2] M. Frid-Adar, E. Klang, M. Amitai, J. Goldberger and H. Greenspan, "Synthetic data augmentation using GAN for improved liver lesion classification," 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018), Washington, DC, USA, 2018, pp. 289-293, doi: 10.1109/ISBI.2018.8363576.

[3] Dahmen J, Cook D. SynSys: A Synthetic Data Generation System for Healthcare Applications. Sensors. 2019; 19(5):1181. https://doi.org/10.3390/s19051181

[4] Dewi, C., Chen, RC., Liu, YT. et al. Synthetic Data generation using DCGAN for improved traffic sign recognition. Neural Comput & Applic 34, 21465–21480 (2022). https://doi.org/10.1007/s00521-021-05982-z

[5] Meister, S., Möller, N., Stüve, J. et al. Synthetic image data augmentation for fibre layup inspection processes: Techniques to enhance the data set. J Intell Manuf 32, 1767–1789 (2021). https://doi.org/10.1007/s10845-021-01738-7

[6] S. K. Zhou et al., "A Review of Deep Learning in Medical Imaging: Imaging Traits, Technology Trends, Case Studies With Progress Highlights, and Future Promises," in Proceedings of the IEEE, vol. 109, no. 5, pp. 820-838, May 2021, doi: 10.1109/JPROC.2021.3054390.

[7] C. Han et al., "GAN-based synthetic brain MR image generation," 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018), Washington, DC, USA, 2018, pp. 734-738, doi: 10.1109/ISBI.2018.8363678.

[8] Tavse S, Varadarajan V, Bachute M, Gite S, Kotecha K. A Systematic Literature Review on Applications of GAN-Synthesized Images for Brain MRI. Future Internet. 2022; 14(12):351. https://doi.org/10.3390/fi14120351

[9] Wang, X., Guo, H., Hu, S., Chang, M. C., & Lyu, S. (2022). Gan-generated faces detection: A survey and new perspectives. arXiv preprint arXiv:2202.07145.

[10] Bhattarai, B., Baek, S., Bodur, R., & Kim, T. K. (2020, May). Sampling strategies for gan synthetic data. In ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 2303-2307). IEEE.

[11] Mendes, J., Pereira, T., Silva, F., Frade, J., Morgado, J., Freitas, C., Negrão, E., Flor de Lima, B., Correia da Silva, M., Madureira, A. J., Ramos, I., Costa, J. L., Hespanhol, V., Cunha, A., & Oliveira, H. P. (2023). Lung CT image synthesis using GANs. Expert Systems with Applications, 215, 119350. https://doi.org/10.1016/j.eswa.2022.119350

[12] Zhan, F., Zhu, H., & Lu, S. (2019). Spatial Fusion GAN for Image Synthesis. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June.

[13] Google Developers. (n.d.). Generator - Generative Adversarial Networks. Retrieved from https://developers.google.com/machine-learning/gan/generator

[14] Xu, Q., Huang, G., Yuan, Y., Guo, C., Sun, Y., Wu, F., & Weinberger, K. (2018). An empirical study on evaluation metrics of generative adversarial networks. arXiv preprint arXiv:1806.07755.