**Co-op Kinectrics**

# Nuclear Fuel Bundle Synthetic Image Generation using DCGAN

**BY**:
Bhumika P. Shukla (101389078)

**DATE**: 12th August 2024

**Under the guidance of : -**
**Prof. Dr. Mahdieh Khalilinezhad**

## Table of Contents

1.1 Introduction ........ 3
1.2 PROBLEM STATEMENT ........ 3
1.3 SOLUTION ........ 3
1.4 DATA ACQUISITION & FEATURE EXPLANATION ........ 4
1.5 DATA CLEANING AND DATA PREPROCESSING ........ 4
1.6 DATA DISTRIBUTION ........ 5
1.7 EXPLORE DATA ANALYSIS (EDA) ........ 5
1.8 MODEL ARCHITECTURE – DCGAN ........ 6
1.9 MODEL COMPILATION ........ 9
1.10 MODEL TRAINING ........ 10
1.11 MODEL EVALUATION ........ 11
1.12 MODEL PREDICTION ........ 12
1.13 CONCLUSION ........ 12

## 1.1 Introduction

The goal of this research is to create an analysis method for nuclear fuel bundle photos based on deep learning. A Generative Adversarial Network (GAN) is used in the system to produce artificial images that are intended to closely mimic real fuel bundle images. In addition to helping with data augmentation for training, this method improves the model's ability to identify minute anomalies that would not have been visible in the original dataset. Incorporating image augmentation with a strong GAN architecture guarantees that the model is capable of managing the intricacies involved in fuel bundle inspection, hence augmenting the safety and efficacy of nuclear energy generation.

## 1.2 PROBLEM STATEMENT

The goal of the project is to build a Generative Adversarial Network (GAN) that can produce synthetic images of nuclear fuel bundles that are realistic. By adding these artificial images to the dataset, the model becomes more adept at identifying fuel bundle flaws, which are crucial for guaranteeing nuclear power generating safety.

The problem is that there isn't much real-world data available to train deep learning models in this field. The research seeks to get around this restriction by using GANs to increase defect detection accuracy and dependability, which would ultimately lead to safer and more effective nuclear power operations.

## 1.3 SOLUTION

This study uses a Generative Adversarial Network (GAN) to produce realistic synthetic images in order to overcome the problem of limited real-world data for training deep learning models in nuclear fuel bundle defect detection. A discriminator model that discerns between real and artificial images and a generator model that produces synthetic images make up the GAN. The generator gradually becomes better at producing realistic, high-quality images through adversarial training.

Subsequently, the artificial images produced by the GAN are employed to enhance the pre-existing dataset, hence augmenting the variety and quantity of training data. With this addition, the model is better able to identify fuel bundle faults even in the presence of the natural fluctuations in real-world data. In the end, data augmentation contributes to safer nuclear power generation by enhancing model performance and reducing the hazards related to training on small, constrained datasets.

## 1.4 DATA ACQUISITION & FEATURE EXPLANATION

The data for our project was acquired through an

Combined 2 Dataset (Collected images through google)

**Total folders**: - 2

**Total Valid Images**: - 315

**Total Invalid Images**: - 131

The dataset comprises two image folders: "Valid" and "Invalid". The "Valid" folders contains images of Nuclear Fuel Bundle that were searched and downloaded from, while the "Invalid" column consists of images that looks like Nuclear Fuel Bundle images but actually are something else.

## 1.5 DATA CLEANING AND DATA PREPROCESSING

The first step of the project is to extract the images from a zip file that has been compressed and contains pictures of nuclear fuel bundles. These images are kept for later processing in a designated directory. There were not many explicit data cleansing methods necessary because the dataset was structured. Making sure all images were imported correctly and that the dataset contained no faulty or unreadable files was the main responsibility.

### DATA PREPROCESSING

To prepare the data for training the Generative Adversarial Network (GAN), several crucial processes were engaged in the preprocessing phase:

1. **Image Resizing**: To maintain consistency throughout the dataset—a prerequisite for deep learning model training—all images were scaled to a consistent goal size of 64x64 pixels.

2. **Normalisation**: By deducting 127.5 from each pixel value and dividing by 127.5, the image pixel values were adjusted to fall within the range of [-1, 1]. The stabilisation and acceleration of the GAN's training process depend on this normalisation step.

3. **Data Augmentation:** Data augmentation techniques were used to improve the robustness of the model and diversify the training set. Random rotations, changes in width and height, and horizontal flips were all part of the augmentation. These additions enhance the model's capacity for generalisation by simulating a range of real-world scenarios**.**

4. **Batch Generation**: To produce batches of augmented images for training, a bespoke data generator was put into place. This generator ensures that the model trains on a diverse and dynamic dataset by continuously providing enhanced data.

## 1.6 DATA DISTRIBUTION

The dataset utilized for this project is organized into two primary folders: "Valid" and "Invalid."

**Valid Images**: This folder contains a total of 315 images, all depicting Nuclear Fuel Bundles. These images were carefully selected to ensure their relevance to the subject matter.

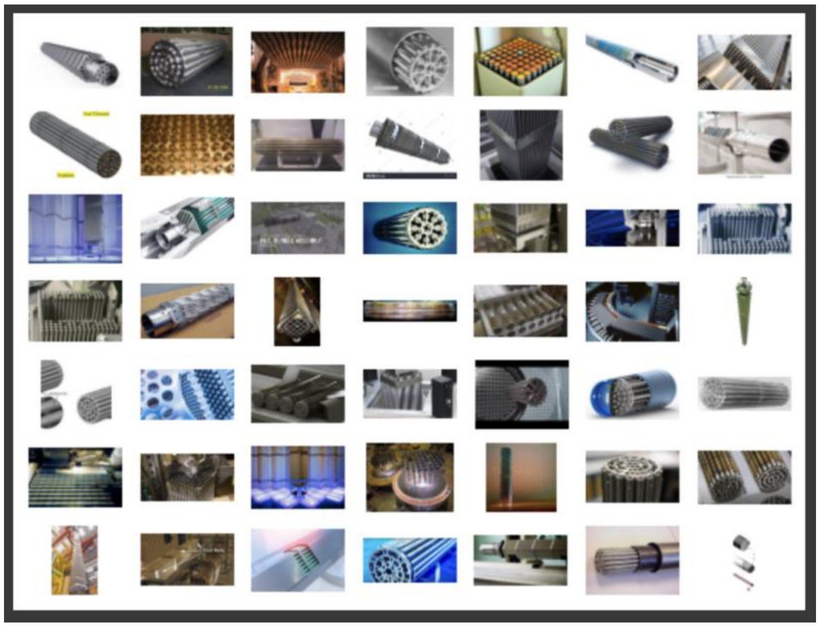
**Invalid Images**: Comprising 131 images, this folder includes images that may visually resemble Nuclear Fuel Bundles but do not accurately represent them.

This distribution highlights the importance of distinguishing between accurate representations and similar-looking yet irrelevant images, which is crucial for the model's ability to perform precise classifications.

## 1.7 EXPLORE DATA ANALYSIS (EDA)

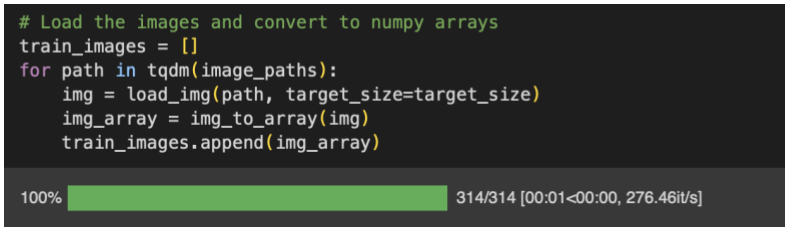
- **Data Visualization**

To understand the dataset, a subset of images (49 images) was visualized. This helps in verifying the dataset's quality and diversity.



- **Image Pre-processing**

Images were resized to 64x64 pixels to standardize their dimensions. They were also normalized to the range [-1, 1]



```
# Load the images and convert to numpy arrays
train_images = []
for path in tqdm(image_paths):
    img = load_img(path, target_size=target_size)
    img_array = img_to_array(img)
    train_images.append(img_array)

100%  314/314 [00:01<00:00, 276.46it/s]
```

- **Data Augmentation**

Data augmentation was applied to increase the diversity of the training images and improve model generalization. Augmentation techniques included rotation, width and height shifts, and horizontal flips

## 1.8 MODEL ARCHITECTURE – DCGAN

**Overview:**

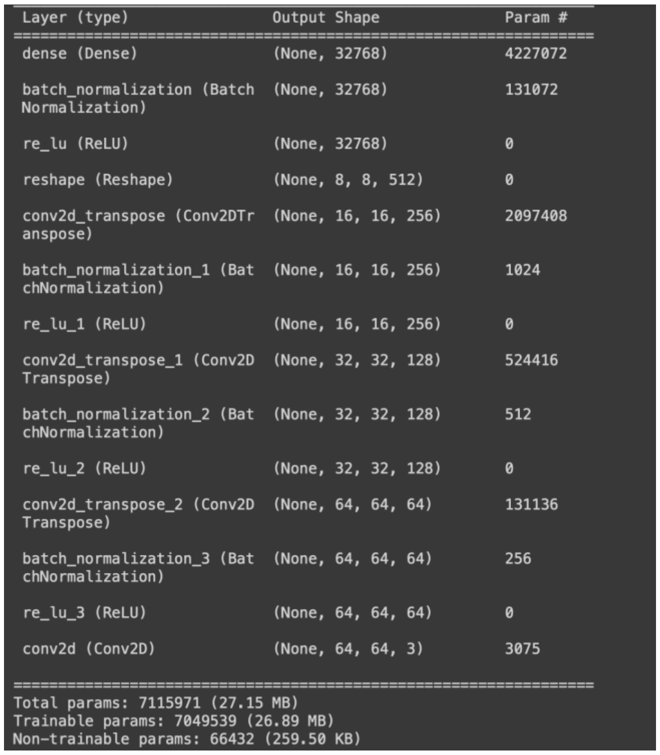
For this research, a Deep Convolutional Generative Adversarial Network (DCGAN) serves as the foundation for the model architecture. There are two primary parts to it: the Discriminator and the Generator. The Generator and Discriminator are two networks that are meant to operate in tandem, with the Generator producing artificial images and the Discriminator trying to discern between actual and artificial images.

**Generator:**

A random noise vector is fed into The Generator, a deep neural network, which uses it to create a synthetic image. The architecture is made up of multiple layers that work together to gradually upscale and smooth this noise and create a realistic image.

**Layer of Input**: The generator receives as input a 128-bit noise vector.
**Dense Layer**: The input is reshaped into a small spatial size with numerous feature maps by the fully connected first layer.
**Reshape Layer**: To create the first tiny feature map, the dense output is reshaped into a 3D tensor.
**Transposed Convolutional Layers**: The feature map is upsampled to the required output size of 64x64 pixels using a sequence of Conv2DTranspose (or deconvolution) layer.

Following each transposed convolution is:
**Batch Normalisation**: By normalising the convolutional layer outputs, this technique helps stabilise and accelerate training.
**ReLU Activation**: To encourage non-linearity, the Rectified Linear Unit (ReLU) is employed as the activation function.
**Output Layer**: The last layer generates an output image with pixel values between [-1, 1] by using a tanh activation function.

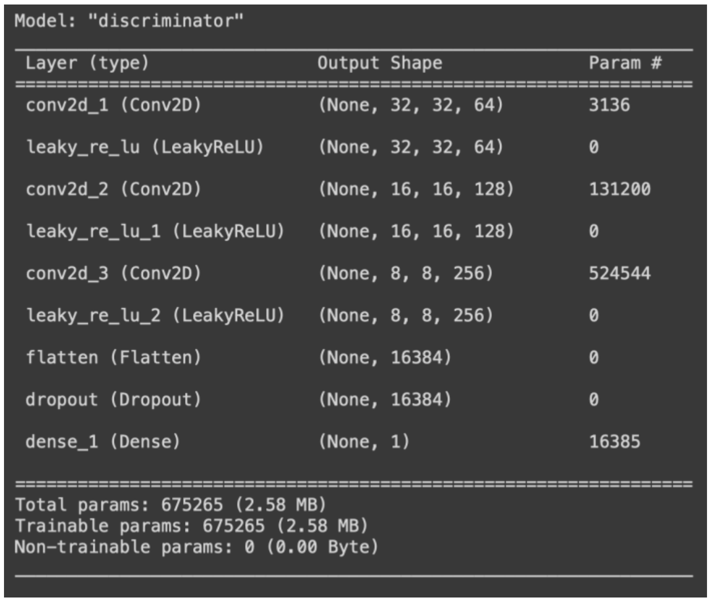


```
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32768)             4227072

 batch_normalization (Batch  (None, 32768)             131072
 Normalization)

 re_lu (ReLU)                (None, 32768)             0

 reshape (Reshape)           (None, 8, 8, 512)         0

 conv2d_transpose (Conv2DTr  (None, 16, 16, 256)       2097408
 anspose)

 batch_normalization_1 (Bat  (None, 16, 16, 256)       1024
 chNormalization)

 re_lu_1 (ReLU)              (None, 16, 16, 256)       0

 conv2d_transpose_1 (Conv2D  (None, 32, 32, 128)       524416
 Transpose)

 batch_normalization_2 (Bat  (None, 32, 32, 128)       512
 chNormalization)

 re_lu_2 (ReLU)              (None, 32, 32, 128)       0

 conv2d_transpose_2 (Conv2D  (None, 64, 64, 64)        131136
 Transpose)

 batch_normalization_3 (Bat  (None, 64, 64, 64)        256
 chNormalization)

 re_lu_3 (ReLU)              (None, 64, 64, 64)        0

 conv2d (Conv2D)             (None, 64, 64, 3)         3075

=================================================================
Total params: 7115971 (27.15 MB)
Trainable params: 7049539 (26.89 MB)
Non-trainable params: 66432 (259.50 KB)
```

**Descriminator:**

A deep convolutional neural network called the Discriminator divides input images into two categories: synthetic and real. In essence, it is a binary classifier that has been trained to maximise the precision of this differentiation.
Input Layer: A 64x64x3 image is accepted as input by the discriminator.
Convolutional Layers: There are multiple Conv2D layers in the architecture, and each one is followed by:
**Leaky ReLU Activation**: To avoid the dying ReLU issue, a tiny gradient is permitted when the unit is not active by using Leaky ReLU in place of conventional ReLU.
**Layer Flattening**: A 1D vector is created by flattening the 3D feature map.
**Dropout Layer**: During training, dropout is used to minimise overfitting by arbitrarily changing a portion of input units to 0 at each update.

**Output Layer:** To output the probability that the input image is real, a dense layer with a sigmoid activation function is employed.



```
Model: "discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 32, 32, 64)        3136

 leaky_re_lu (LeakyReLU)     (None, 32, 32, 64)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 128)       131200

 leaky_re_lu_1 (LeakyReLU)   (None, 16, 16, 128)       0

 conv2d_3 (Conv2D)           (None, 8, 8, 256)         524544

 leaky_re_lu_2 (LeakyReLU)   (None, 8, 8, 256)         0

 flatten (Flatten)           (None, 16384)             0

 dropout (Dropout)           (None, 16384)             0

 dense_1 (Dense)             (None, 1)                 16385

=================================================================
Total params: 675265 (2.58 MB)
Trainable params: 675265 (2.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

**DCGAN Model Combined**
The Generator and Discriminator are combined into a single framework by the DCGAN paradigm.
While receiving instruction:

The Generator creates realistic-looking graphics in an attempt to trick the Discriminator.
The Discriminator gains the ability to discriminate between images generated by the Generator and actual images.
Through adversarial training, the model aims to enhance both components: the Generator's ability to produce realistic images and the Discriminator's ability to recognise fakes.

**Design Consideration:**

Selection of Activation Functions: Leaky ReLU was employed in the Discriminator to preserve small gradients when the input is negative, avoiding the "dying ReLU" issue. ReLU was used in the Generator to encourage non-linearities and mimic complex functions.
**Normalisation**: To stabilise and expedite training, batch normalisation was used in the Generator.
**Regularisation**: By introducing noise into the training process, Dropout in the Discriminator helps avoid overfitting.

## 1.9 MODEL COMPILATION

**Loss Function**: Binary Crossentropy is the loss function that is employed by the Generator and the Discriminator. The nature of the task, in which the Discriminator is a binary classifier that discerns between genuine and artificial images, drove this decision. Because Binary Crossentropy works well in this binary classification setting, it is a standard choice in GANs.

**Optimizers**:

**Adam Optimiser**: Because of its adjustable learning rate and momentum qualities, which make it appropriate for training deep networks, the Adam optimiser was chosen for both the Generator and the Discriminator. To maintain a balanced learning pace, the learning rates of the Generator and Discriminator were adjusted independently.

**Rates of Learning**:

**Generator (G_LR = 0.0001)**: The Discriminator can keep up by using a lower learning rate, which makes sure the Generator develops progressively without going overboard.
**Discriminator (D_LR = 0.0002)**: To enable the Discriminator to swiftly adjust to the Generator's enhancements, a little higher learning rate was selected.

**Beta_1 (0.5)**: For both optimisers, the beta_1 parameter—which determines how quickly the gradient's moving average decays—was set to 0.5. Stabilising training with GANs is a popular method.

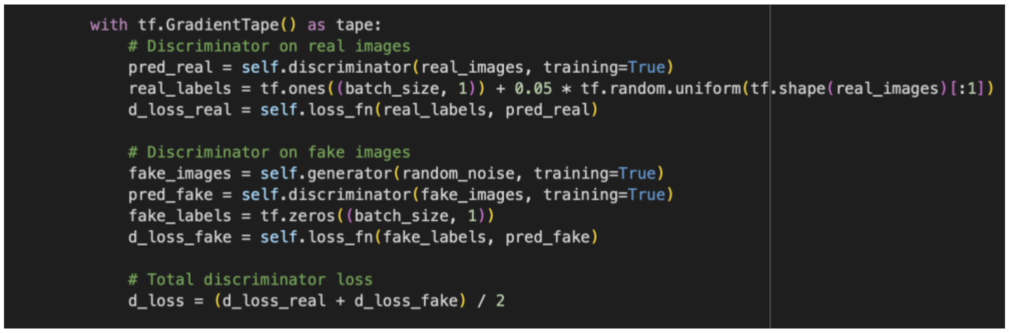
**Batch Size**:

In order to balance the requirements for computational efficiency and steady training, the batch size was set to 32. More memory is needed when the batch size is bigger, but the gradient estimates are more stable. Following testing, it was discovered that a batch size of 32 offered a reasonable compromise between training time and stability.

## 1.10 MODEL TRAINING

**Discriminator Training:**

**Real Images:** A collection of real images from the dataset is used to train the discriminator initially. To keep the Discriminator from being overconfident, noise (1 + 0.05 * random_noise) is added to the real labels in small amounts.
**Fake Images:** Next, a set of fake images produced by the Generator are used to train the discriminator. These fake images have labels set to 0.
The average loss on both actual and false images is used to compute the discriminator loss.



```
with tf.GradientTape() as tape:
    # Discriminator on real images
    pred_real = self.discriminator(real_images, training=True)
    real_labels = tf.ones((batch_size, 1)) + 0.05 * tf.random.uniform(tf.shape(real_images)[:1])
    d_loss_real = self.loss_fn(real_labels, pred_real)

    # Discriminator on fake images
    fake_images = self.generator(random_noise, training=True)
    pred_fake = self.discriminator(fake_images, training=True)
    fake_labels = tf.zeros((batch_size, 1))
    d_loss_fake = self.loss_fn(fake_labels, pred_fake)

    # Total discriminator loss
    d_loss = (d_loss_real + d_loss_fake) / 2
```

**Generator Training:**

By attempting to trick the Discriminator, the Generator learns to create false images, a batch of random noise is run through the Generator. The Discriminator then classifies these pictures.
In order to minimise the discrepancy between the Discriminator's output for fake images and the label 1 (actual), the Generator loss is computed based on the Discriminator's prediction.

**Gradient Updates**:

Backpropagation is used to calculate the gradients of the loss with respect to the model weights. Subsequently, the optimisers adjust the Generator and Discriminator weights in order to minimise their respective losses.

**Monitoring Training Progress:**

To see the Generator's progress, a custom callback called DCGANMonitor is used to generate and display images at the end of predefined epochs.
To prevent memory overflow problems, a different custom callback called ClearMemoryCallback is utilised to clean GPU memory at the end of each epoch.

**Training Duration:**

**Epoches:** Over **5000 epochs** were used to train the model. In order to give the Generator enough time to learn the intricate patterns needed to generate realistic images, a high number of epochs was selected.
**Steps Per Epoch:** This measure, which makes sure every image regularly contributes to the training process, is calculated by dividing the total number of training photos by the batch size.

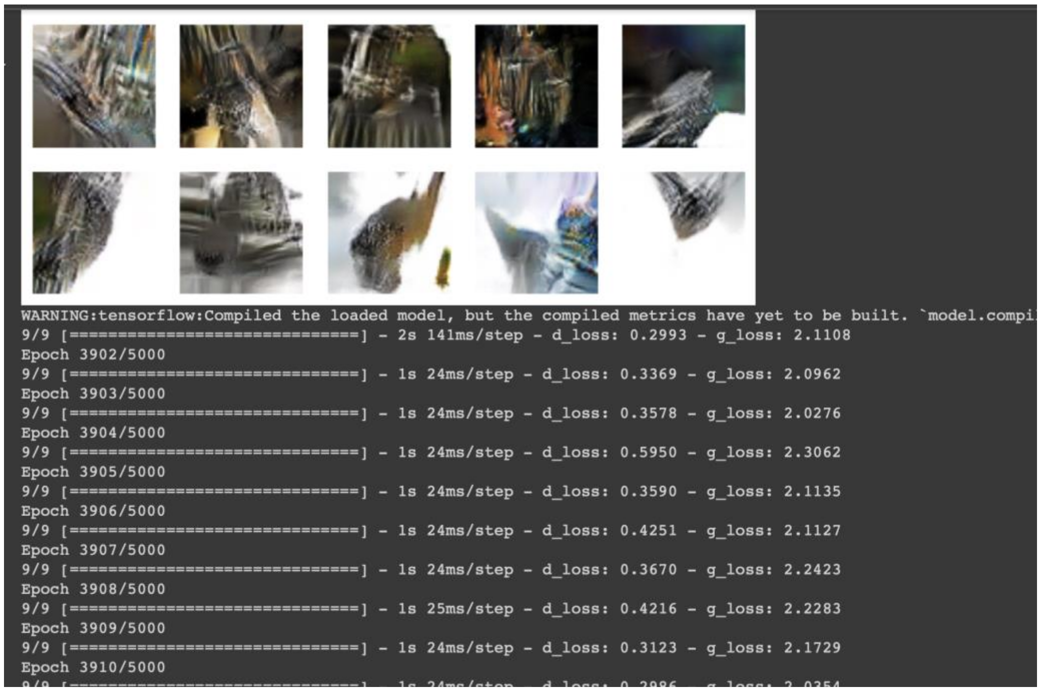
## 1.11 MODEL EVALUATION

### Qualitative Evaluation

Considering the nature of GANs, visually inspecting the images produced by the Generator is one of the most straightforward ways to assess the model. In order to evaluate the realism and diversity of the outputs, sample images were created during the training process at regular intervals (e.g., every 10 epochs) and visually examined.

### Visual Inspection

We looked at things such item shapes' coherence, clarity, and the presence of artefacts in the photos.
The generated images may look noisy or incomplete early in the training process. The visuals should gradually becoming more realistic as training goes on.
A proficient Generator ought to generate visuals that are challenging for the human sight to discern from authentic ones.



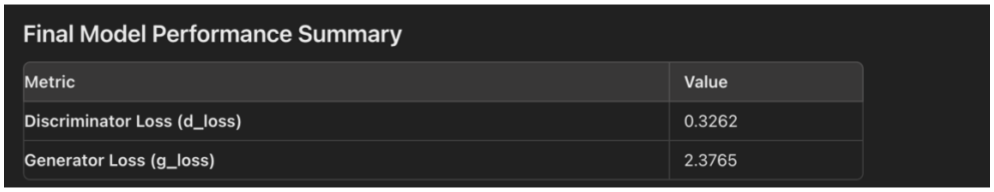
```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compi
9/9 [==============================] - 2s 141ms/step - d_loss: 0.2993 - g_loss: 2.1108
Epoch 3902/5000
9/9 [==============================] - 1s 24ms/step - d_loss: 0.3369 - g_loss: 2.0962
Epoch 3903/5000
9/9 [==============================] - 1s 24ms/step - d_loss: 0.3578 - g_loss: 2.0276
Epoch 3904/5000
9/9 [==============================] - 1s 24ms/step - d_loss: 0.5950 - g_loss: 2.3062
Epoch 3905/5000
9/9 [==============================] - 1s 24ms/step - d_loss: 0.3590 - g_loss: 2.1135
Epoch 3906/5000
9/9 [==============================] - 1s 24ms/step - d_loss: 0.4251 - g_loss: 2.1127
Epoch 3907/5000
9/9 [==============================] - 1s 24ms/step - d_loss: 0.3670 - g_loss: 2.2423
Epoch 3908/5000
9/9 [==============================] - 1s 25ms/step - d_loss: 0.4216 - g_loss: 2.2283
Epoch 3909/5000
9/9 [==============================] - 1s 24ms/step - d_loss: 0.3123 - g_loss: 2.1729
Epoch 3910/5000
9/9 [==============================] - 1s 24ms/step - d_loss: 0.2986 - g_loss: 2.0354
```

## 1.12 MODEL PREDICTION

**Interpretation**:

**Discriminator Loss (d_loss):** A lower value indicates that the discriminator became reasonably good at distinguishing between real and generated images. However, if it's too low, it could mean the discriminator overpowered the generator.

**Generator Loss (g_loss):** A higher value suggests that the generator struggled to fool the discriminator with its generated images. Ideally, this value should decrease over time as the generator improves.



Final Model Performance Summary

| Metric | Value |
|---|---|
| Discriminator Loss (d_loss) | 0.3262 |
| Generator Loss (g_loss) | 2.3765 |

## 1.13 CONCLUSION

Our goal was to build a Generative Adversarial Network (GAN) that could produce realistic, synthetic pictures of nuclear fuel bundles. Carefully preparing the dataset through pre-processing and augmentation to provide the model a wide variety of photos was part of the procedure. The generator in the GAN architecture was in charge of creating artificial images, while the discriminator was in charge of telling them apart from actual ones. Using carefully chosen hyperparameters, such as learning rates, batch sizes, and optimisers, the model was trained across 5000 epochs.

In spite of these attempts, the end outcomes showed that the model was unable to produce synthetic visuals that were compelling. The discriminator's efficacy in distinguishing between actual and artificial images was demonstrated by its stabilisation at 0.3262. Nonetheless, the generator loss persisted at 2.3765, indicating that the generated synthetic images were insufficiently lifelike to fool the discriminator. This disparity draws attention to the difficulties the generator has in raising the image quality.

The result implies that more effort is needed to get the intended outcomes. Enhancing the dataset, investigating more complex GAN variations, and improving the model architecture are some possible avenues for improvement. Notwithstanding the difficulties, this work offers insightful information about the intricacies of GAN training and lays the groundwork for further studies targeted at producing superior synthetic images in this field.