Generative Adversarial Networks Based Scene Generation on Indian Driving Dataset

K. Aditya Shastry*, B.A. Manjunatha, M. Mohan, T.G. Mohan Kumar & D.U. Karthik

Department of Information Science and Engineering,
Nitte Meenakshi Institute of Technology, Bengaluru, 560064, India
*E-mail: adityashastry.k@nmit.ac.in

Abstract. The rate of advancement in the field of artificial intelligence (AI) has drastically increased over the past twenty years or so. From AI models that can classify every object in an image to realistic chatbots, the signs of progress can be found in all fields. This work focused on tackling a relatively new problem in the current scenario-generative capabilities of AI. While the classification and prediction models have matured and entered the mass market across the globe, generation through AI is still in its initial stages. Generative tasks consist of an AI model learning the features of a given input and using these learned values to generate completely new output values that were not originally part of the input dataset. The most common input type given to generative models are images. The most popular architectures for generative models are autoencoders and generative adversarial networks (GANs). Our study aimed to use GANs to generate realistic images from a purely semantic representation of a scene. While our model can be used on any kind of scene, we used the Indian Driving Dataset to train our model. Through this work, we could arrive at answers to the following questions: (1) the scope of GANs in interpreting and understanding textures and variables in complex scenes; (2) the application of such a model in the field of gaming and virtual reality; (3) the possible impact of generating realistic deep fakes on society.

Keywords: *artificial intelligence; deep learning; driving dataset; generative adversial networks; scene generation.*

1 Introduction

The advent of artificial intelligence began in 1943 when the first paper on the concept of a neural network was published. Progress in AI has come a long way, from a theoretical neural network model to models that can detect, classify, predict, and more recently, generate data they have been trained on. The internet has been abuzz with deep fakes and the potential issues such technology could cause with the advent of generative models. However, generative models provide a twofold benefit: they can generate new data in areas lacking suitable data (such as native language processing) and they can be used to interpret and analyze already existing data.

Received August 23rd, 2022, 1st Revision March 4th, 2023, 2nd Revision June 15th, 2023, Accepted for publication July 21st, 2023.
Copyright © 2023 Published by IRCS-ITB, ISSN: 2337-5787, DOI: 10.5614/itbj.ict.res.appl.2023.17.2.4

While other types of models focus on the latter objective, generative models are more promising because they have a lower number of parameters than the total number of features in the data and are hence forced to find the best and most representational features of the lot [1]. This work employed the use of generative adversarial networks (henceforth referred to as GANs) to generate new photorealistic scenes of India. A GAN, first proposed by Ian Goodfellow in 2014 [2], uses two different neural networks competing against each other.

Among various types of generative models, GANs have gained popularity due to their ability to generate high-quality data while using a smaller number of parameters than the total features in the data. GANs consist of two neural networks that compete with each other, resulting in a game-like scenario in which one network generates data and the other network evaluates their authenticity. This paper focuses on the use of GANs to generate photorealistic scenes of India. The research employed the methodology proposed by Ian Goodfellow [2]. This paper will explore the potential of GANs to generate new data and provide a deep analysis of the generated images, which can be useful in various applications such as computer vision, entertainment, and virtual reality.

The research hypothesis was that by using GANs, realistic images can be generated from a purely semantic representation of a scene, specifically the Indian Driving Dataset. We believe that this research will provide valuable insights into the scope of GANs in interpreting and understanding textures and variables in complex scenes. Additionally, we hypothesized that our model could have significant application in the fields of gaming and virtual reality, as it can create immersive and realistic environments. Finally, we predicted that our research would highlight the possible negative impact of generating realistic deep fakes on society, as it could be used for malicious purposes such as spreading misinformation or creating fake news. Overall, we expect that our study will contribute to the further advancement of generative capabilities of AI and provide important insights into the potential applications and risks associated with this technology.

The following are the research contributions of our work:

1. Investigating the generative capabilities of AI, which is a relatively new problem, with a focus on GANs.
2. Developing a GAN-based model that can generate realistic images from a purely semantic representation of a scene.
3. Training the GAN model on the Indian Driving Dataset and evaluating its ability to interpret and understand textures and variables in a complex scene.
4. Exploring potential applications of the proposed model in the fields of gaming and virtual reality.

5. Investigating the possible negative impact of generating realistic deep fakes on society.
6. Providing a detailed comparison of the proposed work with related work.

2 Related Work

Since the original GAN paper [2] was published, there has been tremendous growth in the research interest surrounding GANs. To fully understand what these novel models are capable of, we undertook an extensive survey of papers published in this field. We discovered several types of GANs and their applications in various fields, such as image processing, audio and video processing, 3D model generation, and so on. The next section covers a few of the GANs we surveyed and considered for our work. As per our survey, GANs have not previously been used on the IDD dataset for scene generation. DCGANs were the first major improvement on the GAN architecture. They are more stable in terms of training and generate higher-quality samples. Reference [3] proposed three improvements of classic GAN, namely replacement of any pooling layers with strided convolutions (for the discriminator) and fractional strided convolutions (for the generator); removing the fully connected hidden layer and using batch normalization in both networks; and using LeakyReLU in the discriminator and ReLU activation in the generator.

Conditional GANs [4] were another advancement on the basic GAN framework. While the original GAN used noise as the sole input to the generator, C-GANs employ the use of a condition to control the generated output. The condition could be a class label, a description, or any relevant data. This condition is fed to both the discriminator and the generator as an additional input layer.

Reference [5] built a model popularly known as pix2pix to perform the following tasks: mapping semantic labels to realistic images; aerial images to maps; black and white to color images; edges to images; and day-time images to night-time images. The model employs the use of a DCGAN for the generator and the discriminator. The results produced are realistic, but the training time and the size of the dataset increase drastically with the complexity of the problem. Built to improve on the previous paper, pix2pixHD [6] produces high-resolution images (a relatively new feature in image generation through AI) from semantic maps. The proposed technique converts semantic labels of street views to real images, labels of faces to realistic images, and offers a real-time editing interface with which users can manipulate the semantic labels and immediately see the resultant change in the output. As with pix2pix, the street views used the Cityscapes dataset [7]

Cycle GAN [9] is an architecture used for image-to-image translation. The model captures features in an image collection and figures out how these can be translated into another image collection; this method is also called style transfer. A cycle GAN provides a two-way translation function. G: X $\rightarrow$ Y and F: Y $\rightarrow$ X must be inverses of each other. As mentioned in the previous section, the task of generating realistic driving scenes is not an altogether new idea. We surveyed the methods used for this task to use on the IDD dataset. All the above papers used the Cityscapes dataset [8] for their models. The present work considered the Indian Driving Dataset [10] for similar models.

The method proposed in Reference [11] was aimed at learning a mapping function between an input source video to an output photorealistic video. The input source video consisted of semantic labels of frames from the Cityscapes dataset. A spatio-temporal training method was employed on the model. Two discriminators were used, one to ensure quality of the images and spatial accuracy, and the other to ensure that the frames in sequence are temporally sound. The videos generated by vid2vid are unique in terms of their high quality and accuracy over time. However, due to the higher number of neural networks having to run simultaneously, the training period and the resources used are both on the high side. The same work also released results for pose detection and translation and face to edges to face translation.

Reference [12] proposed using generative neural networks to replicate human driving behavior for autonomous driving. They used a stable GAN architecture to train a controller-trainer network using images and key press data from a video game (Road Rash). The method shows promising results in learning synthetic driving behavior. Reference [13] attempted to generate realistic images from ground and aerial views, which is a challenging computer vision task. The proposed method uses homography to map the images between views and generative adversarial networks to add realism. The study demonstrated that incorporating geometry constraints is a better approach for cross-view image synthesis, adding fine details to the generated images.

Reference [14] proposed a scene generation framework based on GANs that sequentially composes scenes with explicit control over the elements. The framework has separate background and foreground generators, where the foreground objects populate the scene one-by-one. The study demonstrated that the proposed framework could produce more diverse images and handles transformations and occlusion artifacts better than existing approaches through experiments on a subset of the MS-COCO dataset. Reference [15] presents an unsupervised method for generating diverse and realistic images using a class-conditional GAN model. The model is conditioned on automatically derived labels from clustering in the discriminator's feature space to cover different modes. The proposed method outperformed several existing methods in addressing mode collapse and improved image diversity and quality metrics on large-scale datasets such as ImageNet and Places365.

Reference [16] provides a comprehensive overview of image-to-image translation based on GANs and its variants, including state-of-the-art techniques based on multimodal and multidomain representations. Finally, it summarizes open issues and future research directions, including the use of reinforcement learning and 3D modal translation. The review paper [17] provides an overview of the GAN framework for various image and video synthesis tasks, discussing its applications in image translation, image processing, video synthesis, and neural rendering. It also covers important techniques to stabilize GAN training and enable the generation of high-resolution photorealistic images and videos.

Reference [18] provides a comprehensive review of adversarial models for image synthesis using GANs. It covers various categories, such as image-to-image translation, label-to-image mapping, and text-to-image translation, and discusses the base models, architectures, loss functions, evaluation metrics, and datasets used. The review also highlights potential future research directions and includes a collection of loss-variant GANs, evaluation metrics, remedies for several image generation issues, and stable training. Reference [19] provides a review of deep learning-based frameworks used for semantic segmentation of road scenes, discussing their architectures, well-known datasets, data augmentation techniques, and domain adaptation methods. It also includes a quantitative analysis and performance evaluation of different frameworks on reviewed datasets and highlights future research directions in the field of semantic segmentation using deep learning.

The research reported in [20] provides a comprehensive review and meta-analysis of 231 journal papers on the use of GANs in remote sensing (RS) applications. The review covers theories, applications, and challenges of GANs, highlights gaps in the field, and provides insight into the potential of GANs for various RS applications. Reference [21] provides an overview of recent advances in GANs. The paper discusses the challenges faced in training GANs, such as instability, mode collapse, and non-convergence, and how researchers have addressed these issues by modifying the network topology, goal functions, and optimization techniques. The paper also highlights the progress made in GAN architecture and optimization solutions for improving their efficiency in various computer vision applications. The authors suggest that further research is needed in this area to solve real-time computer vision applications using GANs.

The paper [22] provides an overview of the GAN approach and its various model types, as well as their benefits and limitations, and potential model alterations.

The study also highlights the challenges of training GANs and offers suggestions for parameter measurement. The paper reviews various GAN applications in image processing and discusses their potential reach. Overall, this paper offers valuable insight into GANs and their applications in image processing. Reference [23] proposes an unsupervised representation learning method by combining deep neural networks (DNNs) with GANs.

The method involves using encoder networks in addition to generative models for feature extraction, leading to improved performance and faster learning in GANs. The proposed approach was shown to outperform other unsupervised feature learning methods by 2% to 6% in terms of classification accuracy. Reference [24] introduced an unsupervised method for generating diverse images using a class-conditional GAN model that does not use manually annotated class labels. Instead, the model is conditional on labels automatically derived from clustering in the discriminator's feature space, which automatically discovers different modes and requires the generator to cover them. The method outperformed several competing methods when addressing mode collapse and performed well on large-scale datasets such as ImageNet and Places365, improving both diversity and standard metrics compared to previous methods.

The research initiative [25] involved a collaboration between the United States and China with the aim of enhancing the realism of driving simulators. This suggests a shared interest in advancing driving simulation technology. The researchers proposed the use of GANs as a means to achieve this goal. GANs are a type of machine learning model known for their ability to generate realistic synthetic data. The researchers have taken a novel approach to address the challenge of producing photorealistic point-of-view (POV) driving scenarios. They developed a hybrid method that combines the strengths of different approaches. Specifically, they mix the more photorealistic output of cycle GAN-based systems with conventionally generated elements that require more detail and consistency, such as road markings and observed vehicles.

The hybrid system developed by these researchers is called Hybrid Generative Neural Graphics (HGNG). It involves injecting limited output from a conventional computer-generated imagery (CGI)-based driving simulator into a GAN pipeline. The environment generation within this pipeline is facilitated by the NVIDIA SPADE framework. According to the authors, the advantage of using the HGNG system is the potential to create more diverse driving environments, resulting in a more immersive experience. They emphasize that even converting CGI output to photoreal neural rendering output alone cannot address the problem of repetition. The original footage entering the neural pipeline is constrained by the limitations of the model environments, leading to repeated textures and meshes. Overall, this paper presented an innovative approach to improving the realism of driving simulators through the integration of different techniques and the utilization of GANs. The aim was to enhance the immersivity and diversity of driving environments to provide a more realistic experience for users.

Reference [26] proposed a HGNG pipeline to enhance the visual fidelity of driving simulations. This approach involves partially rendering important objects of interest, such as vehicles, and employing generative adversarial processes to synthesize the background and the rest of the image. A novel image formation strategy was introduced to generate 2D semantic images from simple object models without textures within a 3D scene. These semantic images are then transformed into photorealistic RGB images using a state-of-the-art GAN trained on real-world driving scenes. To address the issue of repetitiveness in the generated images, a blending GAN is used to blend the partially-rendered and GAN-synthesized images together.

The researchers demonstrated that the proposed method generates images with higher photorealism compared to conventional approaches. They validated this claim by conducting a semantic retention analysis and measuring the Fréchet Inception Distance (FID) against real-world driving datasets such as Cityscapes and KITTI. In summary, the research presents a HGNG pipeline that aims to improve the visual fidelity of driving simulators. By combining partial rendering, GAN synthesis, and blending techniques, the method achieves higher photorealism, reducing repetitiveness and enhancing immersion. The evaluation against real-world driving datasets supports the superiority of the proposed approach over conventional methods.

Table 1 shows a detailed comparison of the present work with related works. Based on Table 1, we see that most of the related works did not address the following aspects which form the key research contributions of our proposed work: no realistic image generation from semantic scene representation; no gaming/VR applications; no exploration of societal impact; no use of the IDD dataset; no evaluation of texture interpretability; and no discussion of future research.

Table 1 Comparison of the proposed work with related works.

Literature	Comparison with Proposed Work
[3]	No semantic scene generation; deep CGAN constraints; no use of the Indian Driving Dataset; no consideration of deep fakes' societal impact.
[4]	No realistic image generation from semantic scenes; CGAN limited to multi-modal and tagging; no interpretation of textures and variables from IDD.
[5]	General image-to-image translation; no specific dataset mentioned; no exploration of gaming/VR applications or societal impact of deep fakes.

Table 1 Continued. Comparison of the proposed work with related works.

Literature	Comparison with Proposed work
[6]	Limited scope: video-to-video synthesis; no evaluation of societal impact or interpretability of textures/variables; no use of Indian Driving Dataset or discussion of gaming/VR applications.
[7]	No generation of realistic images; ignores generative AI capabilities, and neglects gaming/VR applications and deep fake impact on society.
[8]	Does not explore AI's generative capabilities; uses Cityscapes with fewer labels than the proposed IDD dataset; neglects potential GAN applications in gaming and virtual reality.
[9]	Focuses on high-res photo-realistic image synthesis from label maps; lacks dataset specification; neglects discussion of challenges and future research directions.
[10]	Does not explore gaming/VR applications or impact of realistic deep fakes on society
[11]	Does not include an investigation of the model's impact on society.
[12]	Homogeneous dataset limits generalization; societal implications not explored.
[13]	No realistic image generation; limited homography generalizability; only for cross-view image synthesis.
[14]	No realistic image generation; limited data diversity due to MS-COCO dataset; incomplete evaluation; no societal applications discussed; nongeneralizable without autoencoders.
[15]	Unsupervised with limited accuracy; no exploration of deep fakes or gaming applications.
[16]	Review work without proposal of a model.
[17]	Review work without proposal of a GAN model; does not address the generative capabilities of GAN.
[18]	Review work lacking implementation/evaluation; ignores societal impact/semantic representation.
[19]	No generative exploration; limited dataset/evaluation; no gaming/VR discussion.
[20]	No detailed GAN investigation/solutions proposed; addresses a specific audience.
[21]	Review without GAN implementation; no gaming/VR exploration; no generative model developed.
[22]	Survey without GAN implementation; no specific applications discussed; lacks in-depth discussion.
[23]	Ignores AI generative capabilities in complex scenes; no dataset specification; neglects deep fakes.
[24]	No gaming/VR exploration; neglects societal impact investigation of realistic deep fakes.
[25]	Does not delve into the societal implications of their research.
[26]	Exhibits limitations in terms of dataset dependence; lack of a detailed discussion on the challenges of conventional approaches.

3 Proposed Architecture

The architecture used for this work is described in this section. Our generator was a U-Net, as shown in Figure 1. A U-Net [27] is a classic encoder-decoder model with skip connections. Skip connections directly connect the encoder layer to the decoder layer, allowing the neural network to bypass certain blocks if no useful feature is learnt in them.

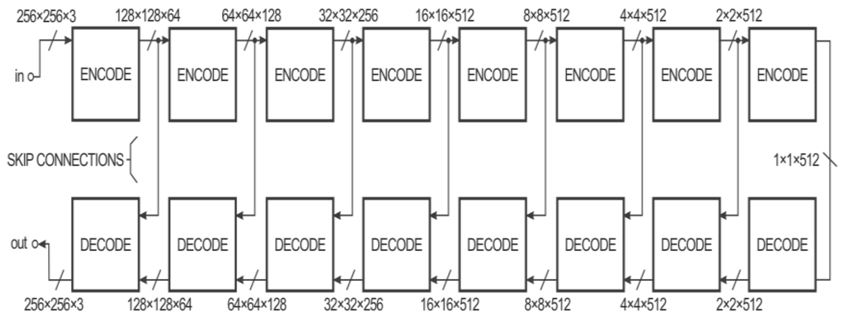

Figure 1 Generator architecture.

The input to the generator is a semantic image and the output is a realistic image. The generator is trained on the losses produced by the discriminator. The discriminator takes in two inputs, the semantic map and the real image or the generated image. It uses these two inputs to generate a probability that the real/generated image is real and corresponds to the semantic map. The architecture of the discriminator is given in Figure 2.

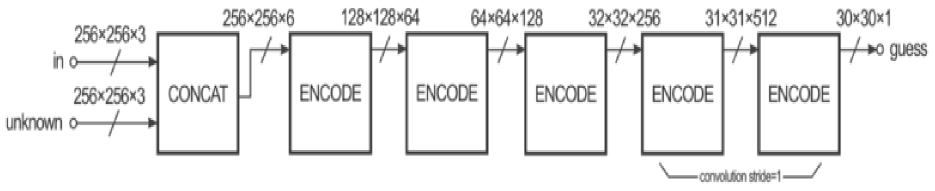

Figure 2 Discriminator architecture.

The discriminator is a PatchGAN. A PatchGAN outputs an image with a size of 30 x 30 x 1 pixels, where each pixel corresponds to the believability of a certain patch in the original image. It is trained to maximize the output of the semantic map and the real image to 1. In mathematical terms it is expressed in Eq. (2):

$$D(SI, RI) \approx 1 \text{ and } D(SI, G(SI)) \approx 0 \qquad (2)$$

3.1 Proposed Algorithm

Algorithm 1 shows the proposed GAN algorithm for generating scenes from the IDD dataset.

Algorithm-1: GAN-scene-generation

```
begin
def make_generator_model(): # Define the generator model
   model ← Sequential
   model.add(Dense-Layers)
   model.add(BatchNormalization())
   model.add(LeakyReLU())
   model.add(Convolution-2D-Layer)
   model.add(BatchNormalization())
   model.add(layers.LeakyReLU())
   model.add(Convolution-2D-Layer)
   model.add(BatchNormalization())
   model.add(LeakyReLU())
   model.add(Convolution-2D-Layer, tanh)
   model.add(Reshape((output_shape))) # Add reshape layer for desired output shape
  return model
def make_discriminator_model(): # Define the discriminator model
    model ← Sequential()
    model.add(Convolutional-2D-Layer)
    model.add(LeakyReLU())
    model.add(Dropout(0.3))
    model.add(Convolutional-2D-Layer)
    model.add(LeakyReLU())
    model.add(layers.Dropout(0.3))
    model.add(Flatten())
    model.add(Dense-Output-Layer(1))
    return model
def generator_loss(fake_output): # Define the loss function for the generator
    return(Losses.sigmoid_cross_entropy(ones_like(fake_output), fake_output))
def discriminator_loss(real_output, fake_output): # Define the loss function for the discriminator
    real_loss ← Losses.sigmoid_cross_entropy(multi_class_labels=ones_like(real_output),    logits=real_output)
    fake_loss ← Losses.sigmoid_cross_entropy(multi_class_labels=zeros_like(fake_output), logits=fake_output)
    total_loss ← real_loss + fake_loss
    return(total_loss)
generator_optimizer ← Train.AdamOptimizer() # Define the optimizer
discriminator_optimizer ← Train.AdamOptimizer()
data ← Load('indian_driving_dataset.npy') # Load the Indian Driving dataset
data = (data / 127.5) – 1 # Normalize the data
BUFFER_SIZE = 60000; BATCH_SIZE = 256; EPOCHS = 50 # Define the batch size and number of epochs
train_dataset ← data.shuffle(BUFFER_SIZE).batch(BATCH_SIZE) # Batch and shuffle the data
generator = make_generator_model() # Define the generator and discriminator models
discriminator = make_discriminator_model()
checkpoint_dir = './training_checkpoints' # Define the checkpoint directory and checkpoint prefix
checkpoint
def generate_random_latent_vector(): # Generate random latent vectors
            return np.random.normal(size=(BATCH_SIZE, latent_dim))
def generate_images(generator, latent_vector): # Generate images from the generator using latent vectors
            generated_images = generator.predict(latent_vector)
            return generated_images
for epoch in range(EPOCHS):
for batch in train_dataset:
# Train the discriminator
noise = generate_random_latent_vector()
with tf.GradientTape() as disc_tape:
generated_images = generator(noise, training=True)
real_output = discriminator(batch, training=True)
fake_output = discriminator(generated_images, training=True)
disc_loss = discriminator_loss(real_output, fake_output)
gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
end
```

Algorithm 1 shows a GAN to generate images from the Indian Driving Dataset. The GAN consists of a generator and a discriminator model. The generator takes random noise as input and generates images that the discriminator then tries to distinguish from real images in the dataset. The two models are trained together, with the generator trying to produce images that fool the discriminator, and the discriminator trying to correctly distinguish between real and fake images. The specific implementation of the generator and discriminator models is as follows.

The generator has an input layer that takes random noise as input, followed by a dense layer. Batch normalization and a LeakyReLU activation function are applied to this output followed by two transposed convolutional layers with batch normalization and LeakyReLU activation functions. The final layer is another transposed convolutional layer with a tanh activation function. The discriminator has an input layer followed by two convolutional layers with LeakyReLU activation functions and dropout. The output is then flattened and passed through a dense layer with a single output, representing the probability that the input image is real. The loss function for the generator is the sigmoid cross entropy between the generated output and a vector of ones. The loss function for the discriminator is the sum of the sigmoid cross entropy between the real output and a vector of ones, and the sigmoid cross entropy between the fake output and a vector of zeros. The models are trained using the Adam optimizer. The dataset is loaded from an npy file, normalized to the range [-1, 1] and then batched and shuffled. The training loop runs for fifty epochs, during which the models are trained on batches of data, with the generator being updated twice for every update of the discriminator. The checkpoints for the models are saved in the 'training_checkpoints' directory.

3.2 Training:

The training process is performed as shown in Algorithm 2.

Algorithm-2: Training

```
        for epoch in range(total_epochs) do
        for batch in range(total_batches), do
Semantic_Map = generate_random_semantic_map()
        Fake_Image= Generate(Semantic_Map)
        err_Fake=Discriminate(Fake_Image)
        Real_Image = get_real_image()
        err_Real=Discriminate(Real_Image)
        Generator_Err = err_Fake
        Discriminator_Err = avg(err_Real, err_Fake)
        Propagate error and optimize models
        Generator_Losses.append(Generator_Err)
        Discriminator_Losses.append(Discriminator_Err)
        Analyze_performance(Generator_Losses, Discriminator_Losses)
End
```

Algorithm 2 trains the model by generating fake images from semantic maps, calculating the errors from the discriminator, and optimizing the generator and discriminator models through error propagation. The training was performed on Colaboratory, Google's free online computing resource. The entire process took five to six days to train. Table 2 specifies the training parameters used for our model.

Table 2 Summary of training parameters.

Training parameters	Training parameter values
Number of epochs	150
Mini batch size	32
Learning rate	2e−4
Optimizer	Adams
Loss Functions	Loss L1 (used for both generator and discriminator). Loss BCE (used for generator)

4 Experimental Setup and Results

Experiments were performed on Google Colab with the software Python 3.6, Pytorch 1.0.1, Cuda 10, and PIL; and the hardware CPU: SSE2 instruction set support (Unity Requirements), and GPU: graphics card with DX10 (shader model 4.0) capabilities.

4.1 Training Specifications:

Our model was trained for 150 epochs until the loss functions of the generator and the discriminator reached equilibrium (the losses remained parallel to each other) as shown in Figure 3. With Google Colab, the model was trained for about six days.

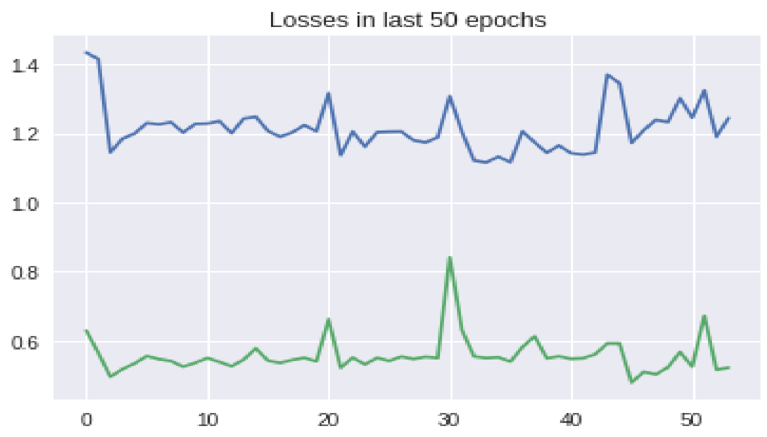

Figure 3 Losses of the networks: generator – blue, discriminator – green.

Figures 4, 5, and 6 show the results generated by our model and the improvement in the quality of the images over epochs. It is clear that there is a distinct improvement in the images generated. In epoch 0, the model seems to have no understanding of colors or details required by the scene. By epoch 75, there is a distinct improvement, where features and colors stand out but are marred by distortion of elements present in the picture. By epoch 150, this distortion has reduced significantly, only showing up when there is ambiguity in the semantic scene such as shadows.

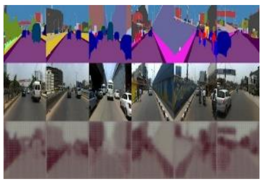
Figure 4 Epoch 0.

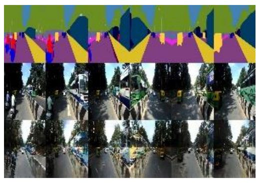
Figure 5 Epoch 75.

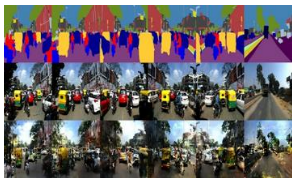
Figure 6 Epoch 150.

4.2 Evaluation of Results

As the Indian Driving Dataset does not currently have a publicly available segmentation model, we were unable to test our results with it and compare the accuracies between our model and the real images from the dataset. Hence, our evaluations of the results were mostly done by simply viewing them and manually deciding if their quality was acceptable and decipherable. While there is a certain amount of noise and distortion in our images, the model was successful in generating objects that could be recognized by humans.

We also used the architecture of pix2pixhd [6-Tang] to compare the images generated by both models, as shown Figures 7 and 8. Table 3 presents the comparison of the two generated images.

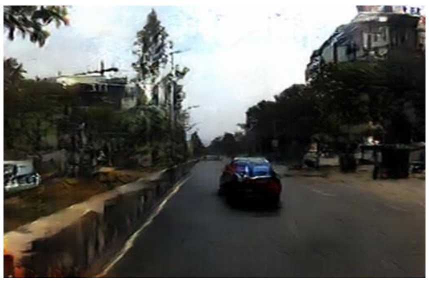
Figure 7 Pix2pix.

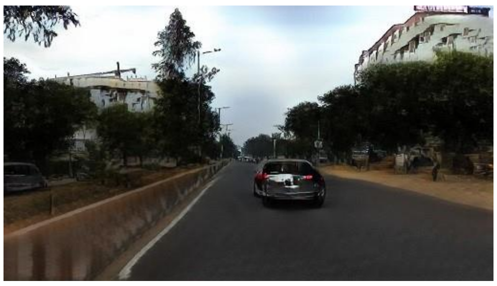
Figure 8 Pix2PixHD.

Table 3 Comparison of Pix2Pix and Pix2PixHD.

Pix2pix	Pix2pixHD
Requires less computational resources (less training time and memory)	More computational resources required (more training time and memory)
Simpler Model (consists of an encoder-decoder)	Complex model (consists of a resnet)
More noise	Less noise

Another method we used for evaluation was through the use of a pre-trained semantic segmentation model, RefineNet. RefineNet is a MATLAB-based neural network meant for identifying objects in scenes and segmenting the images based on them. The pretrained model we used for evaluation was trained on the PASCAL Context dataset. This dataset offers diverse images for the purpose of semantic segmentation 'in the wild'. RefineNet was also one of the top-performing models for semantic segmentation of the Cityscape dataset. Due to these reasons and the lack of an equivalent suitable and publicly available model for the IDD, we chose a RefineNet model for our evaluation. The process flow using the RefineNet model is shown in Figure 9.

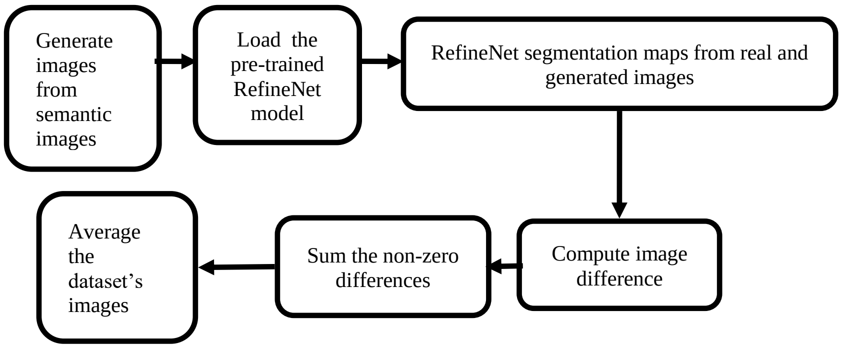
Generate images from semantic images
Load the pre-trained RefineNet model
RefineNet segmentation maps from real and generated images
Compute image difference
Sum the non-zero differences
Average the dataset's images

Figure 9 Process flow of the RefineNet model.

As shown in Figure 9, the process involved the generation of images from semantic images. Subsequently, the real and generated images were fed to the pretrained RefineNet model to obtain the corresponding segmentation maps. Then, the absolute difference between the real and generated images was computed. The total number of non-zero elements in the difference was summed up assuming that this indicated the misclassified number of pixels of the generated image. Finally, the average across all images in the dataset was obtained. Figure 10 depicts the semantic maps of the real and generated images. After running this evaluation method on our dataset, the average error obtained was 40.49%.

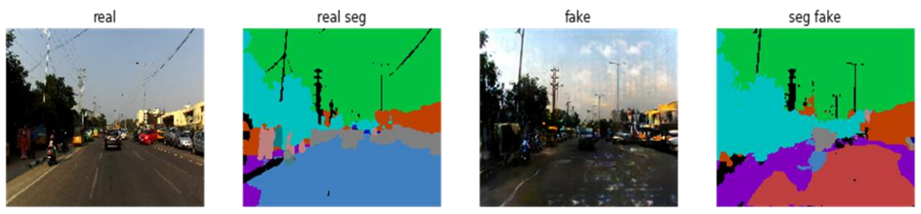

Figure 10 Evaluation using semantic segmentation.

5 Deployment with Unity

Unity is a game development platform for Windows and Mac. Using this platform, we procedurally generated a city made of semantic labels with the addition of an auto rickshaw object. A first-person walkthrough was set up to allow the user to walk around a generated semantically labeled city. By capturing every frame and sending it to our model, we overlayed the city with an equivalent realistic image, allowing users to walk through an AI generated city in first person.

5.1 Design

Figure 11 shows the process flow of Deployment on Unity.

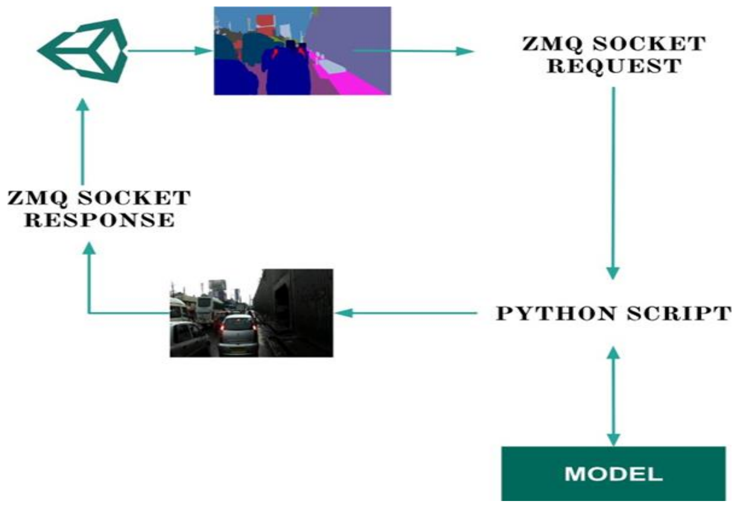

Figure 11 Process flow of Deployment on Unity

We start by generating a city with roads, buildings, pedestrians, and vehicles, with materials corresponding to the semantic labels of the Indian Driving Dataset [7], in a procedural manner in Unity. Then we attach two cameras to the first-person character. The first camera renders the original view of the city and the second camera is used to overlay the generated frames on the screen. The first camera is made to draw the frame into a RenderTexture object, which is in turn read into a Texture object. The Texture object is then encoded as PNG bytes. These bytes are sent over a request-response socket using NetMQ, a library in C# for ZeroMQ. An equivalent ZMQ socket is used in our Python script. The Python script receives the frame in the form of bytes and feeds it to our generator. The generator takes this as input and generates a realistic image based on the frame. This is then encoded as bytes and sent back to the socket in Unity. Unity overlays the received message onto the camera. This process in a loop allows a user to walk through a purely AI generated city.

5.2 Implementation

5.2.1 Unity

Development in Unity was done in C#. To create our ZMQ socket, we used the NetMQ library for Visual Studio. Unity provides an update function that is called every time the frame is updated in the camera. Through this function, we create our socket to send and receive the generated images. This process involves initializing two cameras and connecting to a Python server using a request socket. A Texture object is created to capture the active camera texture, which is encoded as a PNG and sent to the server. Upon receiving a byte array from the server, it is loaded as an image onto another Texture object. The Blit method is then used to efficiently replace every pixel in the second camera’s texture with the new texture. This process is repeated for every frame using the Update method.

5.2.2 Python

While Unity acts as the client in this scenario for sending requests and receiving responses, Python is the server that receives requests, processes them, and sends Unity the appropriate response. We used Python’s PyMQ package to create a ZMQ server socket. The process involves initializing the generator model by loading saved weights, creating a ZMQ socket to connect and listen to a port, and receiving a byte string from a Unity client. The byte string is then converted to a PNG image using the Pillow library’s BytesIO module. The PNG image is pre-processed by resizing to 256 x 256 and removing the alpha channel from it. The pre-processed image is then fed into the generator, and the returned value is stored as an array. The generated image is reshaped to 1024 x 768, and the alpha channel is re-attached. Finally, the image is sent back to Unity over the socket.

5.3 Challenges and Results

One of the key challenges we faced during this phase was maintaining a constant game feed with no latency. As we were running our generator locally, the model utilized the CPU to generate the image and hence caused a visible latency while walking through the city. This problem is easily fixed when the generator is run on a system with a GPU instead. Another challenge we faced was the increase in the noise and distortion in the generated image. We pinpointed this effect since the semantic representations were not accurate, that is, the shapes of the vehicles were rather sharp because cubes were used to create them. When the semantic representation is improved, there is a distinct improvement in the quality of the generated images. Figures 12, 14, 16, and 18 are the equivalent generated frames of Figures 13, 15, 17 and 19.

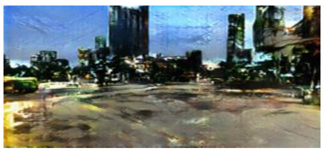
Figure 12 Generated image - first frame.

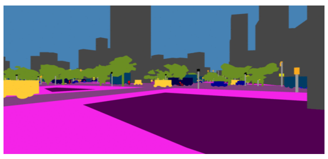
Figure 13 Semantic image – first frame.

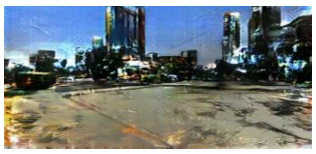
Figure 14 Generated image – second frame.

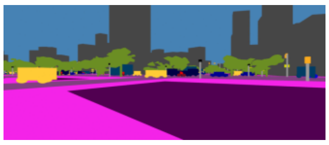
Figure 15 Semantic image – second frame.

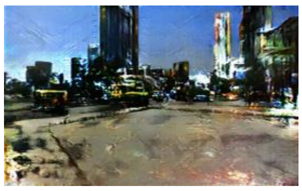
Figure 16 Generated image – third frame.

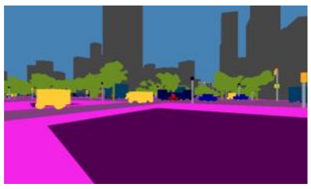
Figure 17 Semantic image – third frame.

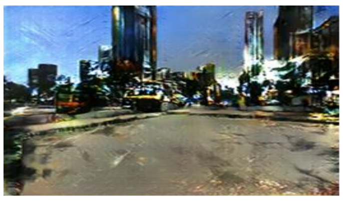
Figure 18 Generated image – fourth frame.

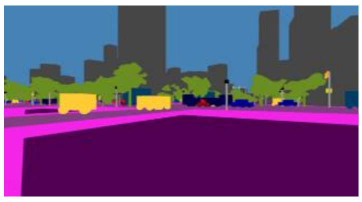
Figure 19 Semantic image – fourth frame.

6 Conclusion and Future Scope

In this research paper, we discussed the use of generative adversarial networks (GAN) to generate realistic images from semantic representations of a scene. We trained our model using the Indian Driving Dataset and explored the scope of GANs in interpreting complex scenes and understanding textures and variables. We also discussed the potential applications of our model in the field of gaming and virtual reality. Finally, we highlighted the possible impact of generating realistic deep fakes on society. We foresee the use of our work in the fields of game design, virtual reality, and autonomous driving. Through our Unity deployment it is clear that the laboriousness of the process of designing landscapes is brought down tremendously, as this model takes over the task of filling in the details. The same applies for designing virtual reality environments. Another application of this work we envision is its use in autonomous driving as the model learns to capture important features and identify elements in a driving scene. We believe this learning can be utilized by autonomous vehicles in their training.

References

[1] Hossam, H., Elgmmal, E. & Elnabawy, R.H., *A Review of Generative Adversarial Networks Applications*, in 4 Novel Intelligent and Leading Emerging Sciences Conference (NILES), pp. 142-146. 2022.

[2] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y., *Generative Adversarial Nets*, in Advances in Neural Information Processing Systems, pp. 2672-2680, 2014.

[3] Radford, A., Metz, L. & Chintala, S., *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016.

[4] Hong, W., Wang, Z., Yang, M. & Yuan, J., *Conditional Generative Adversarial Network for Structured Domain Adaptation*,. in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1335-1344. Salt Lake City, UT, USA, 2018.

[5] Isola, P., Zhu, J. -Y., Zhou, T. & Efros, A.A., *Image-to-Image Translation with Conditional Adversarial Networks*, in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5967-5976, 2017. Honolulu, HI, USA. DOI: 10.1109/CVPR.2017.632. 2017.

[6] Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Liu, G., Tao, A., Kautz, J. & Catanzaro, B., *Video-To-Video Synthesis*, in Advances in Neural Information Processing Systems, **31**, 2018.

[7] Varma, G., Subramanian, A., Namboodiri, A., Chandraker, M. & Jawahar, C.V., *IDD: A Dataset for Exploring Problems of Autonomous Navigation in Unconstrained Environments*, in IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 1743-1751, 2019.

[8] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S. & Schiele, B., *The Cityscapes Dataset for Semantic Urban Scene Understanding*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3213-3223, 2016.

[9] Wang, T. C., Liu, M. -Y., Zhu, J. -Y., Tao, A., Kautz, J. & Catanzaro, B., *High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs.*, in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8798-8807, 2018.

[10] Lin, G., Milan, A., Shen, C. & Reid, I.D., *RefineNet: Multi-path Refinement Networks for High-Resolution Semantic Segmentation*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5168-5177, 2017.

[11] Mottaghi, R., *The Role of Context for Object Detection and Semantic Segmentation in the Wild*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 891-898, 2014.

[12] Ghosh, A., Bhattacharya, B. & Roy Chowdhury, S. B., *SAD-GAN: Synthetic Autonomous Driving using Generative Adversarial Networks*, 30th Conference on Neural Information Processing Systems (NIPS), 2016.

[13] Regmi, K. & Borji, A., *Cross-view image synthesis using geometry-guided conditional GANs*, Computer Vision and Image Understanding, 187, 2019.

[14] Turkoglu, M.O., Thong, W., Spreeuwers, L.J. & Kicanaoglu, B., *A Layer-Based Sequential Framework for Scene Generation with GANs*, AAAI Conference on Artificial Intelligence, 2019.

[15] S Liu, S., Wang, T., Bau, D., Zhu, J.-Y. & Torralba, A., *Diverse Image Generation via Self-Conditioned GANs*, IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) , pp. 14274-14283, 2020.

[16] Alotaibi, A., *Deep Generative Adversarial Networks for Image-to-Image Translation: A Review*, Symmetry, **12**(10), 2020.

[17] Liu, M.Y., Huang, X., Yu, J., Wang, T.C. & Mallya, A, *Generative Adversarial Networks for Image and Video Synthesis: Algorithms and Applications*, Proceedings of the IEEE, 109(5), 2021.

[18] Shamsolmoali, P., Zareapoor, M., Granger, E., Zhou, H., Wang, R., Celebi, M.E. & Yang, J., *Image synthesis with adversarial networks: A comprehensive survey and case studies*, Information Fusion, **72**, pp.126-146, 2021.

[19] Alokasi, H. & Ahmad, M.B., *Deep Learning-Based Frameworks for Semantic Segmentation of Road Scenes*, Electronics, **11**(12), 1884, 2022.

[20] Jozdani, S., Chen, D., Pouliot, D., Johnson, B.A., *A Review and Meta-Analysis of Generative Adversarial Networks and Their Applications in Remote Sensing*, International Journal of Applied Earth Observation and Geoinformation, **108**, 2022.

[21] Pradhyumna, P. & Mohana., *A Survey of Modern Deep Learning based Generative Adversarial Networks (GANs)*, 6th International Conference on Computing Methodologies and Communication (ICCMC), pp. 1146-1152, 2022.

[22] Porkodi, S. P., Sarada, V. & Maik, V., *Generic image application using GANs (Generative Adversarial Networks): A Review*, Evolving Systems, 2022.

[23] Liu, Y., Gal, R., Bermano, A., Chen, B. & Cohen-Or, D., *Self-Conditioned Generative Adversarial Networks for Image Editing*, SIGGRAPH '22: ACM SIGGRAPH 2022, **16**, pp.1-19, 2022.

[24] Mehralian, M. & Karasfi, B., *RDCGAN: Unsupervised Representation Learning with Regularized Deep Convolutional Generative Adversarial Networks*, 9th Conference on Artificial Intelligence and Robotics and 2nd Asia-Pacific International Symposium, pp. 31-38, 2018.

[25] Yurtsever, E., Yang, D., Koc, I.M. & Redmill, K.A., *Photorealism in Driving Simulations: Blending Generative Adversarial Image Synthesis with Rendering*, IEEE Transactions on Intelligent Transportation Systems, **23**(12), pp. 23114-23123, 2022.

[26] Yurtsever, E., Yang, D., Koc, I.M. & Redmill, K.A., *Photorealism in Driving Simulations: Blending Generative Adversarial Image Synthesis with Rendering*, IEEE Transactions on Intelligent Transportation Systems, **23**(12), pp.23114-23123, 2022.

[27] Ronneberger, O., Fischer, P., Brox, T., *U-Net: Convolutional Networks for Biomedical Image Segmentation*, in N. Navab, J. Hornegger, W. Wells, A. Frangi (Eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, Lecture Notes in Computer Science, **9351**, Springer, Cham., 2015.