PHOTO STACK

NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, APPROVED BY AICTE & GOVT.OF KARNATAKA

SYNOPSIS

Submitted in partial fulfilment of the requirement for the award of Degree of

Bachelor of Engineering

in

Computer Science and Engineering

Submitted by:

Nandan M	1NT19CS120
Uday Kiran Chari	1NT19CS204
Veeresh	1NT19CS213
Revanth N Mallol	1NT19CS155

Under the Guidance of
Mrs. Nirmala J.Saunshimath
Assistant Professor, Dept. of CS&E, NMIT

Department of Computer Science and Engineering

(Accredited by NBA Tier-1) 2022-2023

TABLE OF CONTENTS

INTRODUCTION……3

OBJECTIVES……..……..4

METHODOLOGY……..4

RESULTS…….……7

CONCLUSIONS……..7

INTRODUCTION

Background

The crucial job of a Data Scientist is to collect create data as much as possible so that we can train the model and get better accuracy. Used to solve real world problems, pre-collected data is not useful. Insufficient data may lead to low accuracy or inefficient use of the model. ML-based Uber and Google's self-driving cars are trained with the use of synthetic data. In the research department, synthetic data helps you develop and deliver innovative products for which necessary data might not be available.

Brief history of Technology/concept

In 1959, David Hubel and Torsten Wiesel described the "simple cells" and "complex cells" of the human visual cortex. They suggested that both types of cells be used for pattern recognition. A "simple cell" responds to edges and bars in a particular direction. "Complex cells" also respond to edges and bars in a particular direction, but differ from simple cells in that these edges and bars can be moved in the scene and the cells continue to respond. For example, a simple cell may only respond to the horizontal bar at the bottom of the image, and a complex cell may respond to the horizontal bar at the bottom, center, or top of the image. This property of complex cells is called "spatial invariance".

OBJECTIVE

The objective of Generative Adversarial Networks (GANs) in the context of text-to-image synthesis is to generate realistic and visually coherent images from textual descriptions. Specifically, GANs aim to capture the semantic information conveyed by the text and translate it into visually meaningful images that accurately represent the described content. Here are the main objectives of GANs in text-to-image synthesis:

- Realistic Image Generation: GANs aim to capture the essential characteristics, objects, scenes, and attributes mentioned in the text and generate images that are visually convincing and realistic.

- Unsupervised Text-to-Image Learning: GANs enable unsupervised learning in the context of text-to-image synthesis. Unlike supervised approaches that require paired text-image datasets, GANs can learn from unpaired data, where only textual descriptions and corresponding images are available without explicit alignments or annotations.

METHODOLOGY

To run a Generative Adversarial Network (GAN), you will need the following materials:

- Dataset: A collection of data samples that are relevant to your specific task. The dataset should represent the domain you want to generate data in, such as images, text, or audio. Ensure that the dataset is appropriately labeled or organized, depending on the type of GAN you plan to use.

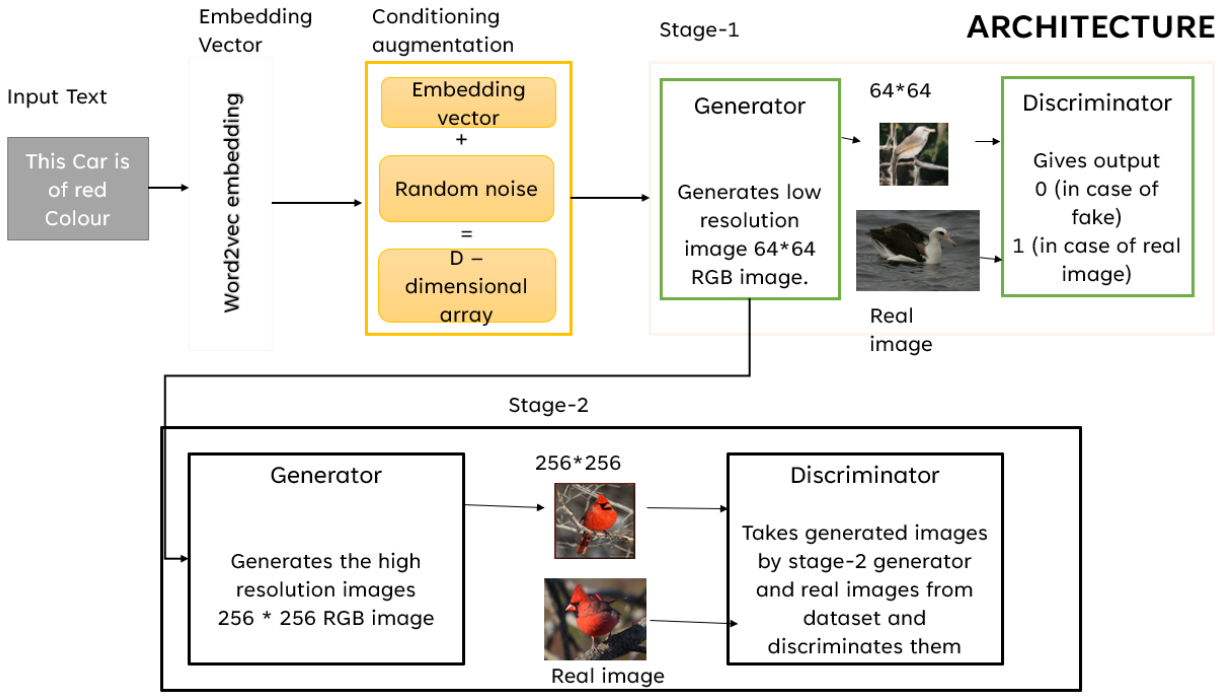
- Hardware: GANs can be computationally intensive, especially for complex tasks or large-scale datasets. Therefore, you will need a computer or server with sufficient computational resources. This typically includes a powerful CPU or GPU, depending on the complexity of the GAN model and the size of the dataset.

- Programming Environment: Set up a programming environment including the chosen deep learning framework and any additional libraries or packages specific to your project.

- Development Tools: Depending on your preferred programming language and environment, you may need code editors or integrated development environments (IDEs) to write, debug, and
- manage your GAN implementation.

- Training Resources: Training a GAN typically requires significant computational resources and time. Ensure that you have access to sufficient memory, storage capacity, and power to support the training process. This includes having access to high-performance GPUs, if possible, to accelerate the training speed.

ARCHITECTURE DIAGRAM



General Overview of proposed work

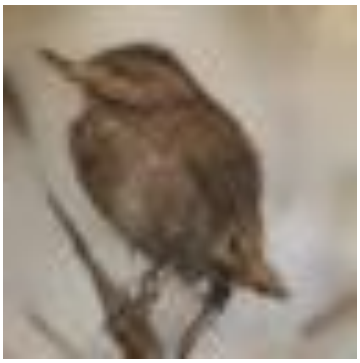
	A	B	C	D	...	Z
Feature 1	0	1	0.2	1	...	0.2
Feature 2	1	0	0	0.5	...	0
Feature 3	0	1	0.3	1	...	0.2
Feature 4	-1	-1	-1	1	..	1

A		B		C		G
1	−	0.2	+	0.2	=	1
0		0		0		0
1		0.3		0.2		1
-1		-1		1		1

```
for epoch in range(start,epochs):
    print("====================================")
    print("Epoch is:", epoch)
    print("Number of batches", int(X_train.shape[0] / batch_size))

    gen_losses = []
    dis_losses = []

    # Load data and train model
    number_of_batches = int(X_train.shape[0] / batch_size)
    for index in range(number_of_batches):
        print("Batch:{}".format(index+1))

        """
        Train the discriminator network
        """
        # Sample a batch of data
        z_noise = np.random.normal(0, 1, size=(batch_size, z_dim))
        image_batch = X_train[index * batch_size:(index + 1) * batch_size]
        embedding_batch = embeddings_train[index * batch_size:(index + 1) * batch_size]
        image_batch = (image_batch - 127.5) / 127.5 # makes the image darker and gets it in range -1 to 1

        # Generate fake images
        fake_images, _ = stage1_gen.predict([embedding_batch, z_noise], verbose=3)

        # Generate compressed embeddings
        compressed_embedding = embedding_compressor_model.predict_on_batch(embedding_batch) #converts the 1024 size to much more less i.e 128
        compressed_embedding = np.reshape(compressed_embedding, (-1, 1, 1, condition_dim)) #-1 is whatever size should be calculated by numpy itself
        compressed_embedding = np.tile(compressed_embedding, (1, 4, 4, 1))

        dis_loss_real = stage1_dis.train_on_batch([image_batch, compressed_embedding],
                                                  np.reshape(real_labels, (batch_size, 1)))
        g_loss = adversarial_model.train_on_batch([embedding_batch, z_noise, compressed_embedding],[K.ones((batch_size, 1)) * 0.9, K.ones((batch_size, 256)) * 0.9
        print("g_loss:{}".format(g_loss))

        dis_losses.append(d_loss)
        gen_losses.append(g_loss)

        """
        Save losses to Tensorboard after each epoch
        """
        # write_log(tensorboard, 'discriminator_loss', np.mean(dis_losses), epoch)
        # write_log(tensorboard, 'generator_loss', np.mean(gen_losses[0]), epoch)

        # Generate and save images after every 2nd epoch
        if epoch % 5 == 0 and index%20==0:
            # z_noise2 = np.random.uniform(-1, 1, size=(batch_size, z_dim))
            z_noise2 = np.random.normal(0, 1, size=(batch_size, z_dim))
            embedding_batch = embeddings_test[0:batch_size]
            fake_images, _ = stage1_gen.predict_on_batch([embedding_batch, z_noise2])

            # Save images
            for i, img in enumerate(fake_images[:10]):
              print("Saving image")
              save_rgb_img(img, "/content/drive/MyDrive/Finalyearproject/Datasets/results_3/gen_{}_{}.png".format(epoch, i))
            stage1_gen.save_weights('/content/drive/MyDrive/Finalyearproject/Datasets/results_3/part1/stage1_generator.h5')
            stage1_dis.save_weights('/content/drive/MyDrive/Finalyearproject/Datasets/results_3/part1/stage1_discrimator.h5')
            ca_model.save_weights('/content/drive/MyDrive/Finalyearproject/Datasets/results_3/part1/stage1_ca.h5')
            embedding_compressor_model.save_weights('/content/drive/MyDrive/Finalyearproject/Datasets/results_3/part1/embedding_compressor.h5')
            adversarial_model.save_weights('/content/drive/MyDrive/Finalyearproject/Datasets/results_3/part1/adversarial_model.h5')
```

RESULTS

Generated Images



CONCLUSION

The project of generating realistic photos using text as input has significant potential in computer vision and artificial intelligence. Deep learning techniques like GANs enable the creation of visually appealing images based on textual descriptions. This bridge between text and visuals opens doors for applications in computer graphics, virtual reality, and content creation. However, generating truly photorealistic images remains challenging, with minor artifacts and inconsistencies. Fine-grained control and higher visual fidelity are areas for improvement. In conclusion, this project offers a promising avenue for research and innovation, pushing the boundaries of visual content generation.

Future Scope

Developing novel network architectures to improve stability and performance.

- Exploring advanced training strategies to address issues like mode collapse and vanishing gradients.
- Enhancing the fidelity and controllability of generated outputs.
- Incorporating domain knowledge to guide the generation process.
- Extending GANs to diverse domains such as natural language processing, video generation, and 3D object synthesis.

- Addressing robustness and fairness concerns in GANs.
- Improving scalability and efficiency to train on larger datasets with reduced computational requirements.
- Advancing techniques for text-to-image synthesis, enabling more accurate and diverse image generation based on textual input.
- Enabling fine-grained control over generated images based on detailed textual descriptions.
- Exploring multimodal approaches that combine text and other modalities (e.g., audio, video) for richer synthesis.
- Investigating methods to generate images conditioned on multiple textual inputs or complex relationships between them.
- Designing architectures that can handle rare or out-of-vocabulary words effectively.
- Developing techniques to improve the interpretability and explain ability of text-to-image synthesis models.
- Incorporating style transfer and artistic rendering capabilities into text-to-image synthesis.