

Text-to-Video GAN Implementation

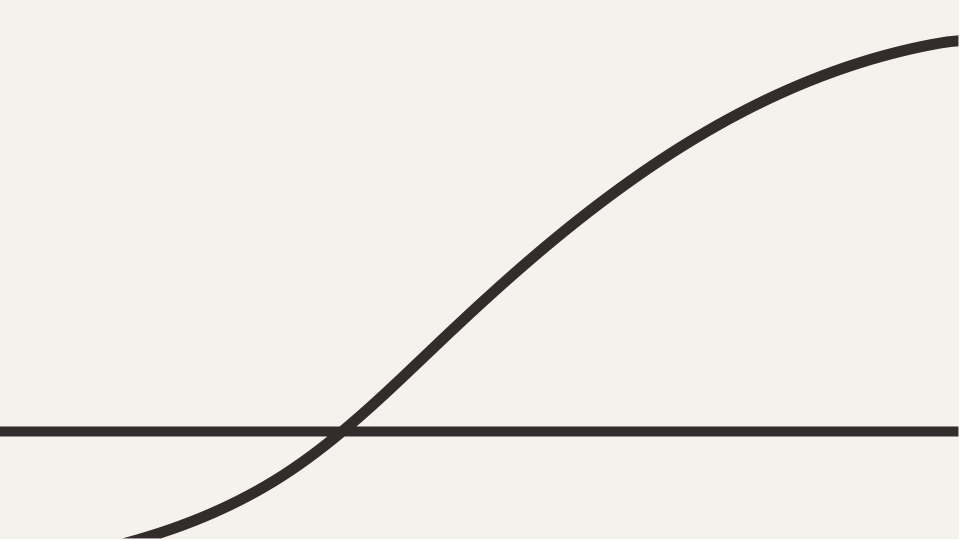
Generating Dynamic Video Content from Text Descriptions

Presented by

Divyansh Garg (21ucs073)

Hiten Ahuja (20ucc051)

Het Patel(21ucc125)



Overview: This project focuses on creating videos dynamically based on text prompts using Generative Adversarial Networks (GANs).

Problem Statement:

Manual video creation is time-consuming.

Need for automated systems for video generation.

Goal: To develop a GAN model that takes text input and generates corresponding videos.



Innovative Features:

- Generate videos from natural language prompts.
- Utilize a GAN architecture for high-quality video synthesis.

Core Capabilities:

- Support for various text prompts describing motions (e.g., "circle bouncing vertically").
- Realistic frame-by-frame continuity in videos.

Applications:

- Media and content creation, gaming, and animation.



Dataset Creation

How the Dataset is Generated:

- Text prompts paired with specific motion descriptions (e.g., "circle moving diagonally up-right").
- Each video contains 10 frames.

Statistics:

- Total videos: 30,000.
- Resolution: 64x64 pixels per frame.

Example Prompts:

- "Circle moving left."
- "Circle zigzagging vertically."

Visuals:

- Add sample images of the dataset showing a few frames.



Data Preprocessing



Process:

- Read image frames and prompts.
- Normalize images to prepare for GAN input.
- Encode text prompts into embeddings using a vocabulary-based encoding.

Tools Used:

- PyTorch for loading and transforming data.

Outcome:

- Dataset ready for training.

GAN Architecture

Generator:

- Combines text embeddings with random noise.
- Upsamples features to generate 64x64 video frames.

Discriminator:

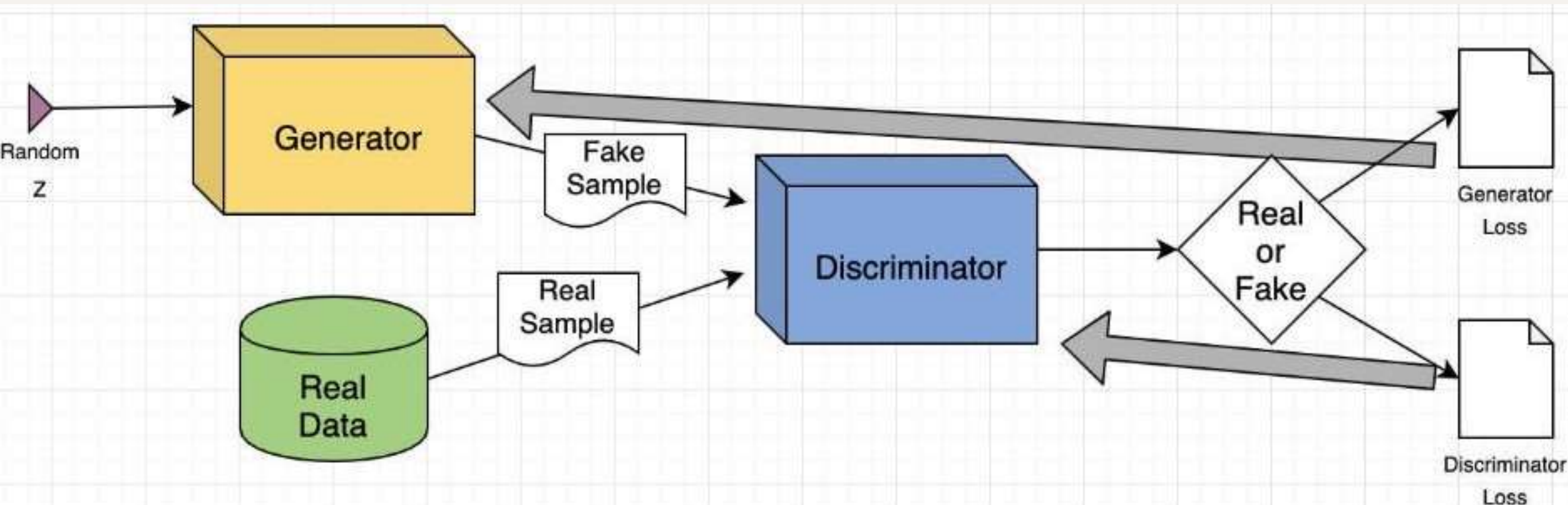
- Distinguishes between real and fake frames.
- Uses convolutional layers for feature extraction.

Text Embedding:

- Converts text prompts into numerical vectors using embedding layers.

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, 4, 2, 1)
        self.conv2 = nn.Conv2d(64, 128, 4, 2, 1)
        self.conv3 = nn.Conv2d(128, 256, 4, 2, 1)
        self.fc1 = nn.Linear(256 * 8 * 8, 1)
        self.leaky_relu = nn.LeakyReLU(0.2, inplace=True)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input):
        x = self.leaky_relu(self.conv1(input))
        x = self.leaky_relu(self.conv2(x))
        x = self.leaky_relu(self.conv3(x))
        x = x.view(-1, 256 * 8 * 8)
        x = self.sigmoid(self.fc1(x))
        return x
```



```
class Generator(nn.Module):
    def __init__(self, text_embed_size):
        super(Generator, self).__init__()
        self.fc1 = nn.Linear(100 + text_embed_size, 256 * 8 * 8)
        self.deconv1 = nn.ConvTranspose2d(256, 128, 4, 2, 1)
        self.deconv2 = nn.ConvTranspose2d(128, 64, 4, 2, 1)
        self.deconv3 = nn.ConvTranspose2d(64, 3, 4, 2, 1)
        self.relu = nn.ReLU(True)
        self.tanh = nn.Tanh()

    def forward(self, noise, text_embed):
        x = torch.cat((noise, text_embed), dim=1)
        x = self.fc1(x).view(-1, 256, 8, 8)
        x = self.relu(self.deconv1(x))
        x = self.relu(self.deconv2(x))
        x = self.tanh(self.deconv3(x))
        return x
```


Training Process

Overview:

- GANs are trained in an adversarial setup: Generator tries to fool the Discriminator.
- Discriminator distinguishes between real and fake data.

Parameters:

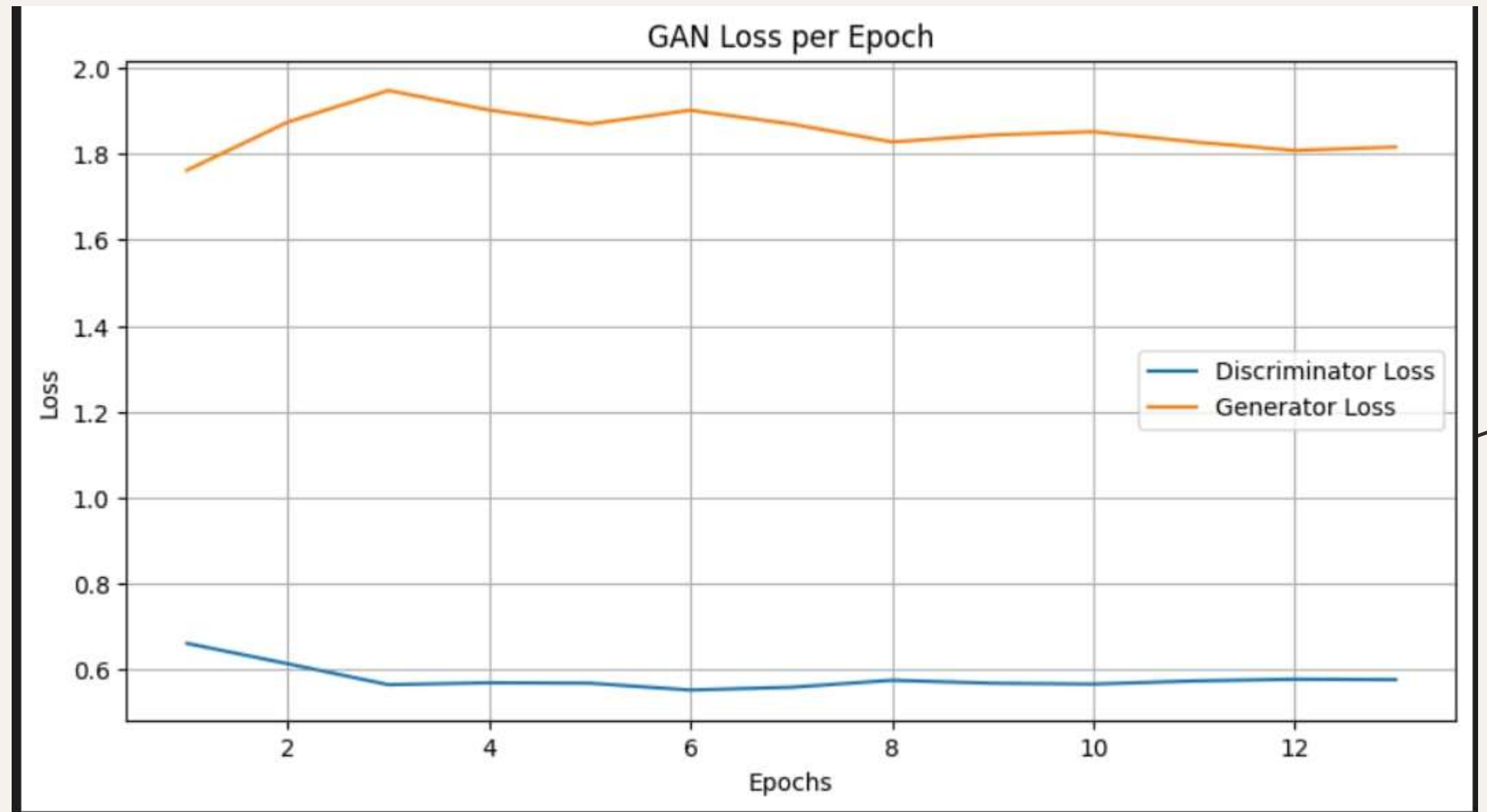
- Batch size: 16.
- Loss function: Binary Cross-Entropy
Loss.Optimizer: Adam with a learning rate of 0.0002.
- Number of epochs: 13.

Training Flow:

- Update Discriminator with real and fake data.
- Update Generator based on Discriminator feedback.



Results



Generated Output

Example Prompt:

- "Circle moving up-right."

Generated Frames:

- Display a few frames from the generated video.



Future Work

Improved Dataset Diversity:

- Incorporate a wider variety of motion patterns, shapes, and objects to increase the model's generalization ability.
- Include more complex video sequences (e.g., multiple objects, interactions).

Higher Resolution Videos:

- Extend the model to generate videos with higher resolution frames (e.g., 128x128, 256x256) for better visual quality.

Advanced Text Embeddings:

- Experiment with more advanced text embedding techniques, such as using pre-trained language models like BERT or GPT, for richer text understanding.

Multi-Object Video Generation:

- Extend the model to handle multi-object video generation, allowing for the synthesis of complex scenes (e.g., animals interacting, people in motion).



Thank You