Text to Image using GANs

CS611 Algorithmic Game Theory - Course Project

Niti Shyamsukha, Vaibhav Singla

OVERVIEW

- Introduction to Generative Adversarial Networks [GANs]
- GANs as a Game Theoretic Concept
- Prerequisites for Building GANs
- Implementation of Single-GAN
- Implementation of GANs [STACK-GAN]
- Conclusions

Introduction to GANs

What is a Generative Model?

- ➔ Generative models can generate new data instances.

  How?

  - By modelling the joint probability of data instances and labels
  - Instead of returning an actual number representing the probability - try to imitate the data; create convincing “fake” data that looks like a part of the data instances.
- ➔ Discriminative models discriminate between two instances.
  - A discriminative model ignores the question of whether a given instance is likely, and just tells you how likely a label is to apply to the instance.

GANs as a Game Theoretic Concept

IDEA Behind GANs

The fundamental idea behind GANs is to train two neural networks, a generator, and a discriminator, in a kind of game or adversarial setting.

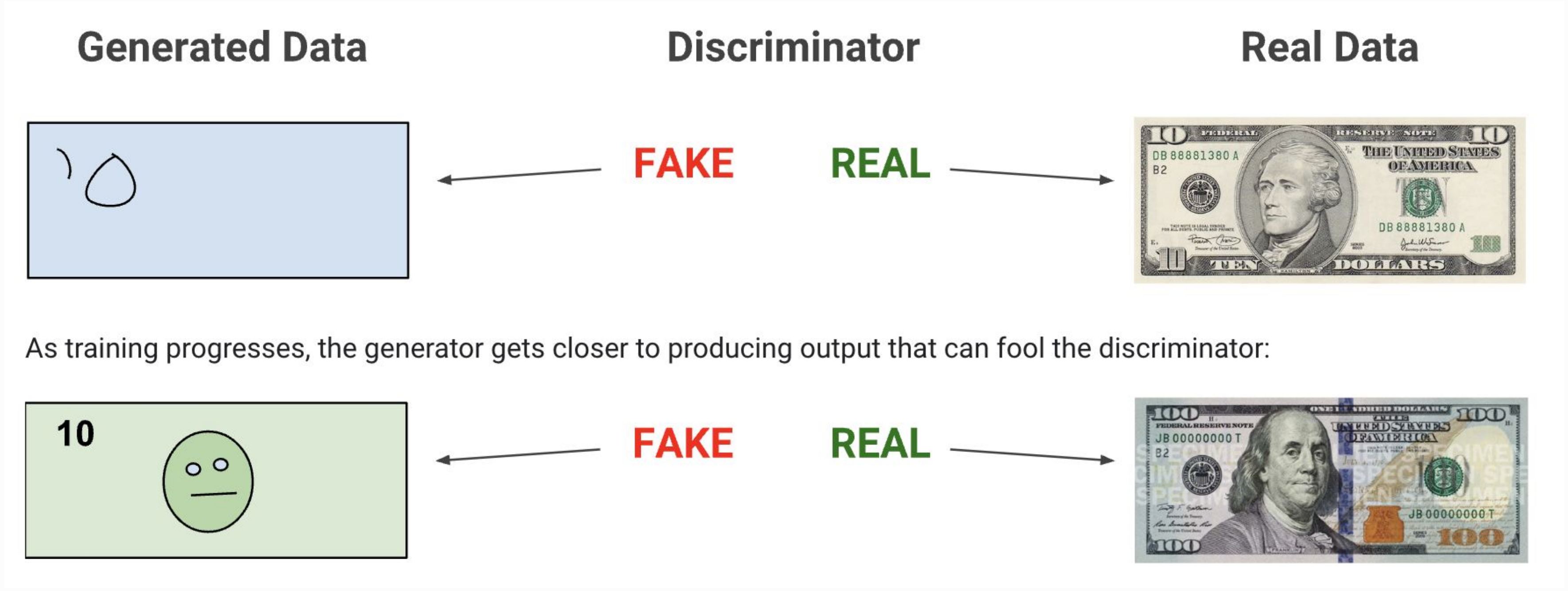
- The generator tries to create data that is indistinguishable from real data. So, the generator learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The discriminator aims to correctly classify real and fake data. The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

IDEA Behind GANs

The training process involves a back-and-forth competition between the generator and the discriminator:

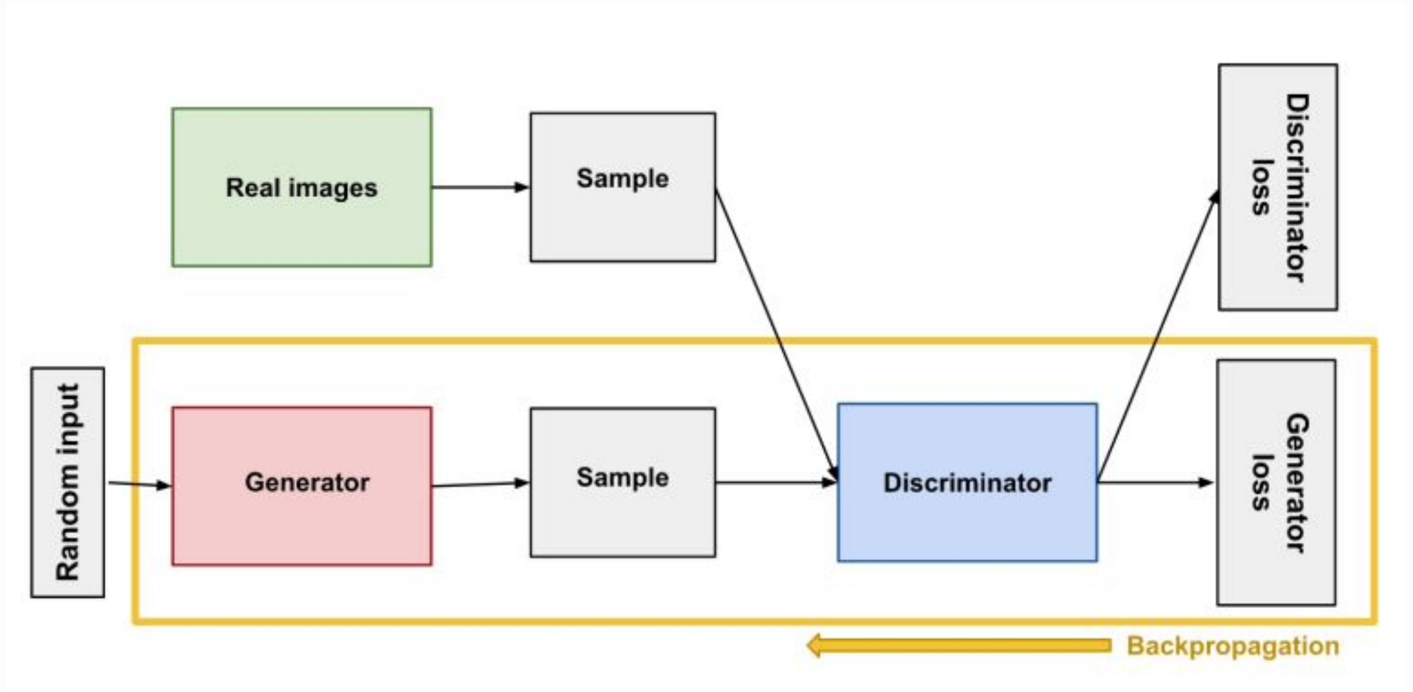
As training progresses, the generator becomes adept at creating realistic data, and the discriminator becomes better at telling the difference between real and generated data. Ideally, this process reaches equilibrium, resulting in a generator that produces high-quality synthetic data.

IDEA Behind GANs



As training progresses, the generator gets closer to producing output that can fool the discriminator:

IDEA Behind GANs



Two-Player Min-Max Game

We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize log(1 – D(G(z))).

Formally, the training procedure is similar to a two-player min-max game with the following objective function,

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

where x is a real image from the true data distribution p_{data}, and z is a noise vector sampled from distribution p_z (e.g., Uniform or Normal distribution). *D(x)* represents the probability that x belongs to the data rather than generator.

[REF: (1, 2)]

Theoretical Results

The generator G implicitly defines a probability distribution p_g as the distribution of the samples G(z) obtained when $z \sim p_z$.

- The minimax game has a global optimum for $p_g = p_{data}$
- Mini-batch stochastic gradient descent training of generative adversarial nets optimizes the Value of the game V(D,G).

Prerequisites for Building GANs

Convolutional Neural Networks

Convolutional Neural Networks are a class of deep neural networks designed for processing structured grid data, such as images and video.

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. These layers apply convolution operations to the input data, using filters or kernels. Convolution involves sliding a small window (filter) over the input data and performing element-wise multiplication and summation. This operation allows the network to detect patterns and features in different spatial locations.

The overall architecture of a CNN typically consists of a stack of convolutional layers, interspersed with activation and pooling layers. This architecture allows the network to automatically learn hierarchical features and spatial hierarchies in the input data

Deconvolutional Neural Networks

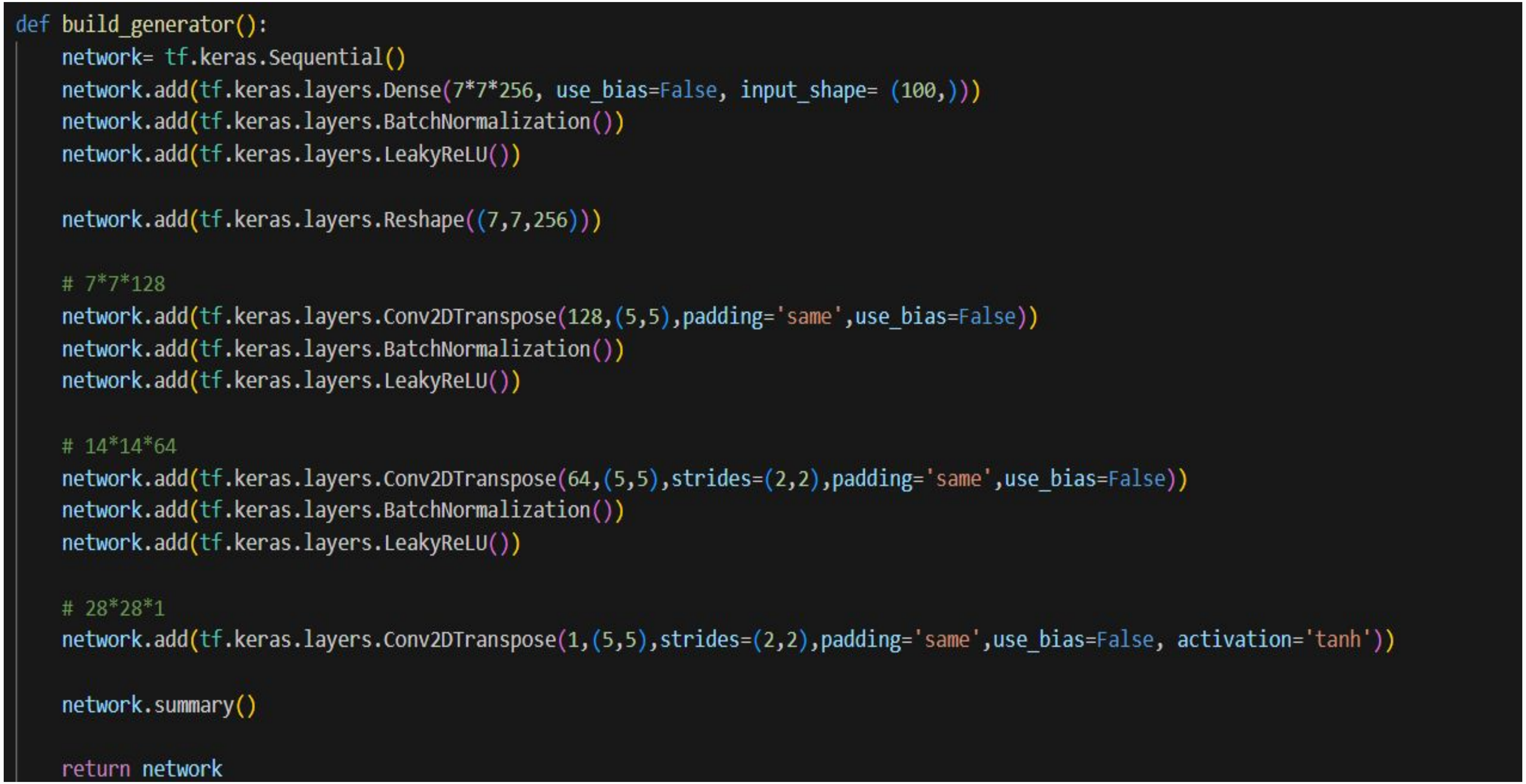
Deconvolutional Neural Networks are a class of neural networks designed to perform the inverse operation of convolutional layers in Convolutional Neural Networks.

Deconvolutional layers are often implemented using transposed convolutions. In transposed convolutions, the operation is similar to convolution, but it involves inserting zeros between the elements of the input and then applying a convolution operation. This process effectively "upsamples" the input.

Implementation of GAN

Generator

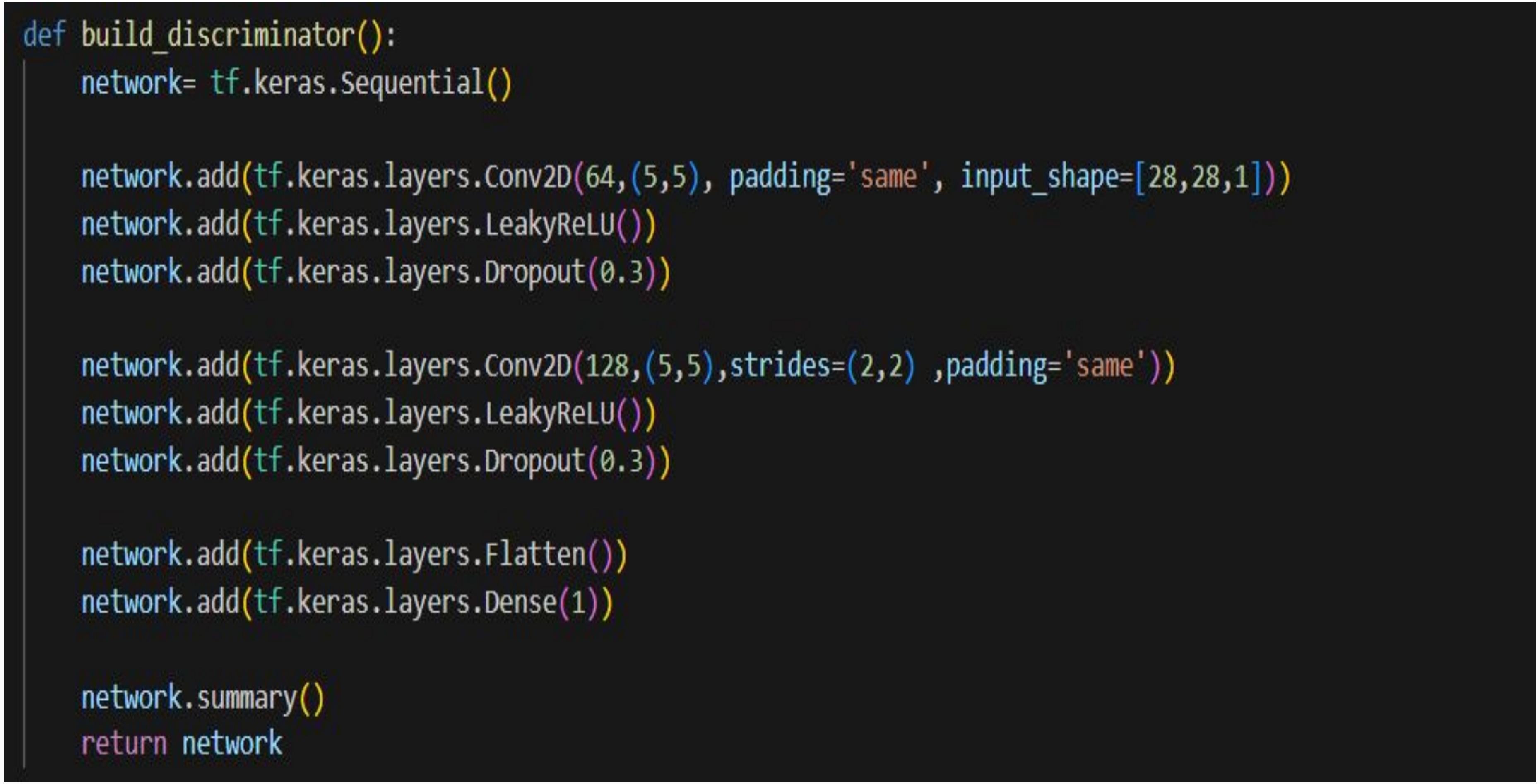
- The generator is responsible for creating artificial data samples, such as images, that resemble real data. It takes random noise as input and transforms it into data that ideally cannot be distinguished from authentic data.
- The generator is implemented as a deconvolutional convolutional neural network. It takes random noise or a seed vector as input and transforms it through a series of layers to produce synthetic data
- The primary goal of the generator is to generate data that is realistic enough to fool the discriminator. It is trained to maximize the probability that the discriminator classifies its generated data as real (assigning a probability close to 1).
- The output of the generator is a synthetic data sample, such as an image. The goal is to make this output indistinguishable from real data when presented to the discriminator.



```python
def build_generator():
    network= tf.keras.Sequential()
    network.add(tf.keras.layers.Dense(7*7*256, use_bias=False, input_shape= (100,)))
    network.add(tf.keras.layers.BatchNormalization())
    network.add(tf.keras.layers.LeakyReLU())

    network.add(tf.keras.layers.Reshape((7,7,256)))

    # 7*7*128
    network.add(tf.keras.layers.Conv2DTranspose(128,(5,5),padding='same',use_bias=False))
    network.add(tf.keras.layers.BatchNormalization())
    network.add(tf.keras.layers.LeakyReLU())

    # 14*14*64
    network.add(tf.keras.layers.Conv2DTranspose(64,(5,5),strides=(2,2),padding='same',use_bias=False))
    network.add(tf.keras.layers.BatchNormalization())
    network.add(tf.keras.layers.LeakyReLU())

    # 28*28*1
    network.add(tf.keras.layers.Conv2DTranspose(1,(5,5),strides=(2,2),padding='same',use_bias=False, activation='tanh'))

    network.summary()

    return network
```

Discriminator

- The discriminator is a neural network responsible for binary classification—determining whether a given input belongs to the real dataset or if it was generated by the generator network.
- Typically a Convolutional Neural Network (CNN), the discriminator takes an input image (either real or generated) and outputs a single scalar value between 0 and 1.
- The primary goal of the discriminator is to correctly classify real images as real (assign a high probability close to 1) and generated images as fake (assign a low probability close to 0).
- The output of the discriminator can be interpreted as the probability that the input image is real. For instance, an output of 0.8 indicates an 80% probability that the input is real.



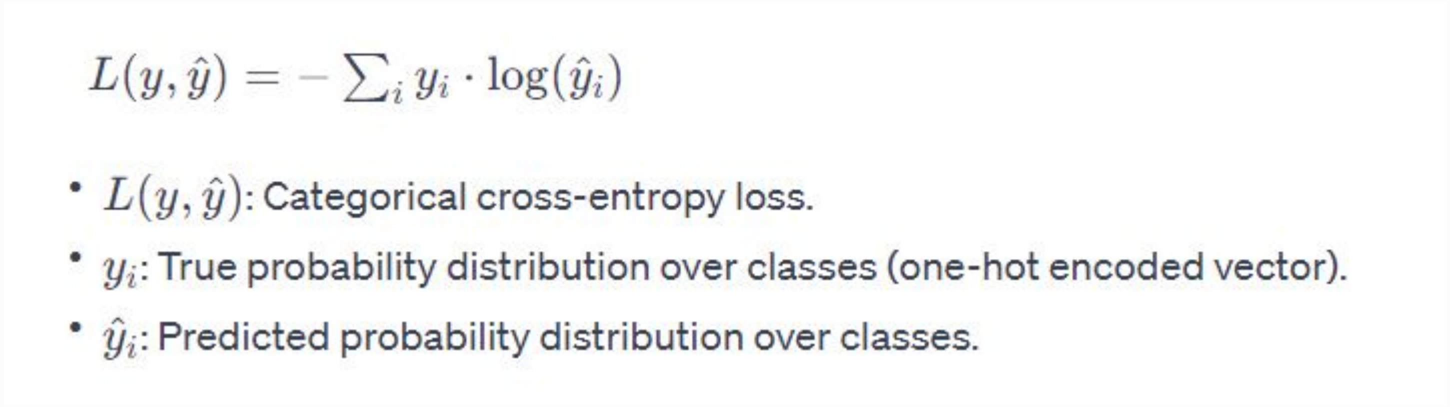
```
def build_discriminator():
    network= tf.keras.Sequential()

    network.add(tf.keras.layers.Conv2D(64,(5,5), padding='same', input_shape=[28,28,1]))
    network.add(tf.keras.layers.LeakyReLU())
    network.add(tf.keras.layers.Dropout(0.3))

    network.add(tf.keras.layers.Conv2D(128,(5,5),strides=(2,2) ,padding='same'))
    network.add(tf.keras.layers.LeakyReLU())
    network.add(tf.keras.layers.Dropout(0.3))

    network.add(tf.keras.layers.Flatten())
    network.add(tf.keras.layers.Dense(1))

    network.summary()
    return network
```

Adversarial Training

- The discriminator's training is intertwined with the training of the generator. As the generator learns to create more realistic images over time, the discriminator adapts to become better at distinguishing real from generated data.
- The adversarial nature of training means that the generator and discriminator are in a constant competition. The generator improves to generate more realistic data, and the discriminator improves to better distinguish real from generated data.
- During training, the GAN alternates between updating the weights of the generator and the discriminator in an attempt to achieve a Nash Equilibrium, where the generator produces high-quality data that the discriminator struggles to differentiate.

Binary Cross Entropy Loss

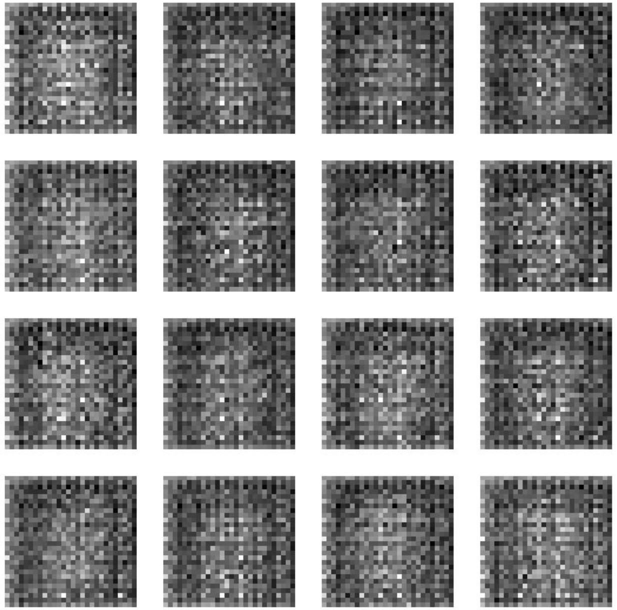
For a binary classification problem, where there are two classes (0 and 1), the cross-entropy loss is calculated as follows:

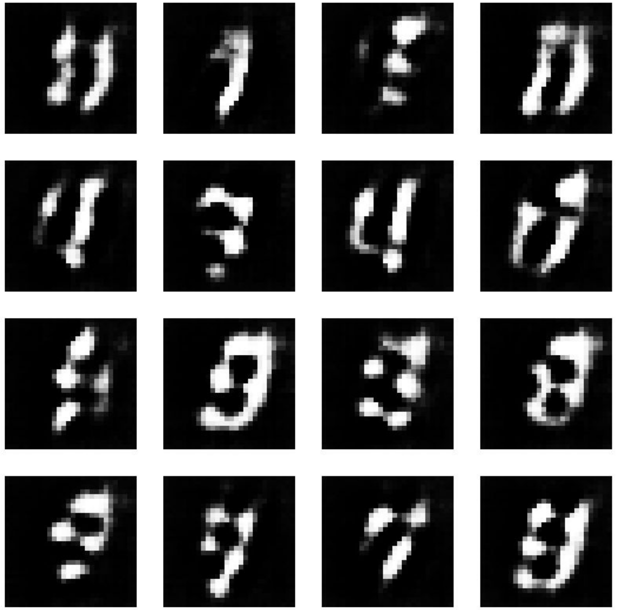


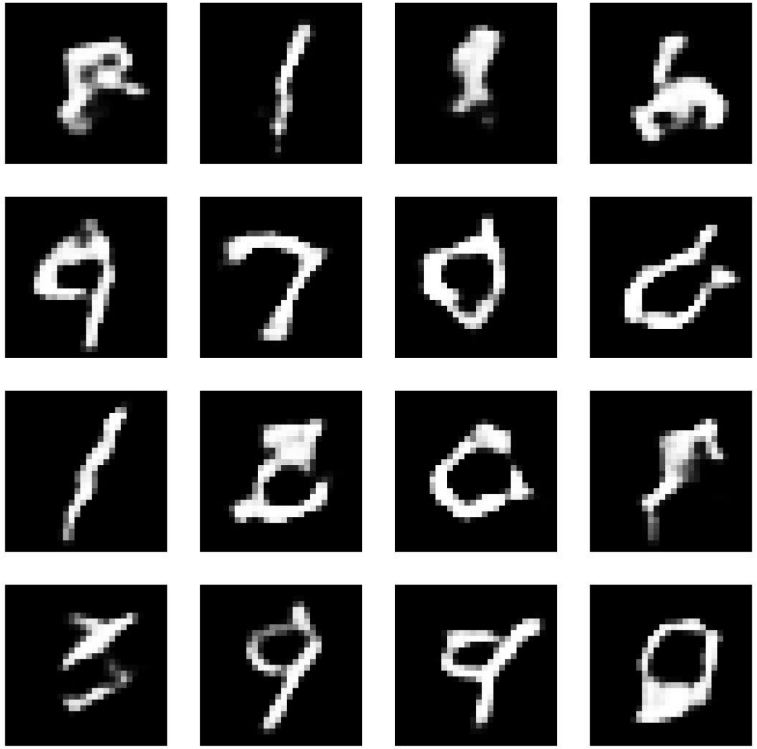
$$L(y, \hat{y}) = -\sum_i y_i \cdot \log(\hat{y}_i)$$

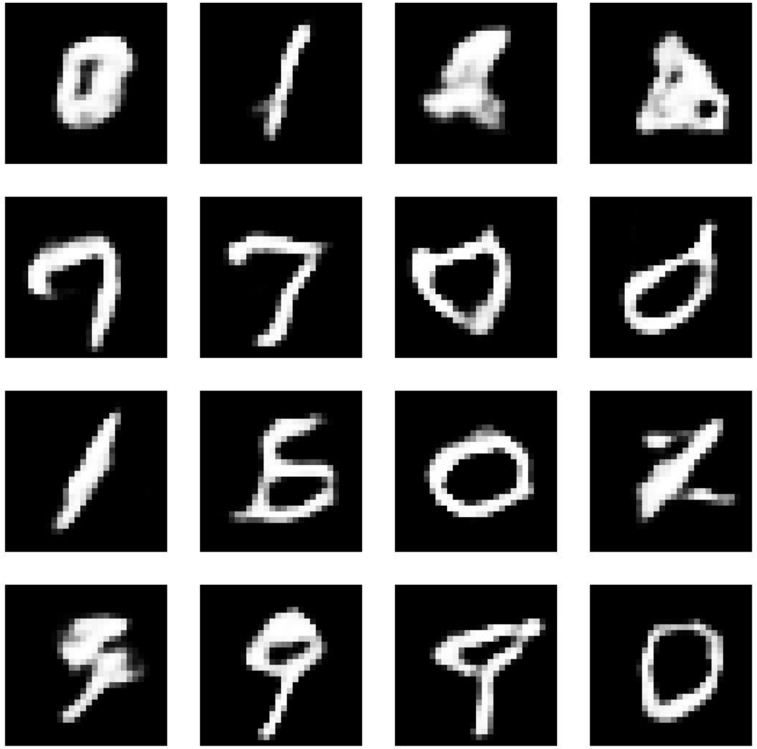
- $L(y, \hat{y})$: Categorical cross-entropy loss.
- y_i: True probability distribution over classes (one-hot encoded vector).
- $\hat{y}_i$: Predicted probability distribution over classes.

Results

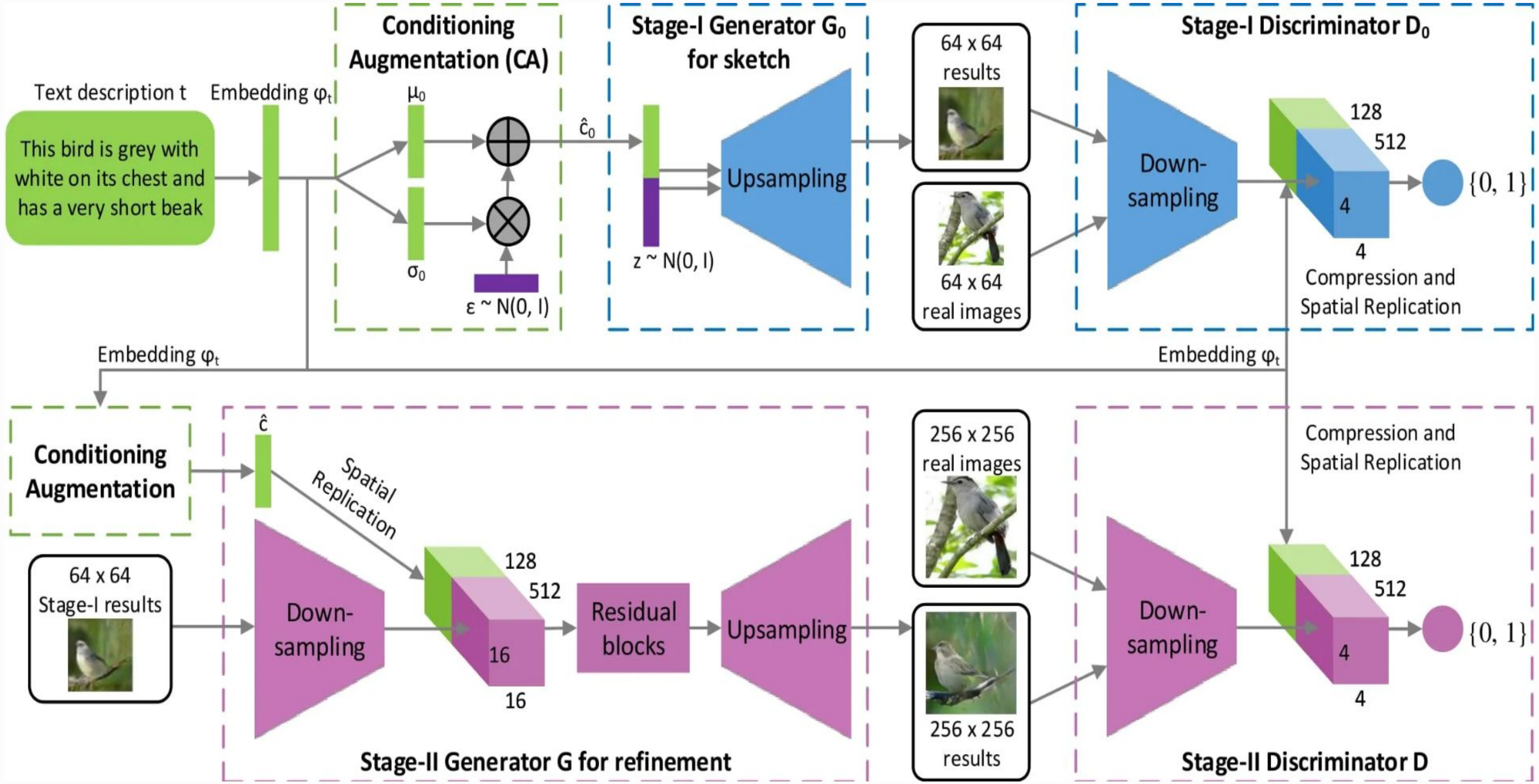

Epoch 1


Epoch 10


Epoch 50


Epoch 100

Implementation of GANs [STACK-GAN]


Text description t
This bird is grey with white on its chest and has a very short beak
Embedding φt
Conditioning Augmentation (CA)
μ0
σ0
ε ~ N(0, I)
ĉ0
Stage-I Generator G0 for sketch
z ~ N(0, I)
Upsampling
64 x 64 results
64 x 64 real images
Stage-I Discriminator D0
Down-sampling
128
512
4
4
{0, 1}
Compression and Spatial Replication
Embedding φt
Embedding φt
Conditioning Augmentation
ĉ
Spatial Replication
64 x 64 Stage-I results
Down-sampling
128
512
16
16
Residual blocks
Upsampling
Stage-II Generator G for refinement
256 x 256 real images
256 x 256 results
Compression and Spatial Replication
Down-sampling
128
512
4
4
{0, 1}
Stage-II Discriminator D

Stage 1 Generator

- In a StackGAN architecture, the Stage 1 generator is responsible for generating a low-resolution image based on a given textual description
- **Text Embedding**: The Stage 1 generator takes as input a textual description of the desired image. This text is often encoded into a fixed-size vector through a text embedding process. The embedding captures semantic information about the input text, enabling the generator to understand the content to be generated.
- **Conditional Augmentation**
- **Generation of Low Resolution Image**: The low-resolution image provides a foundation for subsequent refinement in the later stages of the StackGAN.
- **Upsampling and Refinement**: This involves increasing the resolution of the image and adding more details to create a more realistic and high-resolution final image.
- **Conditional Batch Normalization**: CBN adapts the normalization process to the specific textual condition, helping the generator to better handle different input descriptions.

Conditional Augmentation

- Conditional augmentation is a technique used in StackGANs (Stacked Generative Adversarial Networks) to improve the generation of realistic and diverse images by conditioning the generation process on additional information
- **Addresses Mode Collapse**: It is the situation where the generator produces only a limited variation of images, hence affecting the diversity of the generation. CA mitigates MC and encourages Generator to produce diverse outputs for a single prompt.
- Enhances Realism and Variety
- **Training Stability**: Helps avoid situations where Generator is stuck in producing a limited variety of outputs.
- Improves Generalization in the Model.

Stage 1 Discriminator

- Stage 1 discriminator is responsible for assessing the realism of the generated low-resolution images produced by the Stage 1 generator.
- **Binary Classification**: Performs Binary Classification. Distinguishes between real low-resolution images and images generated by the Stage 1 Generator.
- **Conditional Discrimination**: It is Conditioned on Textual Description used to generate the images. It helps access the realism of images in context of the given text description. It is obtained using Conditional Batch Normalization.
- **Adversarial Training**
- **Guidance for Generator**: Feedback provided by Discriminator guides the training of Generator. The generator adjusts its parameters based on the discriminator's assessment, with the aim of producing more realistic images (better fit for the dataset).
- **Progressive Refinement**

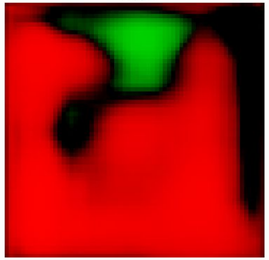
Stage 2 Generator

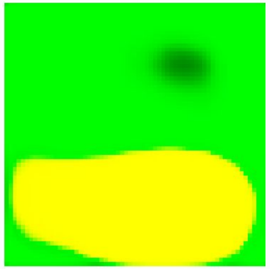
- Stage 2 Generator takes as input the low-resolution images generated by the Stage 1 generator and refines it into high-resolution, more detailed images.
- **Upsampling and Refining**
- **Textual Conditioning**: The conditioning helps align the generated image with the provided textual information, allowing for a more coherent and contextually relevant image synthesis.
- **Conditional Batch Normalization**
- **Feature Fusion**: Uses feature fusion techniques to combine information about the low-resolution image and the conditioning information. It helps ensure that the details added during the upsampling process are consistent with both the low-resolution input and the textual description.
- **Progressive Image Refinement**

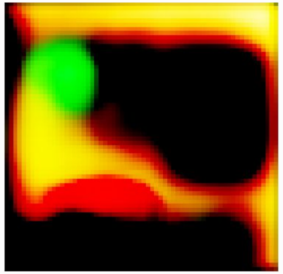
Stage 2 Discriminator

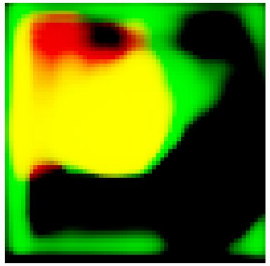
- Stage 2 discriminator is responsible for evaluating the realism of the higher-resolution images generated by the Stage 2 generator.
- **Binary Classification for High-Resolution Images**
- **Conditional Discrimination with Contextual Information**
- **Adversarial Training for Image Realism**
- **Contributing to Progressive Refinement**
- **Feedback Loop for Generator Improvement**

Results

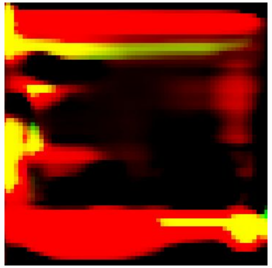

Epoch 1

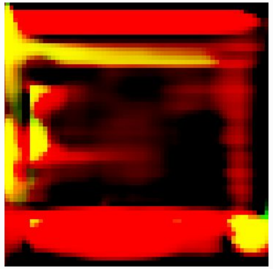

Epoch 10


Epoch 20


Epoch 30

Epoch 50


Epoch 75


Epoch 90

Epoch 100

Conclusion

Conclusion

In conclusion, this project delved into the theoretical foundations, practical implementation [Single GAN], and advanced extensions of Generative Adversarial Networks (GANs) [STACK-GANs].

By dissecting GANs and implementing a Single-GAN, the project laid the groundwork for comprehending the complexities and potential pitfalls in training generative models.

In the realm of STACK-GANs, the study highlighted the scalability and adaptability of GAN architectures for addressing more sophisticated tasks, such as multi-stage image generation. The exploration of STACK-GANs contributed to a broader understanding of the evolving landscape of generative models and their applications.

References

[1] StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, Zhang et al. https://arxiv.org/pdf/1612.03242.pdf

[2] Generative Adversarial Nets, Goodfellow et al. https://arxiv.org/pdf/1406.2661.pdf

[3]https://medium.com/@mrgarg.rajat/implementing-stackgan-using-keras-a0a1b381125e [Slide 25 Reference Image]