DEPARTMENT OF DIGITAL SYSTEMS

NCSR DEMOKRITOS
INSTITUTE OF INFORMATICS AND
TELECOMMUNICATIONS

Deep Learning

GAN: Text to Image

Vasiliki Rentoula mtn2317
Ilias Alexandropoulos mtn2302

Supervisor: George Bouritsas,

Abstract

The scope of this exercise is to create text-to-image (T2I) bird images. For this purpose, we chose the CUB-200-2011p [1] dataset retrieved from Kaggle. We aim to develop a GAN-based model that generates high-resolution, realistic bird images from textual descriptions. The GAN framework is employed, integrating pre-trained text embeddings based on the annotations to guide image generation. Our approach utilizes both a generator and discriminator trained adversarially to improve image realism.

Background

Text-to-Image Synthesis

Text-to-image synthesis is the creation of realistic images that match the input text descriptions. It has many potential applications, such as image editing, video games, and computer-aided design. [3] Recently, text-to-image generation has made huge progress by implementing GANs, which are able to generate realistic images conditioned on given text descriptions. [3]

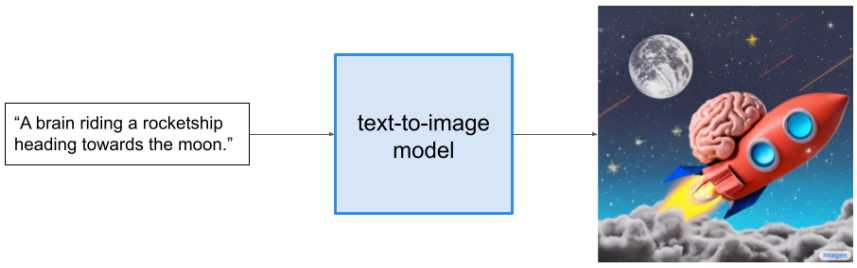

Figure 1: Text-to-Image synthesis

Generative adversarial networks

Generative adversarial networks (GANS) consist of a generator G and a discriminator D that compete in a two-player minimax game. The discriminator tries to divide real training data from synthetic images, and the generator tries to fool the discriminator. [2] (figure 2)

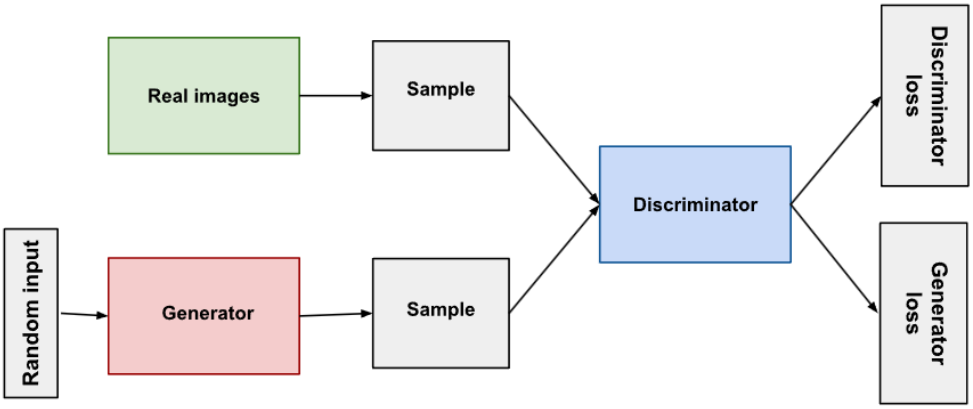

Figure 2: GAN simply architecture

Generator

The generator learns to create fake data by incorporating feedback from the discriminator. The goal is to make the discriminator classify its output as real. [4]

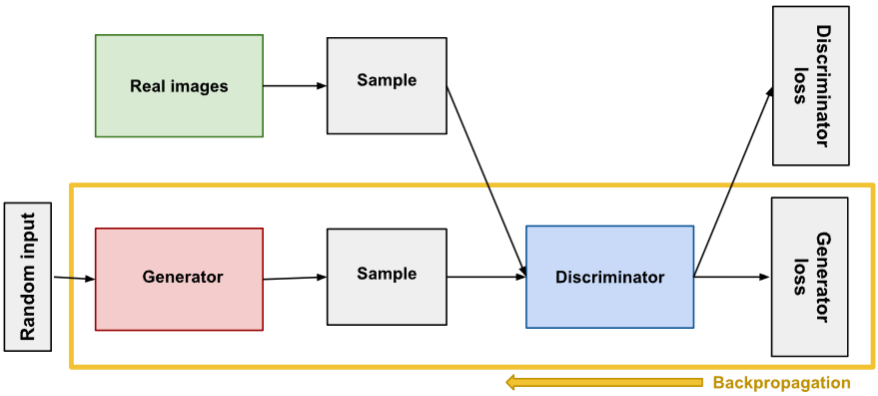

Figure 3: GAN's Generator

Discriminator

The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. [4]

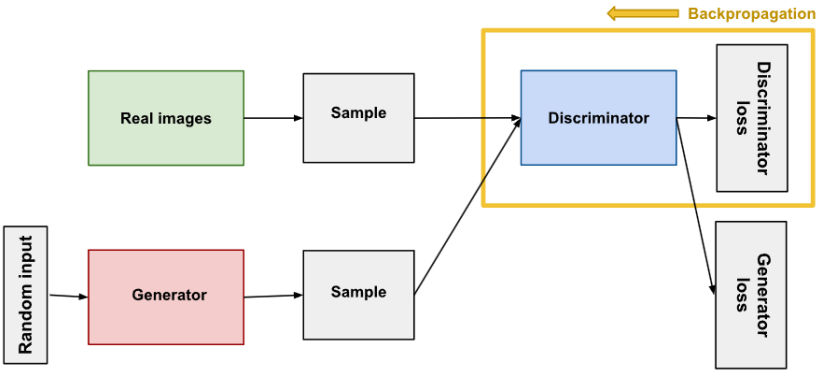

Figure 4: GAN's Discriminator

Implementation

Dataset

For our task, we chose the dataset Caltech-UCSD Birds-200-2011 (CUB-200-2011)[1], an extended version of the CUB-200 dataset, retrieved from Kaggle. It is divided into 200 categories and contains 11,788 bird images. Each image includes detailed annotations, including 15 part locations, 312 binary attributes, and 1 bounding box. We will also use pretrained text s based on the annotations of the images.

Data Prepossessing

After the dataset selection, the next step is the data prepossessing which is important for training our models.

1. **Loading Dataset**
   First, we load the dataset. As we mentioned before, each image is accompanied by text embedding and bounding boxes.

2. **Bounding box extraction**
   Bounding boxes are used to focus the model on the relevant parts of the images, reducing noise and improving the quality of generated images. The bounding boxes are loaded from the corresponding file (bounded_boxes.txt) and mapped to the corresponding image

filenames from the images.txt file. This mapping allows for cropping the images to the regions of interest before feeding them into the model.

3. **Image transformation**
   Next the images are transformed to ensure consistency, suitable for the GAN. The transformations include:

   - Random cropping: Randomly cropping the images to a specified size (64).
   - Horizontal flipping: Randomly flipping images horizontally to augment the dataset and improve the model's generalization.
   - Normalization: Normalizing the images to have pixel values in the range [-1, 1]. This is achieved using the mean and standard deviation values for each color channel.

4. **Text embedding preparation**
   We used pretrained text embeddings, which are loaded from specific files (train/char-CNN-RNN-embeddings.pickle). These embeddings capture the semantic information from the textual description of the images. Note that the text embeddings are split into separate train and test folders. For the purpose of this pipeline, we focus on the train split to train our model effectively.

5. **Data loading and batching**
   Lastly, the processed images and corresponding text embeddings are then loaded into a PyTorch DataLoader, which handles batching and shuffling of the data.We used a batch size of 64.

Model Architecture

For the scope of our excersise we tested three different model architectures. Firstly we examine convolutional neural networks (CNN) to process the input text into a visual representation. Then for the second model architecture we replaced some layers of the CNN with ResBlocks which helps in learning identity mappings that add robustness to deeper architectures. Lastly the third architecture we tested is the addition of the SE blocks, which are employed to adaptively recalibrate the channel-wise feature responses.

1. **CNN implementation:** This is a basic building block (conv3x3) that performs a 3x3 convolution with padding, ensuring spatial dimensions are preserved unless explicitly down sampled. This function is fundamental for both generators and discriminators to process and transform feature maps.

2. **Residual Blocks (ResBlocks):** The residual block helps in preserving the identity of the input, avoiding the vanishing gradient problem in deep networks. Each block performs two consecutive 3x3 convolutions and adds the input directly to the output of these convolutions (residual connection), followed by a ReLU activation.

3. **Squeeze-and-Excitation Blocks (SE Blocks):** Integrated into the network, SE blocks recalibrate channel-wise feature responses by explicitly modelling interdependencies between channels. This mechanism enhances the representational power of the network by allowing it to focus on more informative features, which is especially beneficial in a generative context where every convolutional layer's output can significantly improve the quality of the generated images.

Generator

The generator is denoted as G_0, begins by converting the text description into a text embedding ϕ_t. This embedding is then transformed in order to generate a conditioned latent space where the mean $\mu(\phi_t)$ and variance $\sigma^2(\phi_t)$ are modeled. However, the latent space conditioned on text is usually high-dimensional (> 100 dimensions), having limited data, this can cause discontinuity in the latent data manifold, which is a problem for learning the generator.

To adress this issue, we added a conditioning augmentation technique (figure 5) to produce more conditioning variables for the generator. We randomly sample latent variables from an independent Gaussian distribution $N(\mu(\phi_t), \sum(\phi_t))$, where the mean $\mu(\phi_t)$ and diagonal covariance matrix $\sum(\phi_t)$ (matrxi with $\sigma^2(\phi_t)$ on the diagonal) are functions of the text embedding ϕ_t. The generator combines these conditioned latent variables, with a noise vector sampled from a Gaussian distribution, which then passes through a series of up-sampling blocks to produce a synthetic image.

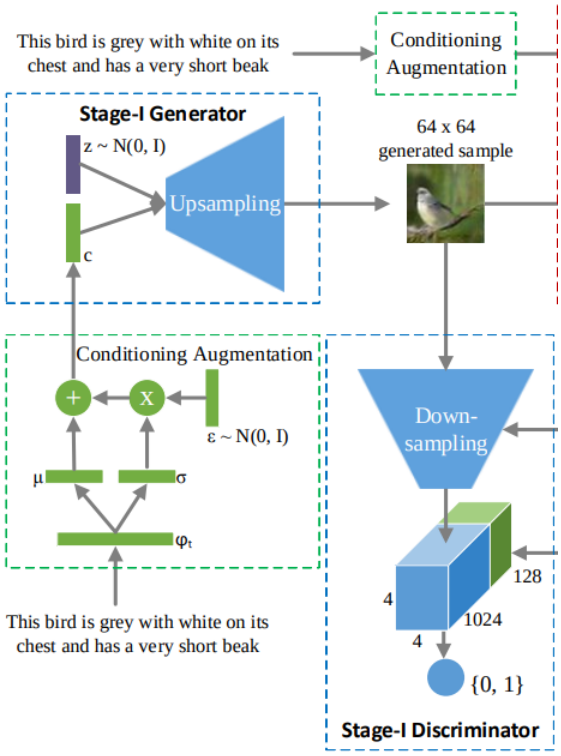

Figure 5: Generator architecture

Discriminator

The discriminator is denoted as D_0, is responsible for categorizing the real images and the generated images from the generator. It uses a convolutional architecture that processes both the image and the conditioned text embedding. The text embedding is compressed and spatially replicated to match the dimensions of the downsampled image features. These combined features are then processed through several convolutional layers to make a binary classification, determining whether the image is real or synthetic.

Training Strategy

We start with initialization by loading our network. Depending on the training model architecture, we load the appropriate generator and discriminator networks. Next the noise and the fixed noise are initialized to generate the synthetic images. Fixed noise is used for evaluating the model progress by generating images that can be compared over different epochs. After that, real labels and fake labels are used to mark the corresponding real and fake images for training the discriminator. Finally, separate Adam optimizes are used for the discriminator and the trainable parameters of the generator with specified learning rates decay parameters.

Training loop

The training loop involves several steps:

1. Learning Rate Adjustment: At the beginning of each epoch, the learning rates for both generator and discriminator are decayed by half every lr decay step epochs to fine-tune the training as it progresses.

2. Batch Processing: each batch of real images and their corresponding text embeddings is loaded. The generator creates fake images using the current batch of text embeddings and noise.

3. Discriminator Training (Update D network): The discriminator's first step is to zero out its gradient by doing so, we ensure that only the gradients from the current batch are used to update the model parameters, leading to a more stable and accurate training process. After that, it computes losses from real images, wrong pairings, and fake images. The discriminator's parameters are updated using backpropagation and the optimizer.

4. Similar to the discriminator we zero out the generator's gradients. The generator loss is calculated using the discriminator's feedback on the fake images and an additional KL divergence loss (kl loss), which penalizes the generator if the learned distribution deviates significantly from the target distribution. This, encourages the generator to produce latent variables (i.e., text embeddings) that follow a target Gaussian distribution (a standard normal distribution with mean zero and variance one). This regularization helps in stabilizing the training and ensuring that the latent space of the generator is well-structured, which can lead to better generalization. The total generator loss is a weighted sum of the GAN loss and the KL loss. The generator's parameters are updated.

Loss functions

Kullback-Leibler Divergence Loss (KL Loss)

The KL divergence loss is particularly important for models that involve a latent space parameterized by a mean μ and a variance $\log(\sigma^2)$, such as a VAEs or GANs that integrate these elements to stabilize and control the output variability. This loss encourages the distribution of the latent variables to approximate a prior distribution, typically a Gaussian distribution. In the context of GANs, it helps in regulating the encoded information, ensuring that the generator does not produce overly divergent outputs that do not align with the training data.

Discriminator loss

The discriminator loss ensures that the discriminator can correctly classify real and fake images and is conditioned properly using the corresponding text or other conditional inputs.

- Binary Cross-Entropy Loss: Used to measure the error between the predicted classes (real or fake) and the true classes.

- Real Loss (errD_real): Calculated between the outputs of the discriminator with real images and the true label ”1” (real).
- Fake Loss (errD_fake): Computed between the discriminator outputs with fake images and the true label ”0” (fake).
- Wrong Pairing Loss (errD_wrong): This is an additional component used in some GAN architectures to penalize the discriminator when it incorrectly classifies real images with mismatched conditions (e.g., wrong text descriptions).
- Aggregate Loss: Depending on whether unconditional logits are used, the discriminator loss might average conditional and unconditional results to balance learning from both conditioned and unconditioned perspectives.

Generator loss

The generator’s loss function is designed to fool the discriminator into thinking that the generated images are real.

- Binary Cross-Entropy Loss: This is also used here, similar to the discriminator, but the target labels for the generated images are ”1” (real), meaning the generator is rewarded for deceiving the discriminator.
- The loss is calculated using the discriminator’s responses to the fake images conditioned on the appropriate embeddings or conditions.
- If unconditional logits are included, their loss is combined to enhance the generator’s ability to produce plausible images irrespective of the conditioning context.

These loss functions collectively help in balancing the adversarial nature of the GAN framework. The KL divergence adds a regularization component that controls the distribution of the latent space, aiding in the stability and coherence of the generated images. The binary cross-entropy components within the discriminator and generator losses ensure that both networks learn accurately and effectively, maintaining a dynamic balance that prevents either network from overpowering the other too quickly, thus fostering sustained learning and improvement throughout training. This structured approach to defining and applying loss functions is critical for the success of complex GAN architectures in producing high-quality, diverse, and realistic outputs.

Configuration

- Latent Space Dimension: Dimensionality of the latent space (input noise vector) = 100
- Image Size: Resolution of the generated images 64x64 for Stage 1 and 256x256 for Stage 2

- Batch size = 32 for the Stage 2 network due to resources limitations and batch size 64 for the rest of the models.
- epoch = 300
- Learning Rate Decay Epoch: Epoch at which the learning rate decays = 300
- Discriminator Learning Rate: Learning rate for the discriminator network = 2e-4
- Generator Learning Rate: Learning rate for the generator network = 2e-4
- KL Divergence Coefficient: Coefficient for the KL divergence loss = 2
- Conditioning Dimension: Dimensionality of the condition vector = 128
- Discriminator Feature Dimension: Base feature dimension for the discriminator = 64
- Generator Feature Dimension: Base feature dimension for the generator = 128

CNN

Generator: This generator starts with a CA_NET module that processes the text embedding to produce a conditioning latent space (mean and variance). This is followed by a series of upsampling blocks, each of which doubles the size of the feature maps, effectively constructing the image in a coarse-to-fine manner. The final output is passed through a 3x3 convolution and a Tanh activation to form the RGB image. The CA_NET handles the text embeddings to produce conditioning variables, allowing the network to better adapt the generation process based on textual input. Finally the upblock utilizes nearest-neighbor upsampling followed by a 3x3 convolution, batch normalization, and ReLU activation. This sequence is crucial for increasing the spatial dimensions of the feature maps while introducing new features.

Discriminator: Comprises several convolutional layers that progressively downsample the input image, extracting increasingly abstract feature representations. The d get logits module computes the final decision of the discriminator, determining whether the input images are real or fake. If conditioned, it integrates text information by concatenating the text embedding across the spatial dimensions of the image features, allowing the discriminator to make more informed decisions based on both visual and textual data.

Training stages

In the figure 5 we have the different train stages for a sample of epoch. We show the output results in the epoch50,150 and 250. As we can see, we have clealer results at the epoch 250.

Figure 6: CNN Train stages

Resblocks

After the preparation of the baseline (cnn implementation), we added the resblocks. Resblocks aids in overcoming the degradation problem typically associated with deeper networks. By adding the input (or a skip connection) directly to the output of two successive 3x3 convolution layers (which include Batch Normalization and ReLU activation between them), it ensures that the gradients flow directly through these connections during backpropagation, limiting the risk of gradient vanishing as the network depth increases.

Advantages of resblocks:

ResBlocks stabilize the training of GANs by maintaining a strong gradient flow across many layers, which is critical in learning detailed and complex patterns necessary for generating high-quality images. By allowing layers to learn modifications to the identity mapping rather than complete transformations, ResBlocks help the network to focus on learning fine details that are crucial for the quality of the generated images, while preserving the overall structure conveyed by earlier layers.

Also, the training of this network was at least 2 times faster than the CNN one which was expected.

Training stages

In this training we can see that we have clearer results than the cnn even from the epoch 150.

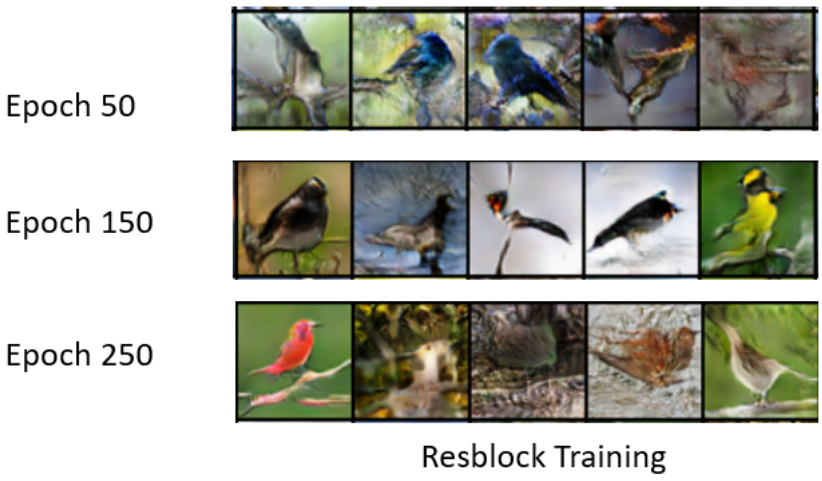

Figure 7: Resblock Training

Stage 2 Resblocks

The Stage 2 generator takes the output from Stage 1 as its input, enhancing the image quality by adding detailed textures and correcting any inconsistencies present in the first stage's output. This process is crucial for achieving high-resolution and photorealistic images.

The hr_joint part of the network combines the upsampled low-resolution image from Stage 1 with a conditioned code derived from the text embedding. The conditioned code is generated by the CA_NET, which modulates the textual information into a dense representation suitable for guiding the image refinement process.

Next the resblocks are used extensively to stabilize training and allow deeper networks by mitigating the vanishing gradient problem. In Stage 2, they help in refining details by allowing the network to adjust its parameters slightly based on the residual errors from the previous layers.

Following the application of residual blocks, multiple upsampling layers are employed to increase the resolution of the image progressively. Each upsampling block doubles the size of the feature maps, effectively enhancing the detail at each step.

The final image is created using a convolutional layer followed by a Tanh activation to ensure the pixel values are normalized between -1 and 1, suitable for image data.

Training stages

In this training we can see that we have clearer results after the epoch 120.

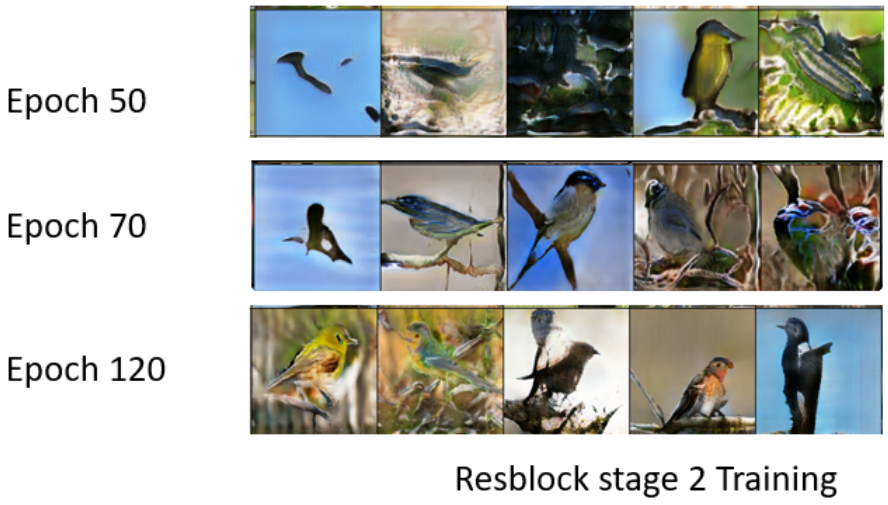

Figure 8: Resblock stage 2

Squeeze-and-Excitation Blocks (SE Blocks):

In this architecture, we enchanced our model by adaptively recalibrating channel-wise feature responses. It operates in two stages:

- Squeezing: Global spatial information of a channel is compressed into a single numerical value (using adaptive average pooling), capturing global distribution of features.
- Excitation: Through a gating mechanism (two fully-connected layers with a ReLU and sigmoid activation respectively), this scalar is transformed to capture channel-wise dependencies, and is used to rescale the original feature maps.

Advantages of SE Blocks:

The SE Blocks allow the network to adaptively adjust the weighting of each channel's feature maps based on the global information, leading to a more dynamic and effective feature representation. This is particularly beneficial in GANs where distinguishing subtle yet crucial patterns can significantly impact the performance of both the generator and discriminator.

Additionaly they focus on informative features, by explicitly modeling interdependencies between channels, potentially enhancing the detail and realism of the generated images.

Training stages

In this training we can see that we have clearer results than the cnn even from the epoch 150.

Figure 9: SE Blocks Training

Results

In this section we made a comparison of our methods on the CUB dataset with representative examples generated by text descriptions by different methods.

Figure 10: Models Resutls

Evaluation Metrics

We calculated the inception score for each model for the generated images from the test embeddings. Although, inception score does not reflect whether the generated images are well conditioned on the given text descriptions we added a 3-different human evaluation for 50 images for each model. The table 1 shows the inception score and the human score (average).

Model	Inception Score	Human Score
CNN	4.74 ± 0.54	0.3
Resblock	4.61 ± 0.52	0.32
SE block	3.51 ± 0.46	0.22
Stage 2	4.85 ± 0.51	0.4

Table 1: Evaluation Metrics Table

Future Work

This project demonstrates the potential of GAN-based models in generating high-resolution bird images from textual descriptions. However, there are several areas for future improvement and exploration to enhance the performance and applicability of our models.

Starting with the model architecture a next step could be integrating more advanced network architectures, such as Transformer-based models or hybrid architectures combining convolutional and attention mechanisms, could potentially improve the quality and diversity of the generated images.

Also, expanding upon the multi-stage generative approach to include more stages GANs might result in even higher resolution and more detailed images.

Finally, employing more advanced conditioning techniques that can better capture the distinctions of textual descriptions and their relationship to image features could enhance the ”accuracy” of the generated images.

Conclusion

In this project, we developed and evaluated a GAN-based model for generating high-resolution bird images from textual descriptions using the CUB-200-2011 dataset. Our approach included several advanced techniques, including the use of pre-trained text embeddings, conditioning augmentation, and various architectural enhancements such as residual blocks (ResBlocks) and squeeze-and-excitation blocks (SE Blocks). Despite the success, our models exhibited some limitations. The generated images occasionally lacked finer details and exhibited artifacts, especially in more complex or ambiguous descriptions. Additionally, while the Inception Score provided a quantitative measure of performance, it did not fully capture the quality of text-image alignment, highlighting the need for more comprehensive evaluation metrics.

References

[1] WeNewOne. (2011). CUB-200-2011 Dataset. Kaggle.

[2] Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H. (2016). Generative Adversarial Text to Image Synthesis. In Proceedings of the 33rd International Conference on Machine Learning (Vol. 48).

[3] Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., Metaxas, D. (2019). StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks. In Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019).

[4] Google Developers. (n.d.). Generative Adversarial Networks (GANs): Generator.