REPORT ON TEXT TO IMAGE GENERATION

AIM :

Developing a text-to-image generative model that requires a combination of advanced natural language processing (NLP) and computer vision techniques.

DIFFERENT WAYS:

Model Architecture

1. Transformer-based NLP Models:
2. Attention Mechanisms:
3. Generative Adversarial Networks (GANs):
4. Diffusion Models:

KERAS vs TORCH

I have used pytorch and it is easily compatible and flexible with the python version and other packages including tree-npm.

While using keras I encountered a couple of compatibility errors.

APPROACHES I TRIED

1. Using GANs

Generative Adversarial Networks (GANs) are a powerful approach for generating images from textual descriptions. In this setup, a conditional GAN (cGAN) is used, where the generator network takes encoded text as input and produces

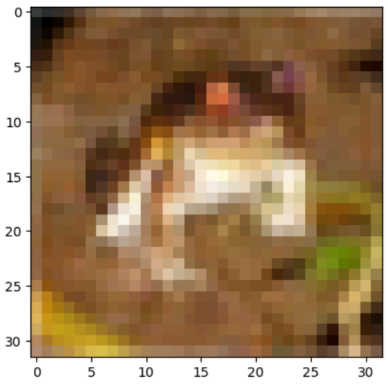
corresponding images. The discriminator network, trained simultaneously, evaluates these images to distinguish between real and generated ones, providing feedback to the generator to improve its outputs.

Failures:

1. Generator was using limited variety of images
2. I couldn't capture the diversity of possible outputs from different textual descriptions.
3. Generator and Discriminator were not giving good quality images. It led to poor-quality image generation.
4. Couldn't train it using large dataset due to very slow CPU processor speed . Had tried on google colab with reduced epochs but it gave very blurred results.
5. Oscillatory behavior in Generative Adversarial Networks (GANs) a situation I encountered during the training process where the performance of the generator and the discriminator fluctuates cyclically without converging to a stable equilibrium.

Providing the link to GAN approached :
https://colab.research.google.com/drive/1qMQYO1SzRLolx_9D7QhfTovzUkgYWQ7N?usp=sharing

Results: (Not the finals one)



2.Using Stable Diffusion Models for Text-to-Image Generation

Stable diffusion models offer a simpler and more stable training process, exhibit remarkable capability in generating high-quality images, and provide a more straightforward mechanism for guiding image generation with textual descriptions.

Simplicity and Stability in Training

One of the primary reasons for choosing stable diffusion models is their simplicity and stability during training. Unlike GANs, which require careful balancing between the generator and discriminator to avoid issues like mode collapse and training instability, diffusion models operate on a fundamentally different principle. They start with noise and iteratively refine the image through a series of denoising steps. This iterative refinement process is inherently stable, as each step incrementally improves the image quality, guided by the underlying diffusion process. Consequently, diffusion models are less prone to oscillatory behavior of GANs.

High-Quality Image Generation Along with ease of text integrations

Stable diffusion models excel in generating high-quality images, particularly when guided by textual descriptions. By

leveraging a series of learned denoising autoencoders, these models can produce detailed and coherent images that faithfully represent the input text. The diffusion process allows for fine-grained control over the generation steps, ensuring that the images gradually evolve to match the semantic content of the description. This capability is crucial for applications requiring precise and accurate image synthesis from complex textual inputs, making diffusion models an excellent choice for such tasks.

- - Straightforward Integration with Textual Descriptions
- - Natural Incorporation of Conditioning Information.Diffusion models can seamlessly integrate text embeddings at each denoising step.
- -Mechanisms for Integration
- Cross-Attention: The model attends to different parts of the text while refining the image.
- Self-Attention: The model focuses on various aspects of the input text during the denoising process.
- - Continuous Guidance: Ensures the generated image remains aligned with the textual description throughout the diffusion process.
- - Simplified Architecture:
- - Ease of Text Conditioning: Simplifies the model architecture.
- - Enhanced Detail CaptureImproves the model's ability to capture intricate details specified in the description.

- - Accurate and Visually Appealing Outputs: Leads to more precise and aesthetically pleasing images that match the input text.

Parts Of Stable- Diffusions:

1. A Generative Model that generates new images by using the known ones like we want a different set of images of dinosaurs given the existing ones , or a new set of images.

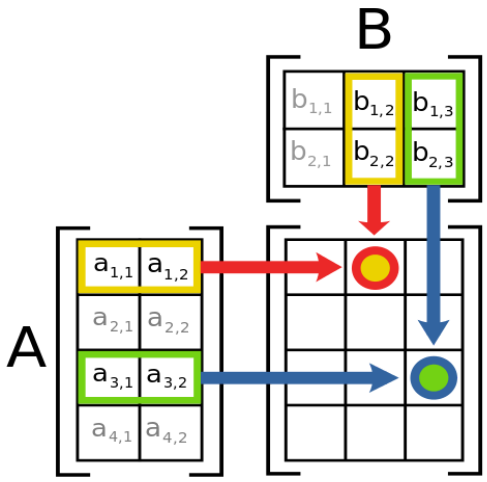
2. While training , in the first iteration we train without any textual descriptions and in the second one we do the training with textual descriptions.

3.We know how to add noise to the images and denoising it is what we look forward to .The image must match the textual description.I used the neural network to refine this process.
I am providing variance and letting the network identify meaning given the noising and denoising parameters (alpha and beta)
The prompt provided by user allows the model to able to understand the amount of noise to be remove in order to move towards image generation process.

I used a classifier free guidance type of model
I train the model giving prompt once and without giving the prompt , so that it would be able to generate new images.

Output = weightage x (promptprovided - noprompt) + noprompt ;

So next step arises is how will the model understand fro the text; Hence a vector representation of texts is used
CLIP - Build by OpenAI
Trained by image associated with texts



Lets say that A represents text and B represents corresponding images

MISTAKE

Basically we compress images of bigger size then learn on this images like perform noising and stuff and then we again convert them into original steps
Initially I performed the conversion of image-compression then to text and expansion back to images using autoencoder. But here there were not measurable differences between the code/vector representation of the text and hence the results were not quiet aligned with prompt.
Amount of noise added is very crucial as if you are adding high noise there is large amount of freedom in the image

generated while adding a very small amount of noise that would lead to not much creativity of the text prompt.
Channel (size , height , width)

Encoder

Used for compressing the images and adding features.
Done using convolutional networks (add suitable striding , padding along with reducing dimensions)
When calculating the deviation noise is fed suitably according to the needs.

Noise Addition:
Gaussian Noise: The noise is added to the original image, and `np.clip` ensures pixel values stay within the valid range
Salt and Pepper Noise:function adds salt and pepper noise by randomly selecting pixels to turn completely white (salt) or black (pepper). The amount of noise added is controlled by the `amount` parameter, which specifies the probability of adding salt or pepper to each pixel.
I am using gaussian noise as i dont want a very random distribution.This will make it difficult to identify noise addition

Decoder

Used for converting back to the image form(RGB) channels

1. **SiLU Activation_Purpose**: SiLU (Sigmoid Linear Unit) is a non-linear activation function that enhances feature representation capability compared to traditional sigmoid or tanh functions.

- **Implementation**: Applied after Group Normalization in each residual block and before the final convolutional layer, SiLU introduces non-linearity, allowing the network to learn complex patterns and representations from the normalized features.

References from : [Variational-Autoencoder/Variational Autoencoder/train.py at main · vignesh-creator/Variational-Autoencoder · GitHub]

To create residual blocks as required

Normalization : (Batch , Group, Layer)

Done to prevent oscillatory behavior . I have used batch normalization so that the features close to one another are more related does resulting in more reasonable outcome

Attention Models ;

- Query
- Key
- Value

1. Softmax is a crucial component in the attention mechanism, transforming the raw attention scores into a probability distribution that weights the values thus providing us the significance.

2. Self Attention: Scalar product of query and key used for the words of the text fed that is intratext scalar products are calculated and then assigned weightage using softmax techniques
3. Cross-Attention:Scalar product of query and key used for the words of the text fed that is intertext scalar products are calculated. Between the generator and discriminator
4. So both of this mechanisms are used combinedly

Embedding

Conversion of unique tokens to dense vectors.Converts tokens to long type.Passes tokens through the embedding layers and then normalizes it. Tokens converted to long long to prevent overflow

UNET-residual block : Process the inputs applied by applying time -features.original input with the processed output through residual connections meaning the features that are extracted from input - spatial orientation of pixels and defining padding , striding.

CLIP Embeddings

The training objective of CLIP is to learn a shared embedding space where images and their corresponding captions are close to each other, while unrelated image-caption pairs are far apart. This is achieved using a contrastive loss.

The contrasive loss is used in such a way that our text prompt is divided into classes and their inter relations between the embeddings are seen

Sampler

Now moving on to remove noise .

Using Markov Chain used to remove noise.There are Beta and alpha values according with the noise addition and removal.

Similar to backtracking the process. For each timestep, a time embedding is generated, and the diffusion model predicts noise which is used to update the latents. If classifier-free guidance is used, the model input is duplicated and the guidance is applied to the model output.Basically using generator the images are compressed along with noise adding and features adding. Along with this word to vector embeddings are also added.It handles tokenization, context embedding, latent space initialization, the diffusion process, and decoding to produce the final image. Various configurations like classifier-free guidance, input image conditioning, and device management are supported.

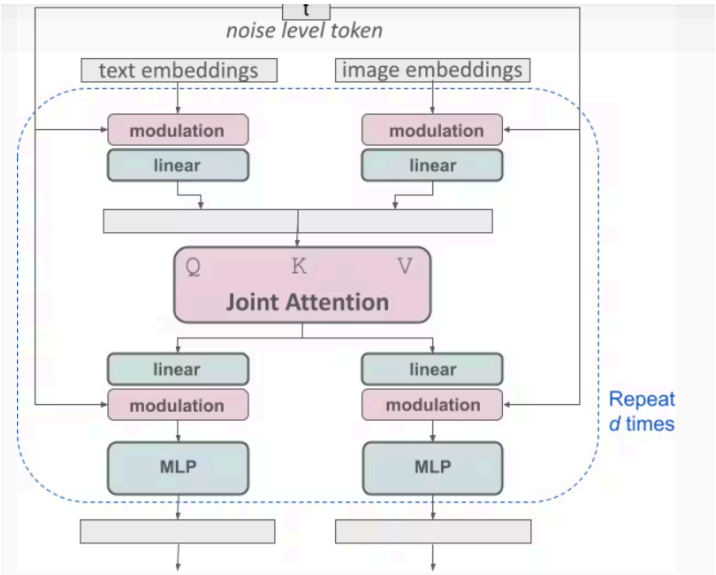
There are different features a user can use like choosing own iteration steps along with using a classifier free guidance model. Few of the values are taken in reference with historical considerations as in said they are good in practice.

Then we load model and then use instances of class and check requirements by ensuring a boolean type

```
static_diffusion.load_state_dict(state_dict['
static_diffusion'], strict=True)
```

Like we created an instance of a class and used its function given the parameters required are matching

Ensuring each model is moved to correct classes and functions step by step along with providing weights ready for further training



This provides a brief idea about the embeddings of the text

And image.

The `load_from_standard_weights` function loads specific weight tensors from a saved PyTorch model checkpoint file (`input_file`) and organizes them into a structured dictionary. It extracts weights corresponding to different modules of the original model, such as

`diffusion`, `encoder`, `decoder`, and `clip`, and maps them to keys within the `converted` dictionary. This function is useful for adapting pretrained model weights to fit a different model structure or for further analysis and manipulation in machine learning tasks.