T2I with an quantitative approach

Ankit Kothari; Sandeep Polavarapu Venkata Naga; Abhishek Kotcharlakota; Isha Asalla

ABSTRACT

The Text to Image conversion problem has seen promising results with the extent of improvements with Neural networks, especially with GANS and diffusion models. The state-of-the-art Text Image generation models include Dall-E, Lafite, and Imagen. We propose a text Image generation framework that caters to numerical data in the input prompt. This will be an improvement over the existing Text to Image(T2I) frameworks' performance in recognizing quantitative attributes associated with an object and generating images that satisfy the number in the text description. For instance: the DALL-E network produces distorted and confused images for an input "4 cats" or "3 oranges". Our project's objective is to address this concern by adding a realistic understanding layer over the text to latent embedding space that would add the essence of quantity to the image generator (GAN).

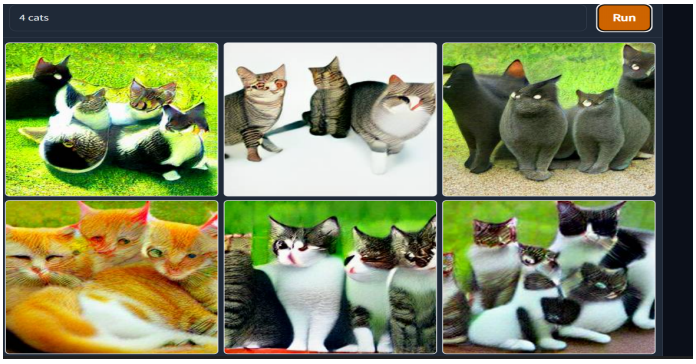


Figure 1: Distorted images generated by Dall-E mini (existing SOTA text to image framework) with "4 cats" as an input prompt.

DATA

The implementation of the aforementioned framework demands a vast dataset with a wide range of images belonging to multiple categories. Additionally, we would also require corresponding text annotations that can be used to match the images while training the model.

COCO Dataset

COCO Dataset is a popular large-scale dataset that comprises 330,000 images with almost one and a half million captions. The COCO dataset provides five captions per image. Some captions have object count, while some are general statements that only identify the object, such as "cats in a park" for an image with two cats. The source for this dataset is provided under the **references section,* which provides the downloadable images and text annotation files individually.

The goal of our project is to reduce the ambiguity so that text-to-image generation is more apparent when the prompt is “Two cats in a park.” by encoding this information into the Textembedding.

Here are a few examples of Images and their captions:

***Figure-2**: Image from COCO Dataset with text annotations*
Captions: A raft full of dogs riding the wave,

1.'a raft full of dogs is riding a wave'
2.'A raft full of different size dogs in the ocean.'
3.'Dogs of all sizes ride on a raft in swift moving water.'
4.'a group of dogs in lifejackets on a raft riding a wave in the water'
5.'Dogs in life jackets ride a board over waves’

Figure-3: Image from COCO Dataset with text annotations
Captions:

1.*‘’A large group of elephants crossing a dirt path.',*
2.'A baby elephant walks through the forest with five adult elephants.',
3.A herd of elephants one running with a baby.',
4.'A herd of elephants, including a tiny baby elephant, cross a dirt road.',
5. 'Four large elephants are watching over their baby in the wilderness.'

***Figure-4:** Image from COCO Dataset with text annotations*
Captions:

Count 3

1.‘Three zebras standing in a grassy field walking',
2. 'Three zebras are standing in an open field.',
3.'Three zebra are walking through the grass of a field.',
4.'Three zebras standing on a grassy dirt field.',
5.'Three zebras grazing in green grass field area.'

TALLYQA Dataset:

TallyQA uses Question Answering and is based on an open-ended counting algorithm. These contain simple “How many…?” questions that help understand the object recognition ability of the object detector. Further to test the model’s grasp of the relationship between various kinds of objects present in the image, complex questions are framed. Hence TallyQA dataset is a composite of simple and complicated questions that test the depth of object detection.

Here’s an example of how QA works with the images:

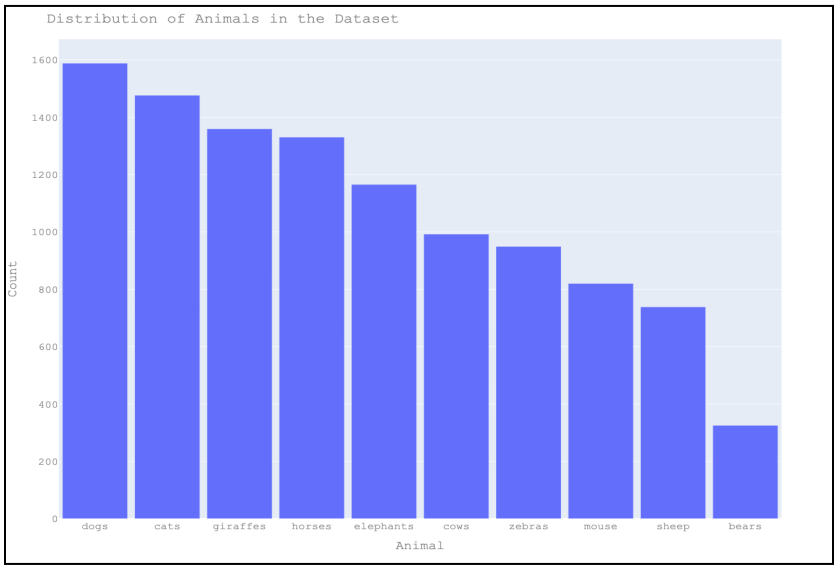
Figure-5: Demonstrating Visual Question Answering on Image from COCO with Tally QA

Question from TallyQA: “How many” dogs? So using the additional information from Tally QA we add an additional label with 8 dogs which is the answer.

DATA PREPARATION:

Each image of the COCO dataset has a label, ImageID, and other properties. However, the count attribute of the object in the image is not considered. The TallyQA dataset has an image_id that can be used to reference an image from the coco dataset. TallyQA contains questions that are framed on the object kind and count. The TallyQA - COCO Data has 135000 rows. For the scope of this project, we only focus on the animal category of the Dataset.

Figure 6 : Distribution of animal images in the Dataset



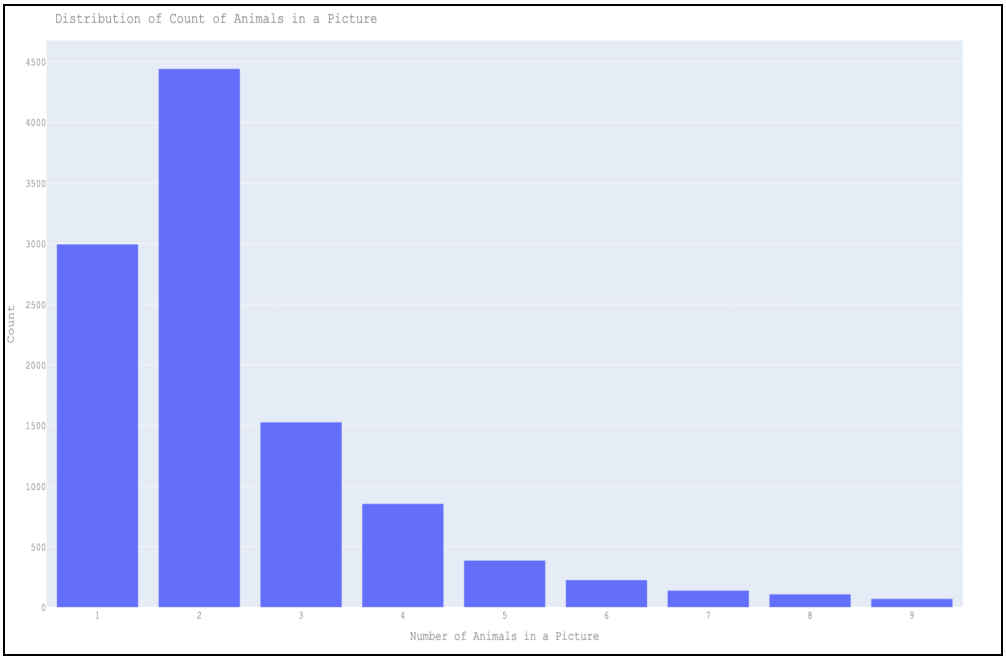
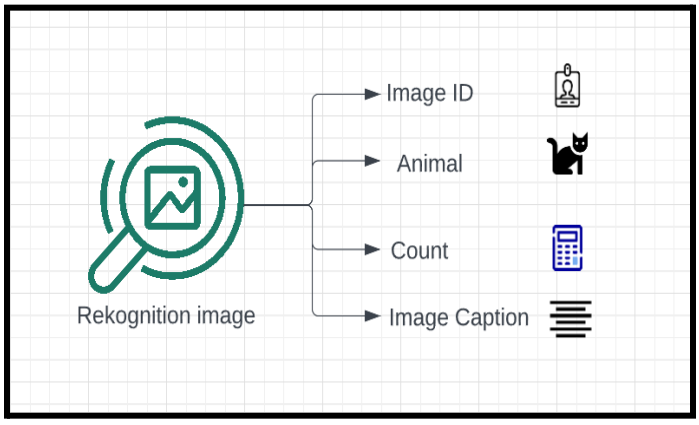


Figure 7: Distribution of animals in an image of the Dataset

TallyQA has the following information pertaining to each image - Image, count, and animal species.

However, the TallyQA data does not have the captions from the original COCO Annotations dataset. Hence we join the TallyQA and COCO with the image id as the primary key.



The final desired dataset consists of the following four labels for each image matched with COCO: image_id, caption, count, and animal_name for over 11000 images. We also filtered thee imaged with count between one and nine. This data for each image will be fed in the form of two sentences to a sentence encoder to get the joint embeddings. The first sentence is the image caption while the second sentence consists of the animal in question along with the count.

Figure -8 : Final Data Format

Text Embeddings

Text or word embeddings are vector representations of words in lower dimensional space. BERT embedding system provides numerical representations of words from the text input with

contextual meaning captured. This would help the model identify between objects with the same word but different meanings. (one popular example being: “bank”). These sentence embedding vectors are then plugged into our GAN model.

The inputs to the BERT Model after tokenization look like this: “*[CLS] two cats sitting in a dark room filled with chairs stacked on top of each other. [SEP] 2 cats [SEP]*”. This data is passed through the pre-trained sentence transformer **‘all-MiniLM-L6-v2’** using the [SEP] token.

Extracted the embeddings of the [CLS] token as the final text representation of the image. The total size of the data is 11000 images with 384 embedding sizes.

Few parameters of the pre-trained sentence transformer are listed below:

Parameter	Value
Transformer model	all-MiniLM-L6-v2
architecture	BertModel
Hidden_activation function	geLu
hidden_dropout_prob	0.1
hidden_size	384
intermediate_size	1536
max_position_embeddings	512
transformers_version	4.20.1
vocab_size	30522
position_embedding_type	absolute

Table -1: Parameters for Sentence Transformer Configuration

Data Augmentation:

By producing altered images of the original photos, image data augmentation is utilized to increase the size of the training dataset. It is possible to increase the accuracy and efficiency of neural network model fitting by training the models using a variety of data.

It entails transforming photos from the training dataset into new images that are members of the same class as the original image. Shifts, flips, zooms, and many more picture alteration techniques are included in the category of transforms.

- Use Keras preprocessing layers
- Use ImageDataGenerator

Following are some of the techniques we plan to implement in the data augmentation:

1. Horizontal and Vertical Shift Augmentation
2. Horizontal and Vertical Flip Augmentation
3. Random Rotation Augmentation
4. Random Brightness Augmentation
5. Random Zoom Augmentation

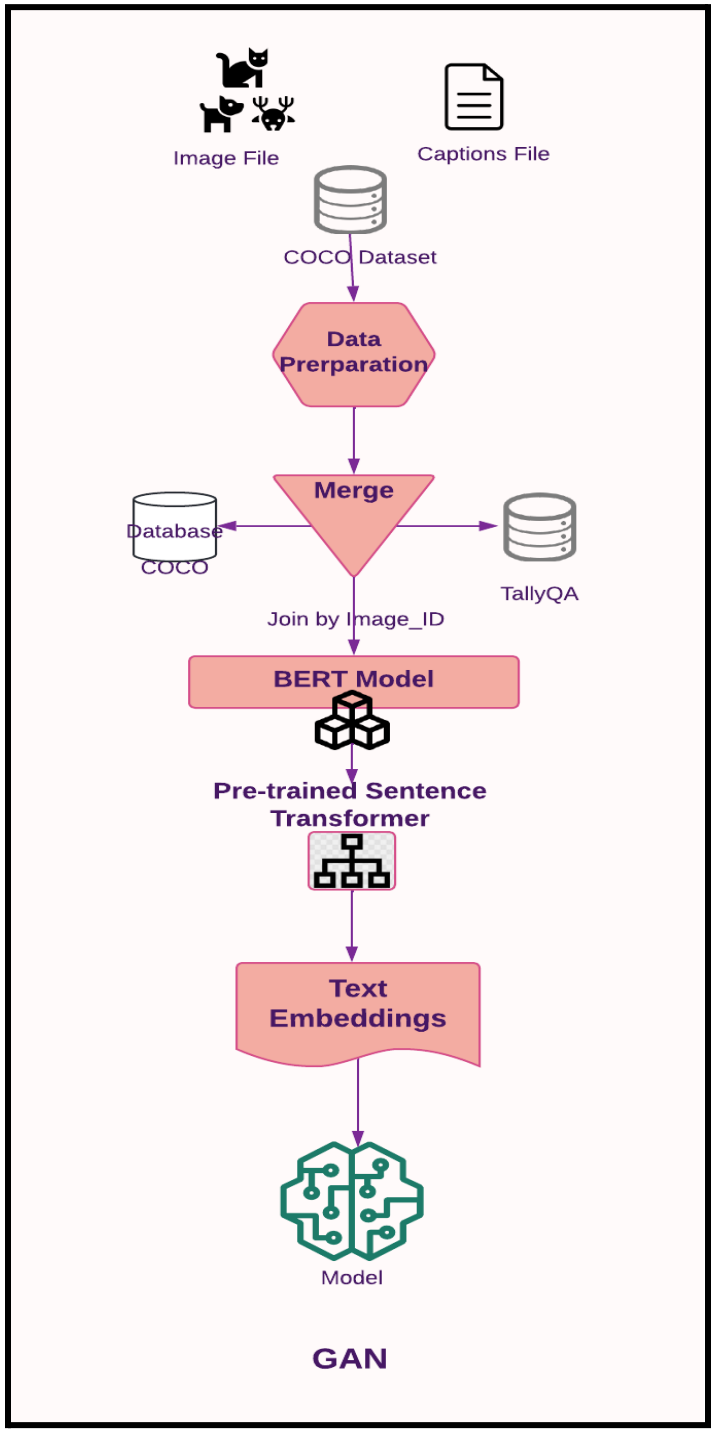


Figure -8: Data Preparation Flowchart

NOVELTY

Text Embeddings

While text embeddings are a regular part of the text-to-image generation , we try to encode additional information into the sentence and train the model using the count associated with the animal name from sentence 2.

So the input to the Sentence Transformer would be the prompt + the "count and animal" eg: *"Cats in a park"* + *"2 cats"*. The output was a text embedding vector with 384 dimensions.

GAN

Additionally, a traditional GAN just looks at the corresponding loss between a generated image, it's real image and the corresponding text.

We propose a methodology which uses an object counting model (detects the number of objects in a given image) to get the difference when compared to the ground truth object number obtained from the TallyQA dataset and propagate it to the loss in a proportionate manner.

The GAN now has a modified generator loss function, which is a weighted addition of generator adversarial loss and mean-squared-error for object count loss. So, the model can also be viewed as a dual discriminator architecture

where 1 discriminator is responsible for the overall quality of the image with respect to the text embeddings and the 2nd is responsible for generating the correct number of objects.

Two mechanisms approached are:

1. Implementation on a traditional CGAN setup
2. State-of-the-art Lafite Text to Image model

Difficulty of the current NN model/ Implementation

The core of our model framework is a Generative Adversarial Network. A generative Neural network is a generative model that creates new data instances that resemble the probability distribution of our training data. GANs use simultaneous training on two models - a generative model that captures training data characteristics and a discriminative model which is used to distinguish between the generated or fake image that is produced by the generator. The aim of GAN is to train the generator enough to deceive the Discriminator into making a mistake.

- **Different types of GAN referred:**
- *Some of the different neural network models we referred to as alternative options to our model:*
    -
    - **Deep Convolutional Generative Adversarial Network**, or DCGAN for short, is another form of the GAN architecture that uses deep CNN for both the generator and discriminator models as well as configurations for the models.

    - **Conditional Generative Adversarial Network (CGAN):** The conditional generative adversarial network(CGAN), is an extension to the GAN architecture that makes use of information in addition to the image as input both to the generator and the discriminator models. If class labels are available, they can be used as input.CGANs are mainly used for image labeling, where both the generator and the discriminator are fed with some extra information y which works as auxiliary information, such as class labels from or data associated with different modalities. The conditioning is usually done by feeding the information into both the discriminator and the generator, as an additional input layer to it.

    - **InfoGAN**: Information Maximizing GAN is an extension to the GAN that attempts to structure the input or latent space for the generator. The latent codes are then used to condition or control specific semantic properties in the generated image.

    - **ViTGAN**: Vision Transformer GAN is an advanced GAN that proceeds to utilize Vision transformers instead of CNNs to generate the image. This also involves additional modifications to the regularization techniques for better learning.

- **Lafite:** A powerful state-of-the-art GAN trained using the popular well-aligned text-image semantic space of CLIP embeddings.

The complexity of this project lies in multiple tiers.

1. The Image embeddings utilize the transformer based BERT-Embeddings.
2. The GAN uses Deep CNNs with multiple improvement techniques like label smoothing, Batch normalization, Leaky ReLU, Gaussian Blur Noise Augmentation
3. As an addition, implementation of an object counting algorithm using opencv to add more insights to the loss calculation.
4. Implementation and understanding of the complex open-source Lafite code and it's usage.

Model Architecture

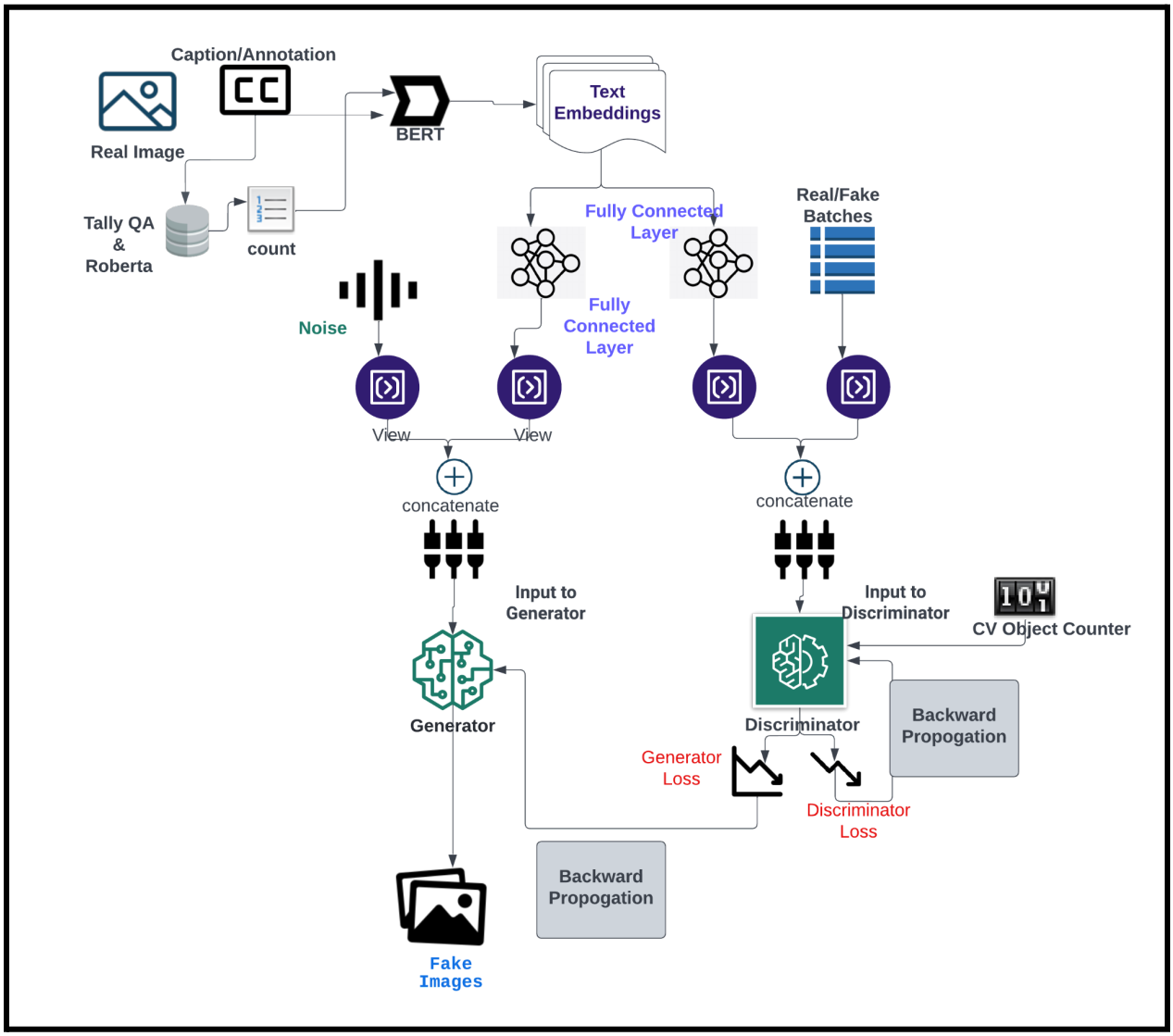


Figure - 9 : Proposed Model Architecture

Implementation:

This project was to implement a Conditional Generative Adversarial Network for Text to Image generation. Since our project aims to try to give higher importance to quantitative context, we design the architecture accordingly.

Architecture Design I
Initially, the text embeddings are put through tally QA to get the number of objects. This is then concatenated to the text and sent to NLP Bert Transformer that gives text embeddings. These embeddings are the conditioned inputs to both Generator and the discriminator.

The version of the GAN used is Conditional , Deep Convolutional GAN (CDCGAN).

Generator:
The generator takes a batch of noise vectors and the text embeddings as input. 1 fully connected layer is connected before they are reshaped and concatenated. The input then passes through a series of ConvTranspose2D layers, Batch normalization layers and activation layers (LeakyRelu). Final Activation in TanH.

Discriminator:
The discriminator takes the text embeddings and real/fake images as input. It passes through a fully connected layer before reshape and concatenation. Then, the input is passed through convolutions, Batch normalization, dropout, activation layers. The expectation of the discriminator is that it must be able to not only identify real images from fake ones, but also identify how well the images are related to the text. Based on these, the discriminator generates losses for the discriminator and the generator. The loss used for the generator is non-saturating. Both losses are computed with adversarial loss.
An OpenCV object counting model is also a part of the discriminator. This model is responsible for understanding the number of objects generated in the fake image, and generating a mean-squared-loss comparing them to the real count, which we get from tallyQA.
This loss is added to the generated loss. This addition is controlled carefully with a hyperparameter alpha.

Discriminator, generator losses for GAN - Adversarial minmax loss.
Modified generator loss = alpha* (generator loss) + (1 - alpha) * (CV count loss)
Currently, we set alpha to 0.9. (or 1-alpha to 0.1)

Keeping this new loss in mind, the model can be viewed as a dual discriminator architecture with 1 discriminator making sure the image generator is good and the other trying to make sure the count for the generated objects is the same.

Architecture Design II (Conceptualized)
We also tried to replicate the Lafite Model and add our quantitative novelty on top of it.
You can import the open source Lafite model from https://github.com/drboog/Lafite. We generated our curated dataset with our custom text+quantitative text as “Prompt. There are <count> <objects>. These were sent to clip to generate text embeddings. The image also was

used to obtain clip image embeddings. The final data as a folder of images and a json file depicting the embeddings is generated.

These feature extracacted inputs are sent to the lafite model which is initialized using the pretrained **MS-COCO with Ground-truth Image-text Pairs, CLIP-ViT/B-32.** The training we proposed was to fine-tune the lafite model with our new feature extracted dataset.

We could generate the required data using CLIP Embeddings and understand the concept to be utilized in Lafite with the pretrained weights. But CUDA and hardware issues made it not feasible to implement on the local system.

Additionally, the object counting function implemented above can be added to the lafite by concatenating the MSELoss for counts to the generator loss.

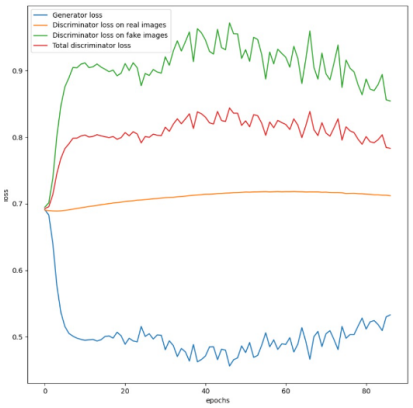
Experimentation Results:
Several Training experiments were performed which resulted in a wide variety of results.

-> Naive training results in either discriminator or generator being too powerful. This results in convergence failure.
When the discriminator was too powerful, we introduced the following to the model:

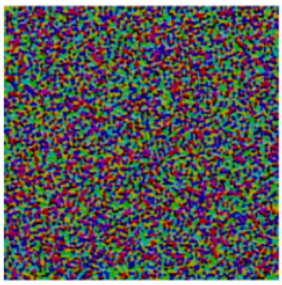
1. Dropout for discriminator.
2. Soft Labels for the discriminator. Instead of 1 for real images and 0 for fake images, we used random values between [0.7 1.1] to real images and random values between [0.0 0.3] for fake images. This has a regularization effect on the discriminator.
3. Noisy labels for the discriminator. Randomly for a small section of the batch, the real and fake labels are swapped. This also makes sure the discriminator doesn't get too powerful.

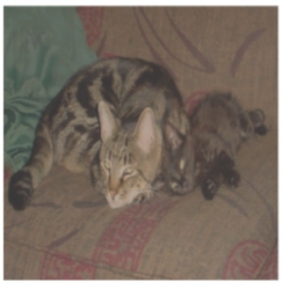
With correct hyperparameters, The loss functions start approaching an equilibrium. However, a lot more epochs on a bigger dataset is needed for a great image generation.

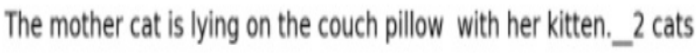
Two giraffes in a grassy area next to trees.

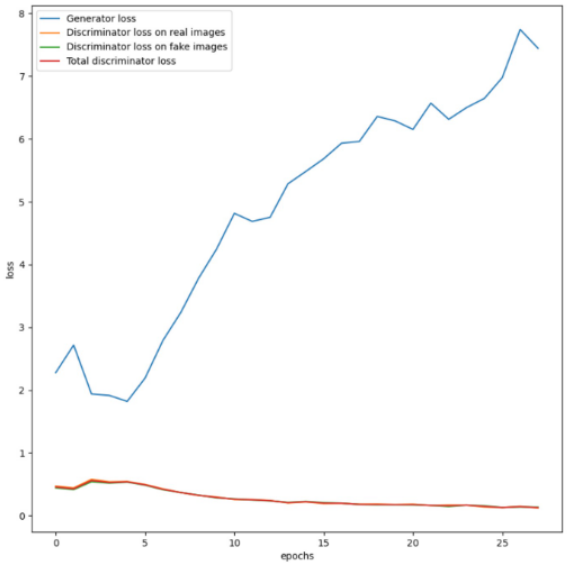


Convergence failure:
When the discriminator is too powerful, the model has convergence failure. The generator is not able to update properly as the discriminator loss doesn't give it enough information.









Ablation Study:

- The current model has 20 million trainable parameters for generator and discriminator. When the discriminator is edited to have 70 Million parameters, the result is convergence failure.
- Deletion of more than 1 'regularization' for discriminator makes it too powerful.

Observations and Challenges

1. GANS are incredibly sensitive to hyperparameters. A slight change in the architecture results in bad / no results.
2. If either part of the GAN is too strong, we get a convergence failure.

3. Different training is required for discriminator and generator if the architecture is different.
4. Saving the models separately, and picking better discriminators each time results in an overall better performance for the generator.
5. Much bigger data with augmentation and several more epochs are required to get better results. This means that greater hardware support is essential for the generation of better images.

Future Scope

1. The quality of the image dataset used could be further improved, and a high pixel image size can be used (1024*1024)
2. Adding Visual Transformers models to the GAN architecture could significantly boost the quality of images.
3. Improve and experiment with the text embeddings. Currently, the last layer [CLS] token embedding is used, but experimentation with different layers embeddings can yield better results
4. Different models, such as Lafite and pre-trained embeddings, can be used to improve our image embeddings further.

Conclusion

1. We tackled a complicated problem of text-to-image formation to disentangle images when given a prompt of multiple objects in a snap. In some of the current models, the images overlap, and we cannot clearly determine the number of cats or objects observed in the Dall-E model.
2. So we encoded numeric information in our text embeddings and passed it into our GAN models, intending to have a better representation of the count of objects in the image.
3. While the initial results were exciting, due to lack of infrastructure, we could not train it for an extended period which would have rendered the images much better.

References

- A Tour of Generative Adversarial Network Models
- The Generator | Machine Learning | Google Developers
- Lverse: [2111.11133] L-Verse: Bidirectional Generation Between Image and Text
- TallyQA Dataset: TallyQA
- COCO Dataset: COCO dataset
- [2107.04589] ViTGAN: Training GANs with Vision Transformers
- https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-tr training-deep-learning-neural-networks/
- Sentence Transformers

- What is Question Answering? - Hugging Face
- GitHub - openai/CLIP: Contrastive Language-Image Pretraining
- Getting started with COCO dataset | by Jakub Adamczyk | Towards Data Science.
- Overview of GAN Structure | Machine Learning | Google Developers
- Imagen
- DALL·E mini by craiyon.com on Hugging Face