Generating Text-Conditional Images using Deep Convolutional GANs

Prashant Kanth
ppk31@scarletmail.rutgers.edu

Arun Subbaiah
as3590@scarletmail.rutgers.edu

Sandesh Mangalore
sm2606@scarletmail.rutgers.edu

ABSTRACT

In this project, we implement a text-to-image synthesis model using Deep Convolutional GAN. Synthesizing high quality images is a challenging problem in Computer Vision. We try to experiment with GANs in a text-conditional setting, which is a difficult task as text and images are characteristically different from one another. Texts come from character space while images reside in pixel space. We use pre-trained models like BERT and CLIP to bring texts from character space to a latent space which can then be translated to pixel space. The baseline of this work is built on methods used in StackGAN [1], where we have Stage-I GAN to sketch primitive shapes and colors of the image based on given text-description and a Stage-II GAN which takes as input the text embeddings and Stage-I output to generate 256×256 *high-resolution images. We make changes to the architecture like using Conv*1×1 *layers and using BERT, CLIP and char-CNN-RNN text encoder for text embeddings. We also perform multiple experiments like removing biases in convolutional layers and using double-sided and one-sided label smoothing to gain training stability and for GAN's convergence.*

I. Introduction

Generation of photorealistic images has many creative, commercial, and forensic applications. On the creative side, it may help artists and graphic designers to translate their thoughts to digital artwork. Commercially, it may open viable alternatives of digital market for designs generated by GANs. In this project, we have attempted to bridge the advances in text and image modeling, translating visual concepts from characters to pixels which can be further extended to multiple domains depending on available training datasets. Image Synthesis is one of the computer vision fields with the most spectacular recent development, but also among those with the greatest computational demands. It is very difficult to train GAN to generate high-resolution photo-realistic images from text descriptions. Thus, it becomes important to track failures early during training. Some of the failure modes are, discriminator loss goes to 0 and generator generates similar image for different inputs, also known as mode collapse. GANs are extremely sensitive to hyperparameters, hence, selection of optimal parameters become very critical for stable training.

Fig. 1. Outputs from Stage-I and Stage-II Generators

II. Related Work

GANs [2] are generative models that learn a mapping from random noise vector x to output image y, $G : z \rightarrow y$. In contrast, conditional GANs learn a mapping from observed input x and random vector z, to y, $G : \{x, z\} \rightarrow y$. Generative Adversarial Networks (GANs) are algorithmic architectures that uses two neural networks, that compete against each other (thus the "adversarial") in order to generate new, synthetic instances of data that can pass for real data. Generator artificially generates fake images in an attempt to fool discriminator for real images. The goal of the discriminator is to identify which outputs it receives have been artificially created. Both models are trained in tandem and follow a cooperative zero-sum game framework to learn with the following objective function,

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

Built upon these generative models, conditional GANs make targeted data generation possible. There have also been works involving image-to-image translation [3], [4], [5] that show promising performance for generating targeted images using conditional GANs. Conditional GAN is an extension of GAN where both the generator and discriminator receive additional conditioning variables c, yielding $G(z, c)$ and $D(x, c)$. This formulation allows G to generate images conditioned on

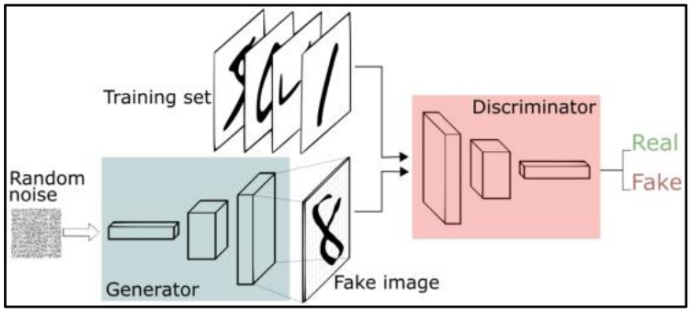


Fig. 2. GAN Architecture

variables c. GANs are always hard to train and there is training instability that needs to be accounted for generating high-resolution images. There are several techniques [11] that have been discussed to stabilize the training process and generate compelling photo-realistic images. Some of these techniques are label noise, label smoothing and instance noise.

III. Dataset

The dataset used for this project is Caltech-UCSD Birds-200-2011 (CUB-200-2011) [6]. It contains 11788 images of 200 bird species. Each image has 10 captions associated with it. As a pre-processing step, we use the bounding-boxes provided along with the dataset to crop the region-of-interest, calculated from center of bounding box, resulting in greater-than 0.75 object-image ratio. We then resize this image to 256×256 to get our final dataset. We keep only 10% of dataset (589 images, 10 captions per image) as test data. Not much data is required to evaluate GANs and to achieve variance in generation, it is only fair that we provide 90% of images (11199 images, 10 captions per image) as training dataset.

IV. Method

In analogy to how painters approach a drawing or painting, first making outlines and then filling in the intrinsic details. The problem of text-to-image generation is also divided into two subproblems where two GAN networks are stacked together to give the final photorealistic image. Stage-I GAN generates low-resolution images that are just the primitive shapes and colors of the resultant image, which are then fed to the Stage-II GAN along with text embeddings to generate final high-resolution images. First Stage-I GAN is trained keeping Stage-II GAN fixed and then Stage-II GAN is trained keeping Stage-I GAN fixed.

A. Stage-I

Text descriptions, t, are fed to pre-trained language model to get their corresponding text embeddings, φ_t, which are passed through conditioning augmentation layer to produce $\mu_1(\varphi_t)$ and $\Sigma_1(\varphi_t)$, which are function of φ_t. Conditioning variables $\hat{c}$ are then sampled from an independent Gaussian distribution $\mathcal{N}(\mu_1(\varphi_t), \Sigma_1(\varphi_t))$, which is concatenated with a noise vector, z, to form the final input for Stage-I Generator $(G1)$, $z \sim \mathcal{N}(0, 1)$. The conditioning augmentation layer increases robustness of the network to small perturbations. For Stage-I discriminator $(D1)$, the real and fake images are passed through a series of downsampling layers to extract their features, and then concatenated with compressed text embeddings, φ_t, to get classified as real (1) or fake (0) image. Conditioning variables $\hat{c}$ are sampled using reparameterization trick,

$$\hat{c} = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{2}$$

The objective of Stage-I GAN is to alternatively maximize $\mathcal{L}_{D1}$ and minimize $\mathcal{L}_{G1}$,

$$\begin{aligned} \mathcal{L}_{D_1} = \mathbb{E}_{(I,t)\sim p_{data}}[logD_1(I, \varphi_t)] + \\ 0.5 * (\mathbb{E}_{(I,t)\sim p'_{data}}[logD_1(I, \varphi'_t)] + \\ \mathbb{E}_{z\sim p_z, t\sim p_{data}}[log(1 - D_1(G_1(z, \hat{c}_0), \varphi_t))]), \end{aligned} \tag{3}$$

$$\begin{aligned} \mathcal{L}_{G_1} = \mathbb{E}_{z\sim p_z, t\sim p_{data}}[log(1 - D_1(G_1(z, \hat{c}_0), \varphi_t))] + \\ \lambda D_{KL}(\mathcal{N}(\mu_1(\varphi_t), \Sigma_1(\varphi_t))||\mathcal{N}(0, I)), \end{aligned} \tag{4}$$

where D_{KL} is Kullback-Leibler divergence (KL divergence) between standard Gaussian distribution and the conditioning Gaussian distribution. This acts as a regularization term to the objective of the generator during training. During training, the discriminator takes real images and their corresponding text descriptions as positive sample pairs (r), whereas negative sample pairs consist of two groups. The first is real images with mismatched text embeddings (w), while the second is synthetic images with their corresponding text embeddings (f). This is calculated as $r + (w + f) * 0.5$

B. Stage-II

Stage-II generator $(G2)$ is conditioned on stage-I low-resolution output (s_0) and same text-embeddings, φ_t, as stage-I. Text embeddings, φ_t, are passed through stage-II conditioning augmentation layer to get $\mu_2(\varphi_t)$ and $\Sigma_2(\varphi_t)$, which is independent from stage-I. Conditioning variables $\hat{c}$ are then samples from an independent Gaussian distribution $\mathcal{N}(\mu_2(\varphi_t), \Sigma_2(\varphi_t))$, which is concatenated with downsampled outputs from stage-I generator to produce latents for stage-II. These latents are passed through series of residual blocks and upsampling layers to finally generate a high-resolution image. Unlike, stage-I we do not use gaussian noise in stage-II. The discriminator $(D2)$ of stage-II has similar structure as stage-I with few extra downsampling layers as the image size is large in this stage. The objective of Stage-II GAN is to alternatively maximize $\mathcal{L}_{D2}$ and minimize $\mathcal{L}_{G2}$,

$$\begin{aligned} \mathcal{L}_{D_2} = \mathbb{E}_{(I,t)\sim p_{data}}[logD_2(I, \varphi_t)] + \\ 0.5 * (\mathbb{E}_{(I,t)\sim p'_{data}}[logD_2(I, \varphi'_t)] + \\ \mathbb{E}_{s_0\sim p_{G_1}, t\sim p_{data}}[log(1 - D_2(G_2(s_0, \hat{c}), \varphi_t))]), \end{aligned} \tag{5}$$

$$\begin{aligned} \mathcal{L}_{G_2} = \mathbb{E}_{s_0\sim p_{G_1}, t\sim p_{data}}[log(1 - D_2(G_2(s_0, \hat{c}), \varphi_t))] + \\ \lambda D_{KL}(\mathcal{N}(\mu_2(\varphi_t), \Sigma_2(\varphi_t))||\mathcal{N}(0, I)) \end{aligned} \tag{6}$$

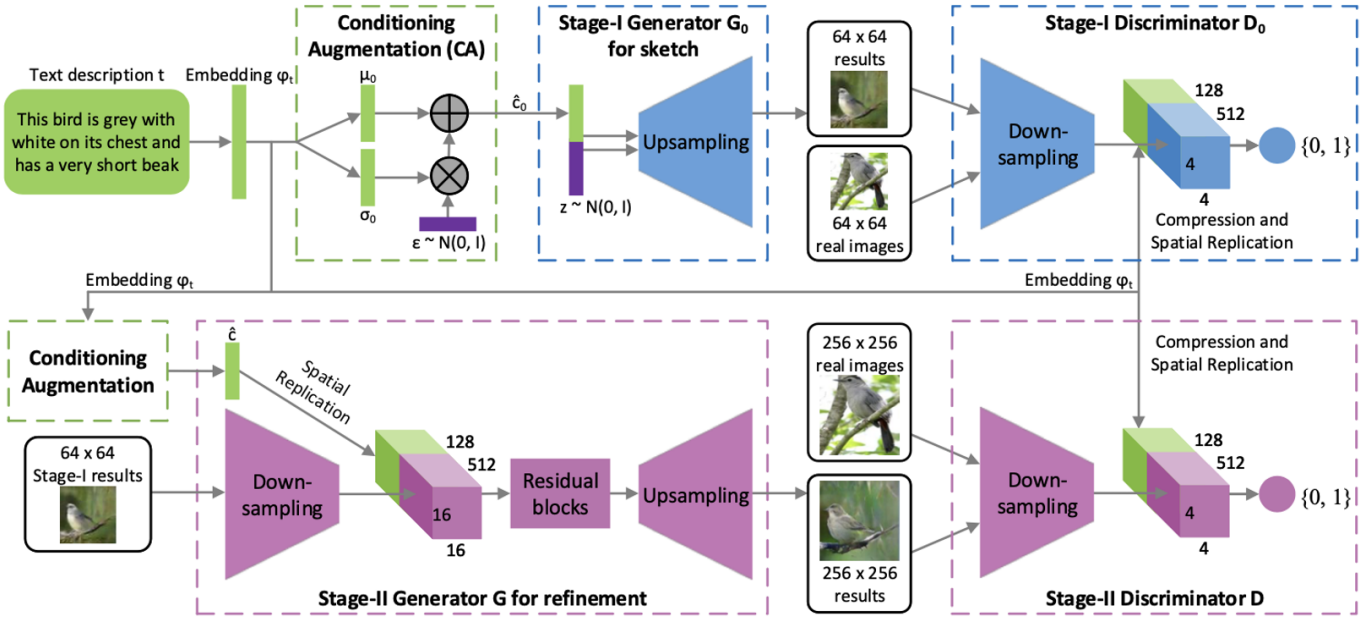


Fig. 3. StackGAN Architecture

V. Implementation Details

The text-encoder that is used in this project are pretrained language models: BERT, CLIP and char-CNN-RNN text encoder. CLIP provides text-embeddings of dimensions 512. For BERT, we use its last hidden state and take its 0^{th} index vector as text-embedding, which is of dimensions 786. char-CNN-RNN text embeddings are 1024 dimensional vector. The conditioning augmentation layer is a fully connected layer that compresses text embeddings to 256-dimensional vector. This 256-dimensional vector is split into two equal parts as μ and σ. Conditioning vector of dimensions 128 is sampled and concatenated with 100-dimensional standard gaussian noise vector to form latent input for Stage-I generator. Stage-I Generator produces a primitive image of shape 64×64. This image along with text-embeddings is then provided as an input to Stage-II Generator to produce a high-dimensional image of 256×256. The Stage-I Generator architecture is displayed in Figure 4. The Augmented Projection for stage-I BERT is displayed in Figure 5. Only input features to linear layer differ in CLIP and char-CNN-RNN text encoder. Upsample Layer consists Conv3 $\times$ 3, BatchNorm and ReLU.

```
----------------------------------------------------------------------------------------------------
           Layer (type)                Output Shape         Param #     Tr. Param #
====================================================================================================
   Augmented_Projection-1     [1, 24576], [1, 128], [1, 128]       5,849,344       5,849,344
            Upsample-2                  [1, 768, 8, 8]      10,619,136      10,619,136
            Upsample-3                [1, 384, 16, 16]       2,655,360       2,655,360
            Upsample-4                [1, 192, 32, 32]         664,128         664,128
            Upsample-5                 [1, 96, 64, 64]         166,176         166,176
              Conv2d-6                  [1, 3, 64, 64]           2,595           2,595
                Tanh-7                  [1, 3, 64, 64]               0               0
====================================================================================================
Total params: 19,956,739
Trainable params: 19,956,739
Non-trainable params: 0
----------------------------------------------------------------------------------------------------
```

Fig. 4. Stage-I Generator

```
Augmented_Projection(
  (fc): Linear(in_features=768, out_features=256, bias=True)
  (relu): ReLU()
  (project): Sequential(
    (0): Linear(in_features=228, out_features=24576, bias=False)
    (1): BatchNorm1d(24576, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
)
```

Fig. 5. Stage-I Augmented Projection for DistilBERT

The Stage-II Genertor architecture is displayed in Figure 6. Only input features to linear layer differ in CLIP and char-CNN-RNN text encoder. DownSample Block consists of series of Conv Layers with BatchNorm and LeakyReLU followed by 4 Residual Blocks. Upsample Layer consists Conv3 $\times$ 3, BatchNorm and ReLU.

```
----------------------------------------------------------------------------------------------------
           Layer (type)                Output Shape         Param #     Tr. Param #
====================================================================================================
         Downsample-1                [1, 192, 64, 64]           5,376           5,376
         Downsample-2                [1, 384, 32, 32]       1,180,800       1,180,800
         Downsample-3                [1, 768, 16, 16]       4,720,896       4,720,896
   Augmented_Projection-4     [1, 128], [1, 128], [1, 128]         196,864         196,864
         Downsample-5                [1, 768, 16, 16]       6,195,456       6,195,456
           ResBlock-6                [1, 768, 16, 16]      10,621,440      10,621,440
           ResBlock-7                [1, 768, 16, 16]      10,621,440      10,621,440
           ResBlock-8                [1, 768, 16, 16]      10,621,440      10,621,440
           ResBlock-9                [1, 768, 16, 16]      10,621,440      10,621,440
          Upsample-10                [1, 384, 32, 32]       2,655,360       2,655,360
          Upsample-11                [1, 192, 64, 64]         664,128         664,128
          Upsample-12               [1, 96, 128, 128]         166,176         166,176
          Upsample-13               [1, 48, 256, 256]          41,616          41,616
            Conv2d-14               [1, 3, 256, 256]           1,299           1,299
               Tanh-15               [1, 3, 256, 256]               0               0
====================================================================================================
Total params: 58,313,731
Trainable params: 58,313,731
Non-trainable params: 0
----------------------------------------------------------------------------------------------------
```

Fig. 6. Stage-II Generator

VI. Experiments

We perform multiple experiments to attain stable GAN training.

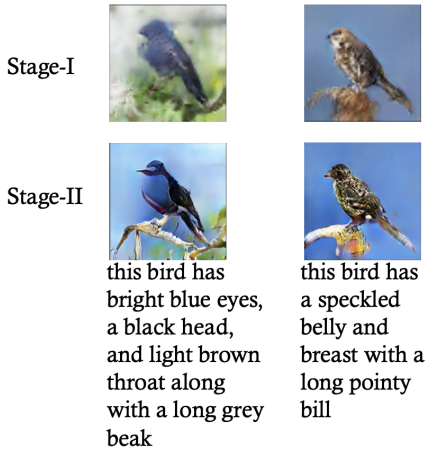


Fig. 7. Stage-I and Stage-II results

A. *Utilizing Only Conditional Loss for Training with μ of text-embeddings shared in Generator and Discriminator*

Here, we experimented by taking μ of text embeddings from Augmented Projection and passing it as condition to discriminator. We only utilise conditional loss in this experiment. i.e., image along with text embeddings as conditions are passed to the discriminator. The training was not stable and generator loss kept on increasing while discriminator loss fell to 0 very quickly which is not desired. Please see Figure 8.

B. *Utilizing both Conditional and Unconditional Loss for Training with μ of text-embeddings shared in Generator and Discriminator*

Here, we experimented by sharing μ of text embeddings from Augmented Projection in Generator with discriminator. We utilise both conditional as well as unconditional loss (only images are passed to the discriminator without any conditional parameter). The training was not stable and generator loss kept shooting up while discriminator loss fell to 0 very quickly which is not desired. Please see Figure 9.

C. *Separate Linear Compression for text-embeddings in Discriminator with bias=True in Conv Layers*

In this experiment we do not share μ of Augmented Projection layer, rather we define a separate linear layer to compress text-embeddings in Discriminator. We also kept $bias = True$ in Upsample and Downsample blocks to observe any benefit in the training. From here on we only use conditional loss during training. This doesn't seem to help in training GAN. Please see Figure 10.

D. *Separate Linear Compression for text-embeddings in Discriminator with bias=False in Conv Layers*

Same as above, we have a separate linear compression layer for text embeddings. Only change being that we keep $bias = False$ as $bias = True$ didn't help much in stable training.Please see Figure 11.

E. *Separate Linear Compression for text-embeddings in Discriminator with bias=False in Conv Layers and Gradient Clipping applied*

After some research, we thought that gradient might be getting too big during backpropagation , thus we decided to run a text with gradient norm clipping. We also keep $bias = False$ from here on. Please see Figure 12.

F. *Separate Linear Compression for text-embeddings in Discriminator with bias=False and Conv1×1 Layer in Discriminator, Label Smoothing in Stage-I and Label Smoothing + Label Noise in Stage-II*

In this experiment, we added a Conv1×1 layer in discriminator instead of Conv3×3 as this will allow discriminator to learn joint distribution of both image and text features giving network a better chance at generating plausible images. We also introduced label smoothing [7], i.e., providing some of the real labels as 0.9 or 0.95 instead of 1 and some of the fake labels as 0.1 or 0.15 instead of 0. This has been proven to introduce stability in GAN, by avoiding mode collapse when generator keeps on producing same result to be passed as real by discriminator. Sometimes, Discriminator becomes too strong too early and in this case generator never learns to produce plausible images, to overcome this we introduced label noise, i.e., randomly flipping labels of real and fake images for $5 - 10\%$ of batch data. Please see Figure 13.

G. *Separate Linear Compression for text-embeddings in Discriminator with bias=False and Conv1×1 Layer in Discriminator, Label Smoothing in Stage-I and Label Smoothing + Label Noise in Stage-II, ConvTranspose2D and Dropout in Upsample Block*

We also experimented with ConvTranspose2D layer in Upsample Block instead scaling and convolution, however this did not help and even after long hours of training, generator couldn't learn the intricacies of real images. Please see Figure 14.

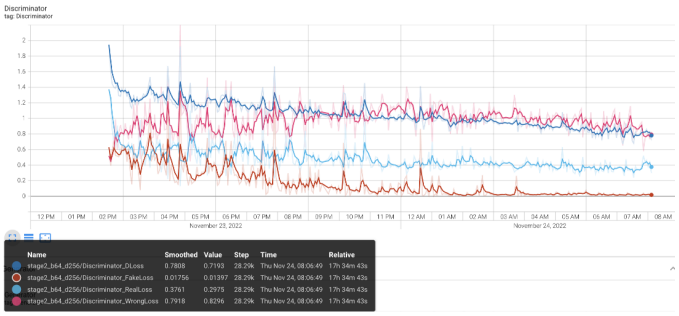
H. *Separate Linear Compression for text-embeddings in Discriminator with bias=False and Conv1×1 Layer in Discriminator, Label Smoothing + Label Noise in both Stage-I and Stage-II*

Here, we introduced both label noise as well as label smoothing [7] in training of both Stage-I and Stage-II and also used char-CNN-RNN text encoder [8]. This helped a lot and the generator seemed to learn to generate plausible images. Please see Figure 15.

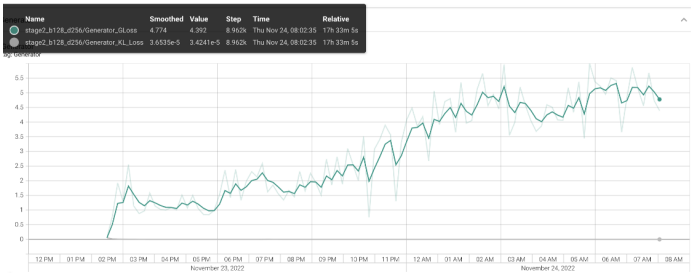
VII. Challenges

We faced a lot of challenges while executing this project. It was very difficult to find a hyperparameter and architecture combination that would result in stable training. Also, due to limited GPU access, getting to train models repeatedly was very tiresome as it took $1-2$ days to train one stage of model. Due to size of the dataset and model architecture (number of trainable parameters), we resorted to parallel GPU training

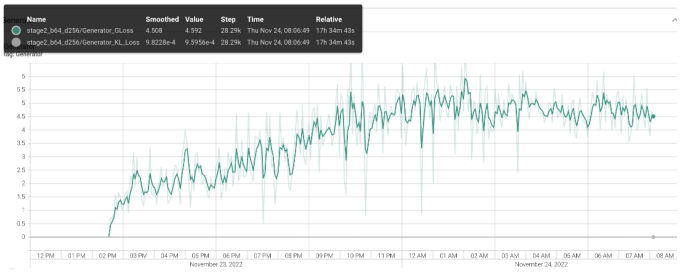
DistilBERT Discriminator Stage-II Training Curve



CLIP Discriminator Stage-II Training Curve

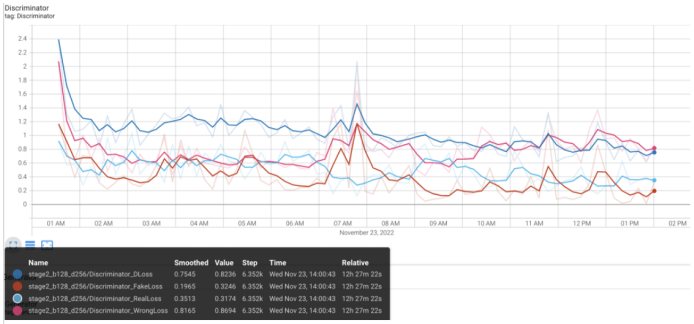


DistilBERT Generator Stage-II Training Curve

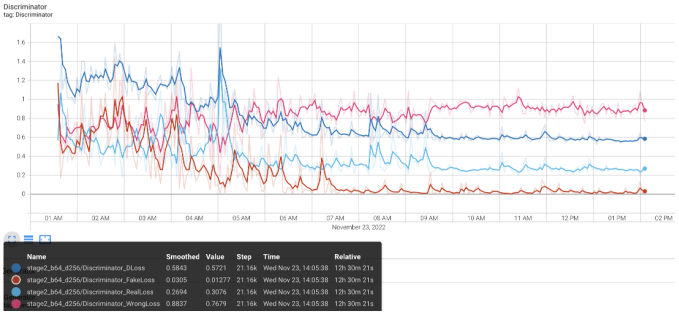


CLIP Generator Stage-II Training Curve

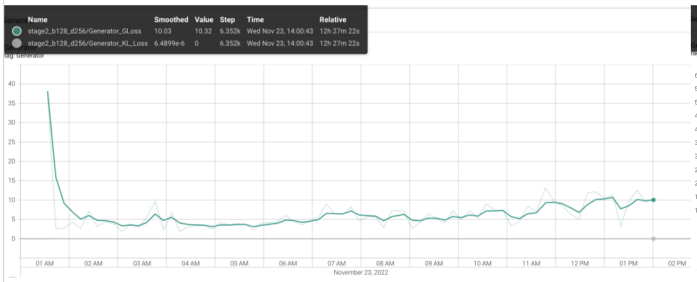
Fig. 8. Experiment A: Stage-II Training Curves of Discriminator and Generator, both CLIP and BERT



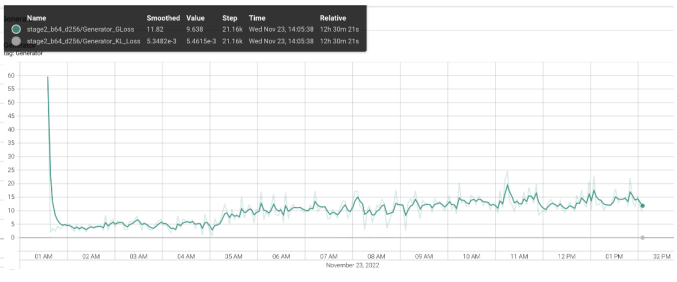
DistilBERT Discriminator Stage-II Training Curve



CLIP Discriminator Stage-II Training Curve

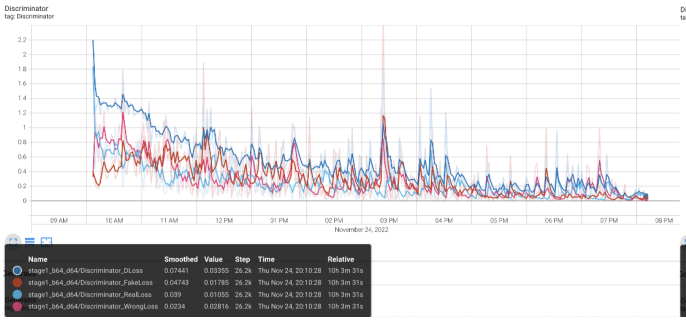


DistilBERT Generator Stage-II Training Curve

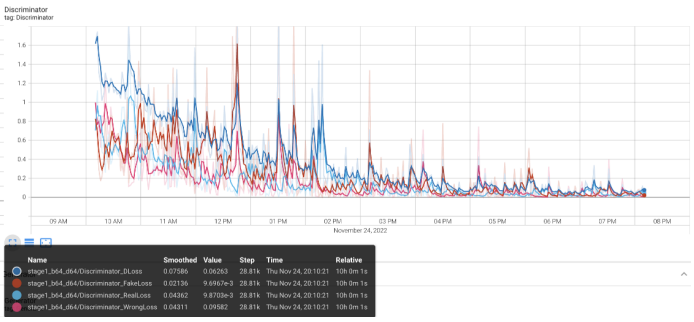


CLIP Generator Stage-II Training Curve

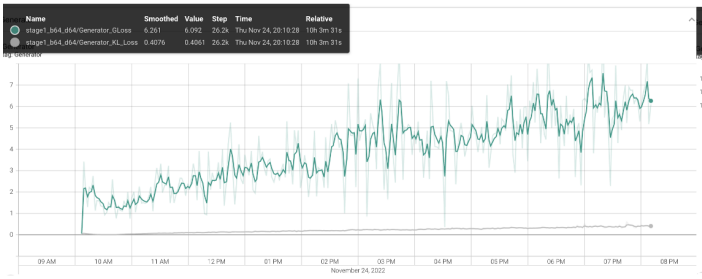
Fig. 9. Experiment B: Stage-II Training Curves of Discriminator and Generator, both CLIP and BERT



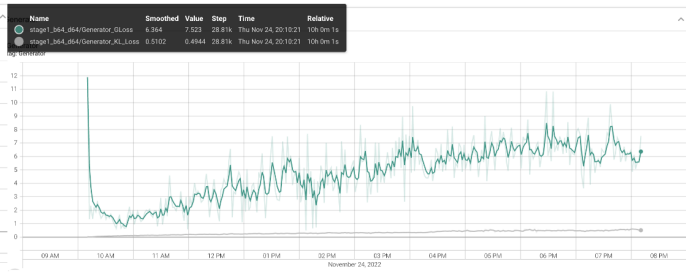
DistilBERT Discriminator Stage-I Training Curve



CLIP Discriminator Stage-I Training Curve



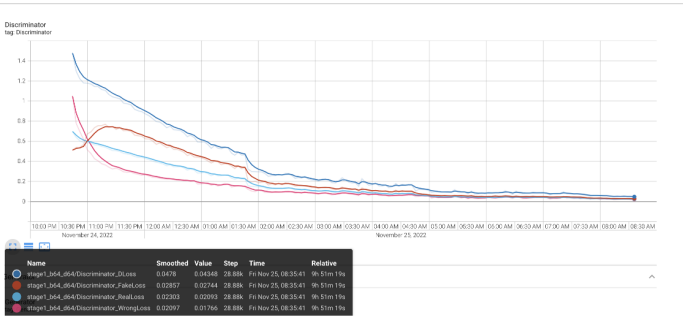
DistilBERT Generator Stage-I Training Curve



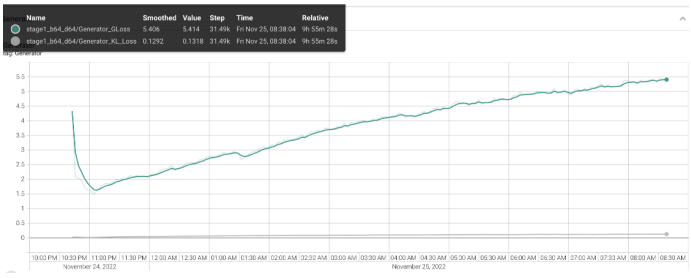
CLIP Generator Stage-I Training Curve

Fig. 10. Experiment C: Stage-I Training Curves of Discriminator and Generator, both CLIP and BERT

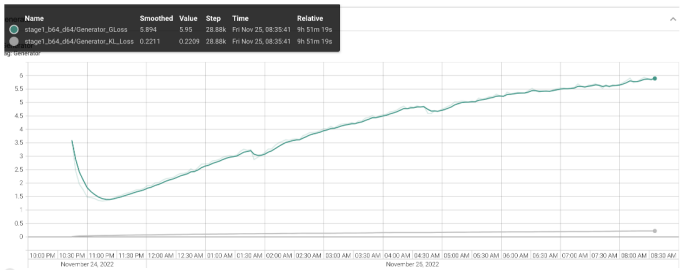
DistilBERT Discriminator Stage-I Training Curve



CLIP Discriminator Stage-I Training Curve



DistilBERT Generator Stage-I Training Curve



CLIP Generator Stage-I Training Curve

Fig. 11. Experiment D: Stage-I Training Curves of Discriminator and Generator, both CLIP and BERT

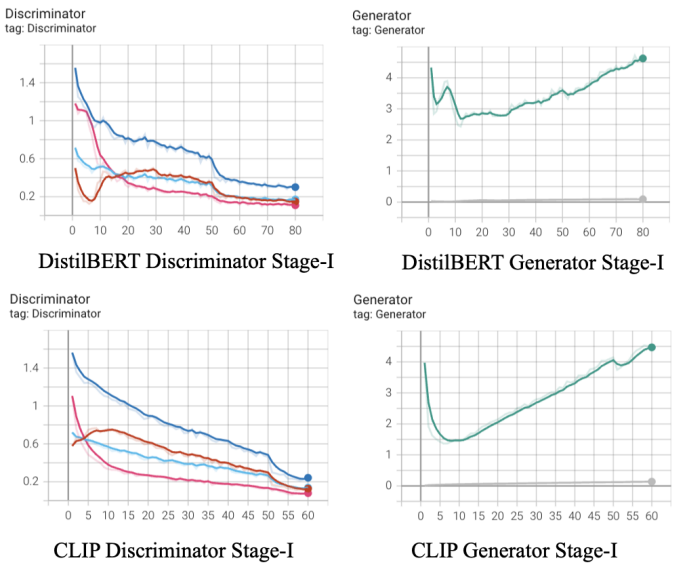


Fig. 12. Experiment E: Stage-I Training Curves of Discriminator and Generator, both CLIP and BERT

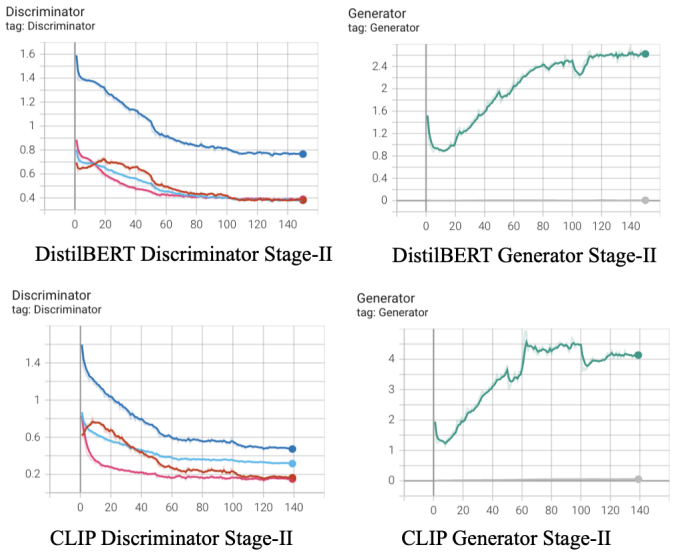


Fig. 13. Experiment F: Stage-II Training Curves of Discriminator and Generator, both CLIP and BERT

(utilising 4 GPUs in parallel). Data pre-processing was a huge task as we had to crop all the images based on their provided bounding boxes such that object-image ratio is optimal. We also had to continuously keep a watch on generated data while training to catch any training anomaly early in the process.

VIII. Technology

- Python as Programming Language
- PyTorch as Deep Learning Framework
- OpenCV for Image Proceesing
- Numpy for Dataset Preprocessing
- Tensorboard for the plots

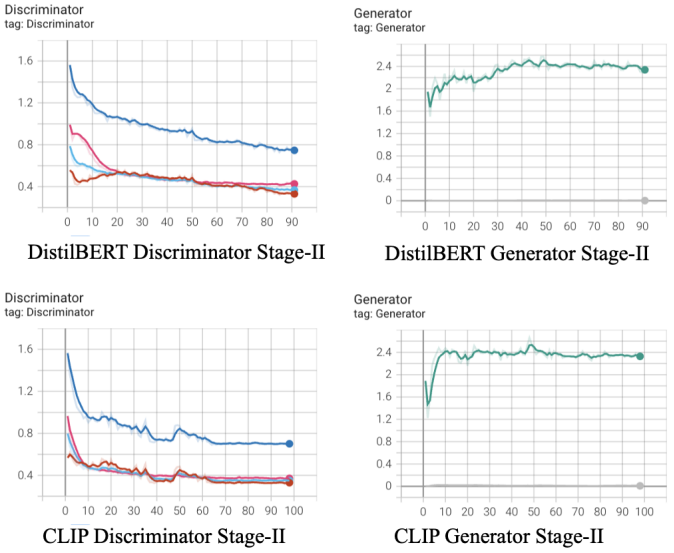


Fig. 14. Experiment G: Stage-II Training Curves of Discriminator and Generator, both CLIP and BERT

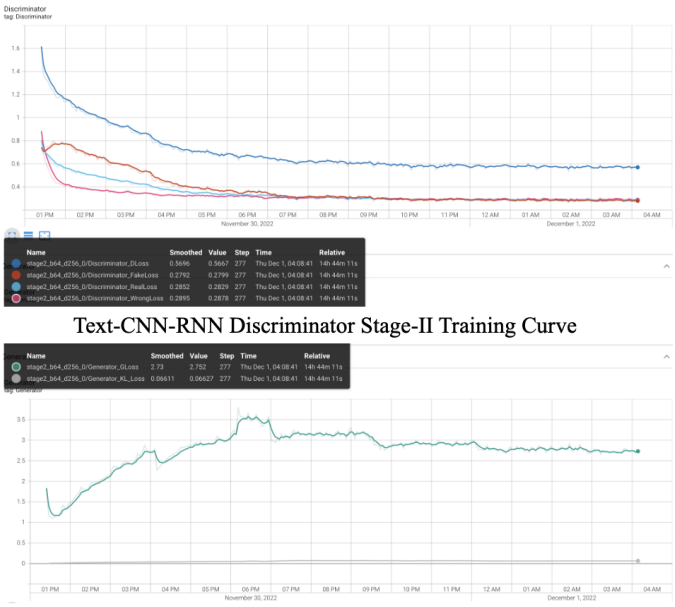


Fig. 15. Experiment H: Stage-II Training Curves of Discriminator and Generator, text-CNN-RNN

IX. Project Timeline

Our project progress is provided in Figure 16

X. Division of Labour

- Ideation: Prashant Kanth, Sandesh Mangalore, Arun Subbaiah
- Dataset Exploration / Preprocessing: Sandesh Mangalore
- Code Design: Prashant Kanth, Arun Subbaiah
- Utilising Text Encoders: Arun Subbaiah
- Multiple Experiments: Prashant Kanth

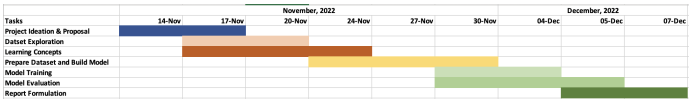


Fig. 16. Project Timeline

- Evaluation: Sandesh Mangalore

XI. Conclusion and Results

During the timeframe of this project, we found that achieving a stable GAN training is difficult because of their hyperparameter-sensitive nature. We had to run a lot of experiments to achieve a stable training and generate close to photorealistic images from text inputs. We learnt a lot about hyperparameter tuning and using different optimizers and losses to produce better results.

In this section, we also show few of the outputs generated by text-to-image GAN model. We show both failure cases in case of mode collapse (Figure 19), training failure (Figure 20) as well as plausible images generated in Stage-I and Stage-II using different text encoders: CLIP, BERT and text-CNN-RNN. We find that text-CNN-RNN produce most plausible images.

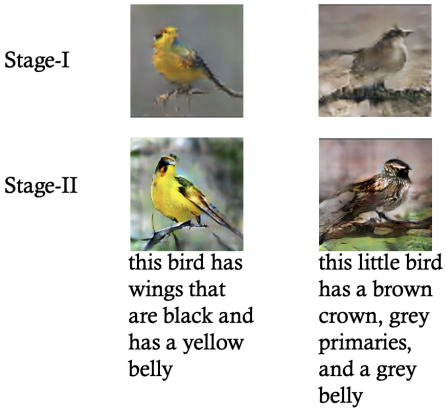


Fig. 17. Results

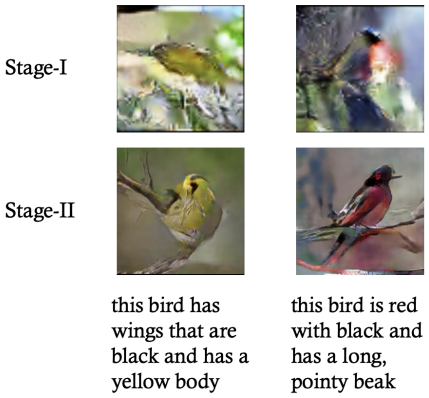


Fig. 18. Results

Fig. 19. Mode Collapse

References

[1] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 5907–5915, 2017.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[3] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[4] J. Gong and M. Mistele, "Conditional generative adversarial networks for transforming face sketches into photorealistic images," tech. rep., Stanford University, 2020.

[5] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

[6] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," Tech. Rep. CNS-TR-2011-001, California Institute of Technology, 2011.

[7] C. K. Sønderby, "Instance Noise: a trick for stabilising gan training," 2016.

[8] S. Reed, Z. Akata, B. Schiele, and H. Lee, "Learning deep representations of fine-grained visual descriptions," in *IEEE Computer Vision and Pattern Recognition*, 2016.

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[10] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," in *International conference on machine learning*, pp. 1060–1069, PMLR, 2016.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[12] C. K. Sønderby, J. Caballero, L. Theis, W. Shi, and F. Huszár, "Amortised map inference for image super-resolution," *arXiv preprint arXiv:1610.04490*, 2016.

[13] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[14] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning*, pp. 8748–8763, PMLR, 2021.

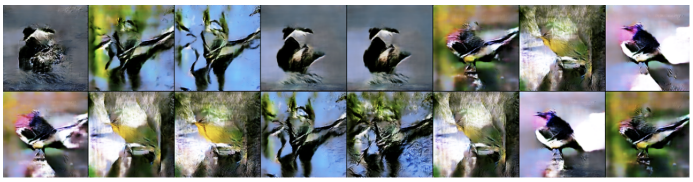


Fig. 20. Training Failure

[15] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[16] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.