TEXT TO IMAGE SYNTHESIS USING TRANSFORMER GANS

BY

YASH PRADEEP GUPTE
A20472798

COMPUTER SCIENCE DEPARTMENT
CS597

Submitted in partial fulfillment of the
requirements for the degree of
MASTER OF SCIENCE in COMPUTER SCIENCE
in the Graduate College of the
Illinois Institute of Technology

Chicago, Illinois
12 / 2021

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Transformer models have shown some significant results in domain of computer vision tasks such as classification, detection, and segmentation. While these tasks mainly focus on the discriminative part of models, we intend to explore transformers on generative adversarial networks (GANs). Our goal is to conduct a study in building a generator model of GAN which is completely free of convolutions, using only pure transformer-based architecture. Our text to image Transformer GAN model generates images from given text input captions. Our model consists of a transformer-based generator that increases feature resolution progressively and convolutional based discriminator model captures semantic contexts and low-level features. We model three types of transformer-based generators and a transitional convolution-based GAN and compare them. Our Transformer GAN is implemented on a specific class of MS COCO dataset that is the Zebra class with image dimension of 64×64.

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

Generating photo-realistic images from text is an important problem and has tremendous applications, including photo-editing, computer-aided design, super resolution, face aging, etc. Recently, Generative Adversarial Networks (GAN) have shown promising results in synthesizing real-world images. However, it is very difficult to train GAN to generate high-resolution photo-realistic images from text descriptions. The main difficulty for generating high-resolution images by GANs is that supports of natural image distribution and implied model distribution may not overlap in high dimensional pixel space.

1.2 PROPOSED SOLUTION

The Transformer is an architecture that uses Attention to significantly improve the performance of deep learning NLP translation models. It was first introduced in the paper "Attention is all you need" and was quickly established as the leading architecture for most text data applications. However, its applications to computer vision remain limited. In this project, we will build up Transformer GAN model to solve text to image synthesis problem. Our goal is to conduct a study in building a GAN completely free of convolutions, using only pure transformer-based architecture. Our contribution is to build a transformer-based generator architecture and experiment on their variations (Model II, Model III and Model IV). We also build a transitional GAN (Model I) to compare with above mentioned models.

CHAPTER 2

REVIEW OF LITERATURE

2.1 DRAWBACKS OF CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) have shown brilliant computations when it comes to image processing and computer vision. CNNs have the ability to learn features with an element of spatial invariance. Object recognition accuracy of CNNs is the best, but there are some notable drawbacks as well. The MaxPooling operation used in CNN disregards the spatial orientation of involved features. The criteria to classify an image is the mere presence of a necessary object while disregarding its placement with respect to image boundary and other present features.

A question can arise whether we can build a strong GAN which is completely free of convolutions? Fundamentally, CNNs cannot process long term dependencies unless passing through enough layers, as the convolution operator has a local receptive field. However, that is not efficient, causing loss of feature resolution and fine details, in addition to difficulty of optimization. The spatial invariance possessed by convolutions poses a bottleneck on its ability to adapt spatially varying or heterogeneous visual patterns. This gave rise to the success of relational network, dynamic filters, and kernel prediction methods.

2.2 ATTENTION GANS

Attention GANs - Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks

This paper introduced a text to image synthesis task with Attention mechanism which synthesize fine grained details at different sub regions of the image by paying attention to relevant words in the natural language description. They also proposed a novel loss foe generator called as DAMSM loss (Deep Attentional Multimodal Similarity Model). They trained thei models on CUB 200 2011 and MS COCO datasets. The motivation comes from the attention mechanism which has shown significant results in machine translation tasks.The encoder – decoder architecture of attention models is a sequence-to-sequence task which when incorporated with GANs describes image generation as an m-stage process.

The Attention GAN model consists of two novel components.

(a) Attn GAN network – attention mechanism is developed foe generator to draw different subregions of image by focusing on words that are most relevant to the subregion being drawn.
(b) DAMSM – This model is able to compute the similarity between the generated image and the sentence using both the global sentence level information and the fine-grained word level information.

Overall, this model had three generators and three discriminators for different dimension of images to be generated. The results of their experiments were great but to train models it was computationally heavy [1].

2.3 STYLE GANS

STYLE GAN V1:

They proposed an alternative generator architecture in GANs, which can precisely control the synthesis of image that is clearly separate the pose, identity, face freckles, hair, color, etc. This novel generator architecture can control the style of the generated image. They trained their model on Flickr Face H1(FFHQ) dataset which has 70k images of 1024x1024 resolution consisting of diverse faces. The Mapping network takes latent vector z as input and maps it into another vector w. They also introduce affine transformations 'A' , normal transformations form noise as 'B' and Adaptive Instance Normalization(AdaIN) in the generator network. Mixing regularization technique was implemented, where instead of passing one latent input they pass at least two z1 and z2 and get two vectors w1 and w2. Then these were passed to randomly chosen input styles 'A'. It was observed that mixing regularization improved the tolerance to adverse operations significantly. The adverse operations included randomizing 1 to 4 latent and crossover points between them [2].

STYLE GAN V2:

The instance normalization in style gan v1 caused water droplets like effects in the generated images. This second version of style gan focused on removing these normalization artifacts by restructuring the Adaptive Instance Normalization (AdaIN). They redesigned the style gan architecture and implemented three techniques:

(a) Weight Demodulation: Separate out the addition of gaussian noise 'B' with AdaIN 'A', stating that these might have conflicting interest which by sort block each other out (separate style blocks). The next idea was to get rid of AdaIN and use Weight demodulation layers. This technique significantly removed normalization artifacts.
(b) Perceptual Path Length (PPL): They added PPL to loss function of generator. The high-level idea of PPL is to have changes in the latent vector x, now results too dramatic of changes in the generated image. We want to have smooth semantic changes in generated images rather than having completely different image.
(c) Is Progressive Growing necessary? - ResNet style architecture – isolate 256x256 feature maps and then perform upsampling, then do elementwise addition, at the end convert it into 3 channel RGB image. Input/Output skips – In this method, first convert 256x256 feature map into RGB image by flattening it into 3 channels, then sum this up and perform upsampling and go to the next level [3].

2.4 TRANS GANS

This paper aims to be the first pilot study to build GAN completely free of convolutions, using only pure transformer-based architecture. They have performed image synthesis on datasets such as STL-10, CIFAR-10, CelebA (128x128), CelebA-HQ (256x256), and LSUN-Church. Although pure transformer architecture applied directly to sequences of image patches can perform very well on image classification task, it is unclear whether the same way remains effective in generating images. Their key approach was to start with a memory friendly transformer-based generator by gradually increasing the feature map resolution in each stage. They also improve on the discriminator with a multi scale structure that takes patches of varied size as inputs, which balances between capturing global contexts and local details, in addition to enhancing memory efficiency more.

They also introduced a new module called grid self-attention, that elevates memory bottleneck further when scaling up TransGAN to high resolution generation (eg. 256x256). Their three major contributions were:

(a) Novel architecture design: First GAN using purely transformers and no convolutions in generator and discriminator models.
(b) New Training Recipe: Techniques like data augmentation, modifying layer normalization, and adopting relative position encoding, both for generator and discriminator, were implemented.
(c) Performance and Scalability: They achieved highly competitive performance scores like inception score of 10.43 and FID score of 18.28 on STL-10 dataset, 9.02 inception score and 9.26 FID score on CIFAR-10 [4].

This paper serves as a baseline for our implementation where we make use of the transformer architecture specifically in the generator model.

CHAPTER 3

ANALYSIS AND DESIGN

3.1 DATASET

We have trained and tested our model on MS COCO dataset specifically the Zebra class. The zebra class contains around 2000 images, out of which 1324 images are assigned for train set and 677 images assigned for test set. Each image is associated with 5 captions describing it in detail [5].

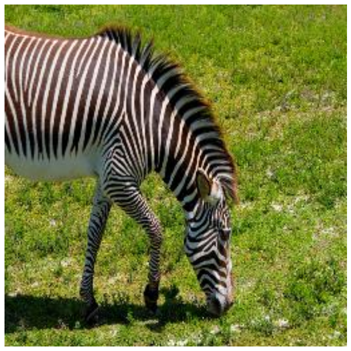


Figure 1: Zebra Class Image

1. a zebra grazing on lush green grass in a field.

2. zebra reaching its head down to ground where grass is.

3. the zebra is eating grass in the sun.

4. a lone zebra grazing in some green grass.

5. a zebra grazing on grass in a green open field.

Captions for Figure 1.

3.2 DATA PREPROCESSING

The input to our Transformer GAN architecture is images and their corresponding captions. We preprocess this data using cv2 [6] and BERT [8] model to generate image and caption vectors. The dataset is split into train and test sets and data preprocessing is performed on images, images masks, and the captions. We convert images and image masks into vectors using cv2 package and then normalize them. Then we import the BERT model and tokenizer to tokenize the captions.

3.2.1 PREPROCESSING CAPTIONS

For all the images, once we get the set of captions (five captions per image) we iterate over them to convert them into the desired model input dimensions. Consider the first caption of the first image

caption: 'a couple of zebra standing next to each other on a field.'

We tokenize this caption using pretrained BERT tokenizer and add ‘CLS’ and ‘SEP’ tokens at the start and end of caption. Thus, the tokenized text looks as follows,

tokenized caption: ['CLS', 'a', 'couple', 'of', 'zebra', 'standing', 'next', 'to', 'each', 'other', 'on', 'a', 'field', '.', 'SEP’]

The next step is to convert these tokens to ids using the same BERT tokenizer methods; thus, our indexed tokens look as follows,

Indexed_tokens: [100, 1037, 3232, 1997, 29145, 3061, 2279, 2000, 2169, 2060, 2006, 1037, 2492, 1012, 100]

We create a tokens_tensor and segements_tensors which are passed into the pretrained BERT Model.

tokens_tensor : tensor([[100, 1037, 3232, 1997, 29145, 3061, 2279, 2000, 2169, 2060, 2006, 1037, 2492, 1012,   100]])
segments_tensors - > tensor ([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])

The output of BERT Model are our desired token embeddings of shape (15,12,768). For each token in out sentence, we iterate and create a captions vector and mask vector. Our maximum length of a sentence is set to 16 and we fill empty words and their mask with zeros and ones respectively in the vector data. Finally, our captions vector (16,768) and mask vectors (16,) are ready to be passed into our generator and discriminator models.

3.2.2 PREPROCESSING IMAGES

Data preprocessing for images is quite easier compared to captions. We use Open CV framework to read images and return their vector data. We implement the cv2.imread(image_file_path) to process the image and then normalize the image. Image masks are also processed in the same way. These are then passed into generator and discriminator models.

3.3 MODEL ARCHITECTURES

3.3.1 OVERVIEW

Text to image - Transformer GANs architecture

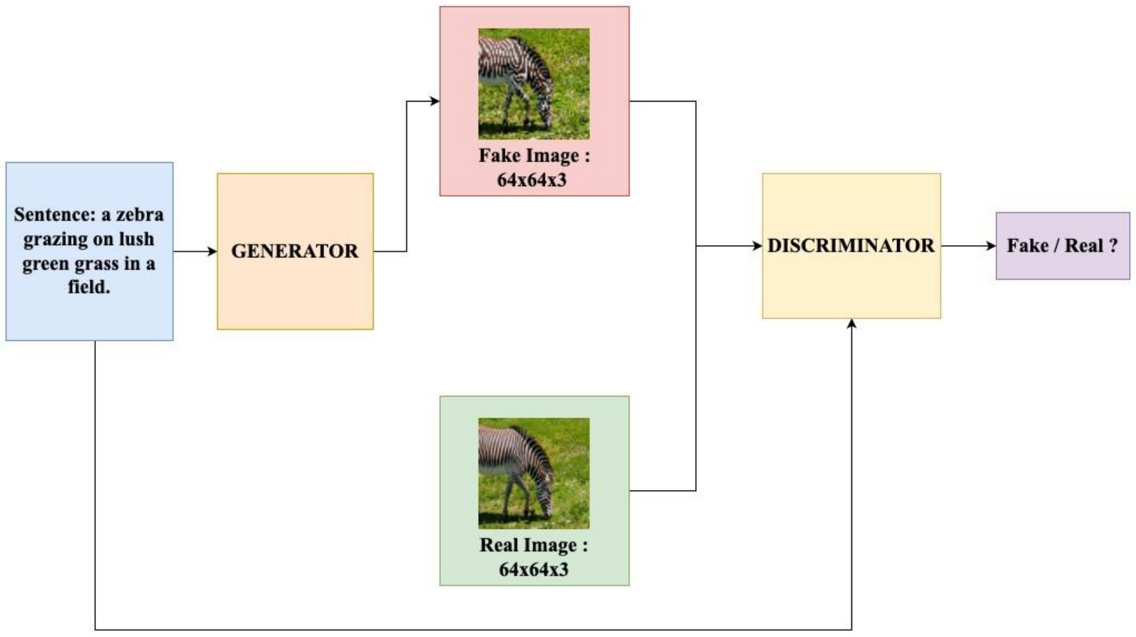


Figure 2. Text to image - Transformer GANs architecture

From abstract view of our entire GAN architecture (Figure 2.), it is similar to other GAN architectures. We send the text caption to the generator architecture and the GENERATOR model generates a FAKE IMAGE. Next, this FAKE IMAGE, along with REAL IMAGE from our dataset is given to the DISCRIMINATOR model. The DISCRIMINATOR model performs operations with an objective to produce a score of how real or fake the generated image is. These two models work simultaneously as adversaries of each other towards an objective to improve the GENERATOR model, which can generate photo-realistic image synthesis from text input.

3.3.2 GENERATOR MODEL I

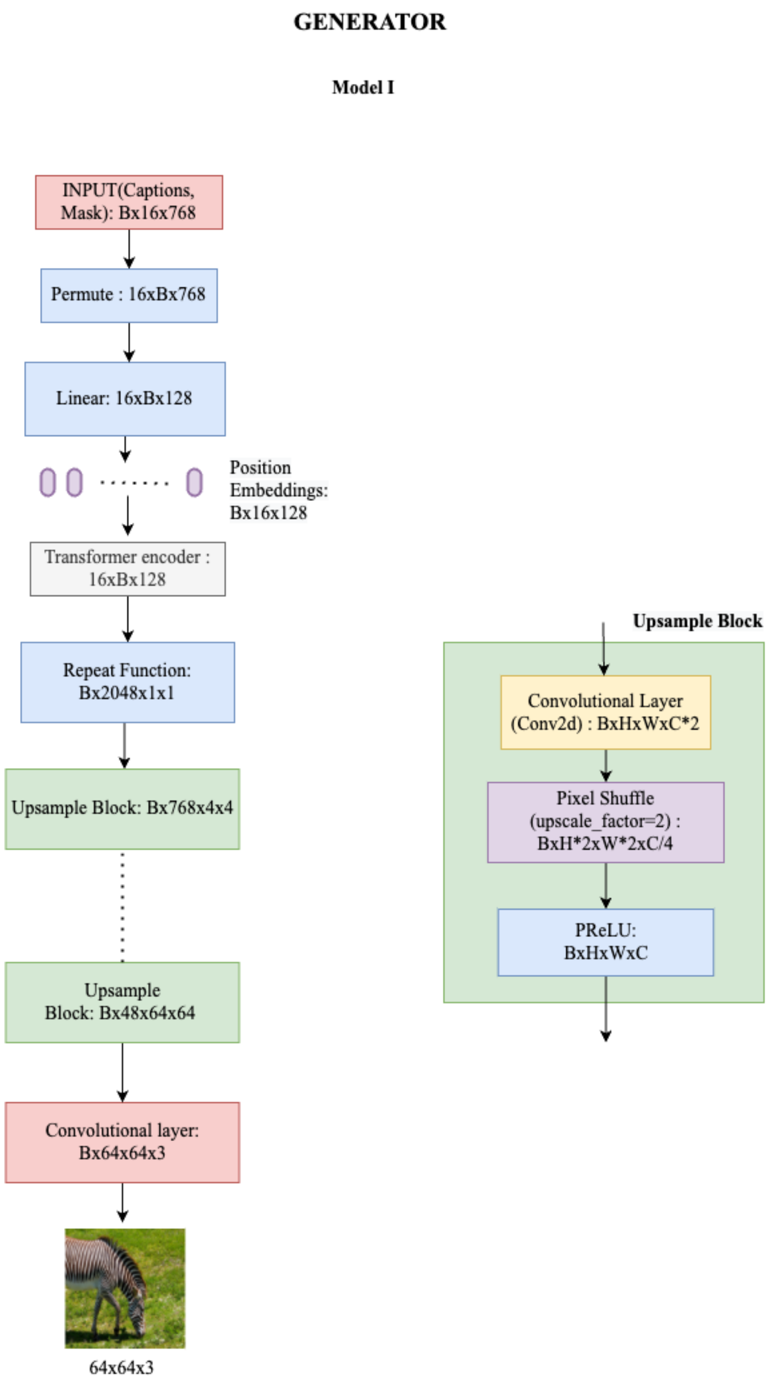

GENERATOR
Model I
INPUT(Captions, Mask): Bx16x768
Permute : 16xBx768
Linear: 16xBx128
Position Embeddings: Bx16x128
Transformer encoder : 16xBx128
Repeat Function: Bx2048x1x1
Upsample Block: Bx768x4x4
Upsample Block: Bx48x64x64
Convolutional layer: Bx64x64x3
64x64x3
Upsample Block
Convolutional Layer (Conv2d) : BxHxWxC*2
Pixel Shuffle (upscale_factor=2) : BxH*2xW*2xC/4
PReLU: BxHxWxC

Figure 3: GENERATOR MODEL I

Figure 3. illustrates the first generator model made up of convolutional layers for upscaling the dimensions. We start with our input viz. captions vector and the mask vector. This input is of dimension Bx16x768, where B is batch size. We permute the dimensions and pass it through a linear layer which outputs a tensor of shape 16xBx128. Next, we provide position embeddings to our tensor, and this is then passed to a Transformer Encoder. The transformer encoder encodes the input caption vector along with position encodings and generates the entire sentence embeddings. Next, we recursively un-squeeze the tensor from 16xBx128 dimension to get a target tensor dimension of Bx2048x1x1. Then we perform a deconvolution operation to upscale the feature map to Bx768x4x4 shape. The next part is a Upsample Block which is constructed as follows, it consists of a Convolutional layer, a Pixel Shuffle layer followed with Parametric ReLU activation. The key feature over here is that we are using the Pixel Shuffle technique with upscale factor of 2 to upsample the inputs. The output of this first upsample block is Bx384x8x8 tensor and then we perform multiple Upsample blocks till we get a tensor of shape Bx48x64x64. Lastly, we forward this tensor to a convolutional layer which outputs an image of shape Bx3x64x64.

This generator model is a transitional GAN generator based on convolutional layers stacked up on each other to upsample text description embeddings to an image.

3.3.3 GENERATOR MODEL II

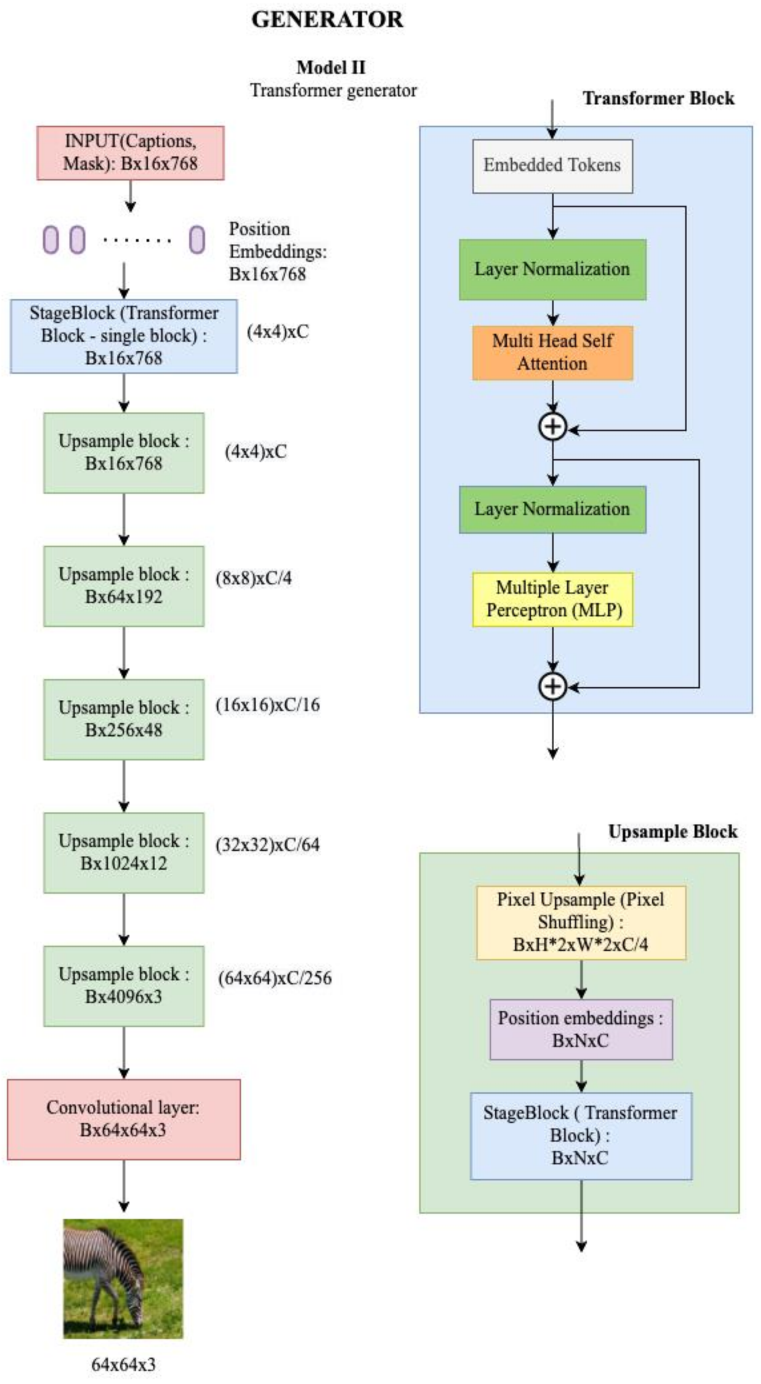

GENERATOR
Model II
Transformer generator
INPUT(Captions, Mask): Bx16x768
Position Embeddings: Bx16x768
StageBlock (Transformer Block - single block) : Bx16x768
(4x4)xC
Upsample block : Bx16x768
(4x4)xC
Upsample block : Bx64x192
(8x8)xC/4
Upsample block : Bx256x48
(16x16)xC/16
Upsample block : Bx1024x12
(32x32)xC/64
Upsample block : Bx4096x3
(64x64)xC/256
Convolutional layer: Bx64x64x3
64x64x3
Transformer Block
Embedded Tokens
Layer Normalization
Multi Head Self Attention
Layer Normalization
Multiple Layer Perceptron (MLP)
Upsample Block
Pixel Upsample (Pixel Shuffling) : BxH*2xW*2xC/4
Position embeddings : BxNxC
StageBlock (Transformer Block) : BxNxC

Figure 4: GENERATOR MODEL II

Figure 4. illustrates generator model II made up of pure transformer architecture which takes caption vector and mask vector as input. This architecture is inspired from the TransGAN generator architecture [4]. Here we first combine the first relative positional embeddings with the input caption vector.

Then comes the next important module of our architecture which is the Stage block which contains a transformer encoder block. An encoder is a composition of two parts, the first part is constructed by a multi-head self-attention module and the second part is a feed forward MLP with GELU nonlinearity. The normalization layer is applied before both of the two parts which also employ residual connections.

By stages we gradually increase the feature map resolution until it meets target resolution $H \times W$(64x64). We define $H0 = 4$ and $W0 = 4$ as height and width which is a breakdown of our captions vector from 16 to 4x4. Thus, we actually treat a feature map (Bx16x768 or Bx4x4768) as a length 16 sequence of 768 dimensional tokens, combined with learnable relative positional encoding.

To scale up to higher resolution images, we insert an upsample block stage wise, consisting of pixel shuffle and stage block mechanism. The pixel shuffle [9] layer upsamples the resolution of feature map by $2 \times$ ratio and reduces the embedding dimension to quarter of the input. Once we have the output of previous layer, we add relative positional embeddings to it and pass it through a transformer encoder block again. This basically is the structure of Upsample block.

We repeat multiple stages of upsample blocks until it reaches the desired target resolution and then we project the embedding dimension to 3 to obtain the RGB image of dimension Bx3x64x64.

While classical transformers used deterministic position encoding or learnable position encoding, the relative position encoding exploit lags instead of absolute positions. Consider a single head of self-attention layer,

$$Attention(Q, K, V) = softmax(\left(\frac{QK^T}{\sqrt{d_k}}\right) V)$$

Where $Q, K, V \in \mathbb{R}^{(HxW)xC}$ represents query, key, and value matrices, $H, W, C,$ denotes the height, width, and embedded dimension of input feature map. The difference in coordinate between each query and key on H axis lies in range of $[-(H-1), H-1]$, similarly for W axis. By simultaneously considering both H and w axis, the relative position encoding can be represented by parameterized matrix $M \in \mathbb{R}^{(2H-1)\times(2W-1)}$. Per coordinate, the relative position encoding E is taken from matrix M and added to the attention map QK^Tas as bias term as follows,

$$Attention(Q,K,V) = softmax(\left(\frac{QK^T}{\sqrt{d_k}} + E\right)V)$$

Relative position encoding learns a stronger relationship between local contents, bringing important performance gains in large scale cases. Hence, we apply it on top of the learnable absolute positional encoding for the generator.

3.3.4 GENERATOR MODEL III

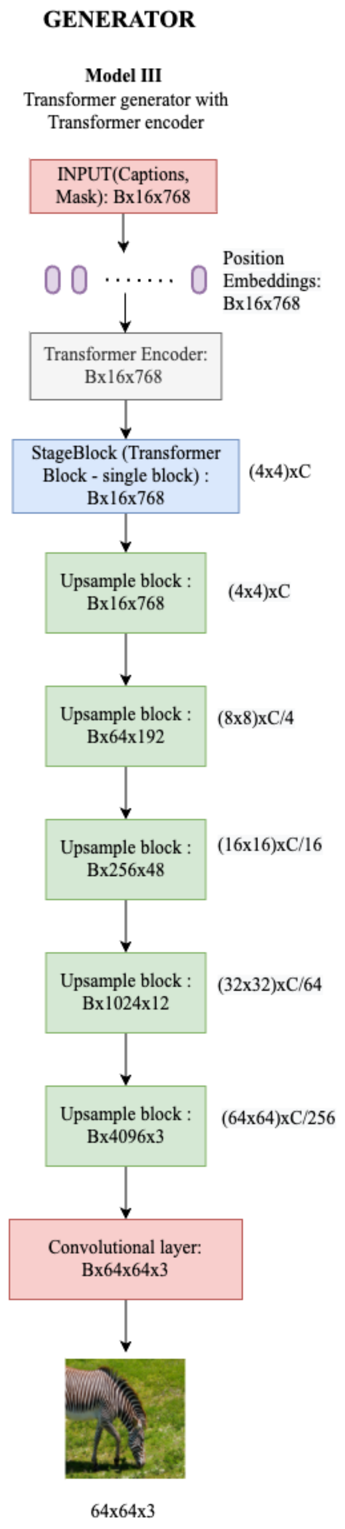

GENERATOR
Model III
Transformer generator with Transformer encoder
INPUT(Captions, Mask): Bx16x768
Position Embeddings: Bx16x768
Transformer Encoder: Bx16x768
StageBlock (Transformer Block - single block) : Bx16x768
(4x4)xC
Upsample block : Bx16x768
(4x4)xC
Upsample block : Bx64x192
(8x8)xC/4
Upsample block : Bx256x48
(16x16)xC/16
Upsample block : Bx1024x12
(32x32)xC/64
Upsample block : Bx4096x3
(64x64)xC/256
Convolutional layer: Bx64x64x3
64x64x3

Figure 5: GENERATOR MODEL III

Figure 5. illustrates generator model III made up of pure transformer architecture which takes caption vector and mask vector as input. This architecture is based on Model II architecture. We take the same Model II and add a transformer encoder layer. As we have seen that in Model II, we add the position embeddings to the input captions and then pass them directly to the Stage Block. In Model III, we will pass the output of position embeddings through a Transformer Encoder layer which will encode these position embeddings along with the mask vector which we pass as an input to the transformer encoder layer. The later layers in Model III are similar to that of Model II. We train this model and evaluate its results to check for any improvements in the generator network.

3.3.5 GENERATOR MODEL IV

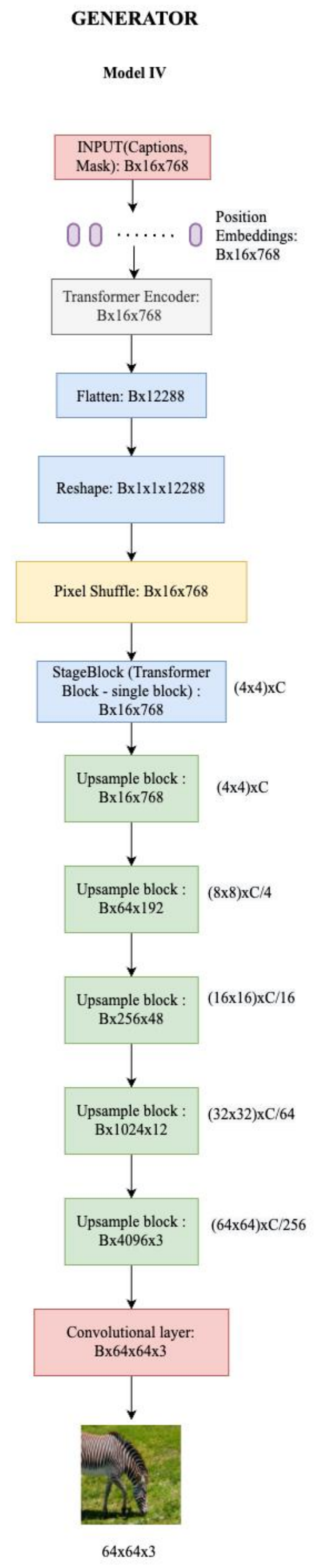

GENERATOR
Model IV
INPUT(Captions, Mask): Bx16x768
Position Embeddings: Bx16x768
Transformer Encoder: Bx16x768
Flatten: Bx12288
Reshape: Bx1x1x12288
Pixel Shuffle: Bx16x768
StageBlock (Transformer Block - single block) : Bx16x768
(4x4)xC
Upsample block : Bx16x768
(4x4)xC
Upsample block : Bx64x192
(8x8)xC/4
Upsample block : Bx256x48
(16x16)xC/16
Upsample block : Bx1024x12
(32x32)xC/64
Upsample block : Bx4096x3
(64x64)xC/256
Convolutional layer: Bx64x64x3
64x64x3

Figure 6: GENERATOR MODEL IV

Above Figure 6. illustrates generator Model IV constructed of a pure transformer which takes captions and mask vector as the input. Model IV is a version of Model III where we add several layers in the former part of the architecture. The combination of Transformer encoder followed by Pixel shuffle technique improves the upsampling procedure efficiently. In this model, we transform the input which we pass to the Stage Block module. We first start with our input captions and add the positional embeddings into it. Then we pass this output through a transformer encoder which is then flattened into a shape of Bx12288. Then, we reshape that tensor into Bx1x1x12288. Next, we perform the pixel shuffle technique with an upscale factor of 4 which upscales the dimensions to Bx16x768. This tensor is now passed to the Stage Block and proceed with the forward layers which are similar to Model III.

3.3.6 DISCRIMINATOR MODEL

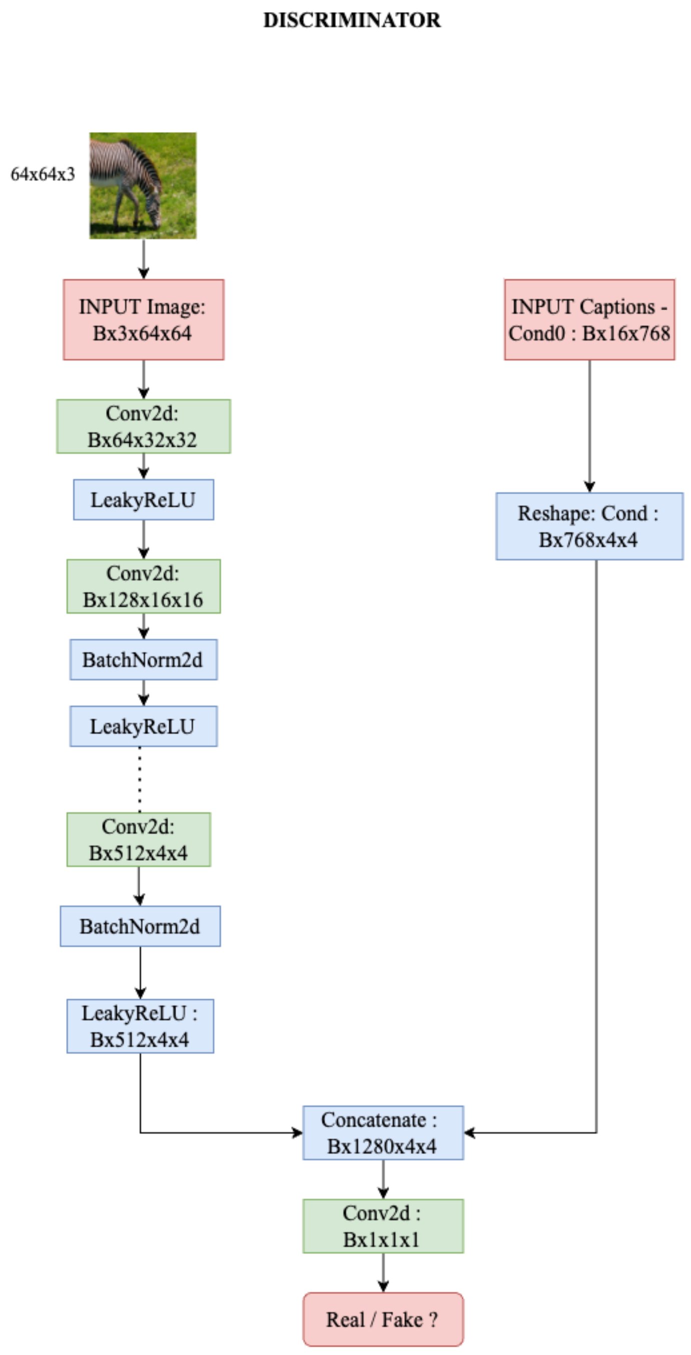


Figure 7: DISCRIMINATOR MODEL

The DISCRIMINATOR MODEL is constructed of convolutional layers stacked after each other. The input to this model is an image (can be generated image or real image from the dataset). Also, we need to send in the related caption as an input to the discriminator model.

Firstly, we scale down the image tensor from Bx3x64x64 to a resultant tensor of Bx512x4x4. We perform down sampling by placing Convolutional layers with Batch Normalization and leaky ReLU activation layers. Next, we reshape the captions input vector into a tensor of shape Bx768x4x4. Then we concatenate the transformed image information and the reshaped captions vector. This results into a vector of shape Bx1280x4x4 which is then forwarded to a convolutional layer which transforms all the information into Bx1x1x1. The output of the discriminator is a score indicating how real or fake the generated image is.

CHAPTER 4

IMPLEMENTATION DETAILS

4.1 TRANSFORMER ENCODER MODEL

Transformer Models have been exhaustively implemented in all our generator architectures of Model II, Model III and Model IV. A Transformer is a model that uses attention mechanism to boost the speed with which models are trained. The transformer contains encoding and decoding components. The encoding component contains stack of encoders and the decoding components contains stack of decoders [7].

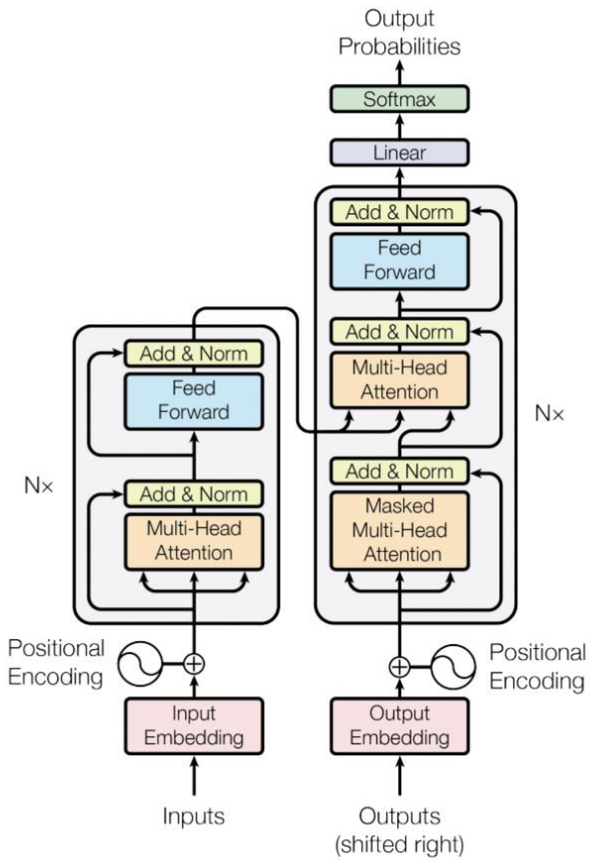


Figure 8: Transformer – Model Architecture

From the above Figure 8., the left part of the model is the encoder and the one on the right is the decoder component. We have used just the encoder part of the transformer in our implementation

4.2 BERT MODEL

BERT (Bidirectional Encoder Representations from Transformers) is a transformer encoder stack, which takes input a sequence of words which keeps flowing up the stack. Each layer applied the self-attention mechanism and passes the results through a feed forward network and then hands it on to the next encoder. We use the BERT Tokenizer and BERT Model to first tokenize the input captions and then generate word feature embeddings respectively. We perform BERT mechanism in the Data Preprocessing part [8].

4.3 PIXEL SHUFFLE

Pixel Shuffle is an operation used in super-resolution models to implement efficient sub-pixel convolutions with a stride of $1/r$. Specifically, it rearranges elements in a tensor of shape $(*, C \times r^2, H, W)$ to a tensor of shape $(*, C, H \times r, W \times r)$. This method achieves image super resolution in rather ingenious manner where the feature maps are extracted in a low resolution space[9] .

.

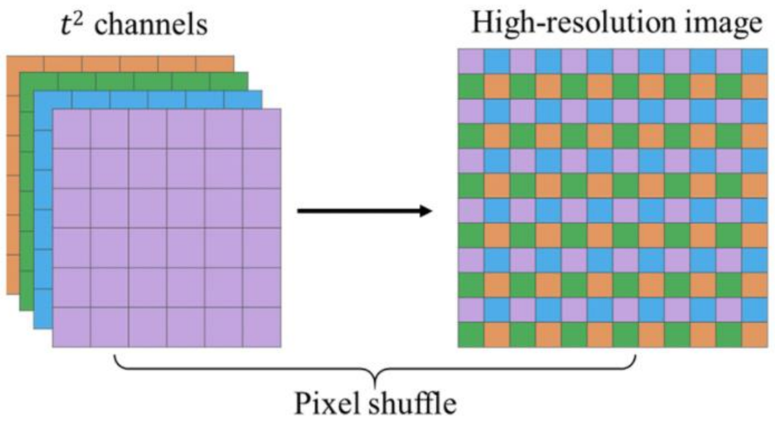


Figure 9: Pixel Shuffle Operation

4.4 TRAINING STRATEGY

We train all our models over the same dataset with a fixed train and test split for 1000 epochs. We keep the same DISCRIMINATOR model throughout our Text to image Transformer GAN and experiment with different sets of GENERATOR models viz. Model I, Model II, Model III, and Model IV. The batch size for our training and test implementations is set to 32.

We calculate two loss functions mainly the Discriminator loss and the Generator loss. The Discriminator loss is calculated as follows,

We first calculate the errorD_real as the output of discriminator which takes input as real image and its corresponding caption.

Then we get fake image as an output of the generator which has inputs the captions and its mask. Next, we get errorD_fake as the output of discriminator model which takes input as the above generated fake image and the captions vector. Thus, the total discriminator loss is the different between errorD_real and errorD_fake. Our objective is to maximize $D(x) - D(G(z))$.

Now, to calculate Generator loss, we first generate the fake image. Next, we get the Adversarial loss which is output of discriminator model with input as the fake image and captions vector. We also get Content loss which is just a MSE loss between the fake image and real image. Finally, we calculate the Total Generator loss as

$$\textit{Total Generator loss} = \textit{Adversarial Loss} + 100.0 \times \textit{Content Loss}$$

For both the loss terms we use the Adam optimizer with learning rate as 0.0002, momentum set to 0.5 and beta value of 0.999.

4.5 CODING STANDARDS

Pytorch: Pytorch is an open-source machine learning library for python that was developed by Facebook’s AI research group. It supports both CPU and GPU computations and offers scalable distributed training and performance optimization in research and production. We have performed out entire implemented using Pytorch utilities and libraries.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 QUANTITATIVE RESULTS

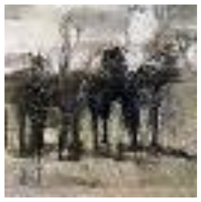
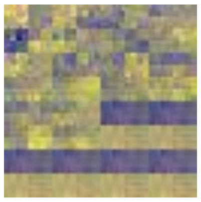
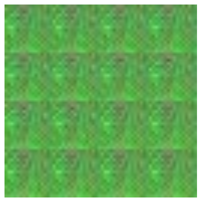
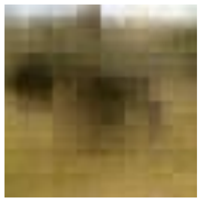
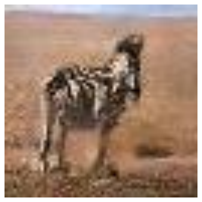
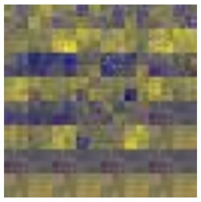
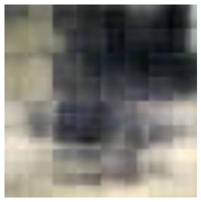
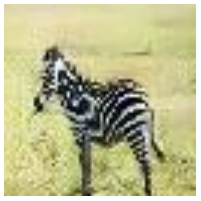
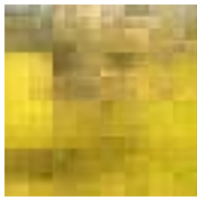
Table 1. QUANTITATIVE RESULTS

Model	**Inception Score ↑**	**Fréchet Inception Distance ↓**
Attn GAN	1.67	195.36
Model I	2.53	154.42
Model II	1.22	301.42
Model III	1.05	398.06
Model IV	1.63	324.49

Inception Score ↑ - denotes the higher the better
Fréchet Inception Distance ↓ - denotes the lower the better

5.2 QUALITATIVE RESULTS

Table 2. QUALITATIVE RESULTS

Image size: 64x64x3	**MODEL I**	**MODEL II**	**MODEL III**	**MODEL IV**	**GROUND TRUTH IMAGE**
Sentence I: a group of zebras are standing in a field.					
Sentence II: a zebra standing on a dry grass covered field.					
Sentence III: a zebra stands in a field of grass.					

5.3 DISCUSSION

The Quantitative results gives us a comparison between a novel Attn GAN architecture, a convolution-based generator Model I and transformer-based generators Model II, Model III and Model IV. We can infer that although we did not achieve state of the art results, the transformer-based generator models have managed to generate images. Out of the three transformer models, the model IV seems to achieve better performance and better-quality image looking at the Qualitative results. Model IV has achieved to learn the background and the most attentive field in the image accurately till now. The performance of these models can be increased by adding more layers, optimizing the hyper parameters of the network, etc.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

Performing text to image synthesis using transformer models shows us that transformers have vast potential in the computer vision domain. We examined the transformer model and replaced the convolutional layers in the generator architecture with transformer encoder architecture. We modeled different architecture models, trained them, and evaluated their performance on our input data. Upsampling techniques like Pixel Shuffle paired with transformer blocks gave us an alternative way to up scale the features. Although we tried a different approach to synthesize image from captions since our Transformer GAN hasn't got better performance yet.

6.2 FUTURE WORK

We state that constructing a transformer model for Discriminator can further achieve and increase in performance of the generated image quality. Doing so will make the entire GAN completely convolutions free. Fine tuning model hyper parameters, adding more layers to the model and implementing different training strategies like data augmentation, training on more data that is multiple classes of the MS COCO dataset and not just on a single class (Zebra class), would be the next step of our work. We have seen that transformers works brilliant in the NLP domain and that they have potential to generate photo realistic images, motivates us to try multiple techniques in the future.

REFERENCES

[1] T. Xu *et al.*, "AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1316-1324, doi: 10.1109/CVPR.2018.00143.

[2] T. Karras, S. Laine and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4396-4405, doi: 10.1109/CVPR.2019.00453.

[3] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen and T. Aila, "Analyzing and Improving the Image Quality of StyleGAN," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 8107-8116, doi: 10.1109/CVPR42600.2020.00813.

[4] Yifan Jiang, Shiyu Chang and Zhangyang Wang, " TransGAN: Two Transformers Can Make One Strong GAN", 2021 *arXiv eprint: 2102.07074*, available at: https://arxiv.org/pdf/2102.07074.pdf

[5] MS COCO Common Objects in Context, available at: https://cocodataset.org/#home

[6] Open CV, available at: https://opencv.org/

[7] Vaswani, Ashish and Shazeer, Noam and Parmar *et al.*, "Attention is All you Need", 2017 *Advances in Neural Information Processing Systems*

[8] Devlin et.al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 2019 *Association for Computational Linguistics*, doi: 10.18653/v1/N19-1423

[9] W. Shi *et al.*, "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 1874-1883, doi: 10.1109/CVPR.2016.207.