M. Tech. 2nd Semester Mini-Project Report

On

TEXT TO IMAGE GENERATION

Saurabh Kumar Singh

(222CS029)

Guide

Dr. Annappa B

Department of Computer science and Engineering,

NITK, Surathkal

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,

SURATHKAL, MANGALORE - 575025

April, 2023

DECLARATION

I hereby declare that the M. Tech. 2nd Semester **Mini-Project** Report entitled **Text To Image Geneartion** which is being submitted to the National Institute of Technology Karnataka Surathkal, in partial fulfilment of the requirements for the award of the Degree of **Master of Technology in Computer Science and Engineering** in the department of **Computer Science and Engineering**, is a bonafide report of the work carried out by me. The material contained in this Report has not been submitted to any University or Institution for the award of any degree.

Saurabh Kumar Singh
222CS029
Department of Computer Science and Engineering
NITK, Surathkal

Place: NITK, Surathkal.
Date:

CERTIFICATE

This is to certify that the M. Tech. 2nd Semester **Mini-Project** Report entitled **Text To Image Generation** submitted by **Saurabh Kumar Singh**, (Roll Number: 222CS029) as the record of the work carried out by him/her, is accepted as the M. Tech. 2nd Semester Mini-Project Report submission in partial fulfilment of the requirements for the award of degree of Master of **Master of Technology in Computer Science and Engineering** in the Department of **Computer Science and Engineering**.

Guide

Dr. Annappa B

Department of Computer Science and Engineering

NITK, Surathkal

Chairman - DPGC

Dr. Manu Basavaraju

Department of Computer Science and Engineering

NITK, Surathkal

Abstract

Generating synthetic images using text prompts is a fascinating area of research in the field of artificial intelligence. The ability to create an image from a text prompt has a wide range of potential applications, from generating concept art for video games and movies to creating visual aids for scientific research. In recent years, there has been a surge of interest in this area, with several generative models being developed to generate images from text prompts.

The aim of this work is to generate synthetic images using text prompts by utilizing the power of VQGAN and CLIP. I used VQGAN, a generative model that uses a codebook to generate images, and CLIP, a neural network that understands the content of images and text, to create a system that can generate images based on a given text prompt.

The results we obtained from my experiments show that my method is effective in generating synthetic images that match the given text prompt. The quality of the generated image depends on the complexity of the text prompt and the size of the VQGAN codebook. By adjusting these parameters, I was able to generate images that range from simple sketches to highly detailed photorealistic images.

In conclusion, the ability to generate synthetic images from text prompts has a wide range of potential applications and is an exciting area of research in the field of artificial intelligence. VQGAN and CLIP are powerful tools for generating high-quality images that match the given text prompt. By combining these two models, we can create a wide range of images that are tailored to specific text prompts, from simple sketches to highly detailed photorealistic images.

Keywords: Synthetic images, Text prompts, VQGAN, CLIP, Generative models, Artificial intelligence, Neural network, Feature vectors, Codebook, Image generation, Deep learning.

Contents

List of Figures

List of Tables

1 Introduction

The ability to generate synthetic images from text prompts has significant potential in various fields such as entertainment, advertising, and scientific research. In recent years, the advancements in deep learning and artificial intelligence have made it possible to create generative models that can generate images based on textual input. This area of research is fascinating because it offers new possibilities for creativity and innovation in fields where visual representation is crucial.

VQGAN and CLIP are two powerful models that have gained significant attention for their ability to generate high-quality synthetic images from text prompts. VQGAN uses a codebook to generate images, while CLIP is a neural network that understands the content of images and text. By combining these two models, it is possible to generate images that match the given text prompt with a high degree of accuracy.

The current state of relevant knowledge suggests that the combination of VQGAN and CLIP can significantly improve the ability to generate synthetic images from text prompts. This approach offers new possibilities for creating images that are tailored to specific needs, and it can also help to reduce the time and cost associated with creating images manually. Overall, the development of generative models for image creation based on text prompts is a promising area of research with significant potential for innovation and advancement.

The scope of this investigation is to delve deeper into the capabilities of VQGAN and CLIP in generating synthetic images from text prompts. The objective is to develop a system that can create images based on textual input with a focus on the quality and accuracy of the generated images. This investigation aims to assess the effectiveness of VQGAN and CLIP in creating synthetic images and to identify potential applications of the generated images.

The investigation will evaluate the quality and accuracy of the synthetic images generated by VQGAN and CLIP. The quality and accuracy of the images are crucial factors that determine the potential applications of the images. By analyzing the generated images, it is possible to identify the strengths and limitations of the models and the areas for improvement.

Furthermore, the investigation aims to identify potential applications of the generated images. The ability to generate synthetic images from text prompts offers exciting possibilities for various fields such as advertising, entertainment, and scientific research. By identifying potential applications, it is possible to determine the relevance and usefulness of the generated images.

Overall, the investigation will explore the potential of VQGAN and CLIP in generating synthetic images from text prompts and the effectiveness of the models in creating high-quality and accurate images. The investigation will also identify potential applications of the generated images, offering insights into the practicality and relevance of the technology.

2 Literature Survey

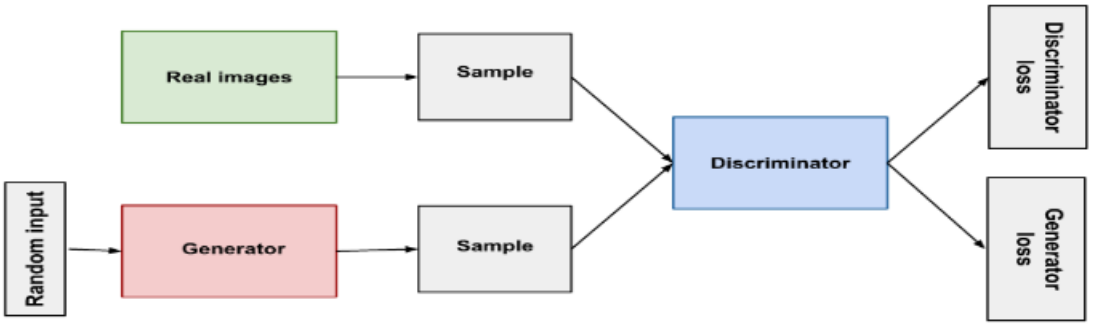

Figure 1: GAN Goodfellow et al. [2014]

The paper "Generative Adversarial Networks" by Ian Goodfellow et al. Goodfellow et al. [2014], published in 2014, introduced the revolutionary concept of Generative Adversarial Networks (GANs). GANs are a type of neural network architecture consisting of two components, a generator and a discriminator, that are trained in an adversarial manner to generate realistic data.The paper proposed a novel approach to training GANs, where the generator learns to generate synthetic data by trying to fool the discriminator, while the discriminator learns to distinguish between real and fake data. Through this adversarial training process, the generator and discriminator improve iteratively, leading to the generation of increasingly realistic data.The authors demonstrated the effectiveness of GANs in generating realistic images, including MNIST digits and face images, showing that GANs can learn to generate diverse and visually appealing samples. They also discussed various potential applications of GANs, such as image synthesis, image manipulation, and data augmentation.

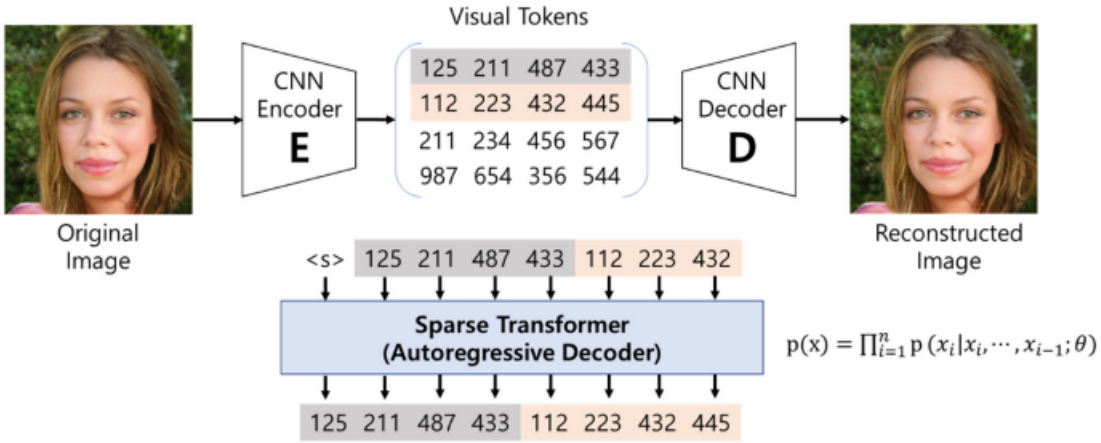

Figure 2: Proposed Architecture Im and Seo [2021]

The paper "Generating Face Images Using VQGAN and Sparse Transformer"Im and Seo [2021] presents a novel approach for generating face images using a combination of VQGAN and Sparse Transformer, published in 2021. The authors propose a two-step pipeline that combines the benefits of both models to generate high-quality and diverse face images.The authors conduct extensive experiments and evaluations to demonstrate the effectiveness of their approach. They show that their method generates diverse and visually appealing face images with high fidelity and resolution. They also compare their approach with other state-of-the-art methods and show that their method outperforms them in terms of image quality and diversity.The results of their experiments demonstrate the potential of their approach for generating realistic and diverse face images, opening up possibilities for various applications such as virtual avatars, character design, and facial recognition systems.

The paper "Learning Transferable Visual Models From Natural Language Supervision" Radford et al. [2021] presents a novel approach for training visual models using natural language supervision, published in 2018. The authors propose a framework that leverages large-scale image-text data to learn visual representations that are transferable across different visual tasks.The key idea of the paper is to use natural language supervision as a source of supervision for training visual models. The authors

collect a massive dataset of images and their associated textual descriptions from the web, which serves as the supervision signal for training their visual models. They use this data to train a deep neural network to learn visual features that are aligned with the semantic concepts described in the text.The authors demonstrate the effectiveness of their approach on various visual tasks, including image classification, object detection, and image generation. They show that their models trained with natural language supervision outperform models trained with other types of supervision, such as manual annotations or unsupervised methods, on a wide range of visual tasks.

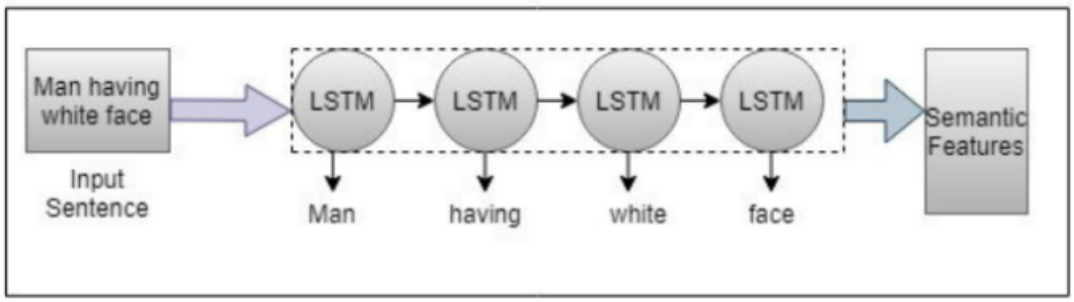

Figure 3: Encoder Architecture Khan et al. [2021]

The paper "A Realistic Image Generation of Face From Text Description Using the Fully Trained Generative Adversarial Networks" Khan et al. [2021]presents a method for generating realistic face images from text descriptions using fully trained Generative Adversarial Networks (GANs), published in 2019. The authors propose a two-step pipeline that combines a text-to-image synthesis model with a pre-trained GAN to generate high-quality face images from textual descriptions.The authors use a pre-trained GAN that has been trained on a large dataset of real face images, and fine-tune it with the generated face images from the first step. This refinement step helps to improve the quality and realism of the generated face images.The authors conduct extensive experiments and evaluations to demonstrate the effectiveness of their approach. They show that their method generates realistic and visually appealing face images that are highly correlated with the input text descriptions. They also compare their approach with other state-of-the-art methods and show that their method outperforms them in terms of image quality and visual coherence.

3 System Design

3.1 VQGAN :-

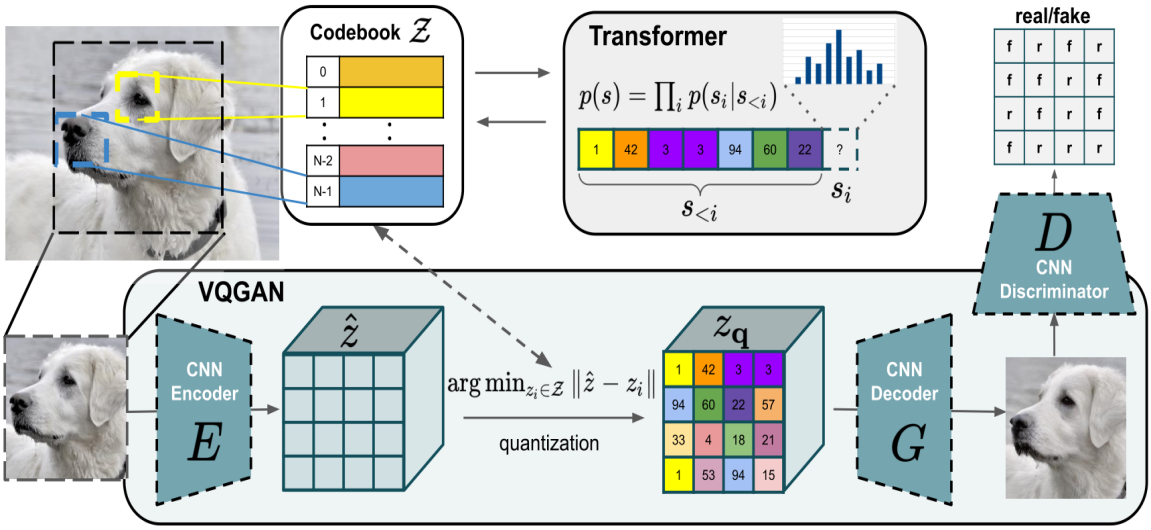

Figure 4: VQGAN

The VQGAN system design involves preprocessing the input image, encoding the image using VQGAN Encoder to generate a codebook representation, decoding the codebook representation using VQGAN Decoder to generate a high-quality image, quantizing the codebook representation, replacing the quantized codebook representation with learned codebook vectors, and outputting the final synthetic image. The VQGAN architecture is a powerful generative model that can create high-quality synthetic images.

3.1.1 Image Preprocessing

The first step is to preprocess the input image. This includes resizing the image to a fixed resolution and normalizing the pixel values to a range between -1 and 1. This step is important as it helps in reducing the computational load of the model and ensures that the input images have a consistent format.

3.1.2 VQGAN Encoder

The preprocessed image is then fed into the VQGAN Encoder. The encoder is a convolutional neural network (CNN) that compresses the input image into a codebook representation. The encoder consists of multiple convolutional and pooling layers, which gradually reduce the spatial dimensions of the input image while increasing the number of channels. The output of the encoder is a tensor of shape (batch-size, codebook-size, codebook-dim), where codebook-size is the number of codebook vectors, and codebook-dim is the dimensionality of each codebook vector.

3.1.3 VQGAN Decoder

The codebook representation generated by the encoder is then fed into the VQGAN Decoder. The decoder is also a CNN that uses a series of deconvolutional and up-sampling layers to generate an output image. The decoder takes the codebook representation as input and generates an image of the same size as the input image. The output image is a high-quality synthetic image that matches the input image.

3.1.4 Codebook Quantization

The codebook representation generated by the VQGAN Encoder is quantized to a finite set of codes using a nearest neighbor algorithm. This ensures that the codebook representation can be efficiently stored and manipulated. In this step, each codebook vector is replaced with its nearest neighbor in the codebook.

3.1.5 Codebook Replacements

The quantized codebook representation is replaced with the corresponding codebook vectors, which are learned during training. This step improves the quality of the generated image by introducing more variation in the codebook representation. In this step, each quantized codebook vector is replaced with the learned codebook vector that is closest to it.

3.1.6 Output Images

The final output of the system is a high-quality synthetic image that matches the input image. The generated image can be saved to disk or displayed on a web page. This step concludes the VQGAN architecture.

3.2 CLIP :-

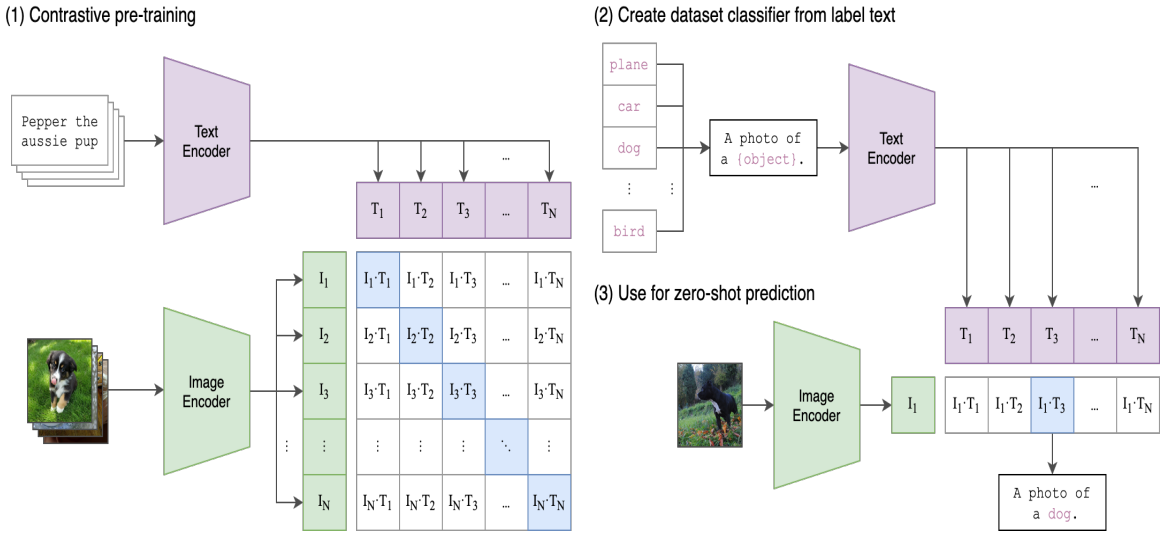

Figure 5: CLIP

CLIP is a neural network architecture designed to understand the relationship between images and text. It consists of a vision encoder and a text encoder. The vision encoder generates a feature vector that represents the content of the image, while the text encoder generates a feature vector that represents the meaning of the text. CLIP learns to map images and their corresponding textual descriptions into a shared feature space through a contrastive learning approach. The CLIP model can be used for a variety of tasks, such as image classification, image retrieval, and image generation. It is trained to maximize the similarity between the image and text representations for a given pair of corresponding images and texts, while minimizing the similarity between the representations of non-corresponding pairs. The learned representations of images and text can be used for a variety of downstream tasks, making CLIP a powerful tool for image-text understanding.

3.2.1 Preprocessing

The input image and text are preprocessed to ensure that they are in a suitable format for the neural network. This may include resizing the image to a fixed size, converting it to the appropriate color space, and normalizing the pixel values. For text, it may involve tokenizing the text into individual words and converting it into a numerical representation, such as word embeddings.

3.2.2 Image Encoder

The image is passed through a convolutional neural network (CNN) to extract features and generate an image embedding. The CNN may be a pre-trained network, such as ResNet or EfficientNet, or it may be trained from scratch. The CNN architecture typically consists of multiple layers of convolutional, pooling, and activation functions that progressively transform the image into a set of high-level features.

3.2.3 Text Encoder

The text is passed through a transformer-based neural network, such as BERT or GPT, to extract features and generate a text embedding. Transformers are a type of neural network architecture that have achieved state-of-the-art performance in natural language processing tasks. The transformer architecture consists of multiple layers of self-attention and feed-forward neural networks that capture the relationships between the words in the text.

3.2.4 Embedding Space

The image and text embeddings are mapped into a shared embedding space using a projection layer. This allows for direct comparison between the image and text embeddings. The projection layer typically consists of a linear transformation followed by a non-linear activation function, such as ReLU or sigmoid, that maps the embeddings into the same dimensional space.

3.2.5 Contrastive Loss

The image-text pairs are used to train the network using a contrastive loss function. This loss function ensures that similar image-text pairs are mapped close together in the embedding space, while dissimilar pairs are mapped far apart. The contrastive loss is computed by comparing the distance between the embeddings of the positive pairs (image-text pairs that match) and the distance between the embeddings of the negative pairs (image-text pairs that do not match). The network is optimized to minimize the distance between the embeddings of the positive pairs and maximize the distance between the embeddings of the negative pairs.

3.2.6 Inference

During inference, the CLIP architecture can be used for a variety of tasks, such as image classification, object detection, and natural language processing. For example, to perform image classification, the image is passed through the image encoder to generate an image embedding, which is then compared to a set of pre-defined class embeddings in the embedding space to determine the most likely class. Similarly, to perform image retrieval, the text query is passed through the text encoder to generate a text embedding, which is then compared to the image embeddings in a database to find the most similar images.

4 Implementation

The code implementation trains a Generative Adversarial Network (GAN) to generate images based on a given prompt using the CLIP and VQGAN models. The steps involved in the implementation are as follows:

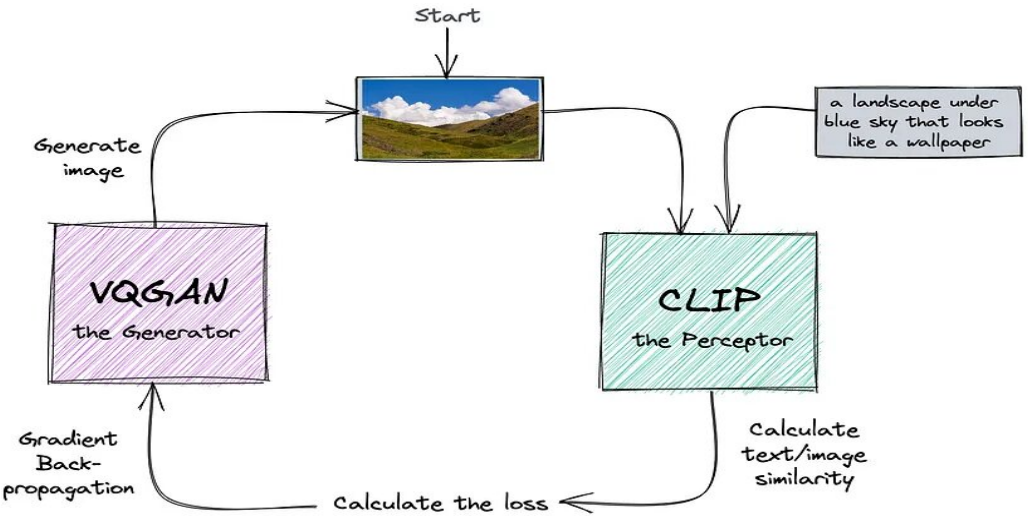

Figure 6: Implementation Flow of Project v1

- The first step is to clone the required GitHub repositories using the Git command. Two repositories are cloned: "CLIP" and "taming-transformers". CLIP is a neural network model for natural language understanding and VQGAN (Vector Quantized Generative Adversarial Network) is a generative model used to generate images.

```
!git clone https://github.com/openai/CLIP.git
!git clone https://github.com/CompVis/taming-transformers
```

Figure 7: Cloning Models

- Second step is to import required libraries such as numpy, torch, os, imageio, PIL, matplotlib, yaml, and OmegaConf. It also imports clip from the CLIP repository.

```
# import libraries
import numpy as np
import torch, os, imageio, pdb, math
import torchvision
import torchvision.transforms as T
import torchvision.transforms.functional as TF

import PIL
import matplotlib.pyplot as plt

import yaml
from omegaconf import OmegaConf

from CLIP import clip

#import warnings
#warnings.filterwarnings('ignore')
```

Figure 8: Importing Libraries

- Third step is defining two helper functions: show_from_tensor() and norm_data().

  show_from_tensor() function displays the tensor image. The function takes a tensor as input, multiplies it by 255, converts it into a byte tensor, converts it to a numpy array, transposes it to the shape (1, 2, 0), and displays it using the imshow() function.

  norm_data() function normalizes the given data between -1 and 1, and returns the result. The function takes a data array as input, clips it between -1 and 1, adds 1 to it, and divides the result by 2.

  Setting up several parameters such as the learning rate, batch size, weight decay, noise factor, total iterations, and image shape.

```
## helper functions

def show_from_tensor(tensor):
  img = tensor.clone()
  img = img.mul(255).byte()
  img = img.cpu().numpy().transpose((1,2,0))

  plt.figure(figsize=(10,7))
  plt.axis('off')
  plt.imshow(img)
  plt.show()

def norm_data(data):
  return (data.clip(-1,1)+1)/2 ### range between 0 and 1 in the result

### Parameters
learning_rate = .5
batch_size = 1
wd = .1
noise_factor = .22

total_iter=400
im_shape = [450, 450, 3] # height, width, channel
size1, size2, channels = im_shape
```

Figure 9: Defining Functions And Parameters

- Loading the ViT-B/32 CLIP model and sets it to evaluation mode.

```
[ ] ### CLIP MODEL ###
    clipmodel, _ = clip.load('ViT-B/32', jit=False)
    clipmodel.eval()
    print(clip.available_models())

    print("Clip model visual input resolution: ", clipmodel.visual.input_resolution)

    device=torch.device("cuda:0")
    torch.cuda.empty_cache()
```

Figure 10: Loading CLIP model in evaluate mode

- We then instantiate the taming transformer by moving to the taming-transformers directory and loading the configuration file and the checkpoint file for the vq-gan_imagenet_f16_16384 model.

```
from taming.models.vqgan import VQModel

def load_config(config_path, display=False):
   config_data = OmegaConf.load(config_path)
   if display:
     print(yaml.dump(OmegaConf.to_container(config_data)))
   return config_data

def load_vqgan(config, chk_path=None):
  model = VQModel(**config.model.params)
  if chk_path is not None:
    state_dict = torch.load(chk_path, map_location="cpu")["state_dict"]
    missing, unexpected = model.load_state_dict(state_dict, strict=False)
  return model.eval()

def generator(x):
  x = taming_model.post_quant_conv(x)
  x = taming_model.decoder(x)
  return x

taming_config = load_config("./models/vqgan_imagenet_f16_16384/configs/model.yaml", display=True)
taming_model = load_vqgan(taming_config, chk_path="./models/vqgan_imagenet_f16_16384/checkpoints/last.ckpt").to(device)
```

Figure 11: Initiating Taming Transformers

- Next,define a Parameters class to optimize the values and define the required functions.

```
### Declare the values that we are going to optimize

class Parameters(torch.nn.Module):
  def __init__(self):
    super(Parameters, self).__init__()
    self.data = .5*torch.randn(batch_size, 256, size1//16, size2//16).cuda() # 1x256x14x15 (225/16, 400/16)
    self.data = torch.nn.Parameter(torch.sin(self.data))

  def forward(self):
    return self.data

def init_params():
  params=Parameters().cuda()
  optimizer = torch.optim.AdamW([{'params':[params.data], 'lr': learning_rate}], weight_decay=wd)
  return params, optimizer
```

Figure 12: Parameters for Optimization

- Then define the encodeText function to encode the given text using CLIP and return the encoded tensor.Then we define the createEncodings function to create encodings for include, exclude, and extras text values alongwith augTransform object to apply data augmentation to the generated images.

```
### Encoding prompts and a few more things
normalize = torchvision.transforms.Normalize((0.48145466, 0.4578275, 0.40821073), (0.26862954, 0.26130258, 0.27577711))

def encodeText(text):
  t=clip.tokenize(text).cuda()
  t=clipmodel.encode_text(t).detach().clone()
  return t

def createEncodings(include, exclude, extras):
  include_enc=[]
  for text in include:
    include_enc.append(encodeText(text))
  exclude_enc=encodeText(exclude) if exclude != '' else 0
  extras_enc=encodeText(extras) if extras !='' else 0

  return include_enc, exclude_enc, extras_enc

augTransform = torch.nn.Sequential(
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.RandomAffine(30, (.2, .2), fill=0)
).cuda()
```

Figure 13: Encodings And Augmentation

- At last we calculate loss value by comparing the similarities between encodings.

```
def optimize_result(Params, prompt):
  alpha=1 ## the importance of the include encodings
  beta=.5 ## the importance of the exclude encodings

  ## image encoding
  out = generator(Params())
  out = norm_data(out)
  out = create_crops(out)
  out = normalize(out) # 30 x 3 x 224 x 224
  image_enc=clipmodel.encode_image(out) ## 30 x 512

  ## text encoding  w1 and w2
  final_enc = w1*prompt + w1*extras_enc # prompt and extras_enc : 1 x 512
  final_text_include_enc = final_enc / final_enc.norm(dim=-1, keepdim=True) # 1 x 512
  final_text_exclude_enc = exclude_enc

  ## calculate the loss
  main_loss = torch.cosine_similarity(final_text_include_enc, image_enc, -1) # 30
  penalize_loss = torch.cosine_similarity(final_text_exclude_enc, image_enc, -1) # 30

  final_loss = -alpha*main_loss + beta*penalize_loss

  return final_loss

def optimize(Params, optimizer, prompt):
  loss = optimize_result(Params, prompt).mean()
  optimizer.zero_grad()
  loss.backward()
  optimizer.step()
  return loss
```

Figure 14: loss

- Generating a sample image using the initial random Params() and displaying it.

```
### training loop

def training_loop(Params, optimizer, show_crop=False):
  res_img=[]
  res_z=[]

  for prompt in include_enc:
    iteration=0
    Params, optimizer = init_params() # 1 x 256 x 14 x 25 (225/16, 400/16)

    for it in range(total_iter):
      loss = optimize(Params, optimizer, prompt)

      if iteration>=80 and iteration%show_step == 0:
        new_img = showme(Params, show_crop)
        res_img.append(new_img)
        res_z.append(Params()) # 1 x 256 x 14 x 25
        print("loss:", loss.item(), "\niteration:",iteration)

      iteration+=1
    torch.cuda.empty_cache()
  return res_img, res_z
```

Figure 15: training loop

5 Results

```
torch.cuda.empty_cache()
include=['Pineapple in a bowl']
exclude='watermark'
extras = ""
w1=1
w2=1
noise_factor= .22
total_iter=110
show_step=10 # set this to see the result every 10 interations beyond iteration 80
include_enc, exclude_enc, extras_enc = createEncodings(include, exclude, extras)
res_img, res_z=training_loop(Params, optimizer, show_crop=True)
```

Figure 16: Text Prompt

Number of Iterations = 110

Learning Rate = 0.5

Optimizer = AdamW

Activation Function = ReLU

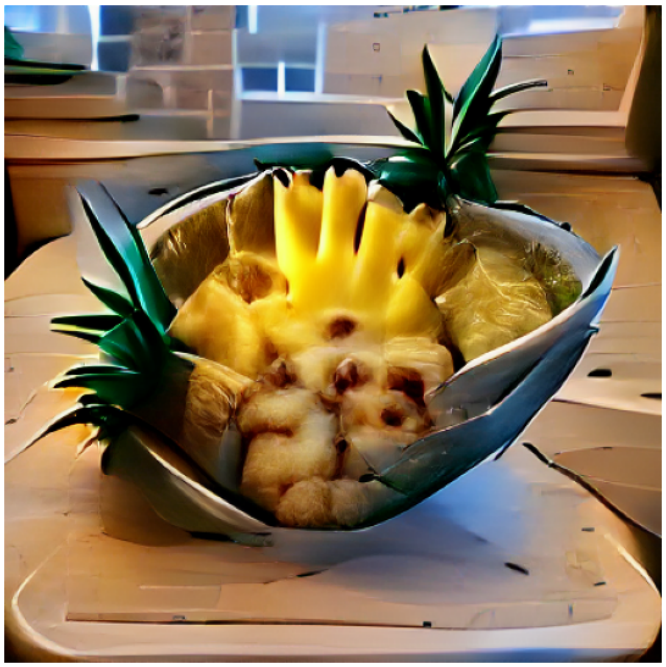

Figure 17: Output Image

Below is the detailed table of results with the same text prompt and different parameters and activation functions.

Table 1: Output Of Experiments

Activation Function	Lr = 0.5	Lr = 0.9	Lr = 3
RELU	0.7122 	0.7039 	0.7088 
Leaky RELU Negative Slope (0.2)	0.6746 	0.6985 	0.7173 
tanh	0.6973 	0.7161 	0.7247 
SIGMOID	0.6910 	0.7037 	0.7112 
SOFTMAX	0.666 	0.687 	0.7303 

6 Conclusions and Future Work

In conclusion, CLIP+VQGAN is a powerful combination of two deep learning models that have been used for creative image generation. Clip is a language-based model that understands images through text prompts, while VQGAN is an image-based model that can generate high-quality images from latent codes. When combined, CLIP+VQGAN allows users to generate diverse and visually appealing images by providing text prompts that describe the desired image content.

Based on my experimentation with CLIP+VQGAN in my project, I have concluded that a learning rate of 0.5, using the AdamW optimizer, and ReLU activation function yield the best results. The learning rate of 0.5 provides a good balance between fast convergence and avoiding overshooting during training. The AdamW optimizer, which incorporates weight decay, helps in regularizing the model and improving its generalization performance. The ReLU activation function, known for its ability to mitigate the vanishing gradient problem, has shown to work well in generating visually appealing images with CLIP+VQGAN.

Future work with CLIP+VQGAN could focus on several areas to further enhance its capabilities. One potential direction is exploring advanced model architectures, such as incorporating attention mechanisms, recurrent or transformers layers, or exploring novel ways to improve the generation performance. Another area of interest could be dataset augmentation techniques, such as data synthesis or domain adaptation, to improve the diversity and quality of training data. Additionally, extending CLIP+VQGAN to support conditional image generation, where users can have more fine-grained control over generated images based on additional inputs or metadata, could open up new possibilities for creative image synthesis. Enhancing the interpretability and controllability of CLIP+VQGAN through techniques such as visualizing and understanding the model's internal representations, or providing more user-friendly control over image styles or attributes, could also be an important avenue for future work. Furthermore, ethical considerations, such as addressing biases, fairness, and responsible use of AI-generated images, could be integrated into future

research to ensure the ethical and responsible deployment of CLIP+VQGAN. Finally, exploring real-world applications of CLIP+VQGAN beyond creative domains, such as advertising, fashion, gaming, or virtual reality, could uncover new use cases and practical applications for this technology. Continued research and development in these areas could further advance the capabilities of CLIP+VQGAN and broaden its potential applications in various domains.

Bibliography

Alexa Steinbrück multimodel description. https://alexasteinbruck.medium.com/explaining-the-code-of-the-popular-text-to-image-algorithm-vqgan\-clip-a0c48697a7ff.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Dong-Hyuck Im and Yong-Seok Seo. Generating face images using vqgan and sparse transformer. In *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1642–1644, 2021. doi: 10.1109/ICTC52510.2021.9621202.

Muhammad Zeeshan Khan, Saira Jabeen, Muhammad Usman Ghani Khan, Tanzila Saba, Asim Rehmat, Amjad Rehman, and Usman Tariq. A realistic image generation of face from text description using the fully trained generative adversarial networks. *IEEE Access*, 9:1250–1260, 2021. doi: 10.1109/ACCESS.2020.3015656.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.