Improving Image Generation Model

Team Number - 12

Jiayu Yuan jyuan60@asu.edu
Chun-Chih Yang cyang218@asu.edu
Robel Abdissa rabdissa@asu.edu
Bala Sujith Potineni bpotinen@asu.edu
Chi-Ao Chen cchen412@asu.edu

Abstract

As image generation models are continuing to play increasingly pivotal role across different domains these days, it's consequentially has given rise to the pressing need for coming up with strategies that provide a more compact models that don't come with the expense of compromising accuracy. The main goal of this project is to augment the performance and versatility of the VQGAN+CLIQ model. By meticulously fine-tuning its parameters and rigorously optimizing its architectural framework, we aim to attain an improved performance. As such by employing detailed examination and adjustments, and not just by relying on tweaking the underlying setup, this project endeavor makes the effort to push the limits of the particular model. Consequently expanding the utility of the VQGAN+CLIP model. Therefore the results of the project we conduct will have significant impact, taking into account the very many possible applications of models like the VQGAN+CLIQ, such as spanning image generation, style transfer and manipulation, and redefining contours of computation aesthetics. In addition, we also tried to explore designing a new image generative model leveraging text and image decoders of CLIP.

1 Introduction

As the impact and influence of image generation models is continuously increasing, the need for exploring ways for optimizing their size without compromising accuracy is becoming very critical. Notably, reducing model scale holds the potential to not only enhance training efficiency but also spur further innovation in image-related fields, such as the development of immersive role-playing games. We selected this topic to address the challenge of scaling down training models while preserving accuracy levels. The importance of the project has a main effect in the various applications that take effect in models like the VQGAN+CLIP, for spanning image generation, style transfer, and manipulation. As such with the architectural enhancements and parameter fine-tuning we conduct the all-encompassing final goal of the project will be to significantly enhance the model's adaptability and efficacy across varying tasks and datasets, with a transformative advancements in computational visual aesthetics.

2 Project Description

2.1 Problem Statement

The main problem we are addressing in this project is the optimization of the VQGAN+CLIP image generation model. Despite its impressive performance in generating images that align with given textual descriptions, the VQGAN+CLIP model is computationally expensive and time-consuming to train due to its large scale. This poses a challenge for applications that require efficient training processes, such as the development of role-playing games we mentioned above. Furthermore, the model's versatility across different tasks and datasets could be improved. Therefore, our research goal is to optimize the scale and performance of the VQGAN+CLIP model without compromising its accuracy too much.

2.2 Technical Background

The VQGAN+CLIP model is a deep learning model that combines Vector Quantized Generative Adversarial Networks (VQGAN) and Contrastive Language–Image Pretraining (CLIP). VQGAN uses a method of vector quantization to generate images, while CLIP is a model that can understand both images and text. By combining these two tools, users can generate or modify images through natural language descriptions, achieving an intuitive mapping from text to image. For example, user can input a text to CLIP, such as “A painting of an apple in a fruit bowl”, then the CLIP model can guide the VQGAN to generate an image that matches the text as in the Figure 1 shown. The application prospects of VQGAN+CLIP are very broad, such as virtual reality, game development, art creation, etc.

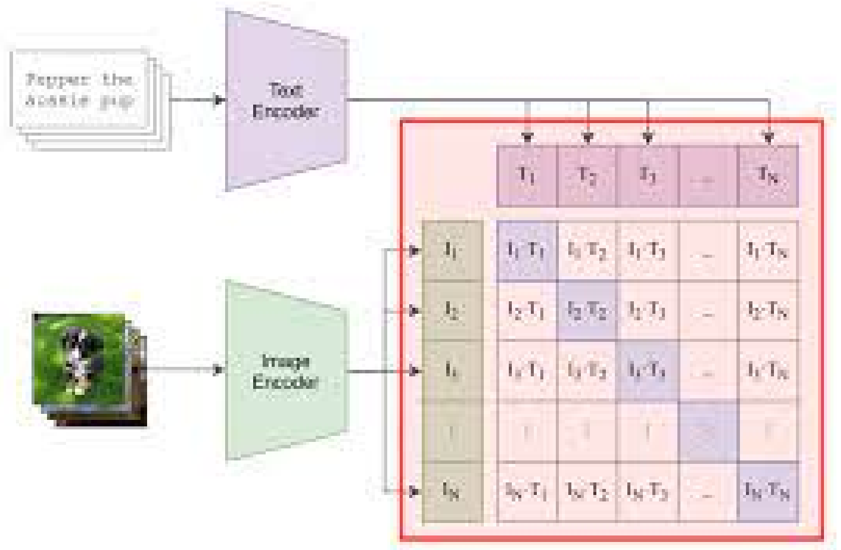
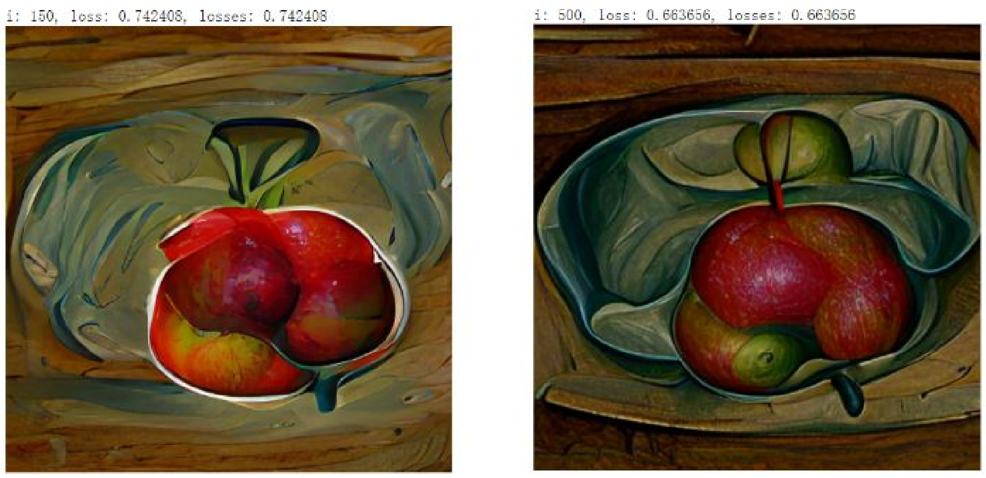


Figure 1: VQGAN+CLIP Architecture

2.3 Related Work

We have found several researches conducted on optimizing deep learning models. One such work was conducted by Song Han, who proposed a methodology called "deep compression", which reduces the size and runtime of a model drastically all the while maintaining a reasonable accuracy. The deep compression method involves pruning unimportant connections in the neural network, quantizing the weights, and employing Huffman coding to compress the weight distribution. But the drawback is that it is yet to be applied to the VQGAN+CLIP model. Also, the impact of the transformer model, by Vaswani et al, extends beyond natural language processing and into the domain of computer vision, specifically influencing the development of models like VQGAN (Vector Quantized Generative Adversarial Network).

Another set of research prioritizes on the improvement of the architecture of the model with the aim of enhancing the performance on a wide variety of tasks and datasets. The EfficientNet model which falls under this category, scales up the network depth, width and resolution in a balanced way to improve the model's accuracy and efficiency. However, it remains an open question how these architectural improvements can be incorporated into the VQGAN+CLIP model to optimize its scale and performance.



```
i: 150, loss: 0.742408, losses: 0.742408
```

```
i: 500, loss: 0.663656, losses: 0.663656
```

Figure 2: Text prompt: "A painting of an apple in a fruit bowl", generated by VQGAN(i is iterations in generation)

3 Approach

3.1 Data Collection and Preprocessing

Evaluating original VQGAN+CLIP model used in this project is built upon the foundational work of Katherine Crowson, specifically her project, "Generate images from text prompts with VQGAN and CLIP (z+quantize method)."[5] In the process of loading the model, we introduced several adjustments to the preprocessing of the image dataset to enhance the diversity and richness of the inputs for the model. These modifications were meticulously chosen to simulate a wide range of variations in the images, thereby enriching the model's ability to generate detailed and varied outputs. For the purpose of distilling CLIP Image Decoder and creating new generative model, we imported the old CLIP model and utilized the helper functions of CLIP to train our model. The dataset used in training the new models is open-source MSCOCO dataset of captioned images. Its size if 118287. The imported images are passed through a custom preprocess function to resize to 224 by 224 RGB image to feed to the model.

3.2 Loading the pre-trained model and modify to our task

By mapping the model parameters to more descriptive names, we've clarified the dataset on which each VQGAN model variant was trained, aiding users in making informed decisions[2]. We also accommodated optional inputs such as seeds for randomness control and the initiation of the process with specific images, enhancing the flexibility and diversity of the generated outcomes.

Consolidating these parameters into a 'name' object was a move to facilitate easier management and transmission of configurations into the image generation function. This decision promotes a more streamlined and efficient process, encouraging users to explore the wide array of creative possibilities that VQGAN models offer. We provide the following test parameters to the model, (text: "birds is flying in the sunny day's sky", width: "400", height: "400", model: "vqgan-imagenet-f16-16384", seed: "42", max-iterations: "500"). The result is depicted in Figure 1.

3.3 Creating ImageEmbeddingNetwork

We successfully designed and implemented the ImageEmbeddingNetwork architecture, aiming to replace the computationally expensive Vision Transformer (ViT) component in Image Decoder of CLIP. The Teacher Model (ViT) uses a 12-layer Transformer encoder architecture with self-attention layers (heads=12) for capturing relationships between image patches. It optionally has a patch embedding layer, positional encoding layer, and an MLP head for image embeddings. The Student Model (Image Embedding Network) uses convolutional layers to extract features, pooling layers to

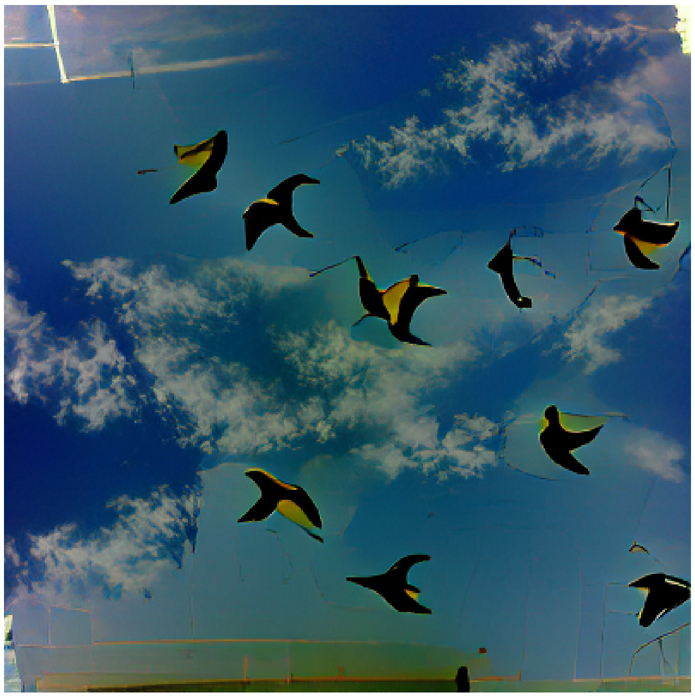


Figure 3: "birds is flying in the sunny day's sky"

reduce complexity, a flattening layer to transform features into a vector, and fully-connected layers to learn relationships between features and generate the 512-dimensional embedding[4]. The number of convolutional and fully-connected layers is not specified, but there are at least 3 convolutional layers, 2 pooling layers, and 2 fully-connected layers. The implementation can be found at: `https://colab.research.google.com/drive/1zhlNjlj3k6U88EhYWRZxn0cYFS-vBiC6?%20usp=sharing#scrollTo=6fXGYcYKcE5T`

3.4 Training on MSCOCO Dataset

The MS COCO dataset comprises of a vast collection of images alongside respective descriptive captions, which is ideal for image captioning tasks that require large-scale data aggregation. The MS COCO dataset has an approximated 118,000 images in the category of the training subset, while it has about 5,000 images in the subset designated for validation, where these subsets have each 5 captions. As such, the dataset serves as a rich source of linguistic and visual data for model training, of particular value to our project. Our preprocessing step here involves the process of tokenizing the captions using the CLIP tokenizer(clip.tokenize), to make sure the seamless integration and application of our dataset into our machine learning architecture. In the data loading step we extract captions and image informations, where it doesn't clearly depict the image loading process, which is included in the training loop. Hence given the diverse images and descriptive captions provided by the MS COCO dataset, it offers a good training ground for learning the image generation task.

3.5 Generating Images from Text Descriptions

The main task in generating images from text descriptions is to train a generative model that can produce images that align with natural language descriptions. This is achieved by employing the concepts of Transformers used for processing both text and images, in a way that extends the capabilities beyond traditional language tasks, CLIP, a pre-trained model that unifies text and images within a shared representational space, which enhances comparison between the two modalities, and finally Contrastive Learning, which is used to in a way direct model to generate images with representation close to their corresponding text descriptions.

3.5.1 Component Breakdown

The generative model we implemented can be broken down into 4 main sub-components, which work in unison to train a generative model intended to produce images that align with their respective natural language descriptions. The data preparation step, as alluded to earlier provides the MS COCO dataset with corresponding images and captions. The second subcomponent, which is the generator model, consists of 3 layers. The first one is a linear layers(blocks), which is a series of linear layers that form a basic building block of the generator. The second is the transformer, which is built from ResidualAttentionBlocks, which is used to process sequences of data, with the goal of capturing relationships between the elements generated by the linear layers. Finally, the convolutional layer, which is a series of nn.Conv2d layers with upsampling, and are used for refining the outputs from the transformer into images with their respective details. The second subcomponent, which is the CLIP model (a pre-trained, and loaded model), embeds the images and text into a common representational space. In the training loop, the text descriptions are encoded into embeddings with the help of CLIP's text encoder. And the generator model produces images based on these provided embeddings. Afterwards, the generated images are also encoded into embeddings using CLIP's image encoder. The contrastive loss penalizes large distances between text embeddings with their corresponding image embeddings, and the Adam optimizer updates the Generator model's parameters to improve on the image generation.

3.5.2 How the model works

After the training step, the image generation works as follows. In the first step the input text undergoes through a tokenization step and embedding step with the help of CLIP's text encoder, which is fed as an input to the trained generator model. The output from the generator model will now be a raw image representation. After which the raw image representation is reshaped and displayed with plotting library such as matplotlib.

3.5.3 Generator model Architecture and Dataflow

The data flow through the generator model follows a series of transformations as it goes from input to output. Firstly, the model takes as an input an embedding vector which represents the input text description. Which itself is an output generated by CLIP's text encoder. Following, the embedded vector traverses through a series of linear layers in the blocks module. In each layer of the series of linear layers, a sequential application of linear transformation and non-linear activation is performed. Which at this step extracts low-level features or patterns that are pertaining to the image description. The next step in the data flow progression is the feeding into the transformer the output from the linear layers. At this stage the transformer analyzes the sequence of features, by emphasizing on the relationships between elements and constructing a more comprehensive representation based on the textual description. Following this, the output from the Transformer is directed into a series of convolutional layers. At the convolutional layers, the data is processed spatially, where learned filters are applied to refine the representation in a continuous way. The resolution subsequently, at this stage, is increased by conducting an upsampling step. At last, the convolutional layer outputs the final model's output, which is, as denoted earlier in the report, is a raw image representation. The final output, which is the raw image generated, will now have dimensions that are corresponding to the image height, width, and color channels. The implementation can be found at:
`https://colab.research.google.com/drive/1byVOeS8tgTPZvvU8j6lIvb-Ey3dQR3y_ ?usp=sharing`

3.5.4 Choice of Architecture

The application of our architecture is by taking into consideration the different ways(methods) that the architecture can utilize given the benefits that arise from its different subcomponents. The linear layers (one of the subcomponents) is utilized for capturing the basic features or patterns from the text embeddings, which provides an initial framework for the subsequent processing stages. Following this, the transformer component is critical for modeling complex relationships within the text description. This is the main step for capturing the visual representation that is provided in the textual input. The convolutional layers used are particular helpful for learning special patterns and through iterations produce(build) a high-resolution image representation. As such with the hybrid architecture we employed for the project, we aimed at bridging the gap between text and image

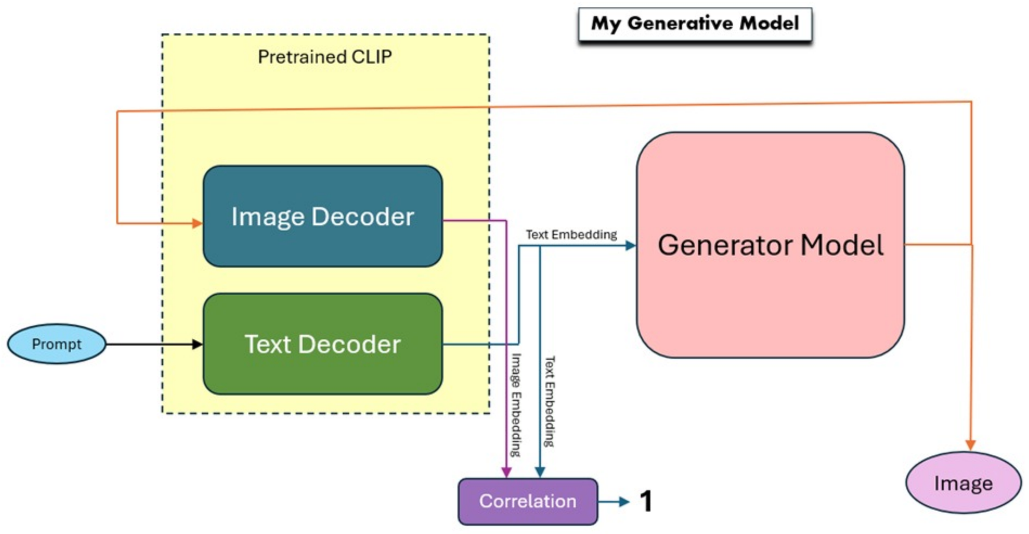


Figure 4: Generative Model

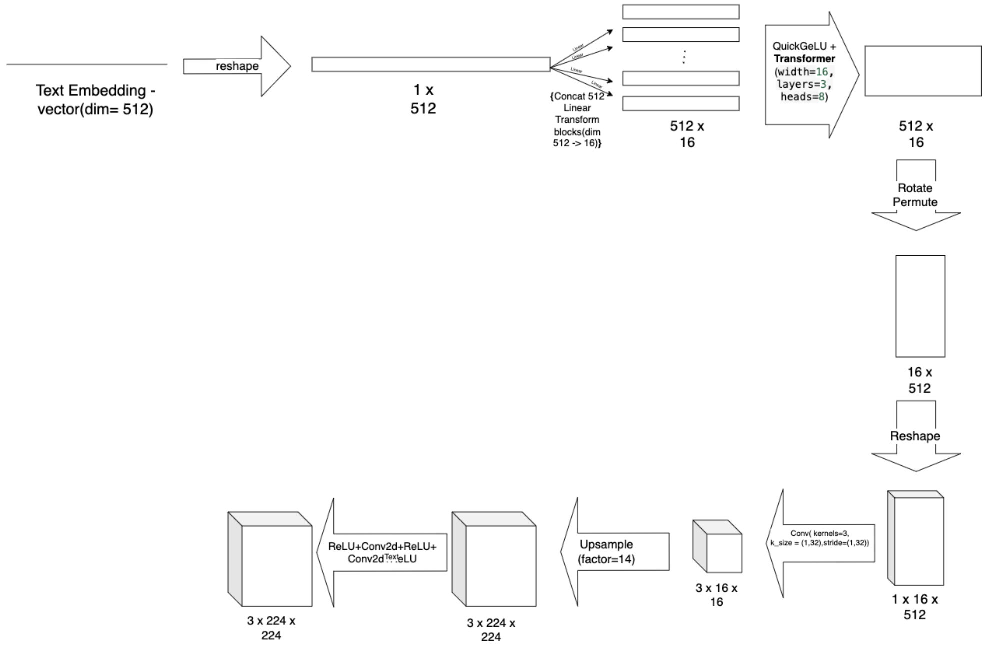

Text Embedding - vector(dim= 512)
reshape
1 x 512
Linear
Linear
Linear
Linear
{Concat 512 Linear Transform blocks(dim 512 -> 16)}
512 x 16
QuickGeLU + Transformer (width=16, layers=3, heads=8)
512 x 16
Rotate Permute
16 x 512
Reshape
1 x 16 x 512
Conv(kernels=3, k_size = (1,32),stride=(1,32))
3 x 16 x 16
Upsample (factor=14)
3 x 224 x 224
ReLU+Conv2d+ReLU+ Conv2d Text eLU
3 x 224 x 224

Figure 5: Generator Architecture

domains. If had just used pure transformers, we wouldn't have been able to get the benefits of getting the inherent spatial awareness of convolutional layers (since pure transformers might struggle to directly generate images). If on the other hand we had used pure convolutional networks, we may not be able to successfully capture the dependencies within the text description, which could lead to images that lack important details. Additionally, hyperparameter tuning, encompassing factors like the number of layers, units within layers, and activation functions, plays a pivotal role in shaping the model's efficacy and is likely fine-tuned throughout the training process.

4 Results

4.1 Results for the Generator model

We used the Mean Squared Error (MSE) loss function as the cost function, which in simple terms calculates the average squared difference between the predicted values and the actual values. As such for our model we penalize larger errors more heavily than the smaller ones. And the test dataset error we obtained is 0.06050895154476166, as we trained our model on 4,647,963 parameters of our designed generator model, and the size of our training dataset and test dataset are 18200 and 6400 respectively. We trained our model for only a 100 epochs, due to the time constraint we faced and the corresponding complexity of our architecture implemented.

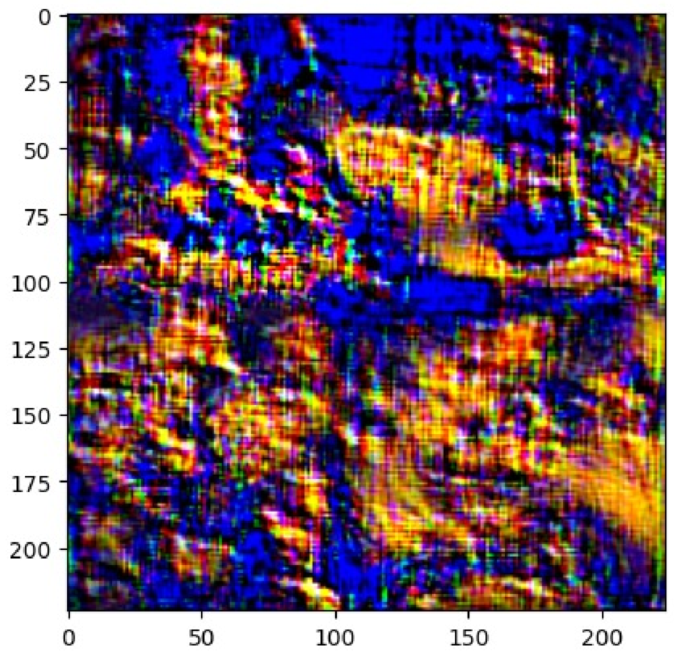


Figure 6: Red Mango

The image generator model, despite the production of noisy images, outputs a low reported training loss. This could be for several reasons. The first one could be overfitting, where the model has memorized a limited set of output patterns during training. This could subsequently be the cause of noisy images across all input texts, as a result of a possible lack of diversity in the dataset. The second probable cause could be mode collapse, which happens usually when employing Generative Adversarial Networks (GANs), where the generator focuses on producing only a few highly realistic-looking outputs, disregarding the diversity present in the dataset, resulting in a low loss value but accompanied with limited output variety. Thirdly, a possible reason could be vanishing gradients, especially when combining diverse components. This makes it difficult for the early layers like the linear layers to be meaningfully updated during training, hindering the model's ability to learn subtle nuances or variations from input text. Finally, a training object mismatch could cause the observed discrepancy. Even though a low loss could be signalling that the model is optimizing according to the defined loss function, there could be a chance that the loss function itself might not fully capture the intended image metrics for calibrating it's quality. As such, it could be the case that the model might be optimizing for a set metric that doesn't directly correspond to the representation of clear visuals.

We could address the issue of noisy image generation by firstly augmenting the dataset through diverse text transformations or image augmentation techniques which could enhance the diversity of our dataset, possibly taking care of the memorization of patterns observed in our model. Employing other loss functions could also be one possible avenue. With our GAN (Generative Adversarial Network), which specifically helps in mitigating the mode collapse problem alluded to earlier. It is also possible to perform architecture hyperparameter tuning, where we can vary a number of

Table 1: Result table of two approaches

Model	Size of test dataset	Error	Error Function
Generator Model	6400	0.0605089515	Mean Squared Error (MSE)
ImageEmbedding Network	118,287	0.0001325494	Mean Squared Error (MSE)

layers, neurons per layer, transformer configuration, the sequence of linear components, transformers, and convolutional components. Finally, we can employ quantitative image quality metrics like FID (Fréchet Inception Distance) and IS (Inception Score) to evaluate the visual quality of generated images independently of the training loss.

4.2 Results for the Image Decoder Distillation Implementation (ImageEmbeddingNetwork)

For the Image Decoder Distillation Implementation we had 512 features for each sample. For the testing we used the entirety of 118,287 samples from COCO dataset, as such the test dataset error we obtained was 0.00013254945224616677. Mathematically, this proves that our developed compact model named *ImageEmbeddingNetwork* performs very good.

4.3 Mathematical Formulas

4.3.1 Mean Squared Error

In this section, we discuss the Mean Squared Error (MSE) formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

where $\hat{y}_i$ represents the predicted value and y_i represents the true value for the i-th sample. In the generative model, we consider $\{\hat{y}_i$: *Generated Image Embedding* , y_i : *Text Embedding*$\}$ while for the ImageEmbeddingNetwork, we consider $\{\hat{y}_i$: *Original CLIP Image Embedding* , y_i : *Compact Model's Image Embedding*$\}$

4.3.2 Cosine Similarity

Cosine Similarity formula shown as:

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Here **A** and **B** represent the vector representations of the two entities being compared.

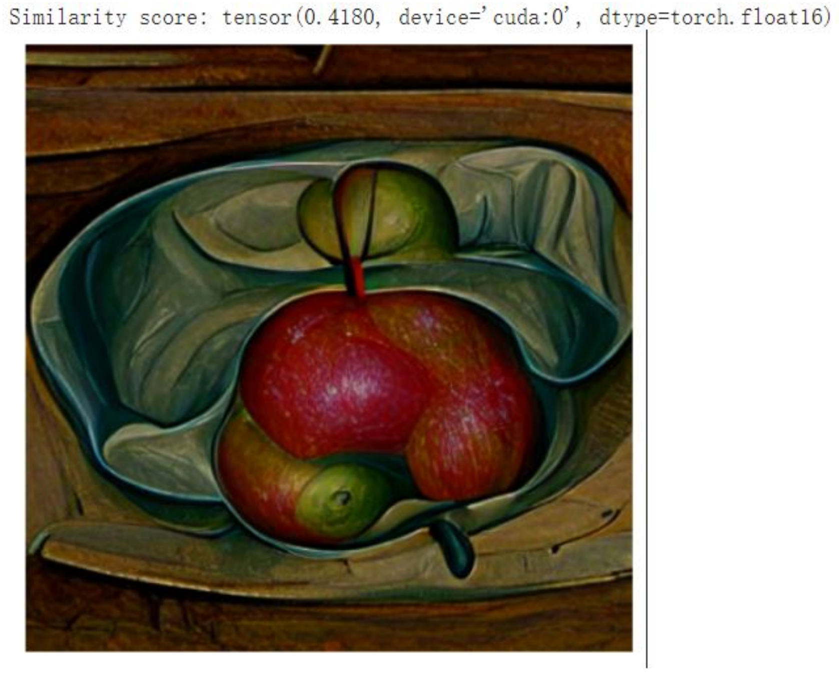
4.4 Similarity Score Evaluation

In our research, we employed both Cosine Similarity and Automated Qualitative Assessment to evaluate the efficacy of Vector Quantized Generative Adversarial Network (VQGAN) models in generating images from textual descriptions. The evaluation framework was designed to provide both quantitative and qualitative insights into the model's performance. Cosine Similarity scores were computed using the CLIP model, which encoded both the input text and the VQGAN-generated images into a shared semantic space. A high Cosine Similarity score indicated that the generated image closely matched the semantic content of the text prompt, as understood by the model. Automated Qualitative Assessment was integrated to facilitate an objective evaluation of the generated images, where a panel of CLIP models was employed to compute similarity metrics at scale. This provided a comprehensive evaluation of the model-generated outputs in comparison to the human textual descriptions, enabling us to analyze the model's capabilities in capturing the essence of the textual prompts. The combination of these assessment methodologies allowed for a robust analysis, encompassing the precision of quantitative metrics and the depth of qualitative insights, thus providing a holistic view of the model's generative performance.

Table 2: Similarity score of Text prompts

Text input	Similarity score	Max iterations
A painting of an apple in a fruit bowl	0.4180	500
Birds is flying in the sunny day's sky	0.1237	500

We compute the cosine similarity between the feature vectors. This is achieved by taking the dot product of the text and image feature vectors, normalized by the product of their magnitudes (L2 norms). The resulting similarity score ranges from -1 to 1, where a score of 1 indicates a perfect match between the image content and text description, 0 indicates no relationship, and -1 indicates complete dissimilarity.



```
Similarity score: tensor(0.4180, device='cuda:0', dtype=torch.float16)
```

Figure 7: Simiarlty score of: "A painting of an apple in a fruit bowl", 0.4180

Finally, the function returns this similarity score, providing a quantitative measure of how well the VQGAN-generated image matches the semantic content of the given text prompt. This score serves as a critical metric in our Automated Qualitative Assessment, allowing us to systematically evaluate the performance of the VQGAN in creating images that are semantically aligned with textual descriptions. By utilizing this approach, we obtain a quantifiable, objective measure to complement our qualitative assessments, thereby enhancing the robustness of our evaluation methodology.

5 Conclusion

This project explored image generation from text and image embedding distillation. The image generation task achieved a low loss value, but the output image quality remained limited. This suggests potential issues with the model architecture, overfitting, or vanishing gradients in the large model. Further investigation is warranted, including exploring alternative architectures, regularization techniques, and extended training. Conversely, the image embedding distillation task was successful, demonstrating that a compact convolutional model can effectively reproduce the embeddings of a larger Vision Transformer. This highlights the potential of model distillation for efficiency improvements. With regards to the proposed approaches, unfortunately we weren't able to successfully finish text embedding distillation task due to limited time. With the proposed milestones we had for our project, we were able to complete the overwhelming majority of our goals. As denoted in the previous sections of our report. However, our group did come short in completing embedding distillation due to the time-constraint we faced in the latter weeks of the semester. All in all our group was able to deliver on almost all the milestones set.

6 Contributions

Table 3: Team member contribution Table

Team Member	Abstract and Research	Data Processing	Model Design and Architecture	Results and Testing
Jiayu Yuan	20%	25%	-	20%
Chun-Chih Yang	20%	-	-	5%
Robel Abdissa	20%	37.5%	25%	35%
Bala Sujith Potineni	20%	37.5%	75%	35%
Chi-Ao Chen	20%	-	-	5%

References

[1] Fine-Tuning Pre-trained Models for Generative AI Applications: `https://www.leewayhertz.com/fine-tuning-pre-trained-models/`.

[2] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

[3] A repo for running VQGAN+CLIP locally: `https://github.com/nerdyrodent/VQGAN-CLIP`.

[4] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean, Distilling the knowledge in a neural network: `https://arxiv.org/abs/1503.02531`

[5] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever, Learning Transferable Visual Models From Natural Language Supervision: `LearningTransferableVisualModelsFromNaturalLanguageSupervision`

[6] Katherine Crowson, Stella Biderman, Daniel Kornis, Dashiell Stander, Eric Hallahan, Louis Castricato, Edward Raff, VQGAN-CLIP: Open Domain Image Generation and Editing with Natural Language Guidance: `https://arxiv.org/abs/2204.08583`

[7] Islam, Mominul, Hasib Zunair, and Nabeel Mohammed. "CosSIF: Cosine similarity-based image filtering to overcome low inter-class variation in synthetic medical image datasets." *Computers in Biology and Medicine* 172 (2024): 108317.

[8] Yu, Jiahui, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. "Vector-quantized image modeling with improved vqgan." *arXiv preprint arXiv:2110.04627* (2021).

[9] Crowson, Katherine, Stella Biderman, Daniel Kornis, Dashiell Stander, Eric Hallahan, Louis Castricato, and Edward Raff. "Vqgan-clip: Open domain image generation and editing with natural language guidance." In *European Conference on Computer Vision*, pp. 88-105. Cham: Springer Nature Switzerland, 2022.

[10] Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." *Advances in neural information processing systems* 27 (2014).

[11] Esser, Patrick, Robin Rombach, and Bjorn Ommer. "Taming transformers for high-resolution image synthesis." In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12873-12883. 2021.

[12] Kang, Minguk, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. "Scaling up gans for text-to-image synthesis." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10124-10134. 2023.

[13] Rodriguez, Juan A., David Vazquez, Issam Laradji, Marco Pedersoli, and Pau Rodriguez. "Ocr-vqgan: Taming text-within-image generation." In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 3689-3698. 2023.