Image Generation from Scene Graphs using Diffusion

Képgenerálás jelenetgráfokból diffúzió alapján

1st Patrik Hujbert
Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics
Budapest, Hungary
patrik.hujbert@gmail.com

***Abstract*—The most popular image generation networks implement a text-based solution (text2image), however there is an approach implementing a generative adversarial network, which takes as input a more structured and specific dataformat, called scene graph. My own solution is meant to improve upon this model by implementing the newer, diffusion-based process. My implemented model uses graph convolution to process input graphs, then the image generation is performed by a diffusion module, which uses the classifier-free diffusion guidance method to sample images based on the processed scene graphs. The training was performed on the COCO-Stuff dataset, which contains images and informations about the complex scenes they describe, achieving promising results. During the evaluation the architecture proved to be suitable to generate images based truly on the given scene graphs. Thus providing potential for future improvements and giving a chance for me to contribute to the further developement of AI Art.**

I. Introduction

With the appearance of diffusion models the general attention to generative models is even bigger than ever before. Their popularity grew not only in the deep learning community, the results had noteworthy impact on the public as well. The artistic sphere has seen a dramatic influx of artificial intelligence resulting in the evolution of AI Art. Despite the mixed feelings of it's reception, AI is set to deeply influence arts future. My project aims to contribute to this field by creating my own image generation model. Some of the recent diffusion based modells, tht achieved superb results in image generation, are **DALL-E-2**, **StableDiffusion** or **DiscoDiffuson**. These solutions generate images basod on text inputs, which is a great, intuitive way to discribe the attributes of an image. However, sentences have a linear structure and therefore a limitation in the aspect of the scenes complexity, that the text describes. From this point of view schene graphs can be a better representation. A scene graph is a graph, that contains objects and relationships between these objects, giving a powreful tool to describe scenes images represent . Given a set of objects O and directed edges E, then the G graf can be described as $G = (O, E)$.

In this paper I am aiming to broden the field of scene graph based image generation.

II. Related Work

The above mentioned scene graph based technique was firstly introduced in 2018 in the paper called **Image Generation from Scene Graphs** [9], shortly **sg2im**.

In this solution the authors the generative adversarial network (GAN) architecture [5] from the generative models. At the time GAN models were state-of-the-art solutions for image generation and almost every models core concept was built around of it's structure. So is the sg2im model as well.

It's main component is the generative network, called **generator**, which was trained against two discriminator network. The image discriminator ensures that the overall appearance of the generated images are realistic. The other discriminator, called object discriminator ensures that the objects appears realistic, complemented by an auxiliary classifier, for the purpose to not only classify the objects real or fake, but to predict each object's category as well.

The generator network uses a graph convolution network, which processes the scene graph inputs, computes a scene layout by predicting bounding boxes and segmentation masks for objects, and converts the layout to an image with a cascaded refinement network.

The model had extraordinary results compared to other, at the time leading GAN architectures like StackGAN

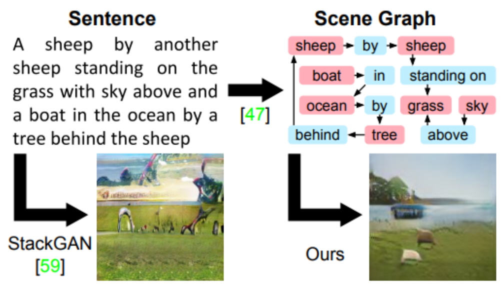
Sentence
A sheep by another sheep standing on the grass with sky above and a boat in the ocean by a tree behind the sheep
[47]
Scene Graph
sheep
by
sheep
boat
in
standing on
ocean
by
grass
sky
behind
tree
above
StackGAN [59]
Ours

Fig. 1. sg2im result compared to StackGAN, source: [9]

III. Architecture

My implementation differs from sg2im in the aspect of generation. My modell contains a diffusion model instead of a GAN architecture. The intention behind my architectural decision is the fact that diffusion-based solutions have taken the lead over the previously state-of-the-art adversial networks, achieving superior image quality [3].

In summary, my model uses sg2im's graph convolution network to process scene graphs, using the transfer learning technique Fine-Tuning. The generation is performed by the diffusion model, which implements a Denoising Diffusion Probabilistic Model (DDPM) [6], using Classifier-Free Diffusion Guidance [8] to take the processed the scene graphs as input guiding to model during training and sampling time as well, what scenes are present in the images.

A. GCN

For the graph convoluton networ I have took in consideration more architectures before choosing the right one to use in my model. The first one was a structure introduced in the paper Semi-Supervised Classification with Graph Convolutional Networks [10]. This approach takes as input two matrices: a feature matrix containing the object's feature vectors, and an adjacency matrix. The network propagates information through the edges within each layer using the $H^{l+1} = f(H^{(l)}, A)$ formula. The problem with this method is that the adjacency matrix doesn't contain information about the edge types. This can be solved using attention mechanism [16], a good example is the Edge Attention-based Multi-Relational Graph Convolutional Network (EAGCN) [14]. Because of time and computing power constraints I chose a computationally lighter solution. Due to the fact that I had access to the trained sg2im model, it was a good alternative for me to use the graph convolution implemented in sg2im, giving me a few extra benefits. This architecture is proved to be able to compute fitting object embedding vectors for the problem. Additionally, I could use the transfer learning technique Fine-Tuning. By loading the trained network parameters (weights) the whole model's training got easier, because with the freezed GCN parameters only the diffuison model's training was needed.

B. Diffusion model

During the implementation of my diffusion model I was following the Denoising Diffusion Probabilistic Models paper [6]. The diffusion consits of two main parts, the forward process and the backward process.

The forward diffuison process is responsible to noise the images with noise samled from a normal distribution, providing a totally noised image following the normal distribution. The noising is an iterative process with 1000 timesteps. The step sizes are controlled by a linear sceduler. Every step can be represented by a timestamp t, thus the image corresponding to a given timestep is x_t, for example x_0, x_{199}, x_{399}, x_{599}, x_{799} and x_{999} can be seen in Fig. 2.

$$q(x_t|x_{t-1}) = N(x_t, \sqrt{\overline{\alpha}_t}x_0, (1-\overline{\alpha}_t)I) \tag{1}$$

The backward process is responsible to reverse the above mentioned diffuison process. I can be described by

$$p(x_{t-1}|x_t) = N(x_{t-1}; \mu_\theta(x_t,t), \sum_\theta(x_t,t)) \tag{2}$$

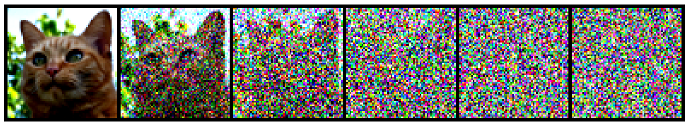

Fig. 2. Noising steps

, where

$$\mu_\theta(x_t,t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(x_t,t)),$$
$$\sum_\theta(x_t,t) = \beta_t$$

. and predicts the noise's expected value between two images with consecutive timesteps (x_{t-1}, x_t). The networks objective, which can be derived from the negative log likelihood, is

$$L_{simple} = \mathbb{E}_{t,x_0,\epsilon}[||\epsilon - \epsilon_\theta(x_t,t)\|^2], x_t = \sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t\epsilon} \tag{3}$$

equation.

My training and sampling method followed the algorithms mentioned in the paper (Fig. 3, Fig. 4).

Algorithm 1 Training

```
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \mathrm{Uniform}(\{1,\dots,T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: until converged
```

Fig. 3. Training algorithm, source: [6]

Algorithm 2 Sampling

```
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
2: for $t = T,\dots,1$ do
3:   $\mathbf{z} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4:   $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\right) + \sigma_t\mathbf{z}$
5: end for
6: return $\mathbf{x}_0$
```

Fig. 4. Sampling alorithm, source: [6]

The model's network implements a U-Net neural network [13], which consists of three main components: decoder, bottleneck, coder. To build the architecture I used the following building blocks [4] (Fig. 5):

- *DoubleConv*: Implements two covolution layer with the possibility to redisual connections.

TABLE I
U-NET ARCHITECTURE

Index	Inputs	Operation(in-, out-channels)	Output Shape
(1)	-	Image	$3 \times 64 \times 64$
(2)	(1)	DoubleConv(3, 64)	$64 \times 64 \times 64$
(3)	(2)	Down(64, 128)	$128 \times 32 \times 32$
(4)	(3)	Down(128, 256)	$256 \times 16 \times 16$
(5)	(4)	Down(256, 256)	$256 \times 8 \times 8$
(6)	(5)	DoubleConv(256, 512)	$512 \times 8 \times 8$
(7)	(6)	DoubleConv(512, 512)	$512 \times 8 \times 8$
(8)	(7)	DobleConv(512, 256)	$256 \times 8 \times 8$
(9)	(8),(4)	Up(512, 128)	$128 \times 16 \times 16$
(10)	(9),(3)	Up(256, 64)	$64 \times 32 \times 32$
(11)	(10),(2)	Up(128, 64)	$64 \times 64 \times 64$
(12)	(11)	Conv2d(64, 3)	$3 \times 64 \times 64$

- *Down*: The main building block for the decoder component, it consists of a max pooling layer and two DoubleConv layer to transform the input images (the first one with a residual connection). it also contains a sequential modul with a SiLU activation and a linear layer to transform the timestep embeddings to the corresponding dimension.
- *Up*: The main building block for the coder component, it consists of an upsample layer and two DoubleConv layer (the first one with residual connection). the same way as in the Down block it caontains a sequential network for the timesteps.

The decoder consists of three Down block, the bottleneck of three DoubleConv, and the coder of three Up block. Between each Down and Up pair I implemented skip connections. The architectre can be seen in Table I

C. Classifier-Free Diffusion Guidance

During sampling the Denoising Diffusion Probabilistic Model generates data such way that it could have been sampled from the original training dataset. However in my case the model needed some guidance the specify what kinf of scene should conatin the generated image. There are different methods for this type of problem. For example a solution can be to use in addition a classifier. The downside is that it makes the model's training computationally more difficult. So I used the technique called Classifier-Free Diffusion Guidance, which provides the guidance for the modell without the classifier. The core concept of the technique that in the decoder and coder components the blocks (Down and Up block) in my case takes as input the guidance as well in form of an embedding vector. This means that I could use my GCN output for guidance, which is exacty what I needed for my project. During training the guidance input had a 90% chance to be the input graph, otherwise I didn't provide any guidance to the model. By sampling the algorithm samples the model with and without the guidance parameter and computes the output image from the two outut with linear interpolation.

IV. METHOD

A. Dataset, data preparation

For the project I used the COCO-Stuff dataset [2], which containes 164K complex images from the COCO dataset [11] with annottations about the objects in the images. The annotations are object categories, bounding boxes, and segmentation masks. It conatins 80 thing classes and 91 stuff classes, with complex spatial context between stuff and things.

While processing the data I needed to transform the images and compute scene graphs for them. The image transformation contained a resize operation (the model was train on images with 64×66 pixel resolution) and a standardisation using the ImageNet dataset's mean and std values. To compute a scene graph to an image I firstly recalculated the bounding box coordinates to match with the new image size. Secondly changed the size of the segmentation masks. Then using the boxes and masks I calculated the center points of the objects. With the help af the centerpoint it was possible to determine the relation between each object, it could be either `__in_image`, `left of`, `right of`, `above`, `below`, `inside` or `surrounding`. The scene graph consists of triples in the form of `[subject, relation, object]`. An important detail is that I added for every object a `[subject, "__in_image__", "image"]` with the purpose to give the model an understanding of what it means for an object to be in the image.

B. Training, evaluation, test

Because of my project's magnitude the training process takes a lot of time, so I only had one chance to train the my model. This meant that without the hyperparameter optimization I had to be extra careful while deciding about the parameters. Most of my architectural and algorithmic decisions were based on a research [1] about Denoising Diffusion Implicit Models. One import aspect of the training, that for simplicity I only used images from the dataset that conatined $2 - 4$ objects for simplicity.

During the training I evaluated my model with two methods. One of them was the tracking of the loss value, and the other is sampling the model from time to time with scene graphs from the training dataset. Each of the methods showed promising signs but with different results. At the beginning of the training the loss converged quickly, as it was expected but after around 40-50 epochs it slowed down drastically and after that it struggled to reach any better results. With sampling the model showed a much more linear trajectory of development. The smpled images gor better and better, and it reached the best results around 300-360 epochs.

I also evaluated the model with sample images using scene graphs from the validation dataset, which showed the same results.

My main goal for this project wasn't necessarily the good image quality, but rather that the generated images should represent the given scene graphs the best way possible. To evaluate the model I used only subjective comparison between

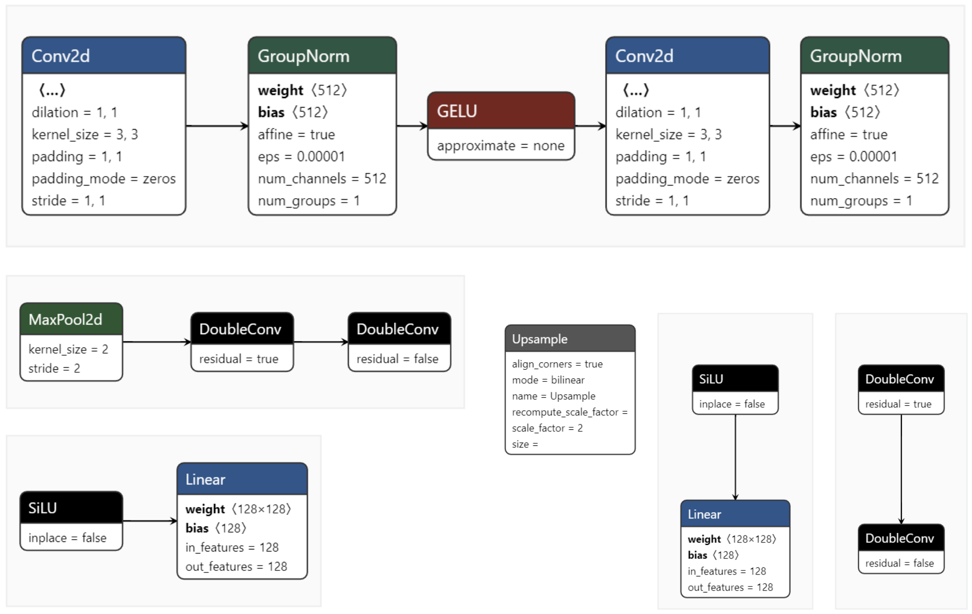

Fig. 5. DoubleConv, Down and Up blocks

the images and it's scenes that the input scene graphs describes. During testing it showed rather good results for the evaluation. In my experience the model works the best with scenes containing 2 objects. Some scenes I generated during testing are:

- beach, with the sea at th top and sand at bottom
- skyscraper with sky at the background
- river, above trees and a rock inside the river
- orange at the table

The corresponding generated images can be seen in Fig 7. For every scene I generated 48 images to better understand the behaviour of the model. It seems that altough the image quality needs improbvements the image groups represents well the scenes attributes, proving that the model learned what each scene means and generates images based on them.

V. Conclusion, Improvement possibilities

In conclusion I can proudly say that the model performes as it was expected and it serves a good baseline for further improvements to generte high quality images based on scene graphs.

Nontheless there are a lot of improvements to be made. First of all the current diffusion model that I use lacks attention mechanism and it is outdated. There are lots of improvements for example changing the linear scheduler to a cosinus based one [12]. The sampling time is really slow with the current solution I use, but it is possible to speed up the process using strided sampling schedule [12] or changing the model to Denoising Diffusion Implicit Models [15]. One other improvement can be to generate higher resolution pixels instead of 64. It can be achieved with a cascaded pipeline of mulitple diffusion models [7].

Changing the way the model processes the scene graphs can cause improvements as well for example with the use of EAGCN or adding further networks like sg2im did.

References

[1] András Béres. Denoising diffusion implicit models. https://keras.io/examples/generative/ddim/#lessons-learned.
[2] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context, 2016.
[3] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021.
[4] dome272. Diffusion-models-pytorch. https://github.com/dome272/Diffusion-Models-pytorch.
[5] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2016.
[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.
[7] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation, 2021.
[8] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022.
[9] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs, 2018.
[10] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.
[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.
[12] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.
[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
[14] Chao Shang, Qinqing Liu, Ko-Shin Chen, Jiangwen Sun, Jin Lu, Jinfeng Yi, and Jinbo Bi. Edge attention-based multi-relational graph convolutional networks. 2018.
[15] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2020.
[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

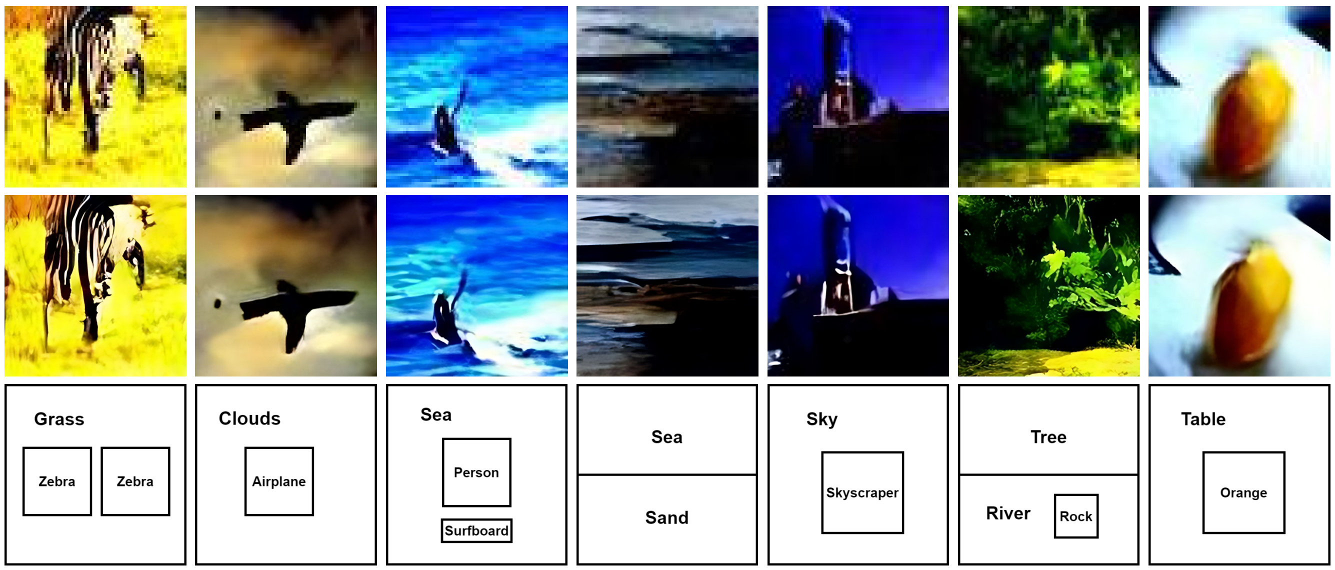

Fig. 6. Generated images and their upscaled versions

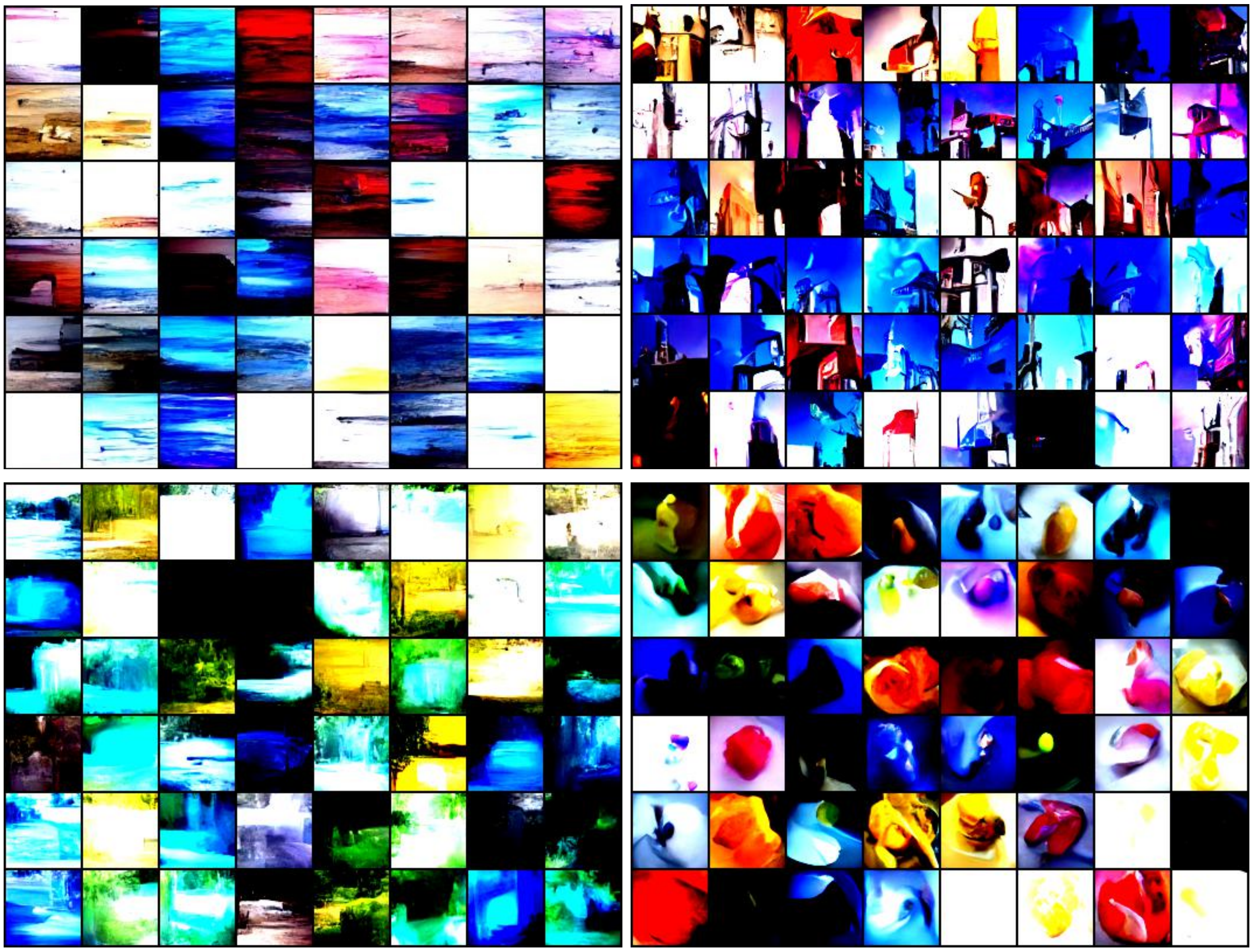

Fig. 7. Test images