Filter-Guided Diffusion for Controllable Image Generation

I. Implementation Details

A. Process Flow

- The process starts by taking a **text prompt** and a **guide image** as the inputs.
- Now, we apply a diffusion model (**Stable Diffusion**) to the text prompt, which generates a noisy image and gradually denoises it.
- FGD integrates within this process by applying a filtering step (**Joint Bilateral Tensor**) at each operation to align the generated image closer to the guide.
- In a few cases, the colors of the guide image may not match the colors specified in the text prompt. For this, FGD provides an optional **color normalization** step, allowing the colors of the generated image to adapt to the text prompt while preserving the structure from the guide image.

B. Parameters

1) **Guidance Strength** - $\boldsymbol{\delta}$ - controls the influence of the guide image. Higher value moves the output image to guide image.
2) **Spatial Filter Size** - $\boldsymbol{\sigma_s}$ - controls scale of structural / spatial changes allowed by the guide.
3) **Value Filter Size** - $\boldsymbol{\sigma_v}$ - controls how much the edges from the guide image are preserved.
4) $\boldsymbol{t_{end}}$ - determines the maximum allowed steps for guidance.
5) **Normalization** - optionl parameter to specify the extent of normalization. Set it to 0 in order to turn off normalization.

II. Datasets

A. Training Dataset

Being a black-box method, FGD does not require any traning datasets since the approach uses pre-trained Stable Diffusion model as base.

B. Evaluation Datasets

For evaluation and comparison following datasets are used

- CelebFaces Attributes (CelebA) - evaluates the ability to manipulate different attributes of human faces.
- Animal Faces-HQ (AFHQ) - used to evaluate how well FGD maintains the characteristics of animals in the generated images.
- Landscape Pictures Dataset - evaluates method's ability to translate and edit natural scenary.

The above datasets can be downloaded using the download.sh file, that I have provided in the github repo of the implementation.

- Also, there are some images that the author has directly used in the paper for some results, those images have been stored in the imgs folder in the repo itself.

III. Results

A. Variation of generated image with *guidance strength parameter* δ -

- Examples have been generated using $\sigma_s = 2$, $\sigma_v = 1$ and $t_{end} = 15$ as parameters, while varying δ from 0.2 to 1.4
- We can see that as we increase the guidance strength the generated image becomes closer to the guide image.

Fig. 1. Variation of generated image with δ - Guide Image: cat.png; Prompt 1: 'a photo of a dog'; Prompt 2: 'a photo of a rabbit'

*B. Effect of **normalization** on the generated image -*

- In some prompts the normalization is not necessary, and our model generates good output images in both cases. But for some prompts color normalization is neccessary in order to generate accurate results, as with the example of red submarine.

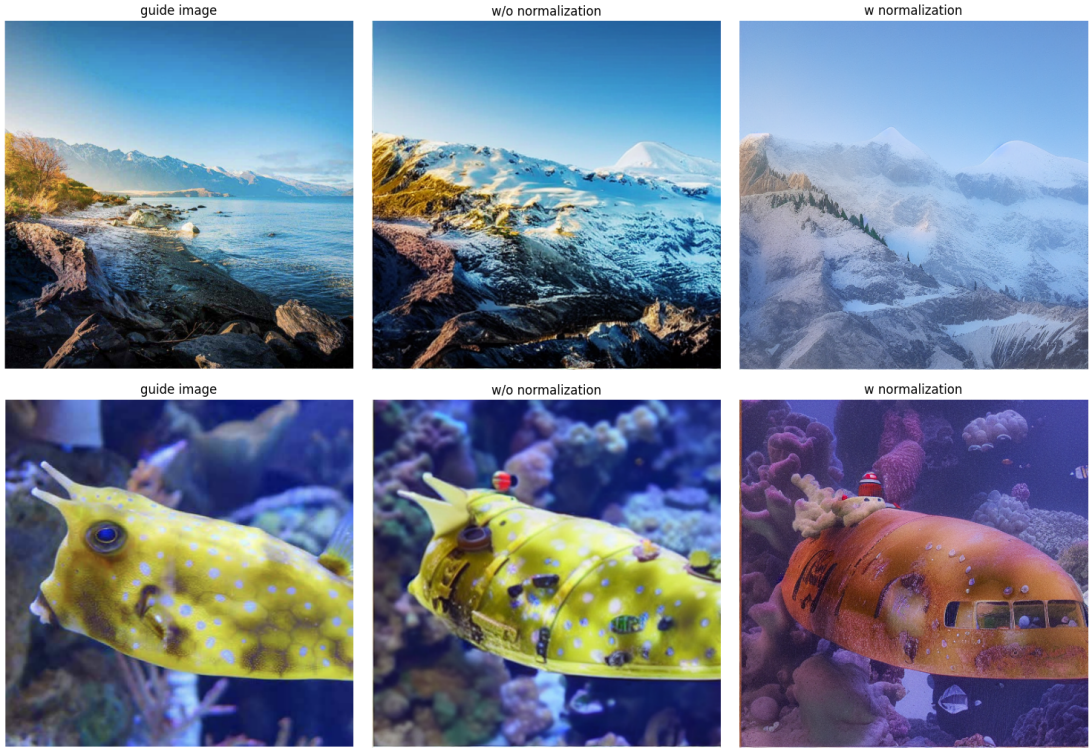

Fig. 2. Effect of normalization on the generated image - Guide Image 1: landscape.png; Prompt 1: 'a photo of a snowy mountain'; Guide Image 2: underwater.png; Prompt 2: 'a photo of a red submarine underwater'

C. Variation of generated image with t_{end} parameter -

- We can see that as we increase the t_{end} while keeping the δ constant the generated image becomes moves away from the guide image and more towards the given prompt, due to increased iterations.

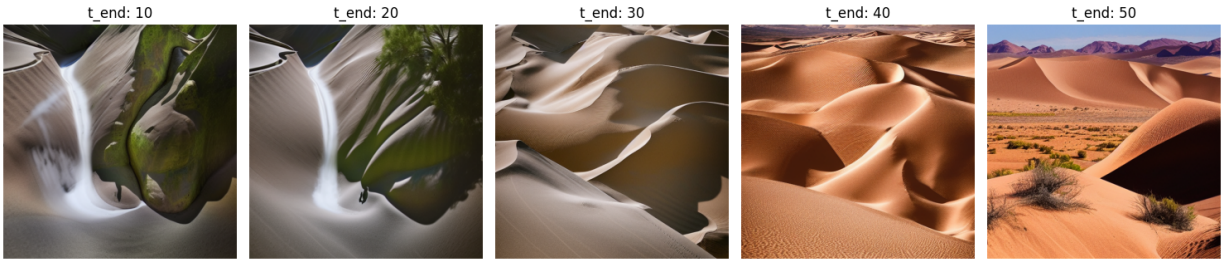

Fig. 3. Variation of generated image with t_{end} - Guide Image: waterfall.png; Prompt: 'a photo of a desert'

*D. **CLIP** and **DINO** score variation on albation of **hyperparameters** -*

It is not practically feasible for me to run this albation as I am doing this in a colab notebook and it has it's API limits for running GPU. Also, it takes up a lot of time, so it was not feasible to plot or show this graph but I have written the code (mentioned towards the end of the notebook) in order to generate the scores which can be in further used to plot the graph.

$\Longrightarrow$ Most of the implementations and results described in this section can be reconstructed / regenerated by following the jupyter notebook [codeResults.ipynb] procedurally in Google Colab using T4 GPU.