AI Beautifier

Group 6
孫承瑞 李欣螢 莊宥瑋 葉家蓁

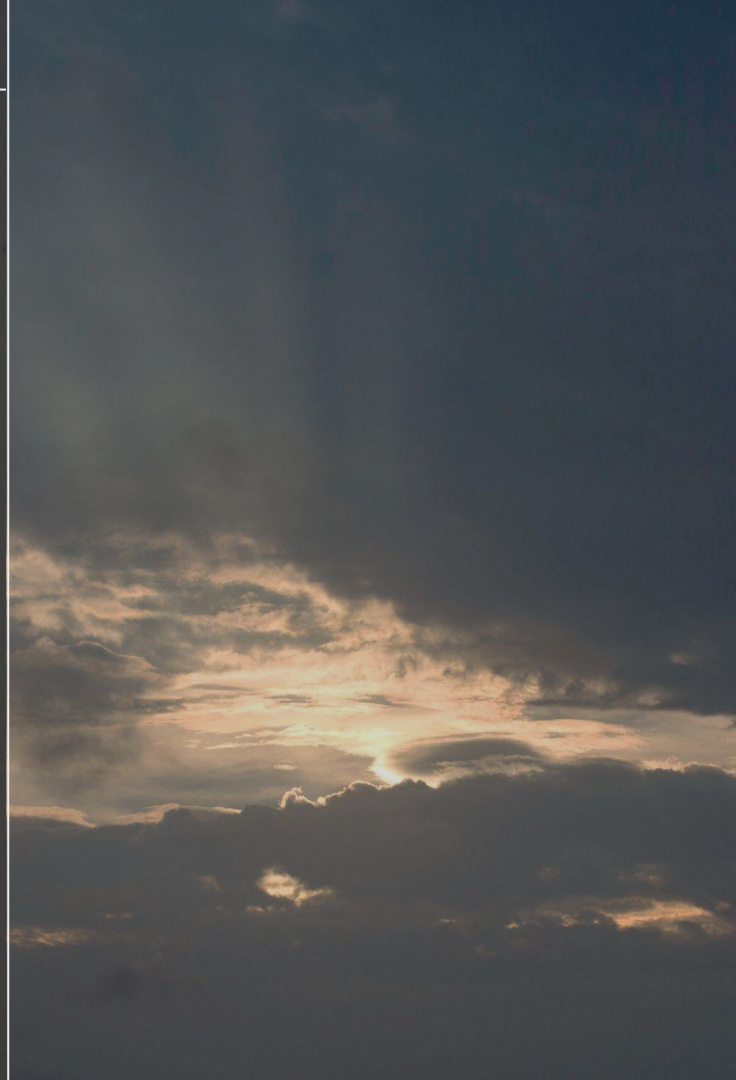


TABLE OF CONTENTS

Introduction

Problem Overview

When taking photos,
we seldom get flawless pictures.

Therefore, We want to
erase the undesirable elements
in the background
and **remove the flares**.

Related work

🔗 Deep Image Prior

- By Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky
- A type of **convolutional neural network** used to enhance a given image with **no prior training data** except for the image itself
- Implement for **inpainting**

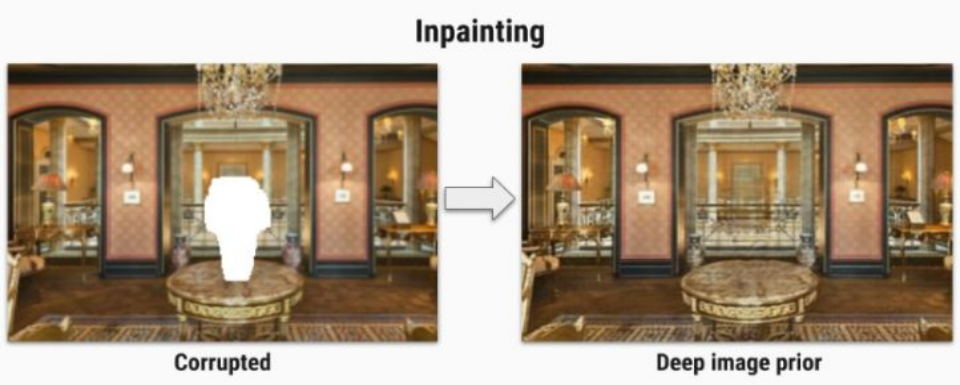
🔗 Flare7K

- By Yuekun Dai, Chongyi Li, Shangchen Zhou, Ruicheng Feng, Chen Change Loy
- **Datasets** of **7k flare patterns** and 24k background images that **add diversity** when training models
- Implement for **removing flare**

Dataset

- Flare7k
  - Introduced by S-Lab, Nanyang Technological University.
  - 5,000 scattering flare images and 2,000 reflective flare images.
  - 25 types of scattering flare and 10 types of reflective flare.
- Flickr24k
  - Sampled from “Single Image Reflection Removal with Perceptual Losses, Zhang et al., CVPR 2018”
  - 23,949 background images.
  - Filtered out most of flare-corrupted or overexposed images.

Baseline - Deep Image Prior



Baseline - Deep Image Prior

- **Encoder-decoder** architecture with **skip connections**

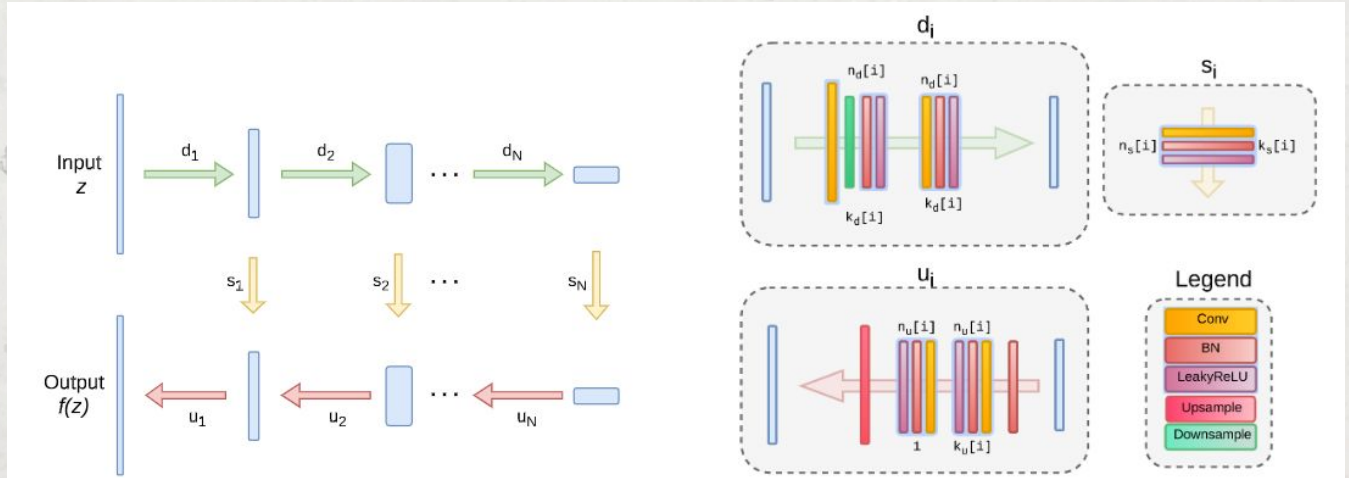


Figure 1: **The architecture used in the experiments.** We use "hourglass" (also known as "decoder-encoder") architecture. We sometimes add skip connections (yellow arrows). $n_u[i]$, $n_d[i]$, $n_s[i]$ correspond to the number of filters at depth i for the upsampling, downsampling and skip-connections respectively. The values $k_u[i]$, $k_d[i]$, $k_s[i]$ correspond to the respective kernel sizes.

Baseline - Deep Image Prior

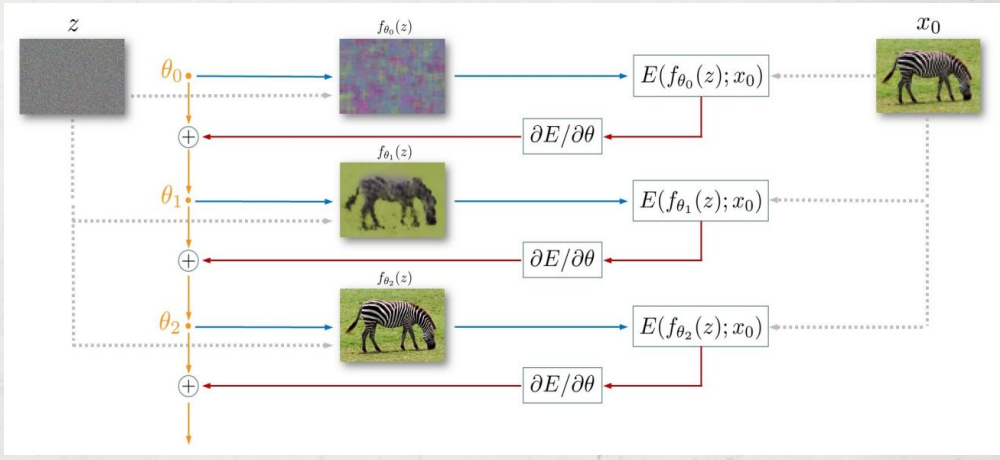
We can formulate image restoration problems as:

$$x^* = \underset{x}{\operatorname{argmin}}\, E(x; x_0) + R(x),$$

- x_0: Input image
- x: Output image
- R(x): Regularizer(Prior), the authors use the implicit prior(local minimizer) here:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, E(f_\theta(z); x_0), \qquad x^* = f_{\theta^*}(z)\,.$$

- z: a fixed randomly-initialized 3D tensor



Baseline - Deep Image Prior

For inpainting problems,

$$E(x; x_0) = \|(x - x_0) \odot m\|^2 ,$$

- m(mask) = {0, 1}

This formula makes the model **restore the mask parts (m=0) from the normal parts (m=1)**, and makes the latter remain the same as possible.

The default architecture:

$z \in \mathbb{R}^{32 \times W \times H} \sim U(0, \frac{1}{10})$ noise

$n_u = n_d = [128, 128, 128, 128, 128]$

$k_u = k_d = [3, 3, 3, 3, 3]$

$n_s = [4, 4, 4, 4, 4]$

$k_s = [1, 1, 1, 1, 1]$

Skip connections

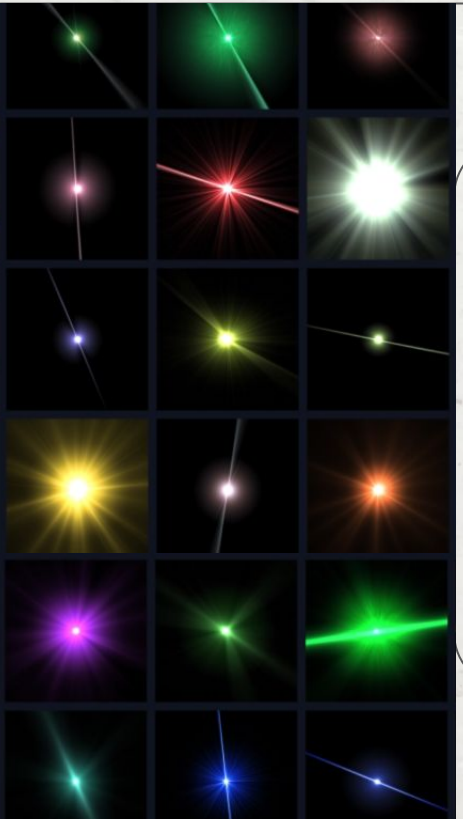
$\sigma_p = \frac{1}{30}$ noise std

num_iter = ~~2000~~ 6000

LR = 0.01

upsampling = bilinear

Baseline - Flare7K



Flares

Flare data Collection	Scattering flare & reflecting flare
Paired test data Collection	Synthetic test data & real test data
Training	Uformer / U-Net
Evaluation	PSNR / SSIM / LPIPS

Main Approach

1

Combination

To beautify our photos, we want the functions to be merged so that they can **become a complete application**.

2

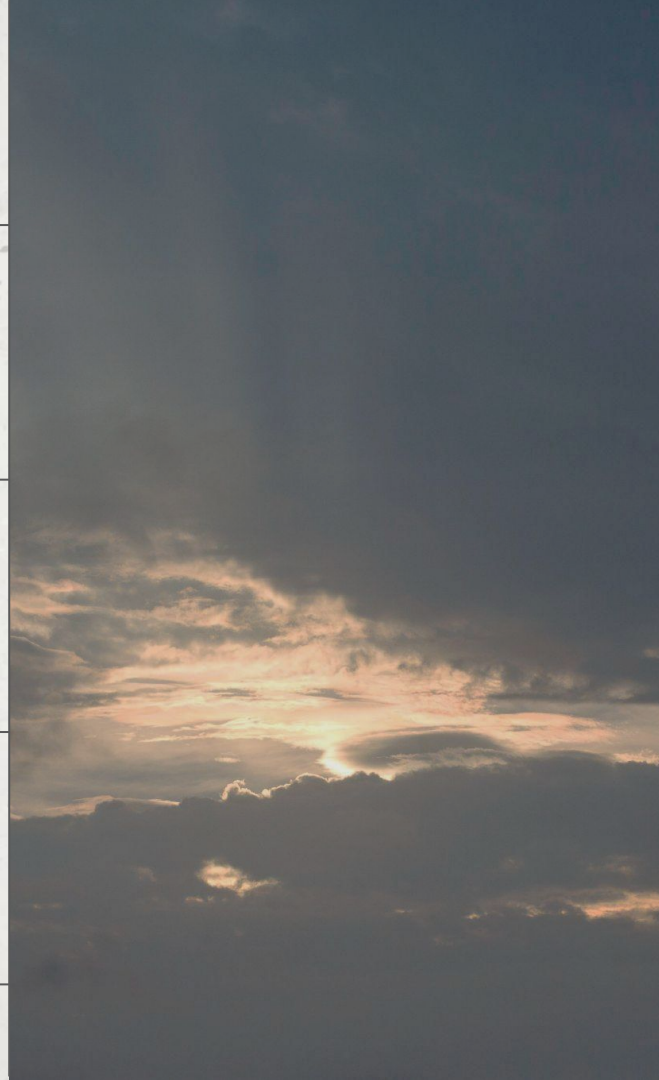
Automatic Mask Generation

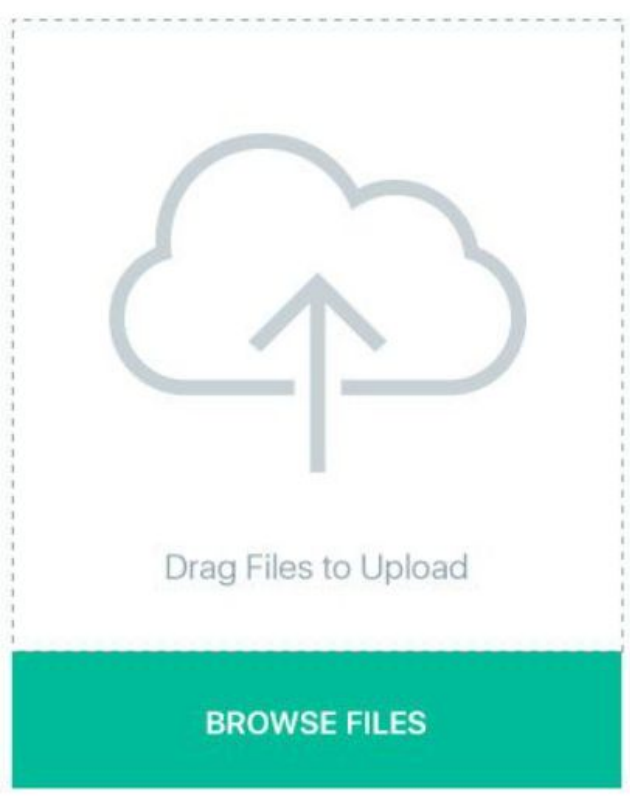
Instead of uploading mask images by ourselves, we provide other **two methods to generate the masks much easier**.

3

Inpainting Setup Comparison

Due to different areas needed to be erased from a photo, we want to **choose the best settings** in each situation.





Combination

Upload your photo

In square size, png/jpg/gif image formats

Deflare / Inpainting

Beautify the photo as you like

Get a 512*512 **png** photo

Without any undesired thing

All the functions can be done in **one Google colab**!

Automatic Mask Generation

Upload mask images (Original)

You may need to use a graphic software and spend lots of time, which is inconvenient.

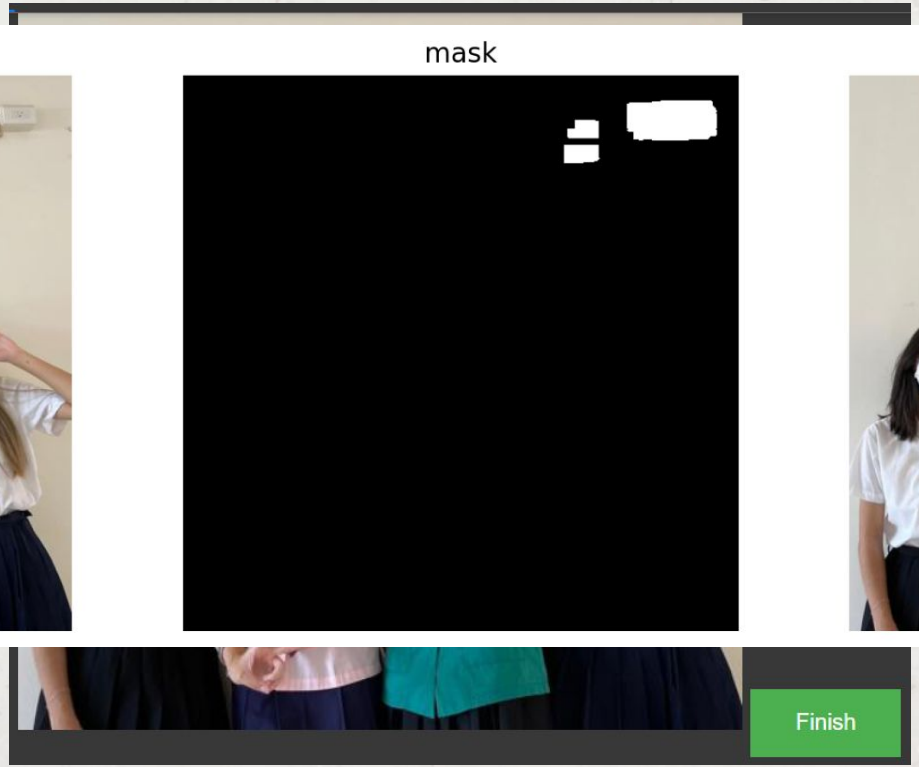
Draw on the photo directly **(New!)**

Now, we can manually draw on the photo to select any area you don't like!

Text-to-mask automatically **(New!)**

Moreover, we can just type in the description of any object in the photo to generate the mask!

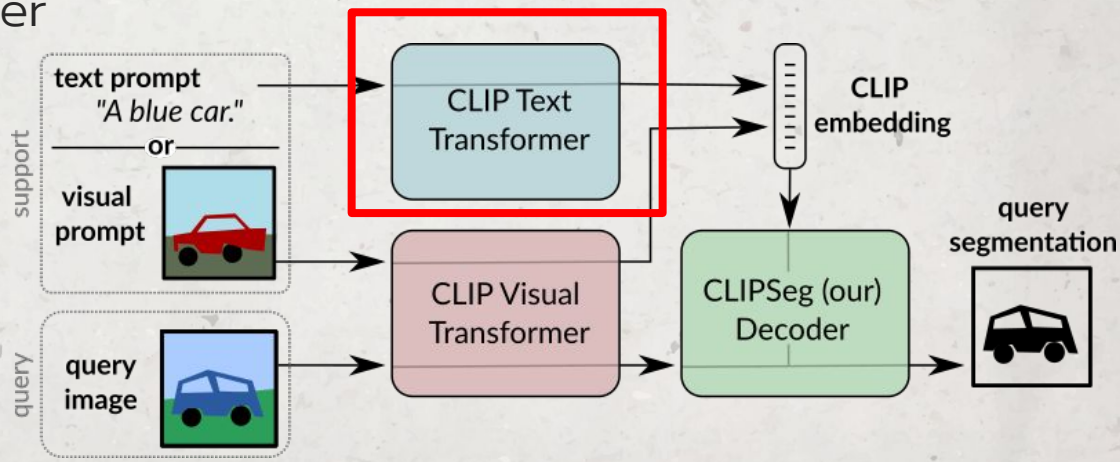
Draw on the photo directly



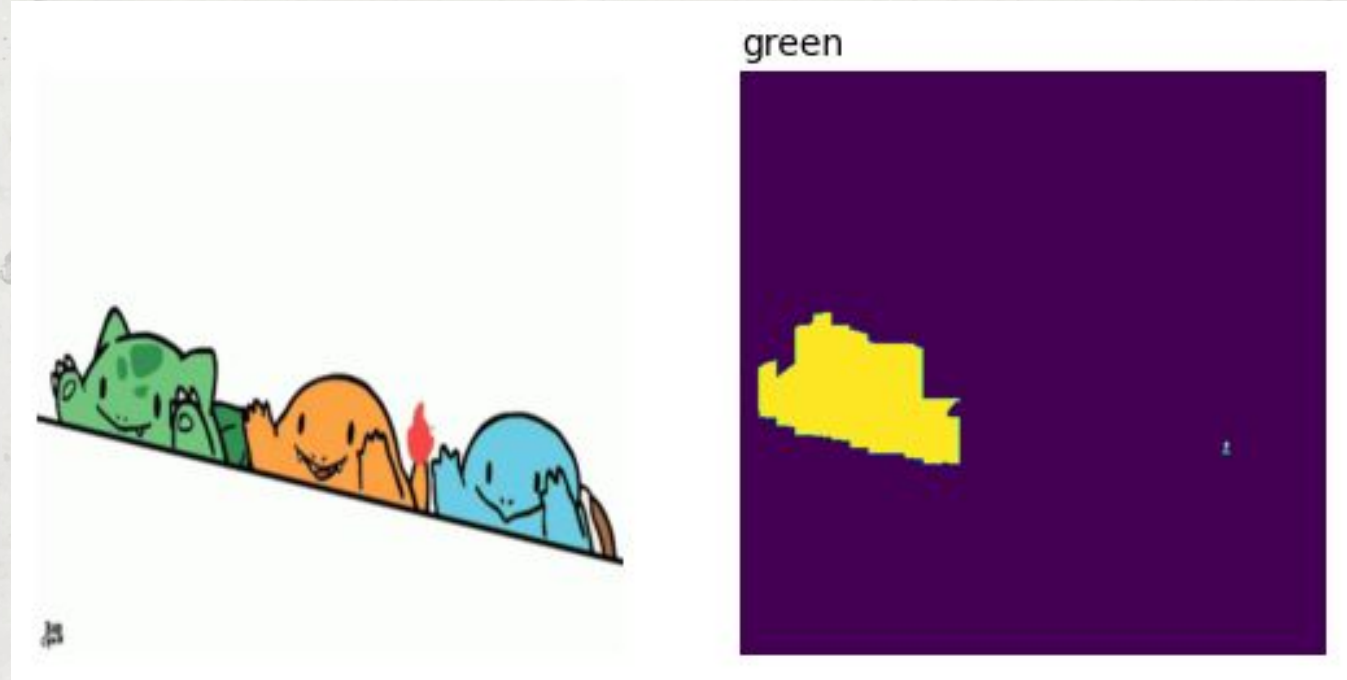
Text-to-mask automatically

CLIPSeg

- By Timo Lüddecke, Alexander S. Ecker
- Image segmentation
- OpenAI's pre-trained CLIP model
- Transformer-based decoder



Text-to-mask automatically

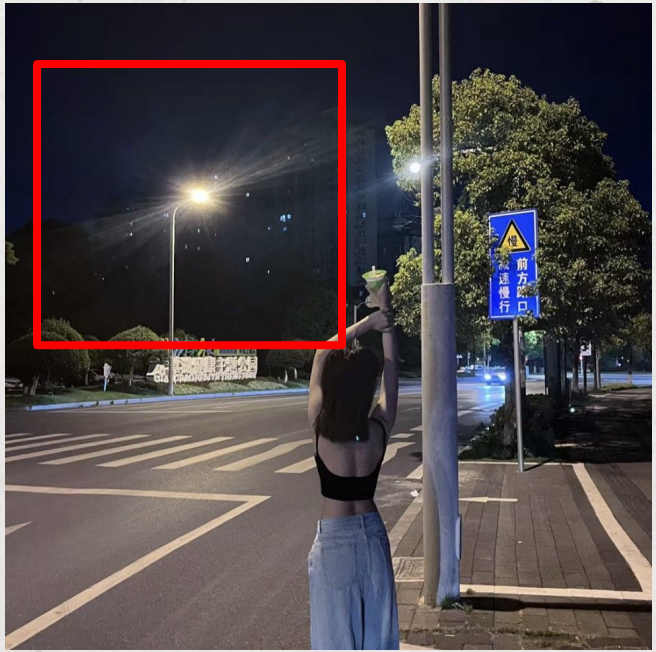


Examples

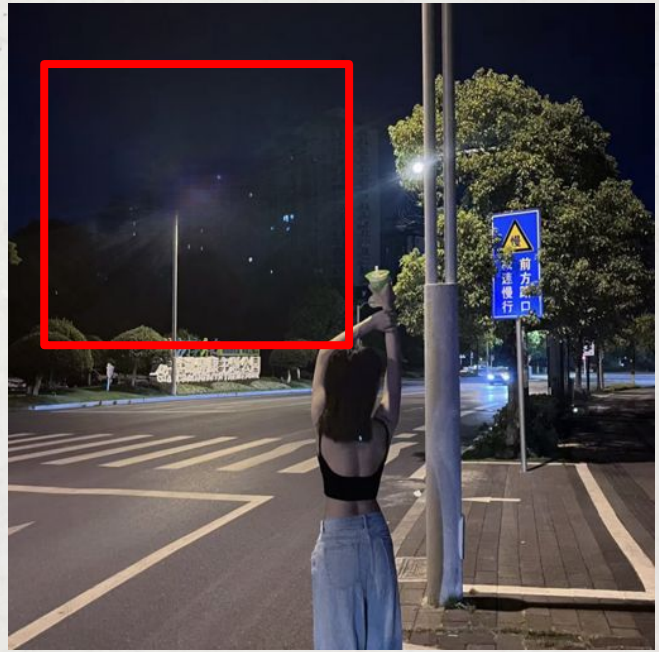
Example - Import Picture

Example - Deflaration

Before

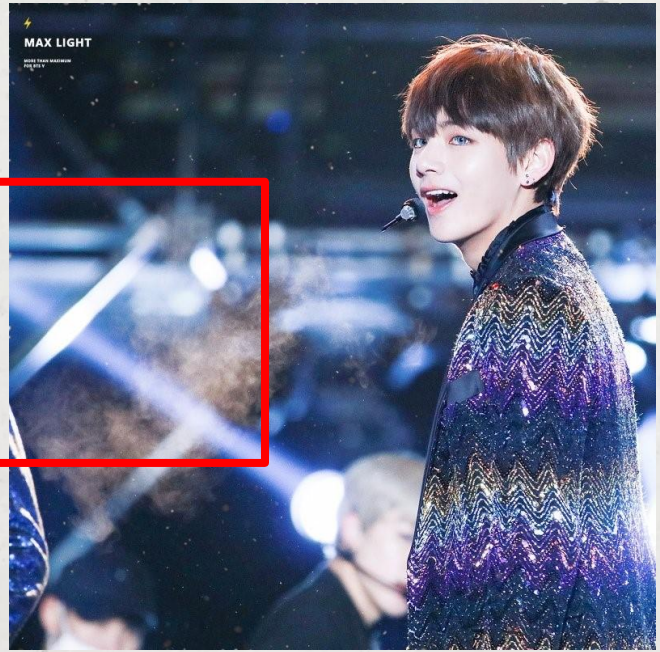


After



Example - Deflaration

Before



After

Example - Automatic text-to-mask

```
prompts = 'tree' #@param {type:"string"}
```

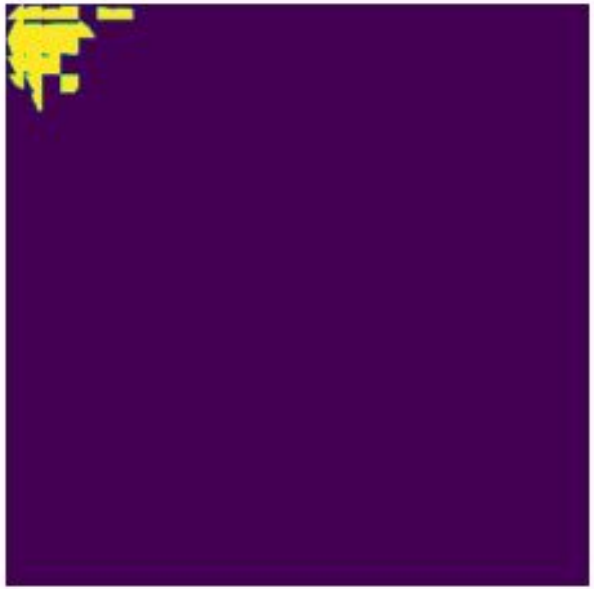
prompts: " tree

Example - Automatic text-to-mask

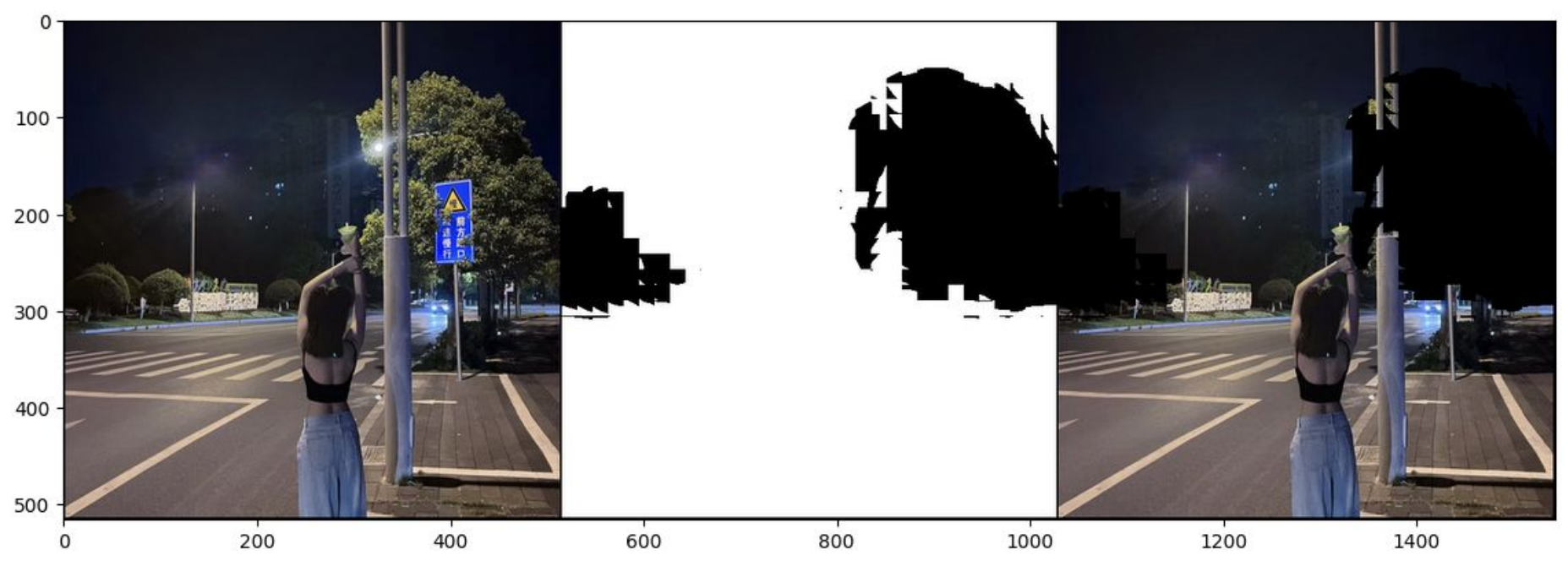
```
prompts = 'alphabets' #@param {type:"s
```

prompts: "alphabets"

alphabets

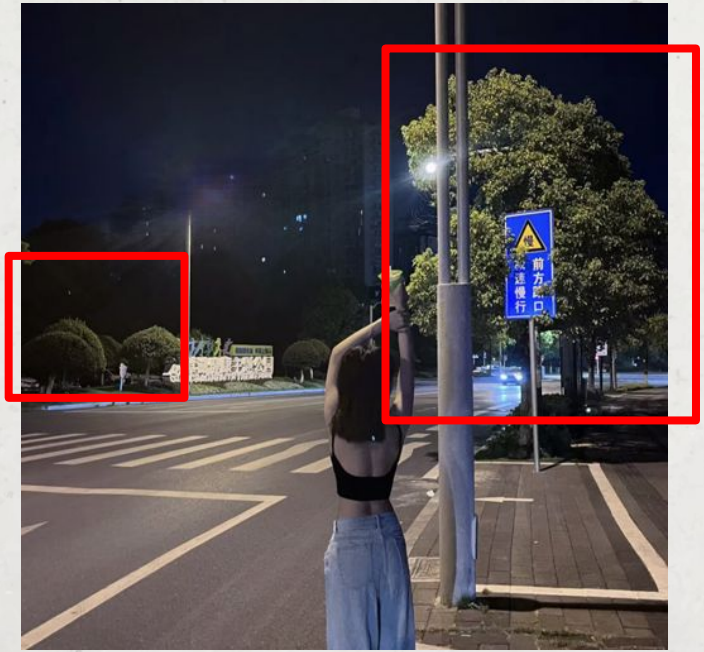


Example - Masked



Example - Inpainting

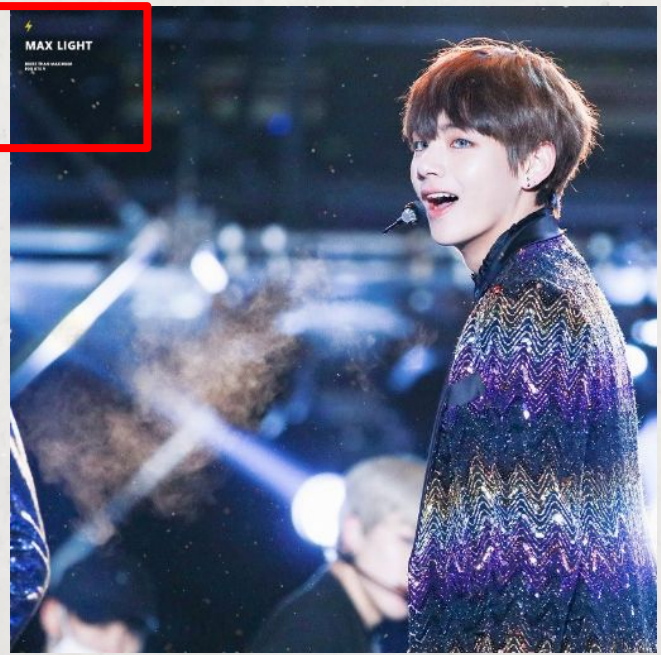
Before



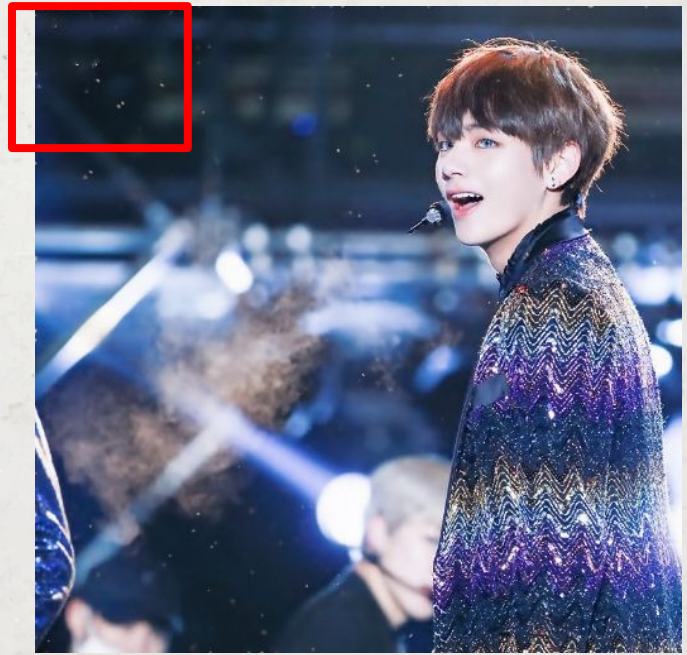
After

Example - Inpainting

Before



After



Example - Evaluation

PSNR	21.55556456
SSIM	0.815952666
LPIP	0.180437713

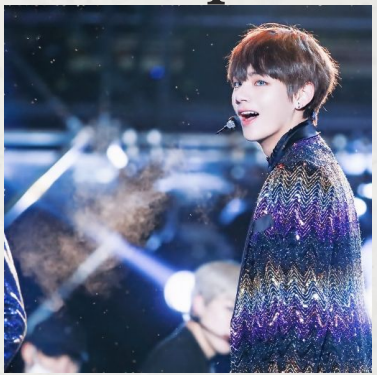
PSNR	31.14807182
SSIM	0.937167919
LPIP	0.082193925

A brief analysis of examples

Example 1

- Performs better at deflaration: The type of flare is included in the training dataset.
- Performs worse at inpainting: The tree to be removed is interfered by a street light and a traffic sign.

Example2



- Performs better at inpainting: The text to be removed has a relatively clean background and is not interfered or covered by other elements.
- Performs worse at deflaration: The “light” at background is not so typical “flare”.

Inpainting Setup Comparison

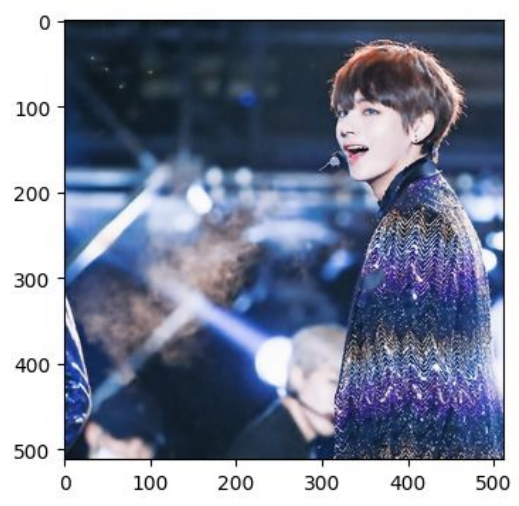
Default

```
NET_TYPE  =  'skip_depth6'

pad  =  'reflection'  #  '
OPT_OVER  =  'net'
OPTIMIZER  =  'adam'#  'LB

INPUT  =  'noise'
input_depth  =  32
LR  =  0.01
num_iter  =  6000
param_noise  =  False
show_every  =  50
figsize  =  3
reg_noise_std  =  0.03
```

- PSNR: 28.518995097589375
- SSIM: 0.894469163747439
- LPIPS: 0.15108217298984528
- Run time: 27m17s



Inpainting Setup Comparison

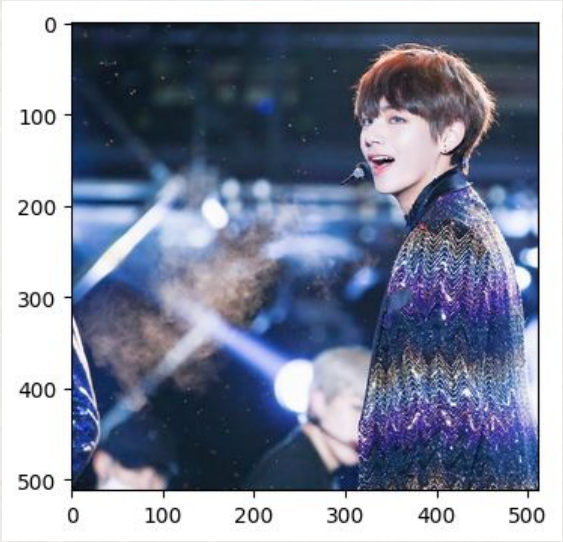
Attempt.1 - Reduce iteration

```
NET_TYPE  =  'skip_depth6'

pad  =  'reflection'  #  '
OPT_OVER  =  'net'
OPTIMIZER  =  'adam'

INPUT  =  'noise'
input_depth  =  32
LR  =  0.01
num_iter  =  3000
param_noise  =  False
show_every  =  50
figsize  =  3
reg_noise_std  =  0.03
```

- PSNR: 31.148071829230748
- SSIM: 0.9371679197165727
- LPIPS: 0.08219392597675323
- Run time: 12m8s



Inpainting Setup Comparison

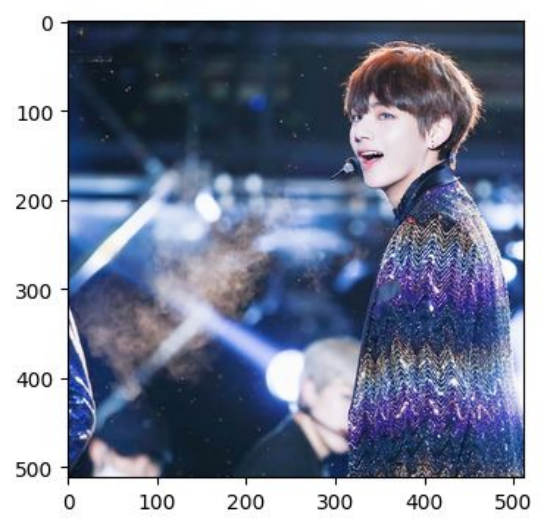
Attempt.2 - A different type of net

```
NET_TYPE  =  'skip_depth2'

pad  =  'reflection'  #  '
OPT_OVER  =  'net'
OPTIMIZER  =  'adam'

INPUT  =  'noise'
input_depth  =  32
LR  =  0.01
num_iter  =  3000
param_noise  =  False
show_every  =  50
figsize  =  3
reg_noise_std  =  0.03
```

- PSNR: 32.12606406514374
- SSIM: 0.9522127244977979
- LPIPS: 0.05374791473150253
- Run time: 12m10s



Inpainting Setup Comparison

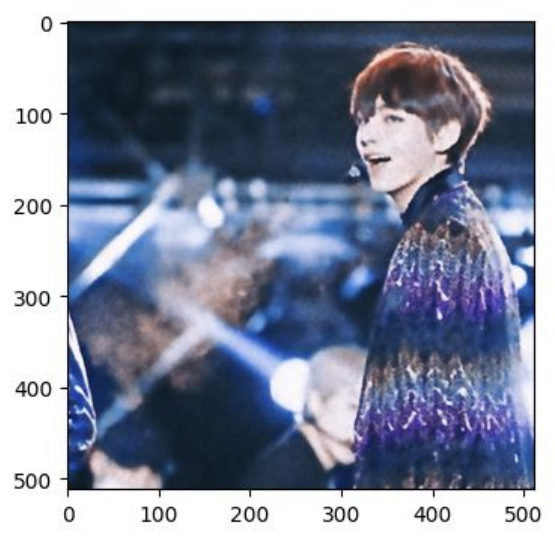
Attempt.3 - A different optimizer

```
NET_TYPE  =  'skip_dept

pad  =  'reflection'  #
OPT_OVER  =  'net'
OPTIMIZER  =  'LBFGS' #

INPUT  =  'noise'
input_depth  =  32
LR  =  0.01
num_iter  =  3000
param_noise  =  False
show_every  =  50
figsize  =  3
reg_noise_std  =  0.03
```

- PSNR: 23.302184233762134
- SSIM: 0.7450587802517482
- LPIPS: 0.28466418385505676
- Run time: 12m48s



Inpainting Setup Comparison

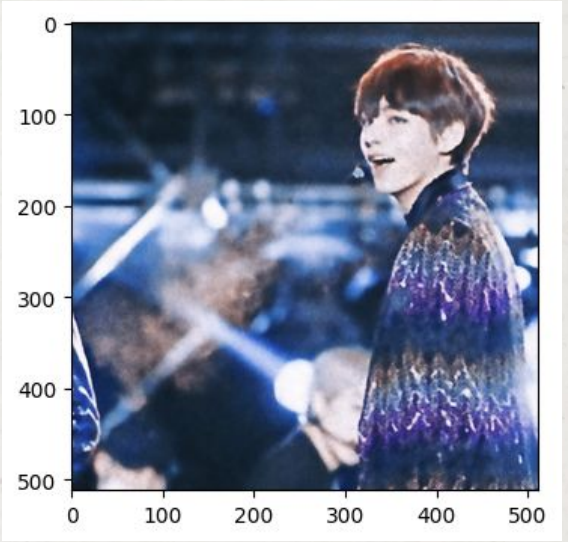
Attempt.4 - A larger learning rate

```
NET_TYPE  =  'skip_dept

pad  =  'reflection'  #
OPT_OVER  =  'net'
OPTIMIZER  =  'adam'#

INPUT  =  'noise'
input_depth  =  32
LR  =  0.1
num_iter  =  3000
param_noise  =  False
show_every  =  50
figsize  =  3
reg_noise_std  =  0.03
```

- PSNR: 26.336750547476633
- SSIM: 0.8412026345982307
- LPIPS: 0.1923815906047821
- Run time: 12m10s



Inpainting Setup Comparison

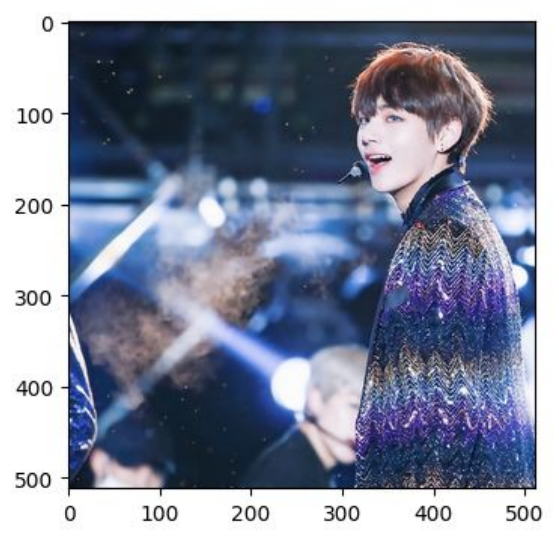
Attempt.5 - A smaller learning rate

```
NET_TYPE  =  'skip_depth6'

pad  =  'reflection'  #  '
OPT_OVER  =  'net'
OPTIMIZER  =  'adam'#  'LB

INPUT  =  'noise'
input_depth  =  32
LR  =  0.005
num_iter  =  3000
param_noise  =  False
show_every  =  50
figsize  =  3
reg_noise_std  =  0.03
```

- PSNR: 30.654185370546074
- SSIM: 0.931728361647156
- LPIPS: 0.09139405190944672
- Run time: 12m12s



Evaluation Metric

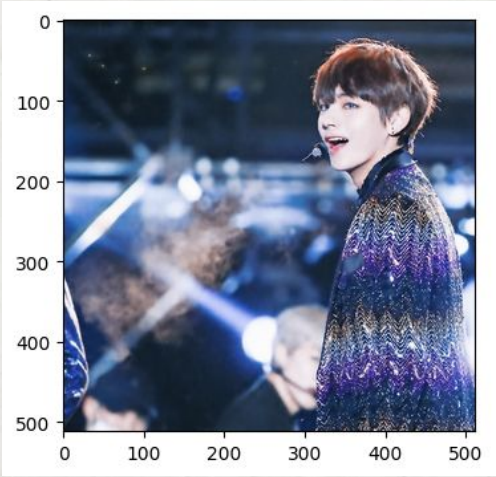
- Performance (quantitative comparison)

	Default	Reduce iteration	Different Optimizer	Different net type	Larger LR	Smaller LR
PSNR	28.51899	31.14807	23.30218	32.12607	26.33675	30.65419
SSIM	0.894469	0.93717	0.745059	0.95221	0.84120	0.93172
LPIP	0.151082	0.08219	0.284664	0.05375	0.19238	0.09139

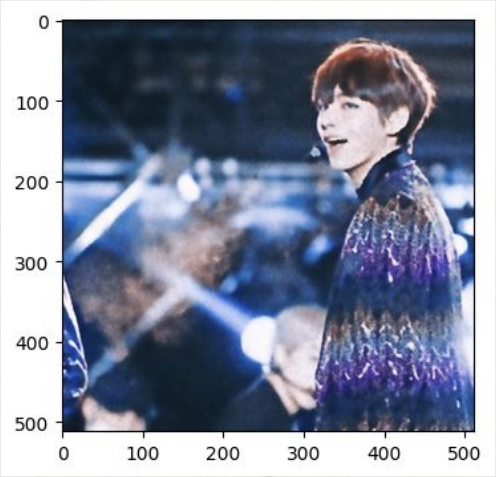
Evaluation Metric

- Performance (qualitative comparison)

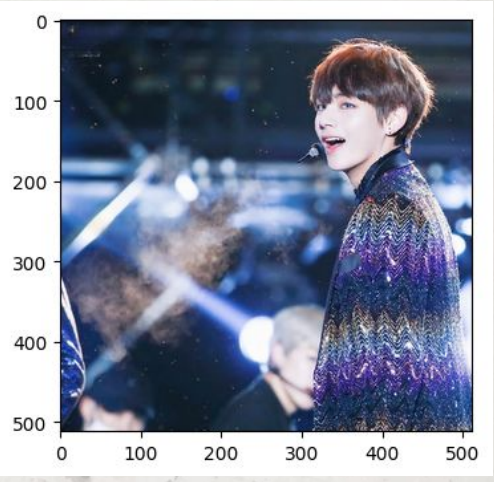
Default



Different Optimizer



Different type of net



Analysis & Limitation

Analysis

According to the evaluation matrices, we obtain two conclusions. Firstly, reducing iteration from **6,000** to **3,000** can improve not only **run time** but also the **overall performance**. Also, comparing to the 3,000 iteration version, implementing a **different type of net** performs better without extending the run time.

Limitation

- The size of output image is fixed to **512*512**, which limits the resolution and might lead to an underestimation of performance. In other words, evaluation outputs might be closer to ideal with a bigger and more clear output image.
- The training process takes tiiiiimme. If willing to reduce training time, a better model would have to be studied and implemented.

Links & References

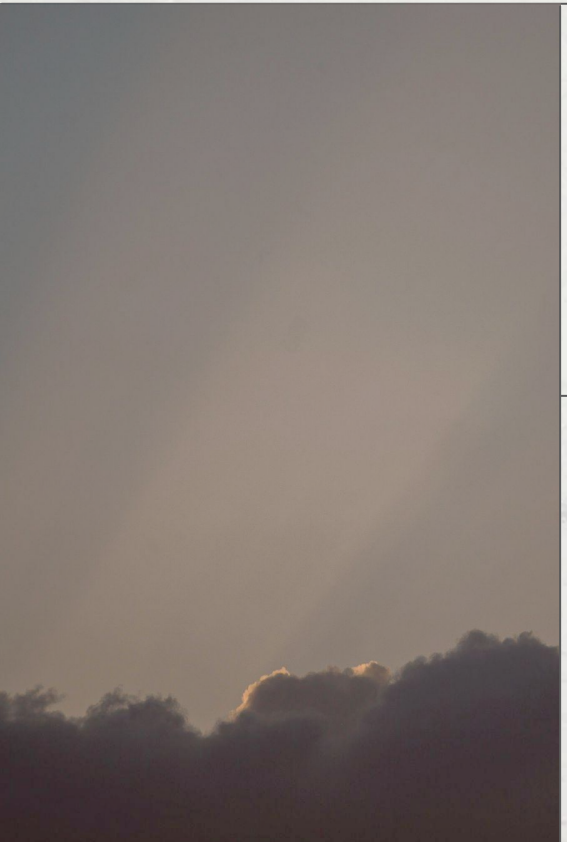
Our GitHub Repo

- https://github.com/Rebeccasun31/AI_Beautifier
- https://colab.research.google.com/drive/1Ppy05GkRldBtVnA4CPnw_FD-MMAg-ZbL?usp=sharing

References

- https://github.com/DmitryUlyanov/deep-image-prior
- https://github.com/ykdai/Flare7K
- https://github.com/timojl/clipseg

Contribution



孫承瑞 110550034 40%	李欣螢 110550033 20%
莊宥瑋 110550111 20%	葉家蓁 110550158 20%