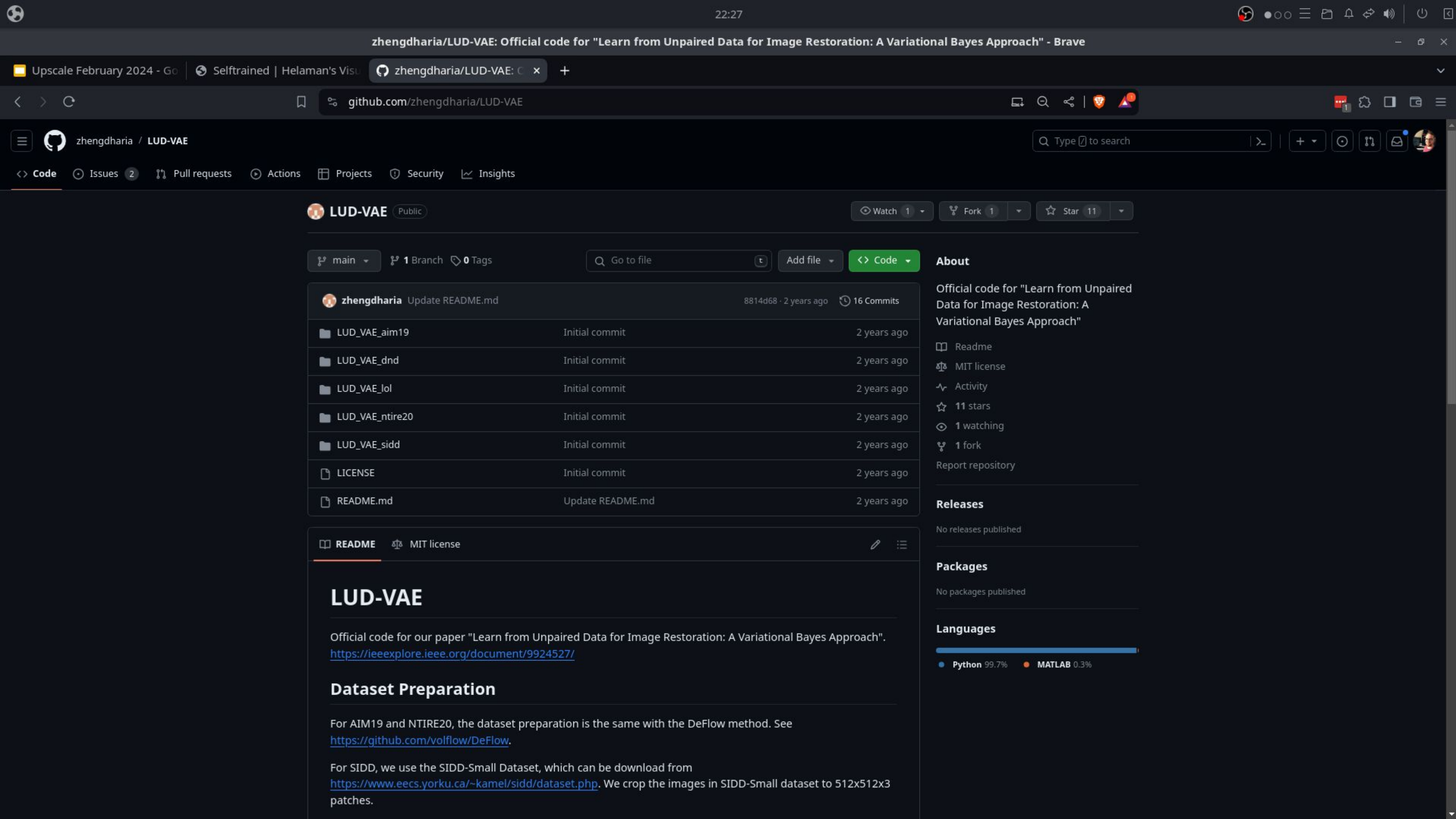
4xRealWebPhoto

Dataset Preparation

My own LUD-VAE model for realistic degradations

The goal here is to create a paired dataset with better realistic photo degradations and then train a photo model to evaluate how well it does.



22:27
zhengdharia/LUD-VAE: Official code for "Learn from Unpaired Data for Image Restoration: A Variational Bayes Approach" - Brave
Upscale February 2024 - Go
Selftrained | Helaman's Visu
zhengdharia/LUD-VAE: C
github.com/zhengdharia/LUD-VAE
zhengdharia / LUD-VAE
Type / to search
Code
Issues 2
Pull requests
Actions
Projects
Security
Insights
Public
Watch 1
Fork 1
Star 11
main
Go to file
t
Add file
Code

LUD-VAE

1 Branch · 0 Tags

zhengdharia Update README.md	8814d68 · 2 years ago	16 Commits
LUD_VAE_aim19	Initial commit	2 years ago
LUD_VAE_dnd	Initial commit	2 years ago
LUD_VAE_lol	Initial commit	2 years ago
LUD_VAE_ntire20	Initial commit	2 years ago
LUD_VAE_sidd	Initial commit	2 years ago
LICENSE	Initial commit	2 years ago
README.md	Update README.md	2 years ago

README · MIT license

LUD-VAE

Official code for our paper "Learn from Unpaired Data for Image Restoration: A Variational Bayes Approach". https://ieeexplore.ieee.org/document/9924527/

Dataset Preparation

For AIM19 and NTIRE20, the dataset preparation is the same with the DeFlow method. See https://github.com/volflow/DeFlow.

For SIDD, we use the SIDD-Small Dataset, which can be download from https://www.eecs.yorku.ca/~kamel/sidd/dataset.php. We crop the images in SIDD-Small dataset to 512x512x3 patches.

About

Official code for "Learn from Unpaired Data for Image Restoration: A Variational Bayes Approach"

Readme
MIT license
Activity
11 stars
1 watching
1 fork
Report repository

Releases

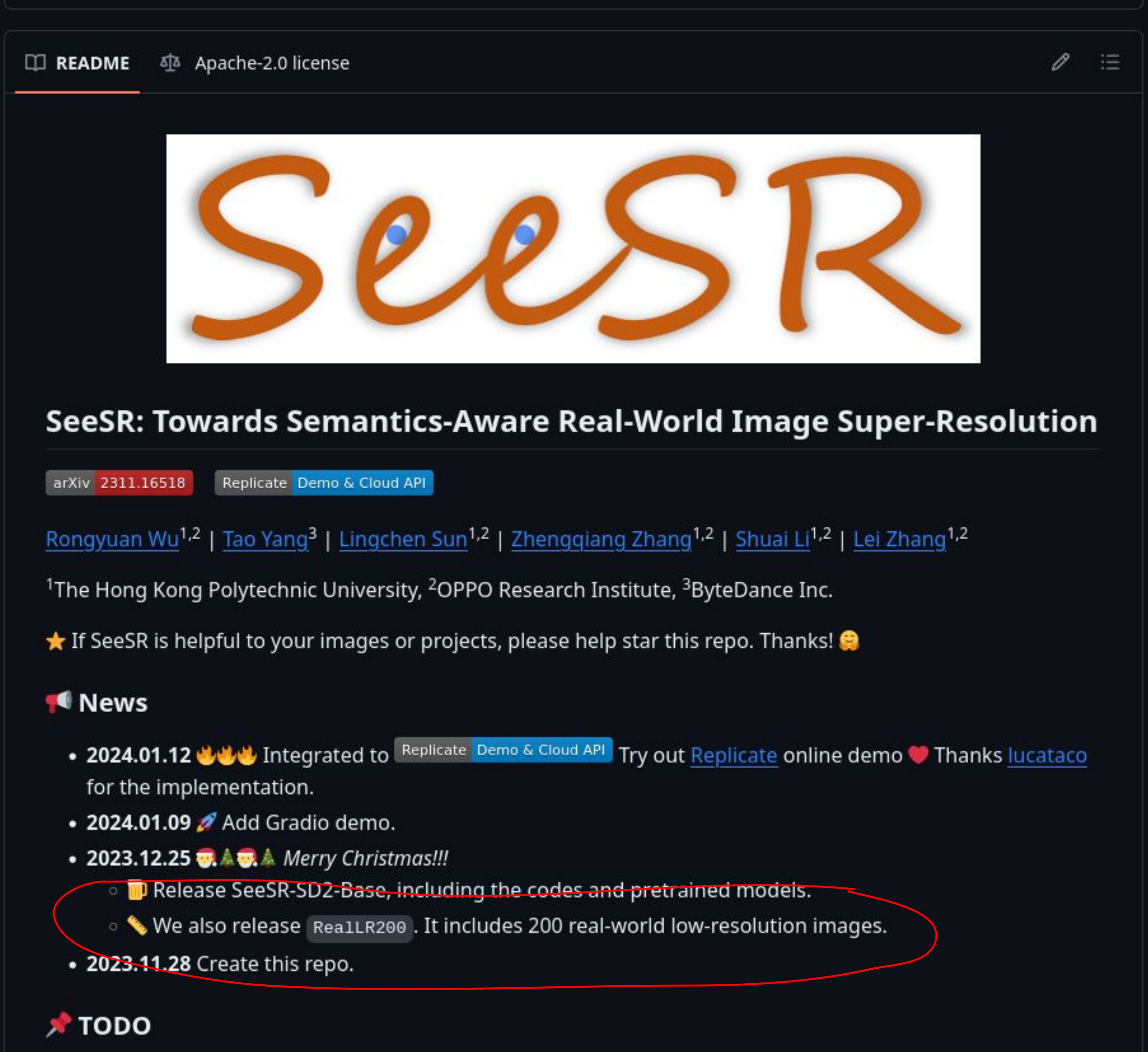
No releases published

Packages

No packages published

Languages

Python 99.7% · MATLAB 0.3%



SeeSR: Towards Semantics-Aware Real-World Image Super-Resolution

arXiv 2311.16518 Replicate Demo & Cloud API

Rongyuan Wu[1,2] | Tao Yang[3] | Lingchen Sun[1,2] | Zhengqiang Zhang[1,2] | Shuai Li[1,2] | Lei Zhang[1,2]

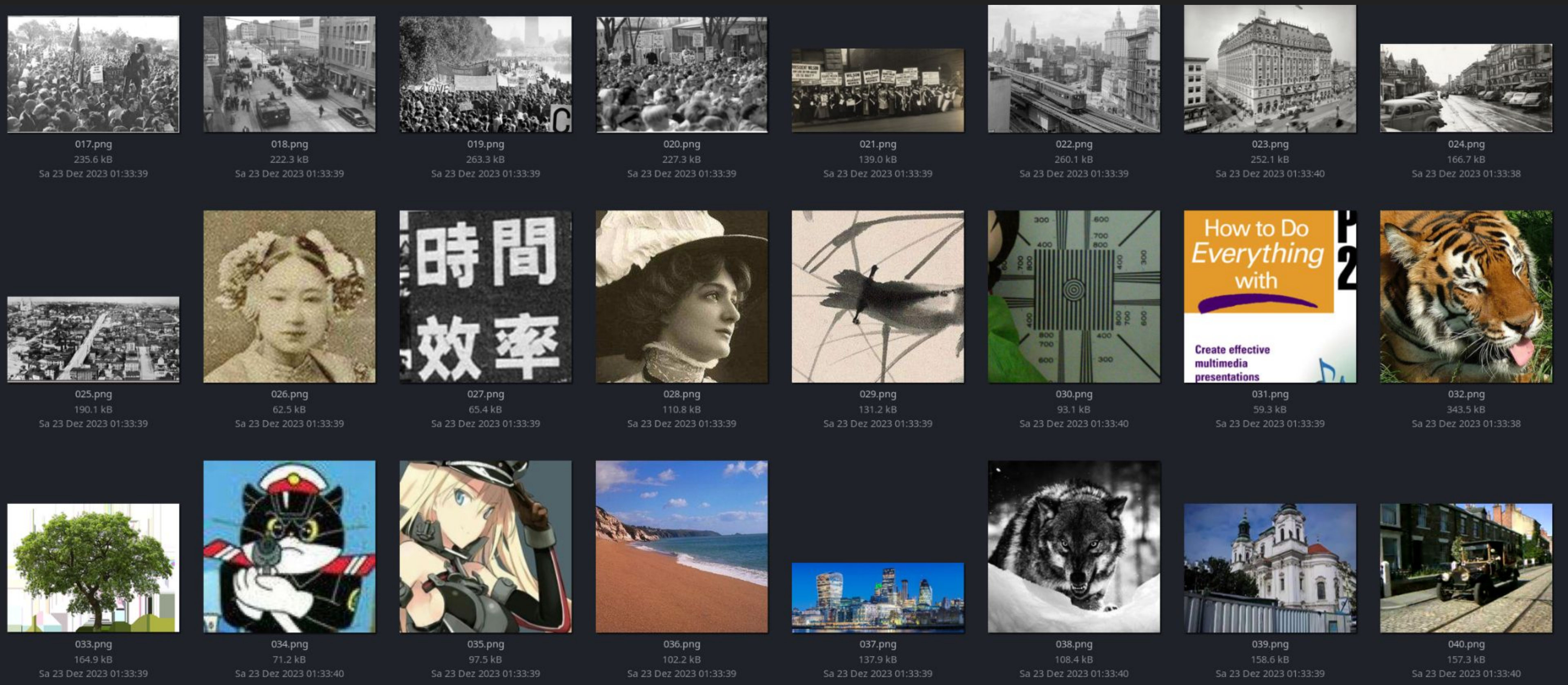
[1]The Hong Kong Polytechnic University, [2]OPPO Research Institute, [3]ByteDance Inc.

⭐ If SeeSR is helpful to your images or projects, please help star this repo. Thanks! 🤗

📢 News

- **2024.01.12** 🔥🔥🔥 Integrated to Replicate Demo & Cloud API Try out Replicate online demo ❤️ Thanks lucataco for the implementation.
- **2024.01.09** 🚀 Add Gradio demo.
- **2023.12.25** 🎅🎄🎅🎄 *Merry Christmas!!!*
  - 🍺 Release SeeSR-SD2-Base, including the codes and pretrained models.
  - 📏 We also release `RealLR200`. It includes 200 real-world low-resolution images.
- **2023.11.28** Create this repo.

📌 TODO



C
時間
效率
How to Do Everything with
Create effective multimedia presentations
P
2

017.png
235.6 kB
Sa 23 Dez 2023 01:33:39

018.png
222.3 kB
Sa 23 Dez 2023 01:33:39

019.png
263.3 kB
Sa 23 Dez 2023 01:33:39

020.png
227.3 kB
Sa 23 Dez 2023 01:33:39

021.png
139.0 kB
Sa 23 Dez 2023 01:33:39

022.png
260.1 kB
Sa 23 Dez 2023 01:33:39

023.png
252.1 kB
Sa 23 Dez 2023 01:33:40

024.png
166.7 kB
Sa 23 Dez 2023 01:33:38

025.png
190.1 kB
Sa 23 Dez 2023 01:33:39

026.png
62.5 kB
Sa 23 Dez 2023 01:33:39

027.png
65.4 kB
Sa 23 Dez 2023 01:33:39

028.png
110.8 kB
Sa 23 Dez 2023 01:33:39

029.png
131.2 kB
Sa 23 Dez 2023 01:33:39

030.png
93.1 kB
Sa 23 Dez 2023 01:33:40

031.png
59.3 kB
Sa 23 Dez 2023 01:33:39

032.png
343.5 kB
Sa 23 Dez 2023 01:33:38

033.png
164.9 kB
Sa 23 Dez 2023 01:33:39

034.png
71.2 kB
Sa 23 Dez 2023 01:33:40

035.png
97.5 kB
Sa 23 Dez 2023 01:33:39

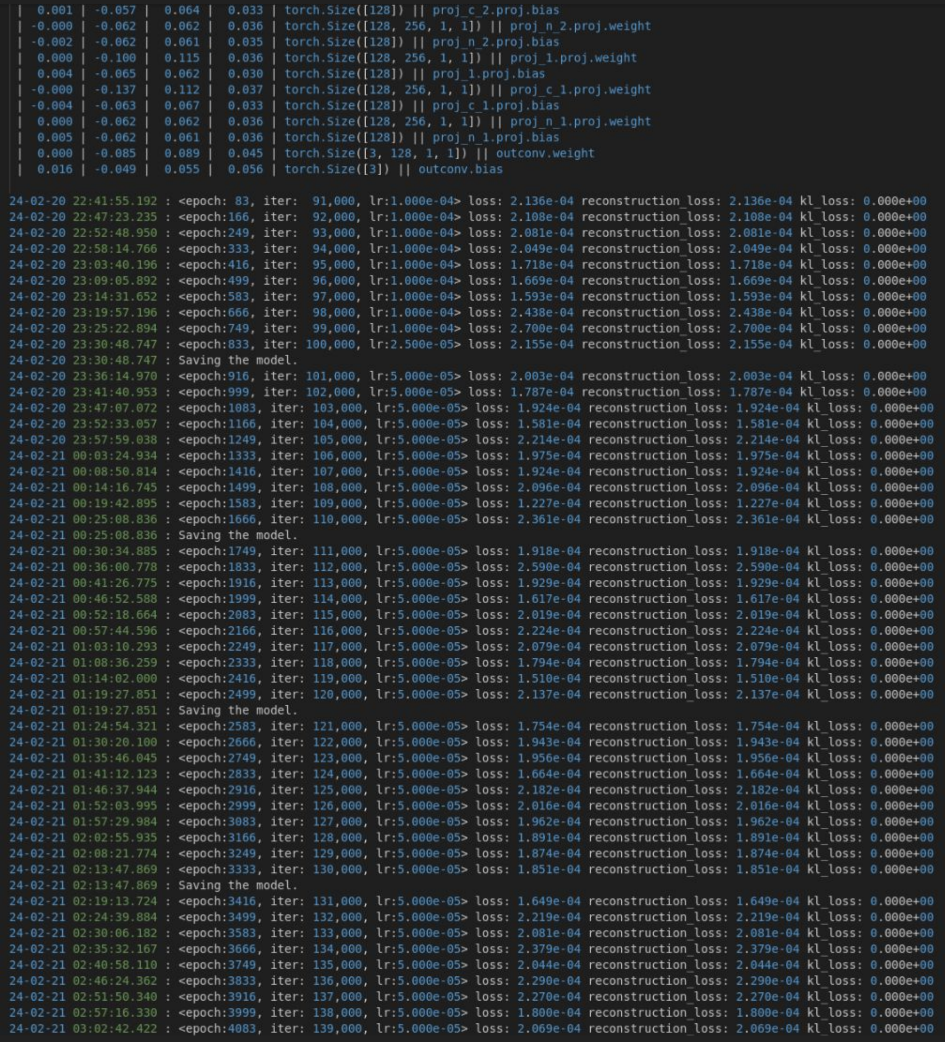
036.png
102.2 kB
Sa 23 Dez 2023 01:33:39

037.png
137.9 kB
Sa 23 Dez 2023 01:33:39

038.png
108.4 kB
Sa 23 Dez 2023 01:33:40

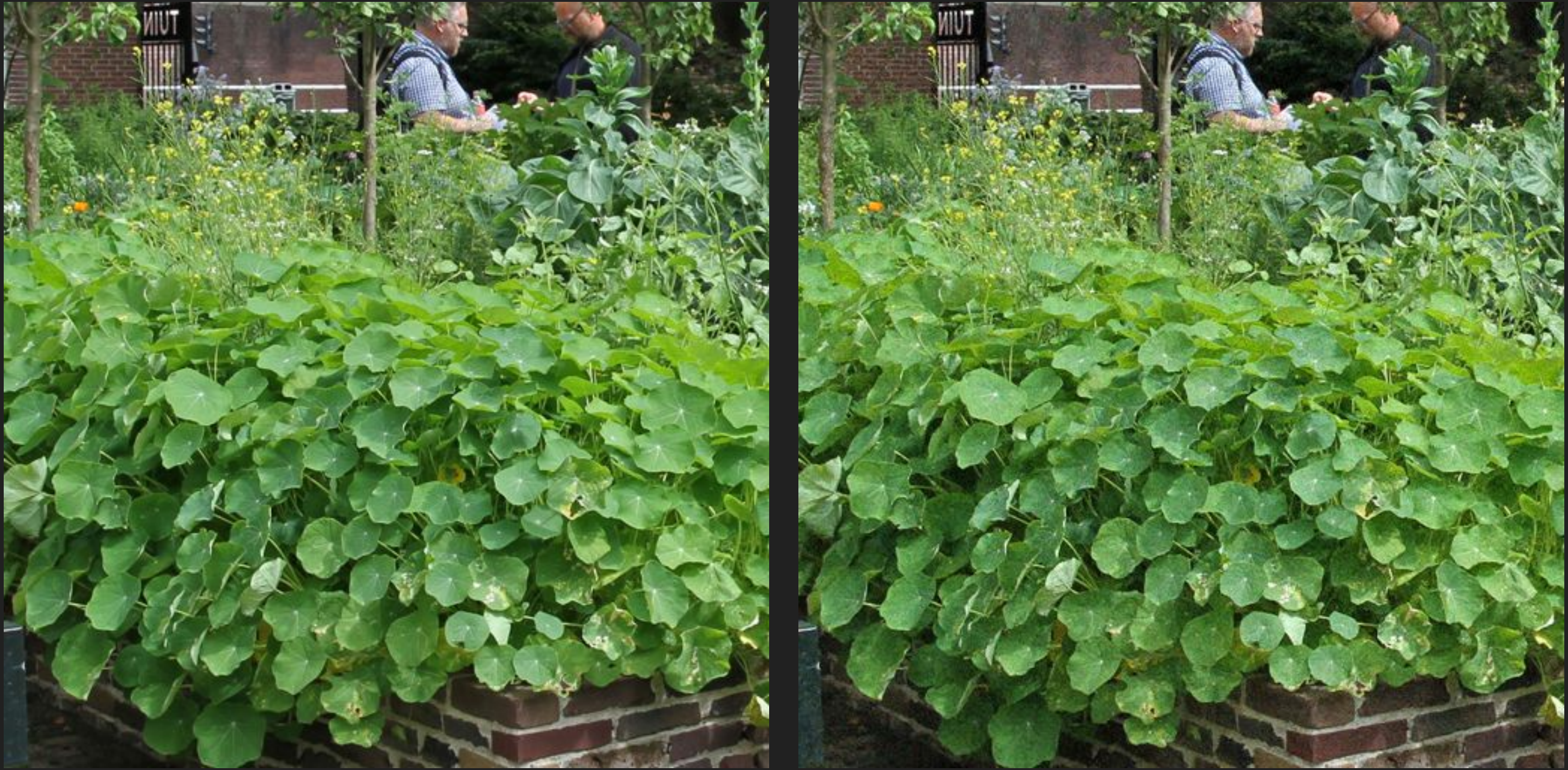
039.png
158.6 kB
Sa 23 Dez 2023 01:33:39

040.png
157.3 kB
Sa 23 Dez 2023 01:33:40



```
|  0.001 | -0.057 |  0.064 |  0.033 | torch.Size([128]) || proj_c_2.proj.bias
| -0.000 | -0.062 |  0.062 |  0.036 | torch.Size([128, 256, 1, 1]) || proj_n_2.proj.weight
| -0.002 | -0.062 |  0.061 |  0.035 | torch.Size([128]) || proj_n_2.proj.bias
|  0.000 | -0.100 |  0.115 |  0.036 | torch.Size([128, 256, 1, 1]) || proj_1.proj.weight
|  0.004 | -0.065 |  0.062 |  0.030 | torch.Size([128]) || proj_1.proj.bias
| -0.000 | -0.137 |  0.112 |  0.037 | torch.Size([128, 256, 1, 1]) || proj_c_1.proj.weight
| -0.004 | -0.063 |  0.067 |  0.033 | torch.Size([128]) || proj_c_1.proj.bias
|  0.000 | -0.062 |  0.062 |  0.036 | torch.Size([128, 256, 1, 1]) || proj_n_1.proj.weight
|  0.005 | -0.062 |  0.061 |  0.036 | torch.Size([128]) || proj_n_1.proj.bias
|  0.000 | -0.085 |  0.089 |  0.045 | torch.Size([3, 128, 1, 1]) || outconv.weight
|  0.016 | -0.049 |  0.055 |  0.056 | torch.Size([3]) || outconv.bias

24-02-20 22:41:55.192 : <epoch: 83, iter:  91,000, lr:1.000e-04> loss: 2.136e-04 reconstruction_loss: 2.136e-04 kl_loss: 0.000e+00
24-02-20 22:47:23.235 : <epoch:166, iter:  92,000, lr:1.000e-04> loss: 2.108e-04 reconstruction_loss: 2.108e-04 kl_loss: 0.000e+00
24-02-20 22:52:48.950 : <epoch:249, iter:  93,000, lr:1.000e-04> loss: 2.081e-04 reconstruction_loss: 2.081e-04 kl_loss: 0.000e+00
24-02-20 22:58:14.766 : <epoch:333, iter:  94,000, lr:1.000e-04> loss: 2.049e-04 reconstruction_loss: 2.049e-04 kl_loss: 0.000e+00
24-02-20 23:03:40.196 : <epoch:416, iter:  95,000, lr:1.000e-04> loss: 1.718e-04 reconstruction_loss: 1.718e-04 kl_loss: 0.000e+00
24-02-20 23:09:05.892 : <epoch:499, iter:  96,000, lr:1.000e-04> loss: 1.669e-04 reconstruction_loss: 1.669e-04 kl_loss: 0.000e+00
24-02-20 23:14:31.652 : <epoch:583, iter:  97,000, lr:1.000e-04> loss: 1.593e-04 reconstruction_loss: 1.593e-04 kl_loss: 0.000e+00
24-02-20 23:19:57.196 : <epoch:666, iter:  98,000, lr:1.000e-04> loss: 2.438e-04 reconstruction_loss: 2.438e-04 kl_loss: 0.000e+00
24-02-20 23:25:22.894 : <epoch:749, iter:  99,000, lr:1.000e-04> loss: 2.700e-04 reconstruction_loss: 2.700e-04 kl_loss: 0.000e+00
24-02-20 23:30:48.747 : <epoch:833, iter: 100,000, lr:2.500e-05> loss: 2.155e-04 reconstruction_loss: 2.155e-04 kl_loss: 0.000e+00
24-02-20 23:30:48.747 : Saving the model.
24-02-20 23:36:14.970 : <epoch:916, iter: 101,000, lr:5.000e-05> loss: 2.003e-04 reconstruction_loss: 2.003e-04 kl_loss: 0.000e+00
24-02-20 23:41:40.953 : <epoch:999, iter: 102,000, lr:5.000e-05> loss: 1.787e-04 reconstruction_loss: 1.787e-04 kl_loss: 0.000e+00
24-02-20 23:47:07.072 : <epoch:1083, iter: 103,000, lr:5.000e-05> loss: 1.924e-04 reconstruction_loss: 1.924e-04 kl_loss: 0.000e+00
24-02-20 23:52:33.057 : <epoch:1166, iter: 104,000, lr:5.000e-05> loss: 1.581e-04 reconstruction_loss: 1.581e-04 kl_loss: 0.000e+00
24-02-20 23:57:59.038 : <epoch:1249, iter: 105,000, lr:5.000e-05> loss: 2.214e-04 reconstruction_loss: 2.214e-04 kl_loss: 0.000e+00
24-02-21 00:03:24.934 : <epoch:1333, iter: 106,000, lr:5.000e-05> loss: 1.975e-04 reconstruction_loss: 1.975e-04 kl_loss: 0.000e+00
24-02-21 00:08:50.814 : <epoch:1416, iter: 107,000, lr:5.000e-05> loss: 1.924e-04 reconstruction_loss: 1.924e-04 kl_loss: 0.000e+00
24-02-21 00:14:16.745 : <epoch:1499, iter: 108,000, lr:5.000e-05> loss: 2.096e-04 reconstruction_loss: 2.096e-04 kl_loss: 0.000e+00
24-02-21 00:19:42.895 : <epoch:1583, iter: 109,000, lr:5.000e-05> loss: 1.227e-04 reconstruction_loss: 1.227e-04 kl_loss: 0.000e+00
24-02-21 00:25:08.836 : <epoch:1666, iter: 110,000, lr:5.000e-05> loss: 2.361e-04 reconstruction_loss: 2.361e-04 kl_loss: 0.000e+00
24-02-21 00:25:08.836 : Saving the model.
24-02-21 00:30:34.885 : <epoch:1749, iter: 111,000, lr:5.000e-05> loss: 1.918e-04 reconstruction_loss: 1.918e-04 kl_loss: 0.000e+00
24-02-21 00:36:00.778 : <epoch:1833, iter: 112,000, lr:5.000e-05> loss: 2.590e-04 reconstruction_loss: 2.590e-04 kl_loss: 0.000e+00
24-02-21 00:41:26.775 : <epoch:1916, iter: 113,000, lr:5.000e-05> loss: 1.929e-04 reconstruction_loss: 1.929e-04 kl_loss: 0.000e+00
24-02-21 00:46:52.588 : <epoch:1999, iter: 114,000, lr:5.000e-05> loss: 1.617e-04 reconstruction_loss: 1.617e-04 kl_loss: 0.000e+00
24-02-21 00:52:18.664 : <epoch:2083, iter: 115,000, lr:5.000e-05> loss: 2.019e-04 reconstruction_loss: 2.019e-04 kl_loss: 0.000e+00
24-02-21 00:57:44.596 : <epoch:2166, iter: 116,000, lr:5.000e-05> loss: 2.224e-04 reconstruction_loss: 2.224e-04 kl_loss: 0.000e+00
24-02-21 01:03:10.293 : <epoch:2249, iter: 117,000, lr:5.000e-05> loss: 2.079e-04 reconstruction_loss: 2.079e-04 kl_loss: 0.000e+00
24-02-21 01:08:36.259 : <epoch:2333, iter: 118,000, lr:5.000e-05> loss: 1.794e-04 reconstruction_loss: 1.794e-04 kl_loss: 0.000e+00
24-02-21 01:14:02.000 : <epoch:2416, iter: 119,000, lr:5.000e-05> loss: 1.510e-04 reconstruction_loss: 1.510e-04 kl_loss: 0.000e+00
24-02-21 01:19:27.851 : <epoch:2499, iter: 120,000, lr:5.000e-05> loss: 2.137e-04 reconstruction_loss: 2.137e-04 kl_loss: 0.000e+00
24-02-21 01:19:27.851 : Saving the model.
24-02-21 01:24:54.321 : <epoch:2583, iter: 121,000, lr:5.000e-05> loss: 1.754e-04 reconstruction_loss: 1.754e-04 kl_loss: 0.000e+00
24-02-21 01:30:20.100 : <epoch:2666, iter: 122,000, lr:5.000e-05> loss: 1.943e-04 reconstruction_loss: 1.943e-04 kl_loss: 0.000e+00
24-02-21 01:35:46.045 : <epoch:2749, iter: 123,000, lr:5.000e-05> loss: 1.956e-04 reconstruction_loss: 1.956e-04 kl_loss: 0.000e+00
24-02-21 01:41:12.123 : <epoch:2833, iter: 124,000, lr:5.000e-05> loss: 1.664e-04 reconstruction_loss: 1.664e-04 kl_loss: 0.000e+00
24-02-21 01:46:37.944 : <epoch:2916, iter: 125,000, lr:5.000e-05> loss: 2.182e-04 reconstruction_loss: 2.182e-04 kl_loss: 0.000e+00
24-02-21 01:52:03.995 : <epoch:2999, iter: 126,000, lr:5.000e-05> loss: 2.016e-04 reconstruction_loss: 2.016e-04 kl_loss: 0.000e+00
24-02-21 01:57:29.984 : <epoch:3083, iter: 127,000, lr:5.000e-05> loss: 1.962e-04 reconstruction_loss: 1.962e-04 kl_loss: 0.000e+00
24-02-21 02:02:55.935 : <epoch:3166, iter: 128,000, lr:5.000e-05> loss: 1.891e-04 reconstruction_loss: 1.891e-04 kl_loss: 0.000e+00
24-02-21 02:08:21.774 : <epoch:3249, iter: 129,000, lr:5.000e-05> loss: 1.874e-04 reconstruction_loss: 1.874e-04 kl_loss: 0.000e+00
24-02-21 02:13:47.869 : <epoch:3333, iter: 130,000, lr:5.000e-05> loss: 1.851e-04 reconstruction_loss: 1.851e-04 kl_loss: 0.000e+00
24-02-21 02:13:47.869 : Saving the model.
24-02-21 02:19:13.724 : <epoch:3416, iter: 131,000, lr:5.000e-05> loss: 1.649e-04 reconstruction_loss: 1.649e-04 kl_loss: 0.000e+00
24-02-21 02:24:39.884 : <epoch:3499, iter: 132,000, lr:5.000e-05> loss: 2.219e-04 reconstruction_loss: 2.219e-04 kl_loss: 0.000e+00
24-02-21 02:30:06.182 : <epoch:3583, iter: 133,000, lr:5.000e-05> loss: 2.081e-04 reconstruction_loss: 2.081e-04 kl_loss: 0.000e+00
24-02-21 02:35:32.167 : <epoch:3666, iter: 134,000, lr:5.000e-05> loss: 2.379e-04 reconstruction_loss: 2.379e-04 kl_loss: 0.000e+00
24-02-21 02:40:58.110 : <epoch:3749, iter: 135,000, lr:5.000e-05> loss: 2.044e-04 reconstruction_loss: 2.044e-04 kl_loss: 0.000e+00
24-02-21 02:46:24.362 : <epoch:3833, iter: 136,000, lr:5.000e-05> loss: 2.290e-04 reconstruction_loss: 2.290e-04 kl_loss: 0.000e+00
24-02-21 02:51:50.340 : <epoch:3916, iter: 137,000, lr:5.000e-05> loss: 2.270e-04 reconstruction_loss: 2.270e-04 kl_loss: 0.000e+00
24-02-21 02:57:16.330 : <epoch:3999, iter: 138,000, lr:5.000e-05> loss: 1.800e-04 reconstruction_loss: 1.800e-04 kl_loss: 0.000e+00
24-02-21 03:02:42.422 : <epoch:4083, iter: 139,000, lr:5.000e-05> loss: 2.069e-04 reconstruction_loss: 2.069e-04 kl_loss: 0.000e+00
```

2 nights, ~ 17 hours,
190’000 iters



Example: Realistic noise added with my own degradation models

Gameplan: 4xRealWebPhoto Paired Dataset

Simulating usecase:

Someone taking a photo (with a bit of noise and blur in it), then uploads it on the web (social media / travel blog / website etc) where the service automatically downscaled and compresses the image for web usage.

Then another Person liking the photo, downloading it, and re-uploading it on the web (where provider automatically scales and compresses for web usage again).

So here we create a paired 4x Dataset and simulate these degradations by taking a photography training dataset and then:

Applying realistic (lens) blur

Applying realistic noise

Downsample to half and then jpg compress. Here we are using multiple downsampling algorithms at random, and also randomized jpg compression between 60-100 (google search image preview would be at around 70, so this would also catch the case of someone downloading the google preview image of a good quality photo). I use multiple downsampling algorithms and a compression range since I do not know what default values a service provider would use so we just handle them all.

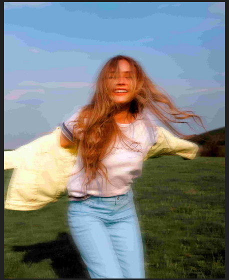
Re-downsampling and re-compressing again (so previous step again). Now the lr’s will be at quarter size for 4x paired training dataset and we have simulated the use case of downloading and upscaling a photo from the web.

Photo, little bit of blur and noise

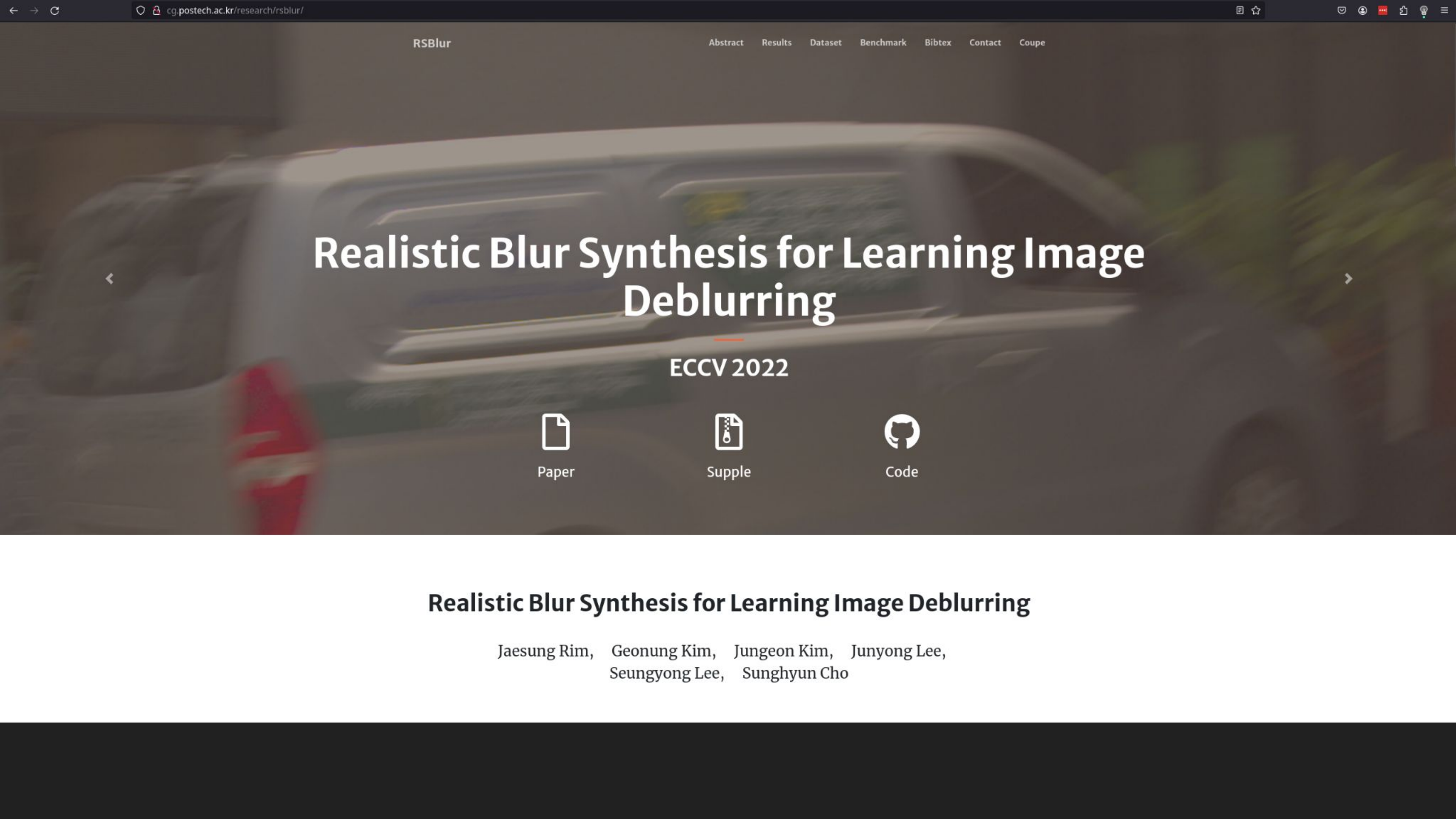
Web Service scales and compresses

Some user downloads the image

Web Service scales and compresses



Some other User downloads image, wants to upscale



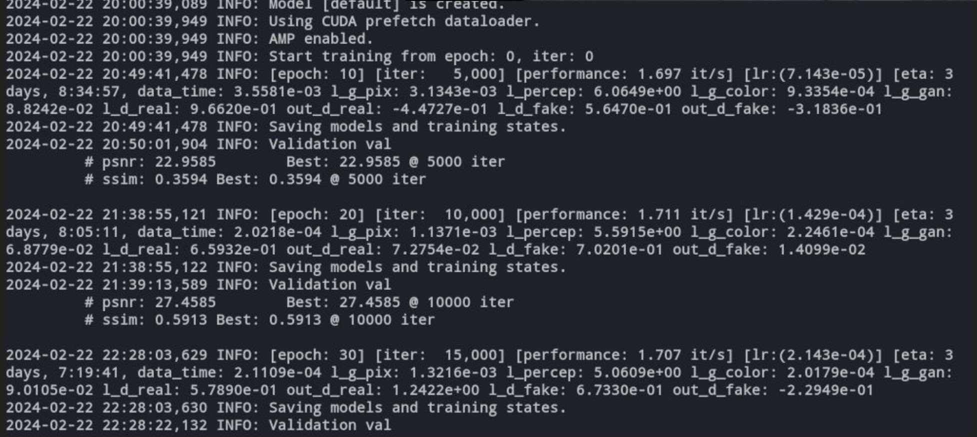
Realistic Blur Synthesis for Learning Image Deblurring

ECCV 2022

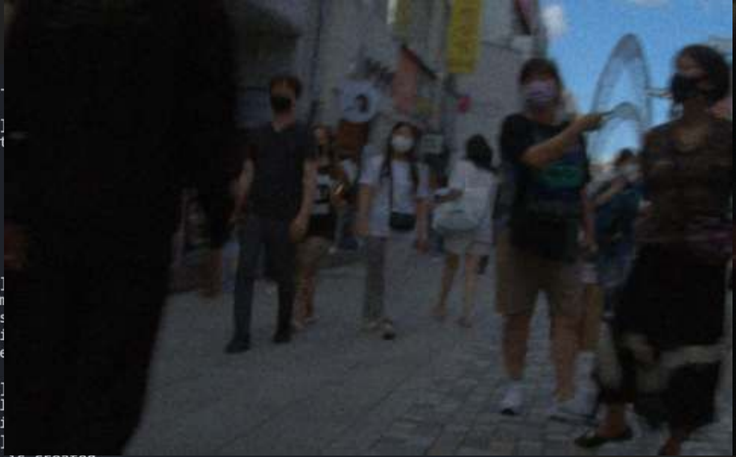
Realistic Blur Synthesis for Learning Image Deblurring

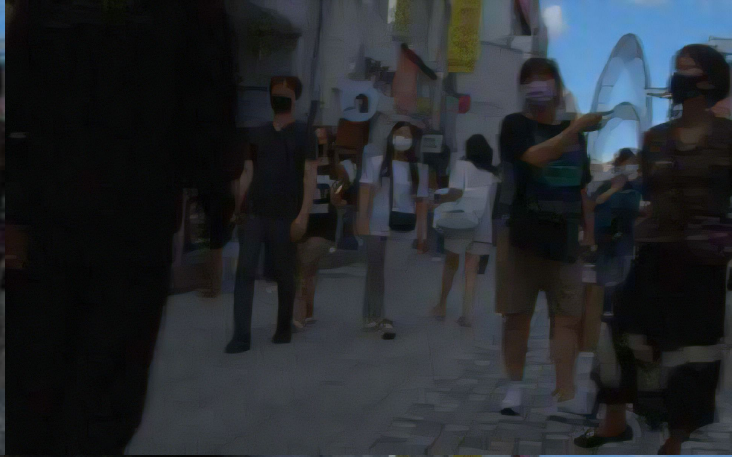
Jaesung Rim, Geonung Kim, Jungeon Kim, Junyong Lee,
Seungyong Lee, Sunghyun Cho

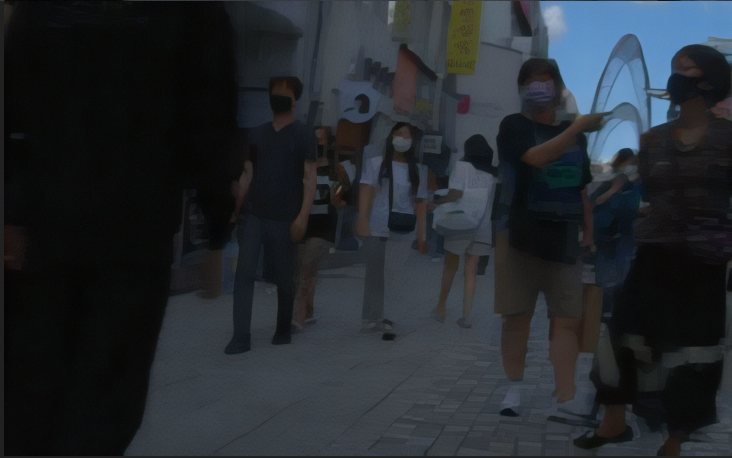
Example images from the RSBlur Dataset after I finished degrading them - more infos in the appendix on this process



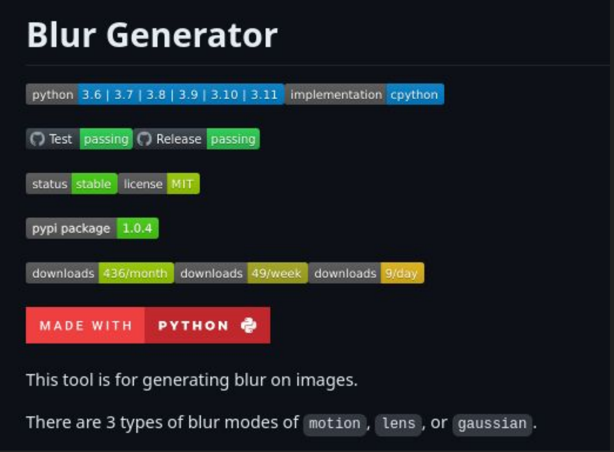
```
2024-02-22 20:00:25,352 INFO:
------------------------ neosr ---------------
Pytorch Version: 2.2.0.dev20230914+cu121
2024-02-22 20:00:35,285 INFO: Dataset [paired]
2024-02-22 20:00:35,285 INFO: Training statist
        Starting model: 4xRealSISR_rgt_s
        Number of train images: 1175
        Dataset enlarge ratio: 5
        Batch size per gpu: 12
        World size (gpu number): 1
        Required iters per epoch: 490
        Total epochs: 1021; iters: 500000.
2024-02-22 20:00:35,285 INFO: Dataset [paired]
2024-02-22 20:00:35,286 INFO: Number of val im
2024-02-22 20:00:36,327 INFO: Network [rgt] is
2024-02-22 20:00:36,421 INFO: Network [unet] i
2024-02-22 20:00:38,597 INFO: Loading rgt mode
[params].
2024-02-22 20:00:38,748 INFO: Loss [HuberLoss]
2024-02-22 20:00:39,080 INFO: Loss [Perceptual
2024-02-22 20:00:39,081 INFO: Loss [GANLoss] i
2024-02-22 20:00:39,081 INFO: Loss [colorloss]
2024-02-22 20:00:39,089 INFO: Model [default] is created.
2024-02-22 20:00:39,949 INFO: Using CUDA prefetch dataloader.
2024-02-22 20:00:39,949 INFO: AMP enabled.
2024-02-22 20:00:39,949 INFO: Start training from epoch: 0, iter: 0
2024-02-22 20:49:41,478 INFO: [epoch: 10] [iter:   5,000] [performance: 1.697 it/s] [lr:(7.143e-05)] [eta: 3
days, 8:34:57, data_time: 3.5581e-03 l_g_pix: 3.1343e-03 l_percep: 6.0649e+00 l_g_color: 9.3354e-04 l_g_gan:
8.8242e-02 l_d_real: 9.6620e-01 out_d_real: -4.4727e-01 l_d_fake: 5.6470e-01 out_d_fake: -3.1836e-01
2024-02-22 20:49:41,478 INFO: Saving models and training states.
2024-02-22 20:50:01,904 INFO: Validation val
         # psnr: 22.9585        Best: 22.9585 @ 5000 iter
         # ssim: 0.3594 Best: 0.3594 @ 5000 iter

2024-02-22 21:38:55,121 INFO: [epoch: 20] [iter:  10,000] [performance: 1.711 it/s] [lr:(1.429e-04)] [eta: 3
days, 8:05:11, data_time: 2.0218e-04 l_g_pix: 1.1371e-03 l_percep: 5.5915e+00 l_g_color: 2.2461e-04 l_g_gan:
6.8779e-02 l_d_real: 6.5932e-01 out_d_real: 7.2754e-02 l_d_fake: 7.0201e-01 out_d_fake: 1.4099e-02
2024-02-22 21:38:55,122 INFO: Saving models and training states.
2024-02-22 21:39:13,589 INFO: Validation val
         # psnr: 27.4585        Best: 27.4585 @ 10000 iter
         # ssim: 0.5913 Best: 0.5913 @ 10000 iter

2024-02-22 22:28:03,629 INFO: [epoch: 30] [iter:  15,000] [performance: 1.707 it/s] [lr:(2.143e-04)] [eta: 3
days, 7:19:41, data_time: 2.1109e-04 l_g_pix: 1.3216e-03 l_percep: 5.0609e+00 l_g_color: 2.0179e-04 l_g_gan:
9.0105e-02 l_d_real: 5.7890e-01 out_d_real: 1.2422e+00 l_d_fake: 6.7330e-01 out_d_fake: -2.2949e-01
2024-02-22 22:28:03,630 INFO: Saving models and training states.
2024-02-22 22:28:22,132 INFO: Validation val
```







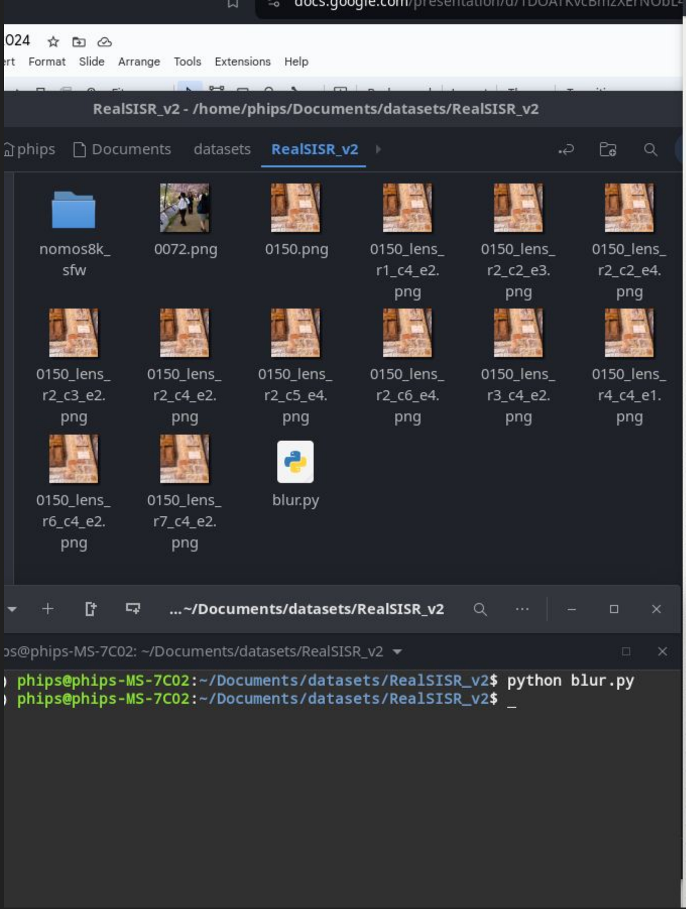
Trained 2 experiment models with it, compact for 40k and rgt_s for 70k.
Degrdataions / Motion blur in dataset is too strong.
Better: Find a method to degrade with realistic blur with adjustable strengths instead of using pre-blurred dataset I cannot adjust



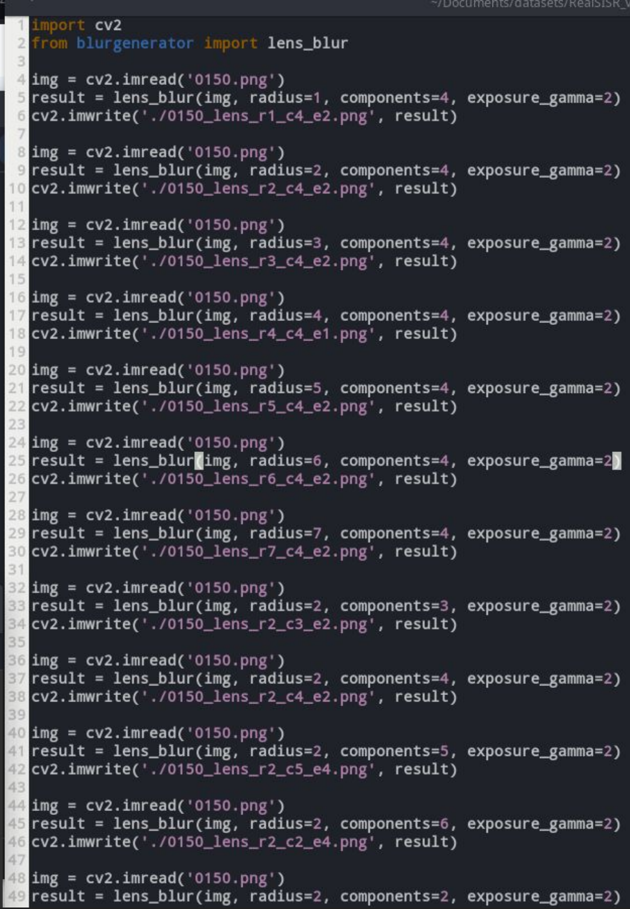
Blur Generator

This tool is for generating blur on images.

There are 3 types of blur modes of `motion`, `lens`, or `gaussian`.

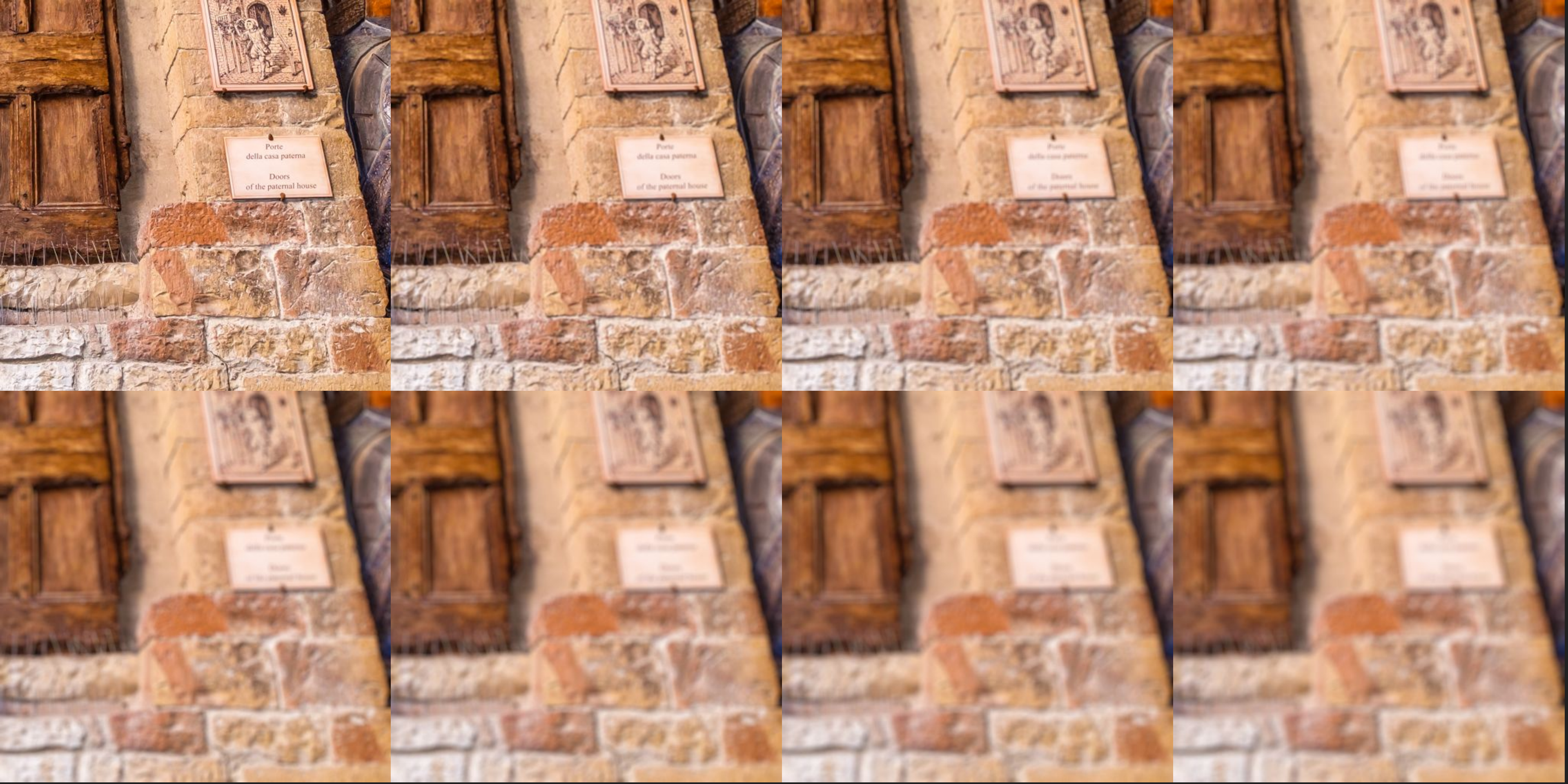


```
) phips@phips-MS-7C02:~/Documents/datasets/RealSISR_v2$ python blur.py
) phips@phips-MS-7C02:~/Documents/datasets/RealSISR_v2$ _
```



```python
import cv2
from blurgenerator import lens_blur

img = cv2.imread('0150.png')
result = lens_blur(img, radius=1, components=4, exposure_gamma=2)
cv2.imwrite('./0150_lens_r1_c4_e2.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=2, components=4, exposure_gamma=2)
cv2.imwrite('./0150_lens_r2_c4_e2.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=3, components=4, exposure_gamma=2)
cv2.imwrite('./0150_lens_r3_c4_e2.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=4, components=4, exposure_gamma=2)
cv2.imwrite('./0150_lens_r4_c4_e1.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=5, components=4, exposure_gamma=2)
cv2.imwrite('./0150_lens_r5_c4_e2.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=6, components=4, exposure_gamma=2)
cv2.imwrite('./0150_lens_r6_c4_e2.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=7, components=4, exposure_gamma=2)
cv2.imwrite('./0150_lens_r7_c4_e2.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=2, components=3, exposure_gamma=2)
cv2.imwrite('./0150_lens_r2_c3_e2.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=2, components=4, exposure_gamma=2)
cv2.imwrite('./0150_lens_r2_c4_e2.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=2, components=5, exposure_gamma=2)
cv2.imwrite('./0150_lens_r2_c5_e4.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=2, components=6, exposure_gamma=2)
cv2.imwrite('./0150_lens_r2_c2_e4.png', result)

img = cv2.imread('0150.png')
result = lens_blur(img, radius=2, components=2, exposure_gamma=2)
```

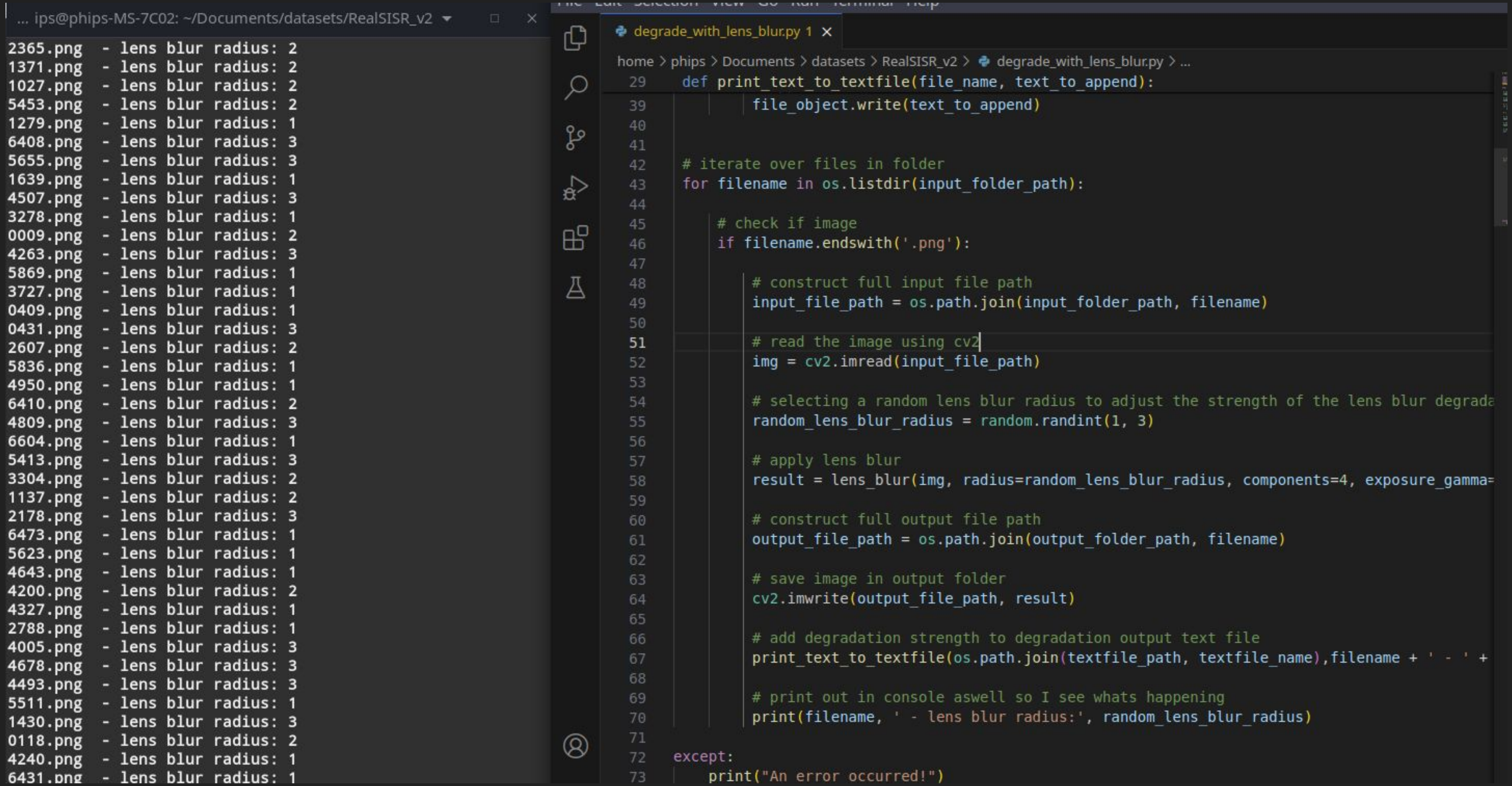
Testing out lens blurs / strengths



Testing strengths visualization lens blur

So after testing and what I learned from last dataset try, I now have a complete degradation pipeline.
This time, I took the nomos8k_sfw dataset from musl, which is a photo dataset.
It has good variety, consists of 6118 images of 512x512 px.
Since its 512x512 I will be able to apply realistic blur, then realistic noise (no out of vram), then downscale+jpg, downscale+jpg like previously.
(and its sfw so I can show here)

Example images nomos8k_sfw


... ips@phips-MS-7C02: ~/Documents/datasets/RealSISR_v2
File Edit Selection View Go Run Terminal Help
degrade_with_lens_blur.py 1
home > phips > Documents > datasets > RealSISR_v2 > degrade_with_lens_blur.py > ...

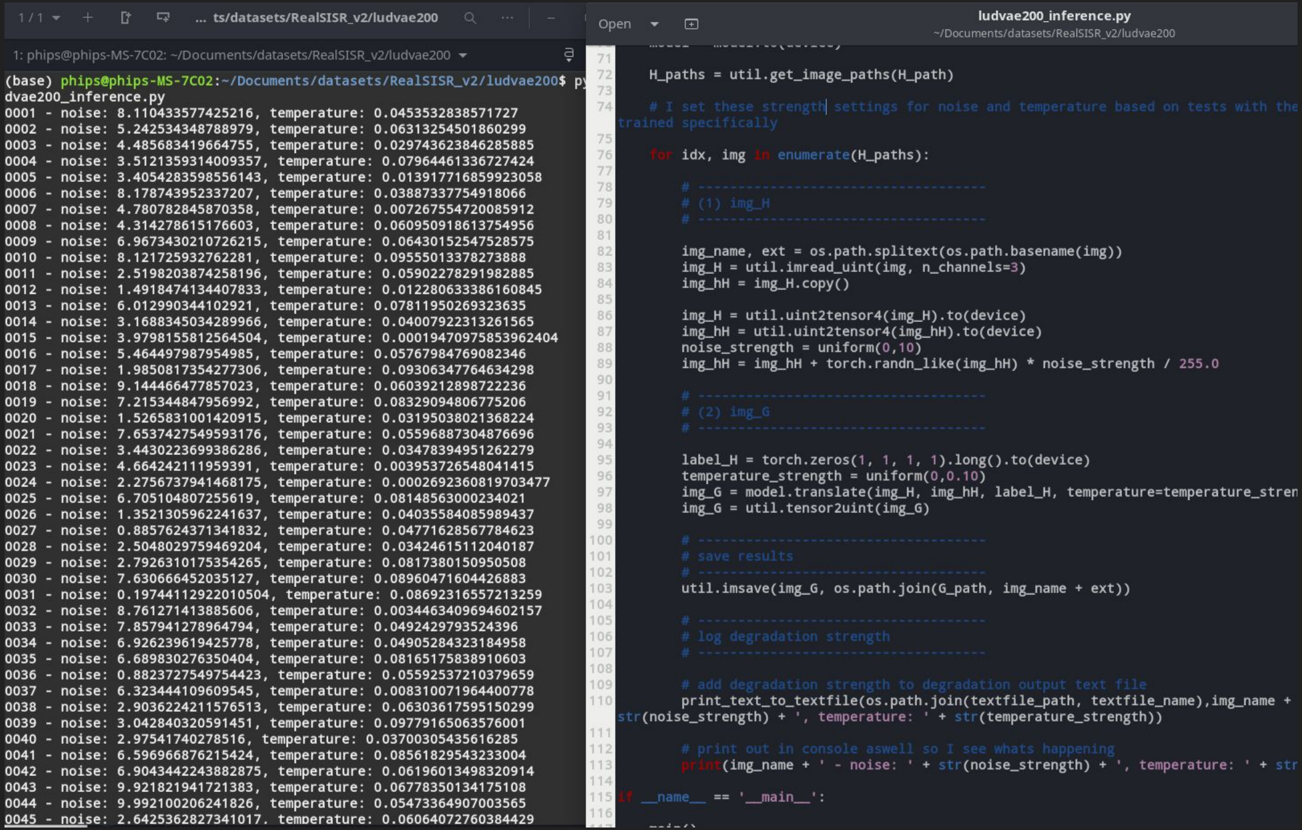
```
2365.png  - lens blur radius: 2
1371.png  - lens blur radius: 2
1027.png  - lens blur radius: 2
5453.png  - lens blur radius: 2
1279.png  - lens blur radius: 1
6408.png  - lens blur radius: 3
5655.png  - lens blur radius: 3
1639.png  - lens blur radius: 1
4507.png  - lens blur radius: 3
3278.png  - lens blur radius: 1
0009.png  - lens blur radius: 2
4263.png  - lens blur radius: 3
5869.png  - lens blur radius: 1
3727.png  - lens blur radius: 1
0409.png  - lens blur radius: 1
0431.png  - lens blur radius: 3
2607.png  - lens blur radius: 2
5836.png  - lens blur radius: 1
4950.png  - lens blur radius: 1
6410.png  - lens blur radius: 2
4809.png  - lens blur radius: 3
6604.png  - lens blur radius: 1
5413.png  - lens blur radius: 3
3304.png  - lens blur radius: 2
1137.png  - lens blur radius: 2
2178.png  - lens blur radius: 3
6473.png  - lens blur radius: 1
5623.png  - lens blur radius: 1
4643.png  - lens blur radius: 1
4200.png  - lens blur radius: 2
4327.png  - lens blur radius: 1
2788.png  - lens blur radius: 1
4005.png  - lens blur radius: 3
4678.png  - lens blur radius: 3
4493.png  - lens blur radius: 3
5511.png  - lens blur radius: 1
1430.png  - lens blur radius: 3
0118.png  - lens blur radius: 2
4240.png  - lens blur radius: 1
6431.png  - lens blur radius: 1
```

```python
    def print_text_to_textfile(file_name, text_to_append):
            file_object.write(text_to_append)


    # iterate over files in folder
    for filename in os.listdir(input_folder_path):

        # check if image
        if filename.endswith('.png'):

            # construct full input file path
            input_file_path = os.path.join(input_folder_path, filename)

            # read the image using cv2
            img = cv2.imread(input_file_path)

            # selecting a random lens blur radius to adjust the strength of the lens blur degrada
            random_lens_blur_radius = random.randint(1, 3)

            # apply lens blur
            result = lens_blur(img, radius=random_lens_blur_radius, components=4, exposure_gamma=

            # construct full output file path
            output_file_path = os.path.join(output_folder_path, filename)

            # save image in output folder
            cv2.imwrite(output_file_path, result)

            # add degradation strength to degradation output text file
            print_text_to_textfile(os.path.join(textfile_path, textfile_name),filename + ' - ' +

            # print out in console aswell so I see whats happening
            print(filename, ' - lens blur radius:', random_lens_blur_radius)

except:
    print("An error occurred!")
```

Made a python script to apply random lens blur strengths to that dataset

Realistic random lens blur strengths applied

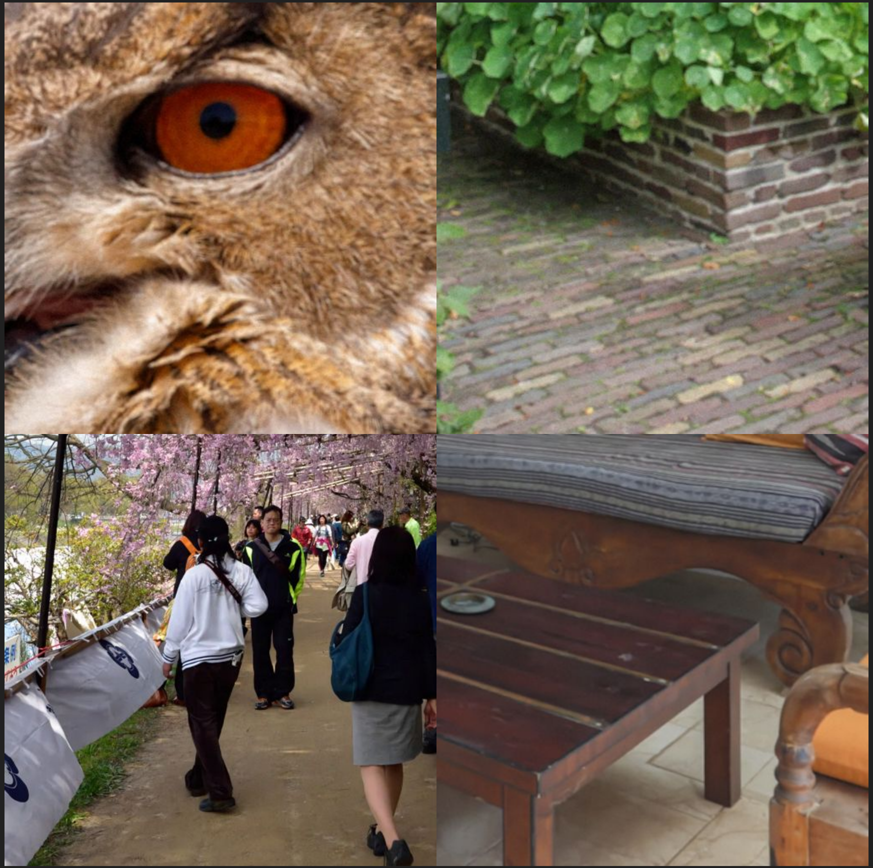
Example images nomos8k_sfw


1 / 1
... ts/datasets/RealSISR_v2/ludvae200
1: phips@phips-MS-7C02: ~/Documents/datasets/RealSISR_v2/ludvae200
Open
ludvae200_inference.py
~/Documents/datasets/RealSISR_v2/ludvae200

```
(base) phips@phips-MS-7C02:~/Documents/datasets/RealSISR_v2/ludvae200$ py
dvae200_inference.py
0001 - noise: 8.110433577425216, temperature: 0.0453532838571727
0002 - noise: 5.242534348788979, temperature: 0.06313254501860299
0003 - noise: 4.485683419664755, temperature: 0.029743623846285885
0004 - noise: 3.5121359314009357, temperature: 0.07964461336727424
0005 - noise: 3.4054283598556143, temperature: 0.013917716859923058
0006 - noise: 8.178743952337207, temperature: 0.03887337754918066
0007 - noise: 4.780782845870358, temperature: 0.007267554720085912
0008 - noise: 4.314278615176603, temperature: 0.060950918613754956
0009 - noise: 6.9673430210726215, temperature: 0.06430152547528575
0010 - noise: 8.121725932762281, temperature: 0.09555013378273888
0011 - noise: 2.5198203874258196, temperature: 0.05902278291982885
0012 - noise: 1.4918474134407833, temperature: 0.012280633386160845
0013 - noise: 6.012990344102921, temperature: 0.07811950269323635
0014 - noise: 3.1688345034289966, temperature: 0.04007922313261565
0015 - noise: 3.9798155812564504, temperature: 0.00019470975853962404
0016 - noise: 5.464497987954985, temperature: 0.05767984769082346
0017 - noise: 1.9850817354277306, temperature: 0.09306347764634298
0018 - noise: 9.144466477857023, temperature: 0.06039212898722236
0019 - noise: 7.215344847956992, temperature: 0.08329094806775206
0020 - noise: 1.5265831001420915, temperature: 0.03195038021368224
0021 - noise: 7.6537427549593176, temperature: 0.05596887304876696
0022 - noise: 3.4430223699386286, temperature: 0.03478394951262279
0023 - noise: 4.664242111959391, temperature: 0.003953726548041415
0024 - noise: 2.2756737941468175, temperature: 0.0002692360819703477
0025 - noise: 6.705104807255619, temperature: 0.08148563000234021
0026 - noise: 1.3521305962241637, temperature: 0.04035584085989437
0027 - noise: 0.8857624371341832, temperature: 0.04771628567784623
0028 - noise: 2.5048029759469204, temperature: 0.03424615112040187
0029 - noise: 2.7926310175354265, temperature: 0.0817380150950508
0030 - noise: 7.630666452035127, temperature: 0.08960471604426883
0031 - noise: 0.19744112922010504, temperature: 0.08692316557213259
0032 - noise: 8.761271413885606, temperature: 0.0034463409694602157
0033 - noise: 7.857941278964794, temperature: 0.0492429793524396
0034 - noise: 6.926239619425778, temperature: 0.04905284323184958
0035 - noise: 6.689830276350404, temperature: 0.08165175838910603
0036 - noise: 0.8823727549754423, temperature: 0.05592537210379659
0037 - noise: 6.323444109609545, temperature: 0.008310071964400778
0038 - noise: 2.9036224211576513, temperature: 0.06303617595150299
0039 - noise: 3.042840320591451, temperature: 0.09779165063576001
0040 - noise: 2.97541740278516, temperature: 0.03700305435616285
0041 - noise: 6.596966876215424, temperature: 0.08561829543233004
0042 - noise: 6.9043442243882875, temperature: 0.06196013498320914
0043 - noise: 9.921821941721383, temperature: 0.06778350134175108
0044 - noise: 9.992100206241826, temperature: 0.05473364907003565
0045 - noise: 2.6425362827341017, temperature: 0.06064072760384429
```

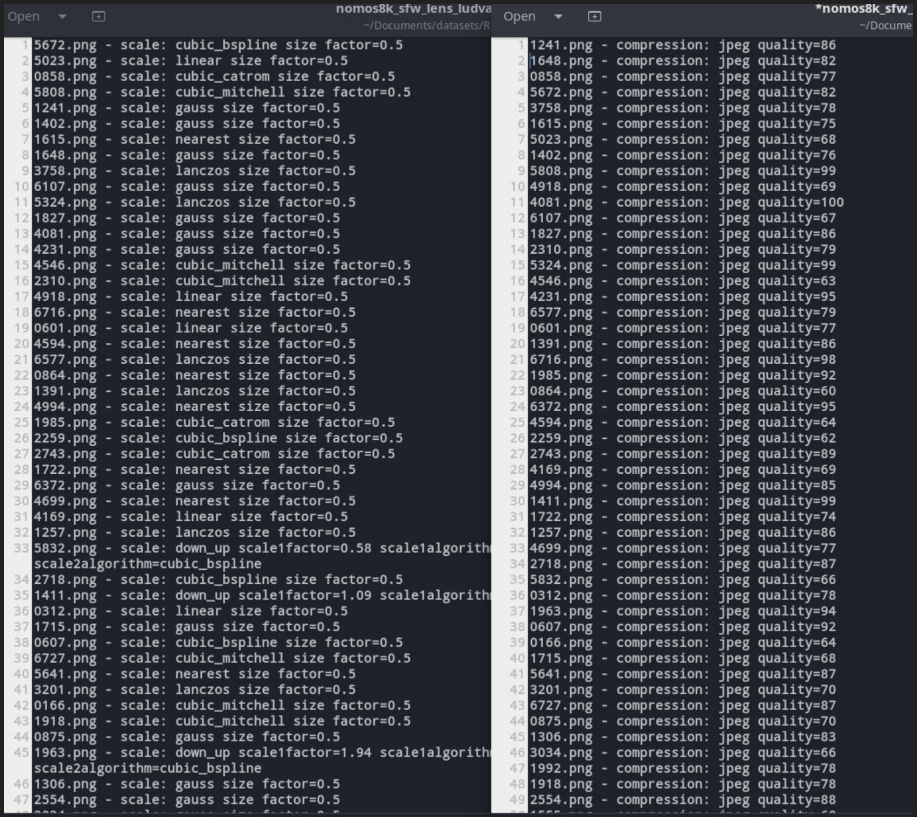
```python

    H_paths = util.get_image_paths(H_path)

    # I set these strength settings for noise and temperature based on tests with the trained specifically

    for idx, img in enumerate(H_paths):

        # ------------------------------------
        # (1) img_H
        # ------------------------------------

        img_name, ext = os.path.splitext(os.path.basename(img))
        img_H = util.imread_uint(img, n_channels=3)
        img_hH = img_H.copy()

        img_H = util.uint2tensor4(img_H).to(device)
        img_hH = util.uint2tensor4(img_hH).to(device)
        noise_strength = uniform(0,10)
        img_hH = img_hH + torch.randn_like(img_hH) * noise_strength / 255.0

        # ------------------------------------
        # (2) img_G
        # ------------------------------------

        label_H = torch.zeros(1, 1, 1, 1).long().to(device)
        temperature_strength = uniform(0,0.10)
        img_G = model.translate(img_H, img_hH, label_H, temperature=temperature_stren
        img_G = util.tensor2uint(img_G)

        # ------------------------------------
        # save results
        # ------------------------------------
        util.imsave(img_G, os.path.join(G_path, img_name + ext))

        # ------------------------------------
        # log degradation strength
        # ------------------------------------

        # add degradation strength to degradation output text file
        print_text_to_textfile(os.path.join(textfile_path, textfile_name),img_name + str(noise_strength) + ', temperature: ' + str(temperature_strength))

        # print out in console aswell so I see whats happening
        print(img_name + ' - noise: ' + str(noise_strength) + ', temperature: ' + str

if __name__ == '__main__':

```

My ludvae200 degradation model applied, extended with logging strengths



Ludvae200 applied

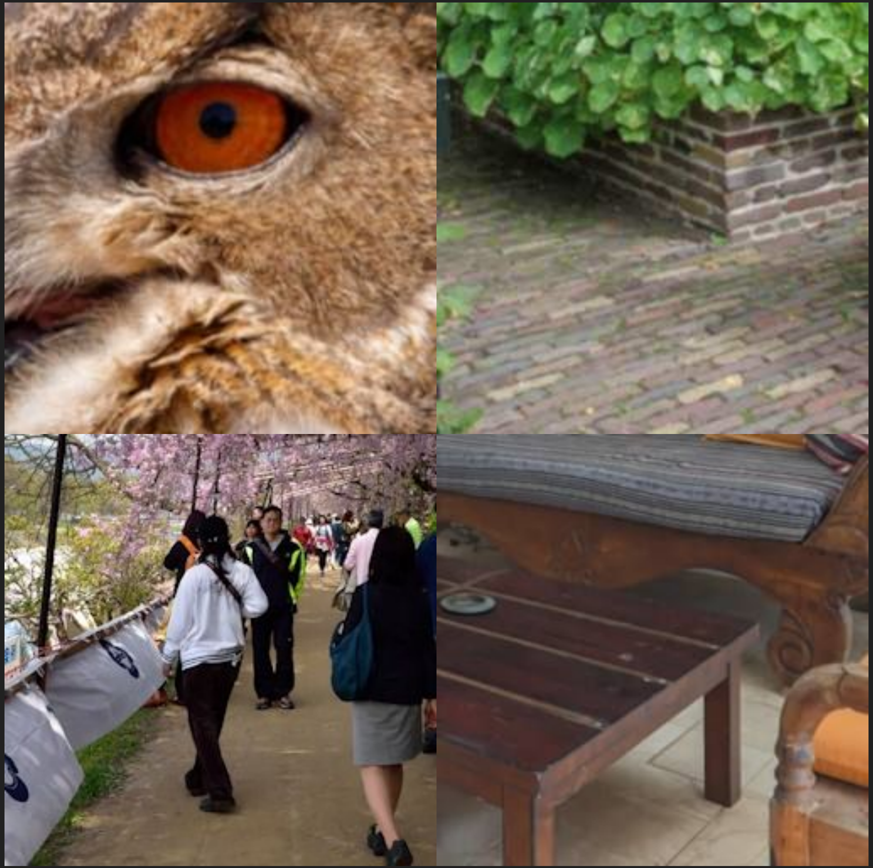
Realistic random lens blur strengths applied



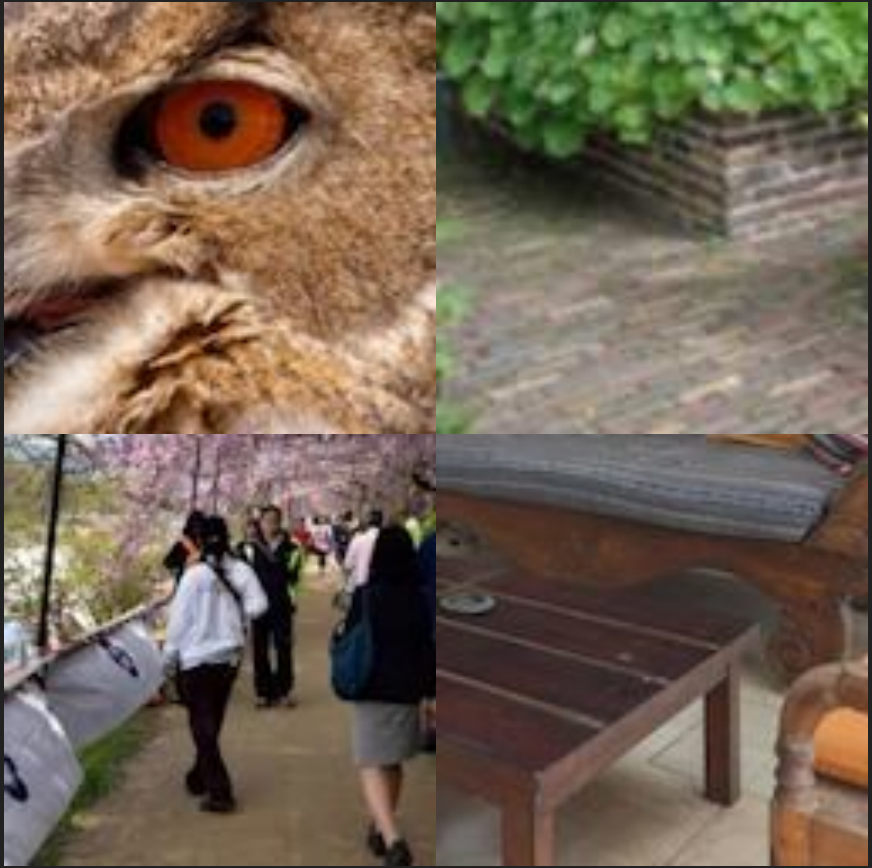
```
5672.png - scale: cubic_bspline size factor=0.5
5023.png - scale: linear size factor=0.5
0858.png - scale: cubic_catrom size factor=0.5
5808.png - scale: cubic_mitchell size factor=0.5
1241.png - scale: gauss size factor=0.5
1402.png - scale: gauss size factor=0.5
1615.png - scale: nearest size factor=0.5
1648.png - scale: gauss size factor=0.5
3758.png - scale: lanczos size factor=0.5
6107.png - scale: gauss size factor=0.5
5324.png - scale: lanczos size factor=0.5
1827.png - scale: gauss size factor=0.5
4081.png - scale: gauss size factor=0.5
4231.png - scale: gauss size factor=0.5
4546.png - scale: cubic_mitchell size factor=0.5
2310.png - scale: cubic_mitchell size factor=0.5
4918.png - scale: linear size factor=0.5
6716.png - scale: nearest size factor=0.5
0601.png - scale: linear size factor=0.5
4594.png - scale: nearest size factor=0.5
6577.png - scale: lanczos size factor=0.5
0864.png - scale: nearest size factor=0.5
1391.png - scale: lanczos size factor=0.5
4994.png - scale: nearest size factor=0.5
1985.png - scale: cubic_catrom size factor=0.5
2259.png - scale: cubic_bspline size factor=0.5
2743.png - scale: cubic_catrom size factor=0.5
1722.png - scale: nearest size factor=0.5
6372.png - scale: gauss size factor=0.5
4699.png - scale: nearest size factor=0.5
4169.png - scale: linear size factor=0.5
1257.png - scale: lanczos size factor=0.5
5832.png - scale: down_up scale1factor=0.58 scale1algorith
scale2algorithm=cubic_bspline
2718.png - scale: cubic_bspline size factor=0.5
1411.png - scale: down_up scale1factor=1.09 scale1algorith
0312.png - scale: linear size factor=0.5
1715.png - scale: gauss size factor=0.5
0607.png - scale: cubic_bspline size factor=0.5
6727.png - scale: cubic_mitchell size factor=0.5
5641.png - scale: nearest size factor=0.5
3201.png - scale: lanczos size factor=0.5
0166.png - scale: cubic_mitchell size factor=0.5
1918.png - scale: cubic_mitchell size factor=0.5
0875.png - scale: gauss size factor=0.5
1963.png - scale: down_up scale1factor=1.94 scale1algorith
scale2algorithm=cubic_bspline
1306.png - scale: gauss size factor=0.5
2554.png - scale: gauss size factor=0.5
```

```
1241.png - compression: jpeg quality=86
1648.png - compression: jpeg quality=82
0858.png - compression: jpeg quality=77
5672.png - compression: jpeg quality=82
3758.png - compression: jpeg quality=78
1615.png - compression: jpeg quality=75
5023.png - compression: jpeg quality=68
1402.png - compression: jpeg quality=76
5808.png - compression: jpeg quality=99
4918.png - compression: jpeg quality=69
4081.png - compression: jpeg quality=100
6107.png - compression: jpeg quality=67
1827.png - compression: jpeg quality=86
2310.png - compression: jpeg quality=79
5324.png - compression: jpeg quality=99
4546.png - compression: jpeg quality=63
4231.png - compression: jpeg quality=95
6577.png - compression: jpeg quality=79
0601.png - compression: jpeg quality=77
1391.png - compression: jpeg quality=86
6716.png - compression: jpeg quality=98
1985.png - compression: jpeg quality=92
0864.png - compression: jpeg quality=60
6372.png - compression: jpeg quality=95
4594.png - compression: jpeg quality=64
2259.png - compression: jpeg quality=62
2743.png - compression: jpeg quality=89
4169.png - compression: jpeg quality=69
4994.png - compression: jpeg quality=85
1411.png - compression: jpeg quality=99
1722.png - compression: jpeg quality=74
1257.png - compression: jpeg quality=86
4699.png - compression: jpeg quality=77
2718.png - compression: jpeg quality=87
5832.png - compression: jpeg quality=66
0312.png - compression: jpeg quality=78
1963.png - compression: jpeg quality=94
0607.png - compression: jpeg quality=92
0166.png - compression: jpeg quality=64
1715.png - compression: jpeg quality=68
5641.png - compression: jpeg quality=87
3201.png - compression: jpeg quality=70
6727.png - compression: jpeg quality=87
0875.png - compression: jpeg quality=70
1306.png - compression: jpeg quality=83
3034.png - compression: jpeg quality=66
1992.png - compression: jpeg quality=78
1918.png - compression: jpeg quality=78
2554.png - compression: jpeg quality=88
```

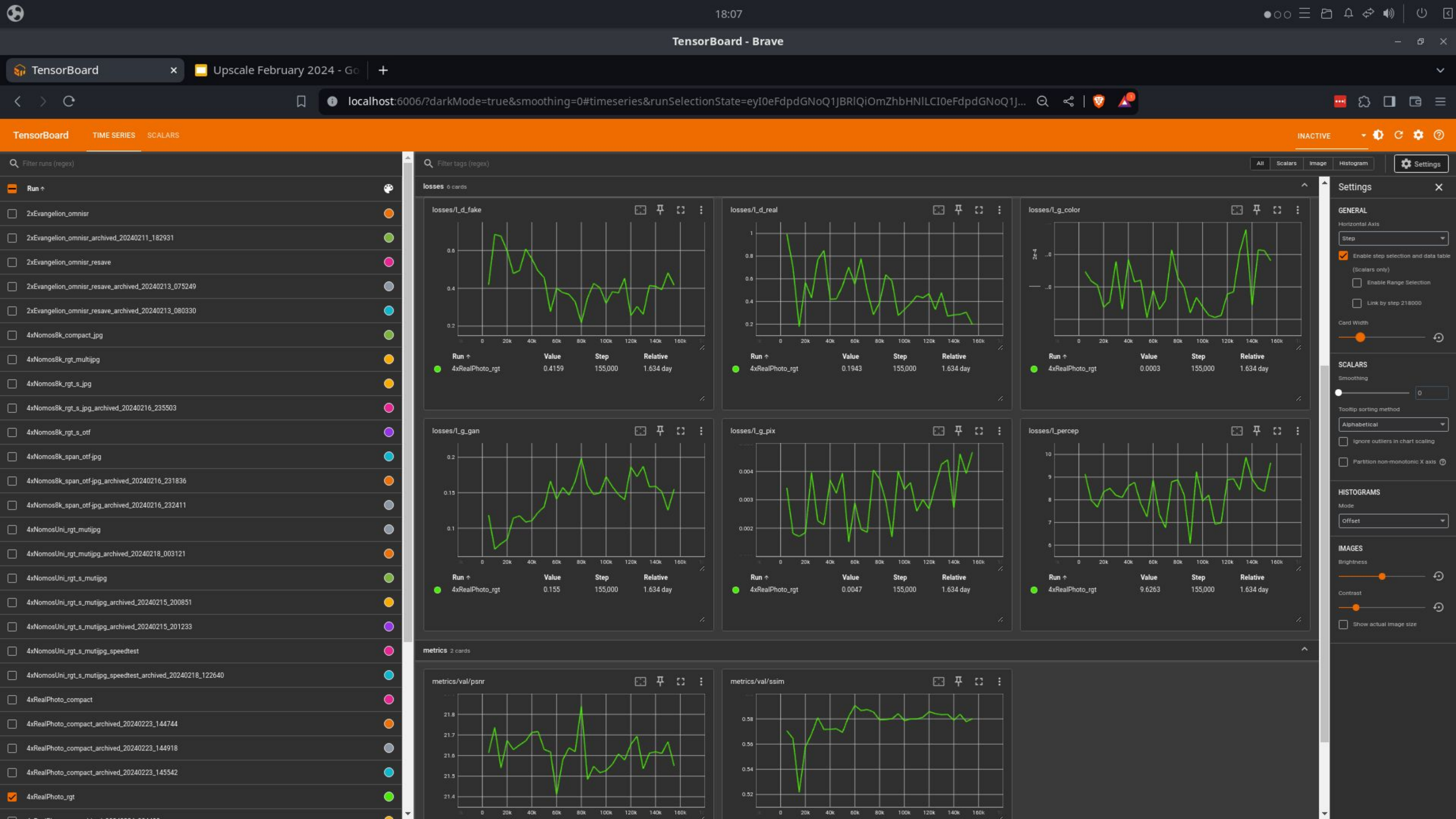
Scale and jpg compression applied



Scale and jpg compression applied



Scale and jpg compression re-applied, final lr’s


18:07
TensorBoard - Brave
TensorBoard
Upscale February 2024 - Go
localhost:6006/?darkMode=true&smoothing=0#timeseries&runSelectionState=eyI0eFdpdGNoQ1JBRlQiOmZhbHNlLCI0eFdpdGNoQ1J...
TensorBoard
TIME SERIES
SCALARS
INACTIVE
Filter runs (regex)
Filter tags (regex)
All
Scalars
Image
Histogram
Settings

Run ↑

- ☐ 2xEvangelion_omnisr
- ☐ 2xEvangelion_omnisr_archived_20240211_182931
- ☐ 2xEvangelion_omnisr_resave
- ☐ 2xEvangelion_omnisr_resave_archived_20240213_075249
- ☐ 2xEvangelion_omnisr_resave_archived_20240213_080330
- ☐ 4xNomos8k_compact_jpg
- ☐ 4xNomos8k_rgt_multijpg
- ☐ 4xNomos8k_rgt_s_jpg
- ☐ 4xNomos8k_rgt_s_jpg_archived_20240216_235503
- ☐ 4xNomos8k_rgt_s_otf
- ☐ 4xNomos8k_span_otf-jpg
- ☐ 4xNomos8k_span_otf-jpg_archived_20240216_231836
- ☐ 4xNomos8k_span_otf-jpg_archived_20240216_232411
- ☐ 4xNomosUni_rgt_mutijpg
- ☐ 4xNomosUni_rgt_mutijpg_archived_20240218_003121
- ☐ 4xNomosUni_rgt_s_mutijpg
- ☐ 4xNomosUni_rgt_s_mutijpg_archived_20240215_200851
- ☐ 4xNomosUni_rgt_s_mutijpg_archived_20240215_201233
- ☐ 4xNomosUni_rgt_s_mutijpg_speedtest
- ☐ 4xNomosUni_rgt_s_mutijpg_speedtest_archived_20240218_122640
- ☐ 4xRealPhoto_compact
- ☐ 4xRealPhoto_compact_archived_20240223_144744
- ☐ 4xRealPhoto_compact_archived_20240223_144918
- ☐ 4xRealPhoto_compact_archived_20240223_145542
- ☑ 4xRealPhoto_rgt

losses 6 cards

losses/l_d_fake

Run ↑	Value	Step	Relative
● 4xRealPhoto_rgt	0.4159	155,000	1.634 day

losses/l_d_real

Run ↑	Value	Step	Relative
● 4xRealPhoto_rgt	0.1943	155,000	1.634 day

losses/l_g_color

Run ↑	Value	Step	Relative
● 4xRealPhoto_rgt	0.0003	155,000	1.634 day

losses/l_g_gan

Run ↑	Value	Step	Relative
● 4xRealPhoto_rgt	0.155	155,000	1.634 day

losses/l_g_pix

Run ↑	Value	Step	Relative
● 4xRealPhoto_rgt	0.0047	155,000	1.634 day

losses/l_percep

Run ↑	Value	Step	Relative
● 4xRealPhoto_rgt	9.6263	155,000	1.634 day

metrics 2 cards

metrics/val/psnr

metrics/val/ssim

Settings

GENERAL

Horizontal Axis: Step

☑ Enable step selection and data table (Scalars only)

☐ Enable Range Selection

☐ Link by step 218000

Card Width

SCALARS

Smoothing: 0

Tooltip sorting method: Alphabetical

☐ Ignore outliers in chart scaling

☐ Partition non-monotonic X axis

HISTOGRAMS

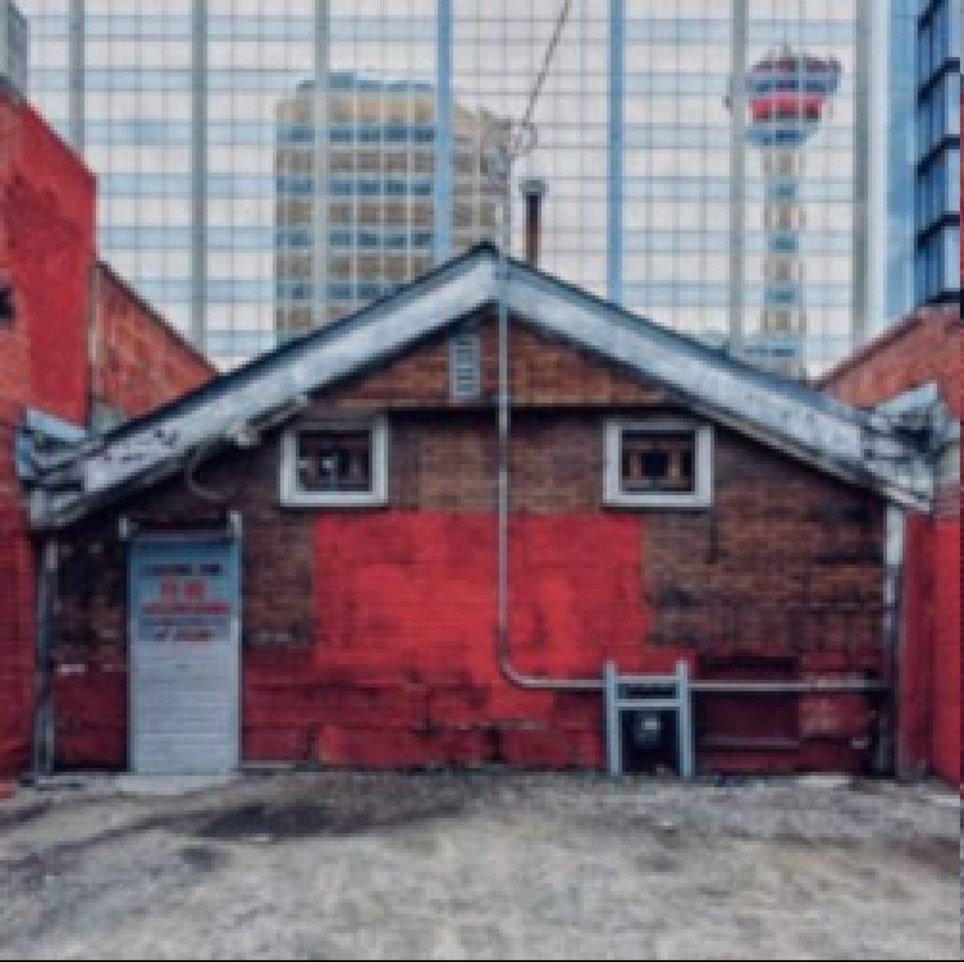
Mode: Offset

IMAGES

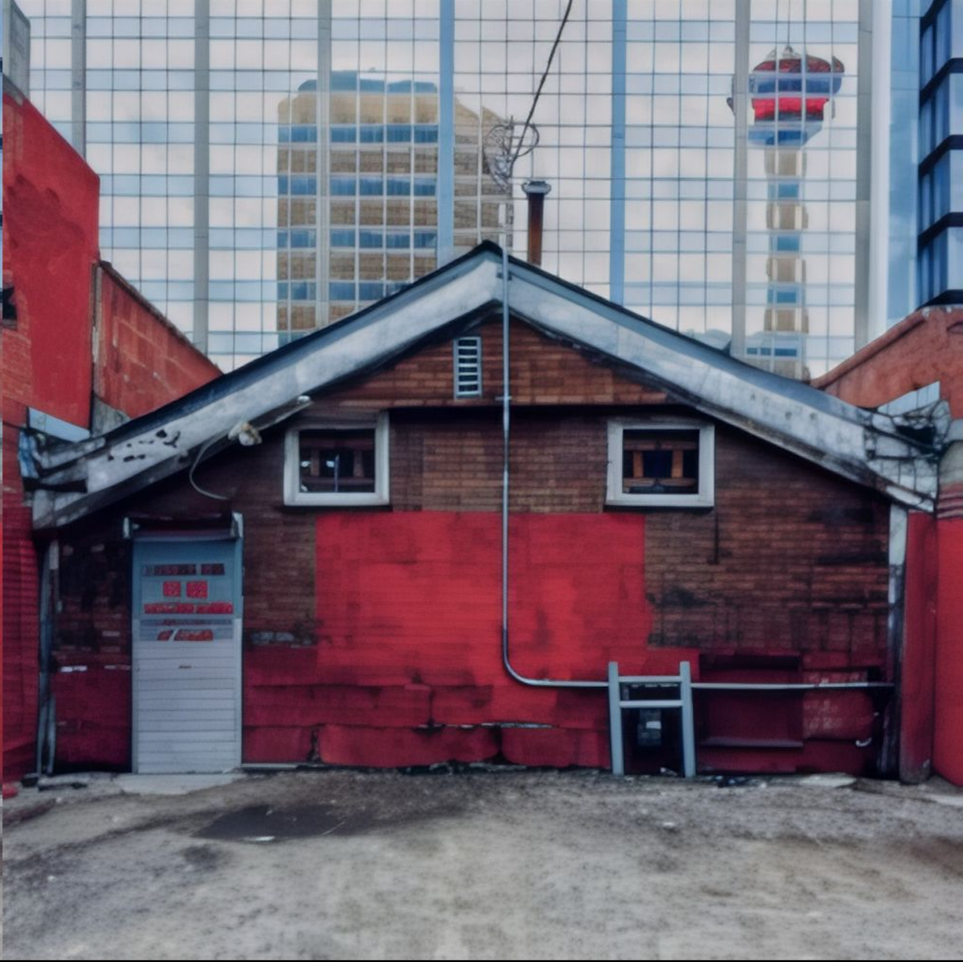
Brightness

Contrast

☐ Show actual image size

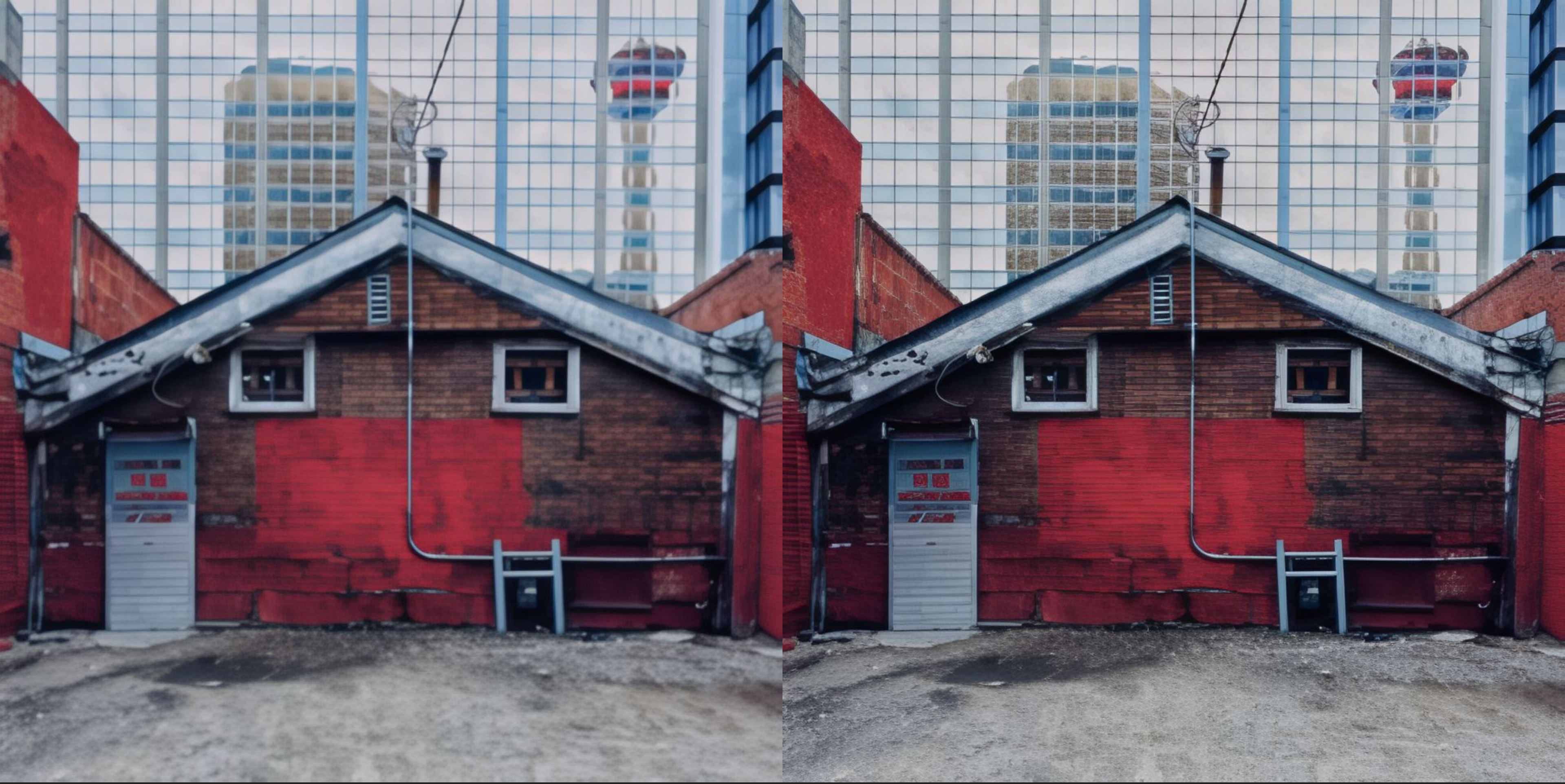


4xNearestNeighbor

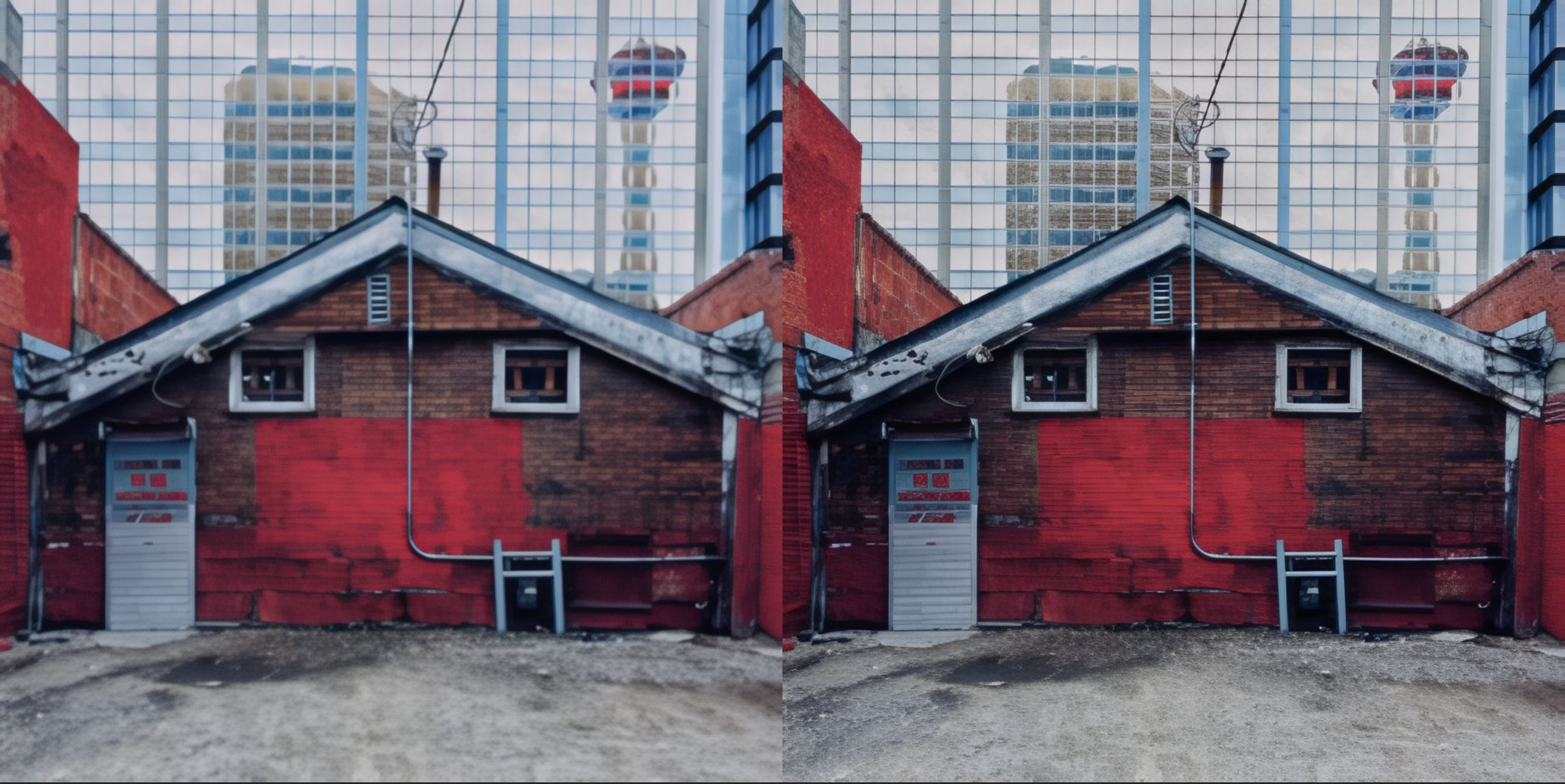


4xRealWebPhoto_RGT_60k

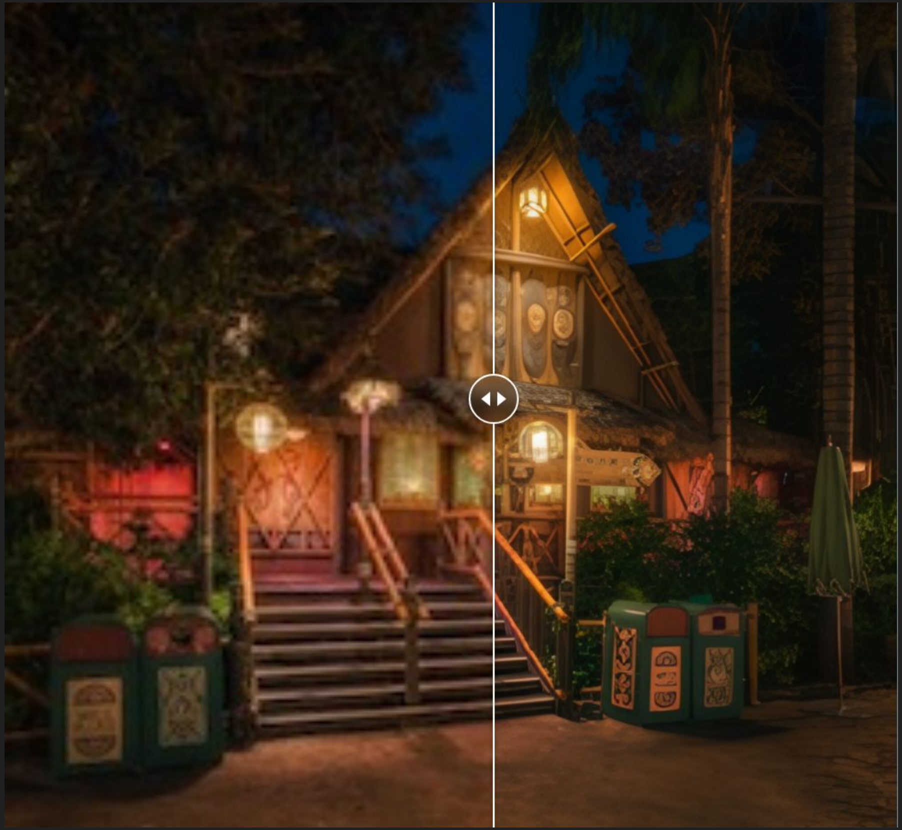
WIP - Training and testing models to see if its a working approach



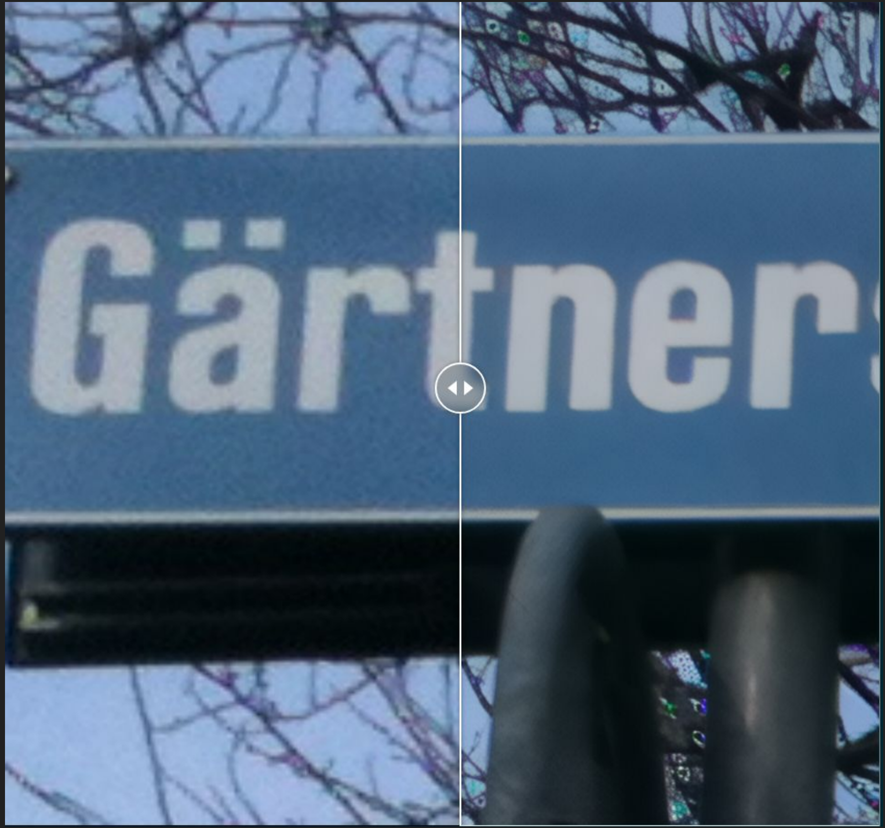
Improvement when increasing gt size from 128 (at 160k iters) to 256 (this is 200k)



Improvement again gt size from 256 to 384 (but 4 hours training for 1 checkpoint of 5k iters)



Validation results, 260k, ~109 hours of training
https://slow.pics/s/1TRW2uBK



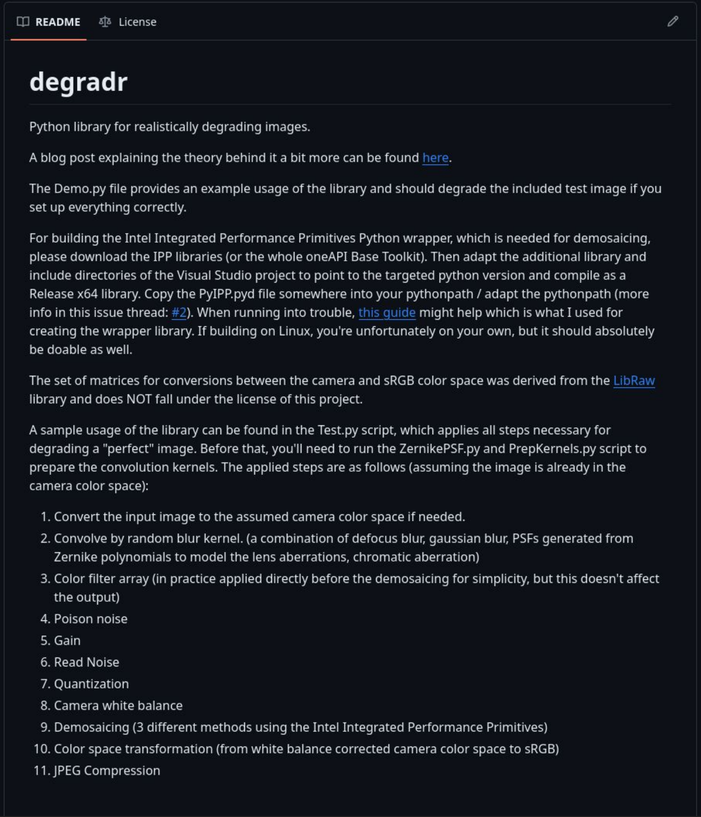
https://slow.pics/s/uCrGDwSe

Thoughts

Tone down lens blur

Play around with degradations

degradr



degradr

Python library for realistically degrading images.

A blog post explaining the theory behind it a bit more can be found here.

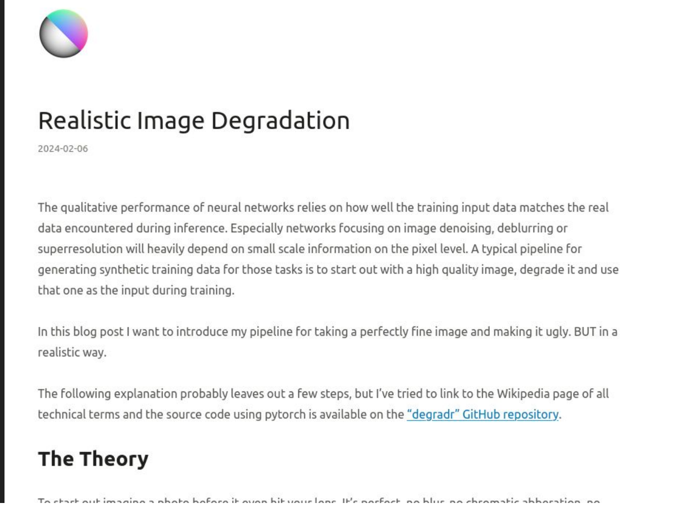
The Demo.py file provides an example usage of the library and should degrade the included test image if you set up everything correctly.

For building the Intel Integrated Performance Primitives Python wrapper, which is needed for demosaicing, please download the IPP libraries (or the whole oneAPI Base Toolkit). Then adapt the additional library and include directories of the Visual Studio project to point to the targeted python version and compile as a Release x64 library. Copy the PyIPP.pyd file somewhere into your pythonpath / adapt the pythonpath (more info in this issue thread: #2). When running into trouble, this guide might help which is what I used for creating the wrapper library. If building on Linux, you're unfortunately on your own, but it should absolutely be doable as well.

The set of matrices for conversions between the camera and sRGB color space was derived from the LibRaw library and does NOT fall under the license of this project.

A sample usage of the library can be found in the Test.py script, which applies all steps necessary for degrading a "perfect" image. Before that, you'll need to run the ZernikePSF.py and PrepKernels.py script to prepare the convolution kernels. The applied steps are as follows (assuming the image is already in the camera color space):

1. Convert the input image to the assumed camera color space if needed.
2. Convolve by random blur kernel. (a combination of defocus blur, gaussian blur, PSFs generated from Zernike polynomials to model the lens aberrations, chromatic aberration)
3. Color filter array (in practice applied directly before the demosaicing for simplicity, but this doesn't affect the output)
4. Poison noise
5. Gain
6. Read Noise
7. Quantization
8. Camera white balance
9. Demosaicing (3 different methods using the Intel Integrated Performance Primitives)
10. Color space transformation (from white balance corrected camera color space to sRGB)
11. JPEG Compression



Realistic Image Degradation

2024-02-06

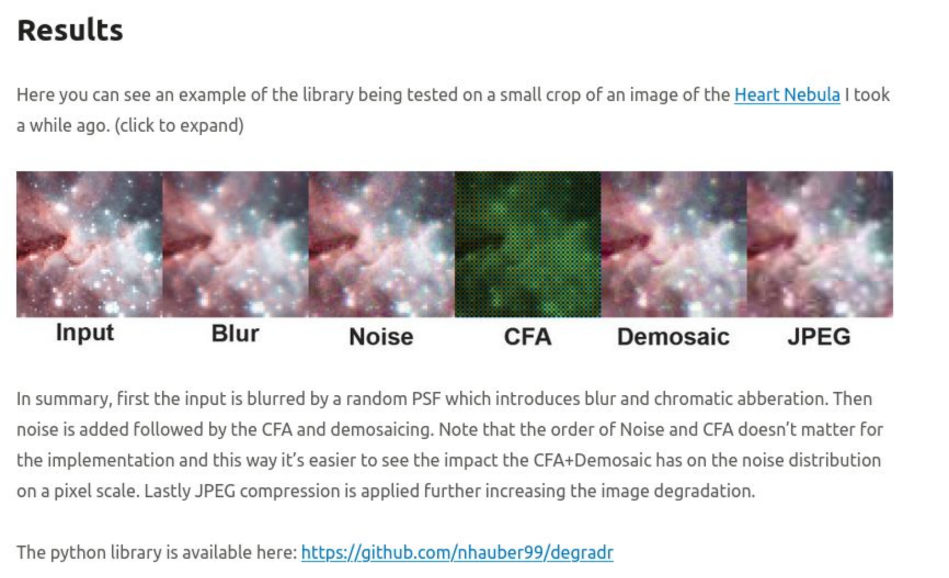
The qualitative performance of neural networks relies on how well the training input data matches the real data encountered during inference. Especially networks focusing on image denoising, deblurring or superresolution will heavily depend on small scale information on the pixel level. A typical pipeline for generating synthetic training data for those tasks is to start out with a high quality image, degrade it and use that one as the input during training.

In this blog post I want to introduce my pipeline for taking a perfectly fine image and making it ugly. BUT in a realistic way.

The following explanation probably leaves out a few steps, but I've tried to link to the Wikipedia page of all technical terms and the source code using pytorch is available on the "degradr" GitHub repository.

The Theory

To start out imagine a photo before it even hit your lens. It's perfect, no blur, no chromatic abberation, no

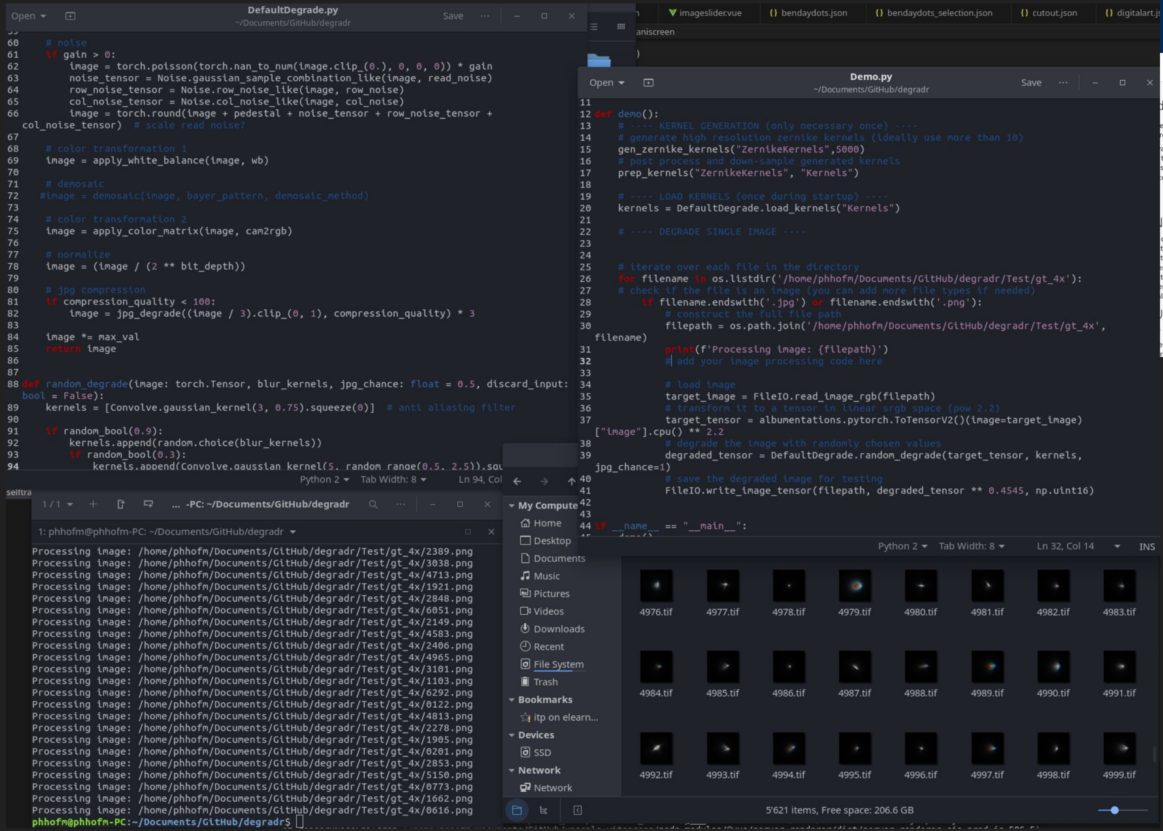


Results

Here you can see an example of the library being tested on a small crop of an image of the Heart Nebula I took a while ago. (click to expand)

In summary, first the input is blurred by a random PSF which introduces blur and chromatic abberation. Then noise is added followed by the CFA and demosaicing. Note that the order of Noise and CFA doesn't matter for the implementation and this way it's easier to see the impact the CFA+Demosaic has on the noise distribution on a pixel scale. Lastly JPEG compression is applied further increasing the image degradation.

The python library is available here: https://github.com/nhauber99/degradr



DefaultDegrade.py
~/Documents/GitHub/degradr

```python
    # noise
    if gain > 0:
        image = torch.poisson(torch.nan_to_num(image.clip_(0.), 0, 0, 0)) * gain
        noise_tensor = Noise.gaussian_sample_combination_like(image, read_noise)
        row_noise_tensor = Noise.row_noise_like(image, row_noise)
        col_noise_tensor = Noise.col_noise_like(image, col_noise)
        image = torch.round(image + pedestal + noise_tensor + row_noise_tensor + col_noise_tensor)  # scale read noise?

    # color transformation 1
    image = apply_white_balance(image, wb)

    # demosaic
   #image = demosaic(image, bayer_pattern, demosaic_method)

    # color transformation 2
    image = apply_color_matrix(image, cam2rgb)

    # normalize
    image = (image / (2 ** bit_depth))

    # jpg compression
    if compression_quality < 100:
        image = jpg_degrade((image / 3).clip_(0, 1), compression_quality) * 3

    image *= max_val
    return image


def random_degrade(image: torch.Tensor, blur_kernels, jpg_chance: float = 0.5, discard_input: bool = False):
    kernels = [Convolve.gaussian_kernel(3, 0.75).squeeze(0)]  # anti aliasing filter

    if random_bool(0.9):
        kernels.append(random.choice(blur_kernels))
        if random_bool(0.3):
            kernels.append(Convolve.gaussian_kernel(5, random_range(0.5, 2.5)).squu
```

Python 2 ▾ Tab Width: 8 ▾ Ln 94, Col

Demo.py
~/Documents/GitHub/degradr

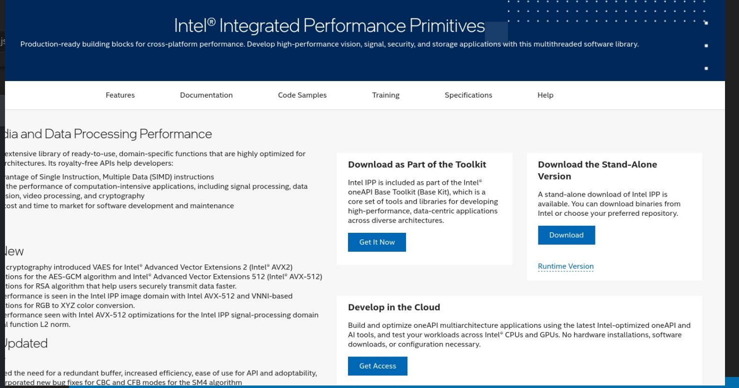
```python

def demo():
    # ---- KERNEL GENERATION (only necessary once) ----
    # generate high resolution zernike kernels (ideally use more than 10)
    gen_zernike_kernels("ZernikeKernels",5000)
    # post process and down-sample generated kernels
    prep_kernels("ZernikeKernels", "Kernels")

    # ---- LOAD KERNELS (once during startup) ----
    kernels = DefaultDegrade.load_kernels("Kernels")

    # ---- DEGRADE SINGLE IMAGE ----


    # iterate over each file in the directory
    for filename in os.listdir('/home/phhofm/Documents/GitHub/degradr/Test/gt_4x'):
    # check if the file is an image (you can add more file types if needed)
        if filename.endswith('.jpg') or filename.endswith('.png'):
            # construct the full file path
            filepath = os.path.join('/home/phhofm/Documents/GitHub/degradr/Test/gt_4x', filename)
            print(f'Processing image: {filepath}')
            # add your image processing code here

            # load image
            target_image = FileIO.read_image_rgb(filepath)
            # transform it to a tensor in linear srgb space (pow 2.2)
            target_tensor = albumentations.pytorch.ToTensorV2()(image=target_image)["image"].cpu() ** 2.2
            # degrade the image with randomly chosen values
            degraded_tensor = DefaultDegrade.random_degrade(target_tensor, kernels, jpg_chance=1)
            # save the degraded image for testing
            FileIO.write_image_tensor(filepath, degraded_tensor ** 0.4545, np.uint16)


if __name__ == "__main__":
```

Python 2 ▾ Tab Width: 8 ▾ Ln 32, Col 14 ▾ INS

... -PC: ~/Documents/GitHub/degradr

1: phhofm@phhofm-PC: ~/Documents/GitHub/degradr ▾

```
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/2389.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/3038.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/4713.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/1921.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/2848.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/6051.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/2149.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/4583.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/2406.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/4965.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/3101.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/1103.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/6292.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/0122.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/4813.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/2278.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/1905.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/0201.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/2853.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/5150.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/0773.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/1662.png
Processing image: /home/phhofm/Documents/GitHub/degradr/Test/gt_4x/0616.png
phhofm@phhofm-PC:~/Documents/GitHub/degradr$
```



Intel® Integrated Performance Primitives

Production-ready building blocks for cross-platform performance. Develop high-performance vision, signal, security, and storage applications with this multithreaded software library.

dia and Data Processing Performance

extensive library of ready-to-use, domain-specific functions that are highly optimized for
rchitectures. Its royalty-free APIs help developers:

antage of Single Instruction, Multiple Data (SIMD) instructions
the performance of computation-intensive applications, including signal processing, data
sion, video processing, and cryptography
cost and time to market for software development and maintenance

lew

cryptography introduced VAES for Intel® Advanced Vector Extensions 2 (Intel® AVX2)
tions for the AES-GCM algorithm and Intel® Advanced Vector Extensions 512 (Intel® AVX-512)
tions for RSA algorithm that help users securely transmit data faster.
erformance is seen in the Intel IPP image domain with Intel AVX-512 and VNNI-based
tions for RGB to XYZ color conversion.
erformance seen with Intel AVX-512 optimizations for the Intel IPP signal-processing domain
l function L2 norm.

Jpdated

ed the need for a redundant buffer, increased efficiency, ease of use for API and adoptability,
rporated new bug fixes for CBC and CFB modes for the SM4 algorithm

Download as Part of the Toolkit

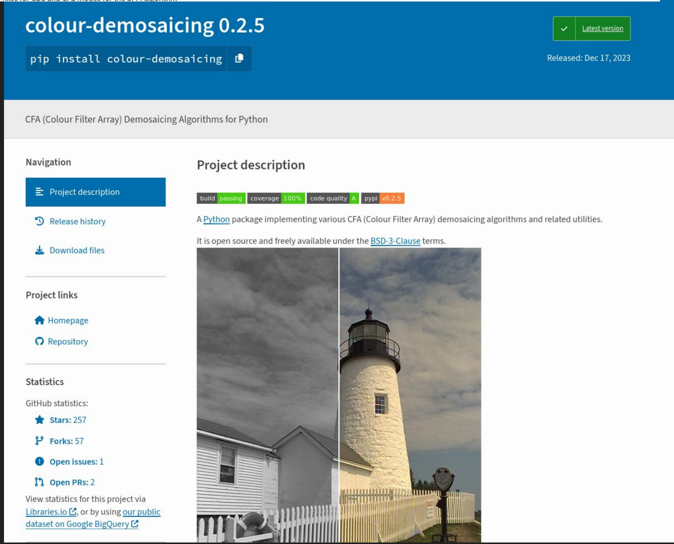
Intel IPP is included as part of the Intel® oneAPI Base Toolkit (Base Kit), which is a core set of tools and libraries for developing high-performance, data-centric applications across diverse architectures.

Download the Stand-Alone Version

A stand-alone download of Intel IPP is available. You can download binaries from Intel or choose your preferred repository.

Develop in the Cloud

Build and optimize oneAPI multiarchitecture applications using the latest Intel-optimized oneAPI and AI tools, and test your workloads across Intel® CPUs and GPUs. No hardware installations, software downloads, or configuration necessary.



colour-demosaicing 0.2.5

```
pip install colour-demosaicing
```

Released: Dec 17, 2023

CFA (Colour Filter Array) Demosaicing Algorithms for Python

Navigation

Project links

Statistics

GitHub statistics:

Stars: 257

Forks: 57

Open issues: 1

Open PRs: 2

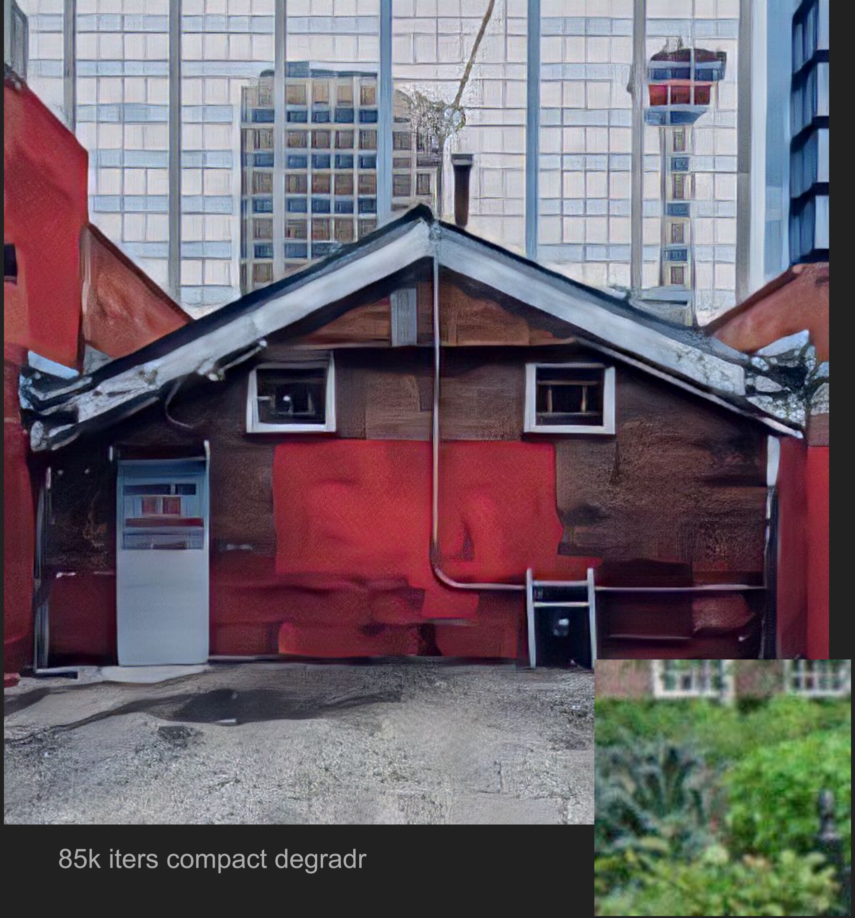
View statistics for this project via Libraries.io, or by using our public dataset on Google BigQuery

Project description

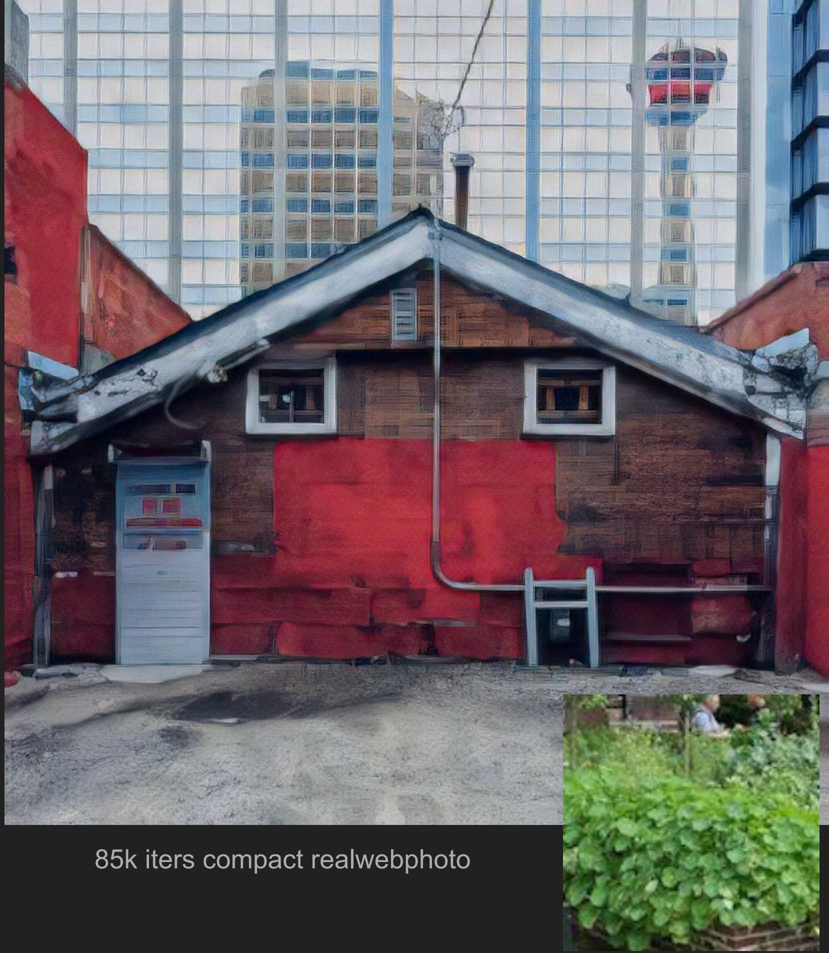
A Python package implementing various CFA (Colour Filter Array) demosaicing algorithms and related utilities.

It is open source and freely available under the BSD-3-Clause terms.

Degrade applied same dataset 5k ZernikeKernels but I removed demosaic step because of more complex dependency build setup



85k iters compact degradr



85k iters compact realwebphoto

Degrade-non-demosaic result Compact

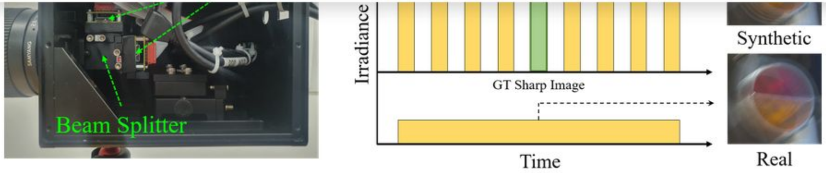
Appendix: RealWebPhoto dataset creation with blur photo dataset - first try (v0)

Realistic Blur Synthesis for Learning Image Deblurring

ECCV 2022

Realistic Blur Synthesis for Learning Image Deblurring

Jaesung Rim, Geonung Kim, Jungeon Kim, Junyong Lee,
Seungyong Lee, Sunghyun Cho



(a) Our dual-camera system

(b) Image acquisition process

Google Drive Link

Google drive link
(recommended)

Postech Link

CG lab server link

RSBlur.zip

13,358 pairs of real/synthetic blurred image and a corresponding GT image.

RSBlur_additional.zip

8,821 addtional images for learning based synthesis, additional synthetic images or etc.
Do not use it as additional real training images.

RSBlur_sharps

All of sharp image sequneces.

GoPro_INTER_ABME.zip

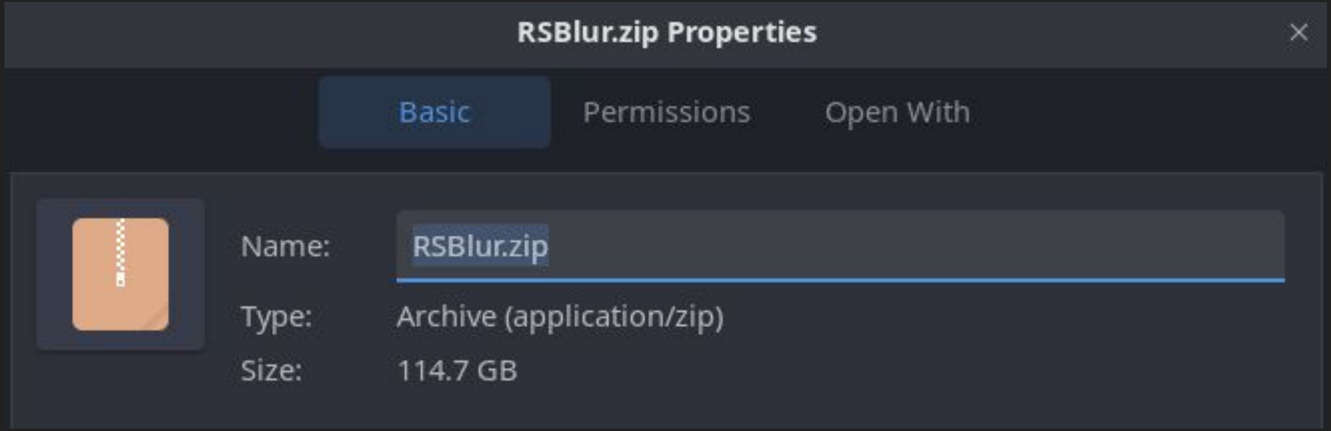
Synthetic blur dataset using the GoPro and the ABME method.

GoPro_U.zip

Synthetic blur dataset using the GoPro and synthetic blur kernels.

Code

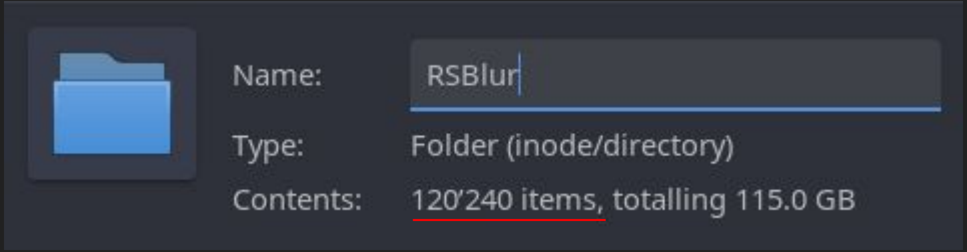
Training and Evaluation code



Name: RSBlur.zip

Type: Archive (application/zip)

Size: 114.7 GB



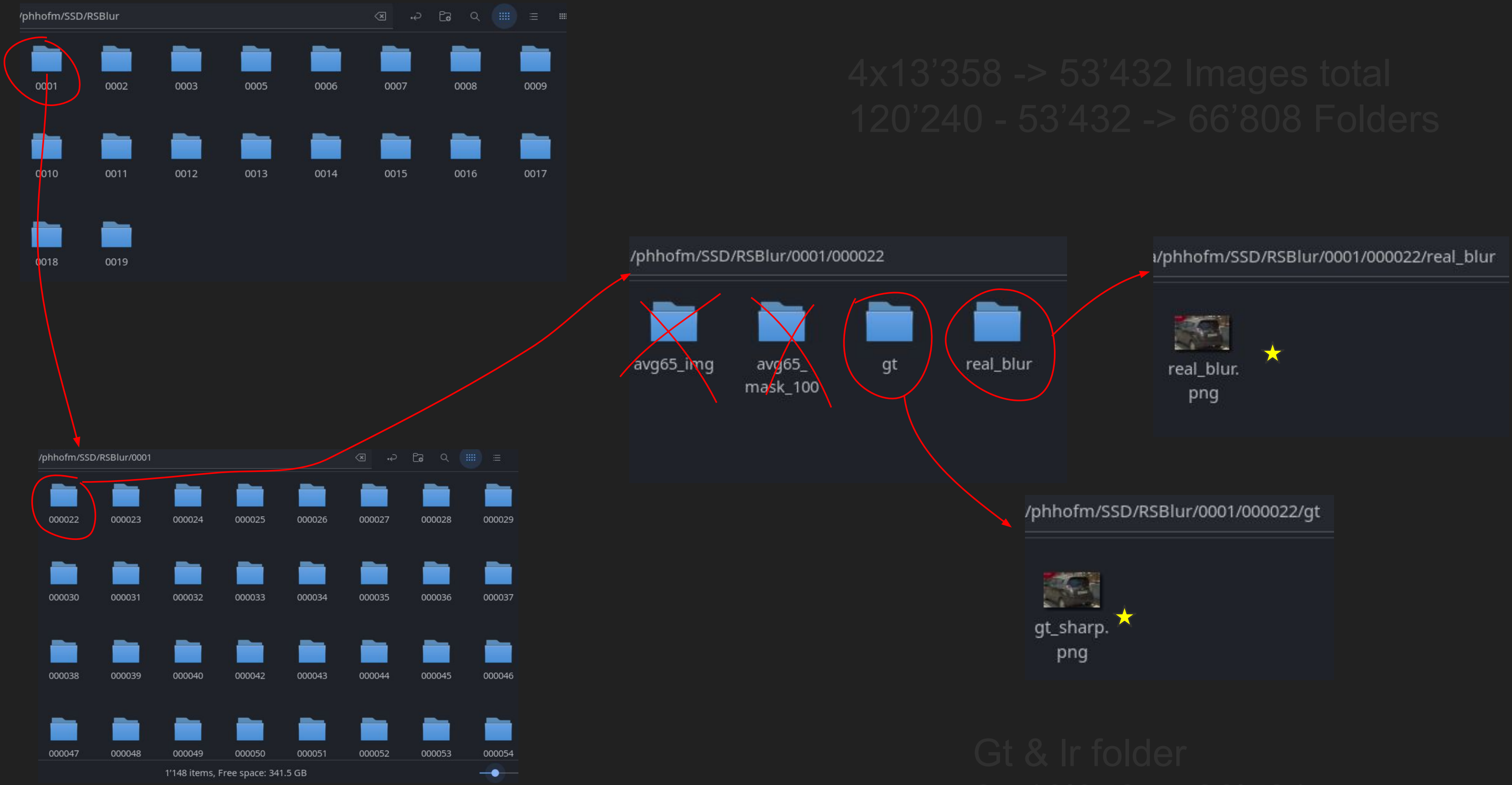
Name: RSBlur

Type: Folder (inode/directory)

Contents: 120’240 items, totalling 115.0 GB

Counts not only Files but also Folders

4x13’358 -> 53’432 Images total
120’240 - 53’432 -> 66’808 Folders



/phhofm/SSD/RSBlur

/phhofm/SSD/RSBlur/0001

1’148 items, Free space: 341.5 GB

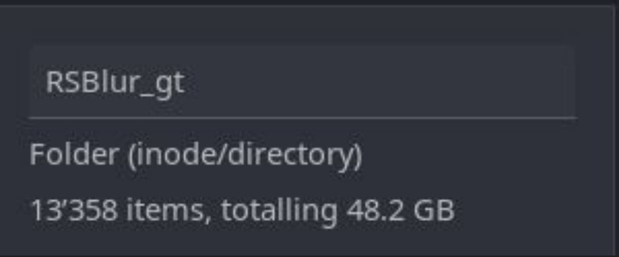
/phhofm/SSD/RSBlur/0001/000022

/phhofm/SSD/RSBlur/0001/000022/real_blur

/phhofm/SSD/RSBlur/0001/000022/gt

Gt & lr folder
2x 13’358 -> 26’721

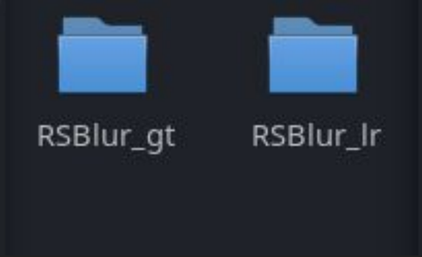
26’721 Images, 94 GB

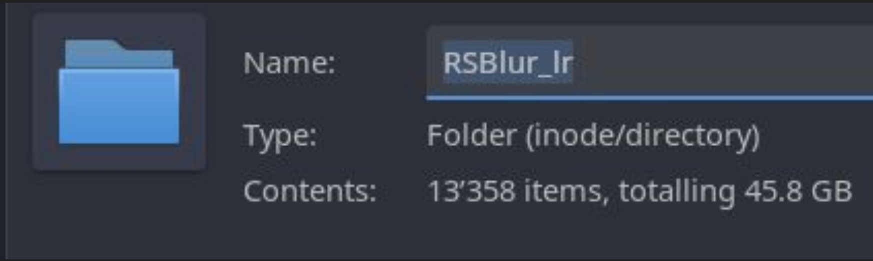


RSBlur_gt

Folder (inode/directory)

13’358 items, totalling 48.2 GB

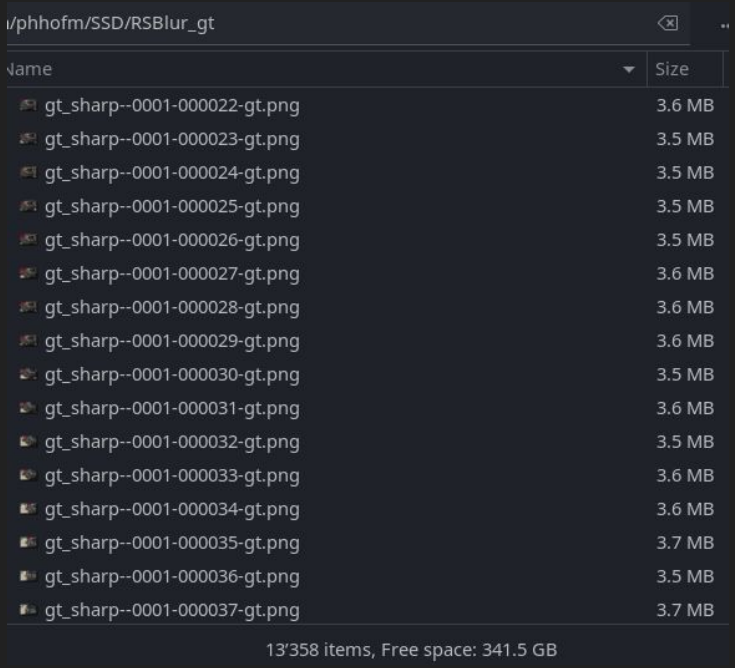




Name: RSBlur_lr

Type: Folder (inode/directory)

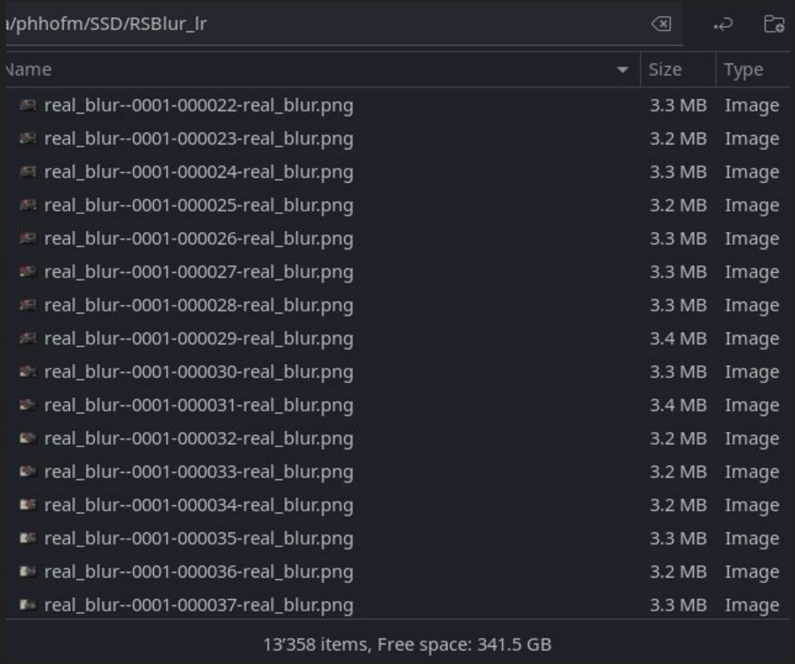
Contents: 13’358 items, totalling 45.8 GB



/phhofm/SSD/RSBlur_gt

Name	Size
gt_sharp--0001-000022-gt.png	3.6 MB
gt_sharp--0001-000023-gt.png	3.5 MB
gt_sharp--0001-000024-gt.png	3.5 MB
gt_sharp--0001-000025-gt.png	3.5 MB
gt_sharp--0001-000026-gt.png	3.5 MB
gt_sharp--0001-000027-gt.png	3.6 MB
gt_sharp--0001-000028-gt.png	3.6 MB
gt_sharp--0001-000029-gt.png	3.6 MB
gt_sharp--0001-000030-gt.png	3.5 MB
gt_sharp--0001-000031-gt.png	3.6 MB
gt_sharp--0001-000032-gt.png	3.5 MB
gt_sharp--0001-000033-gt.png	3.6 MB
gt_sharp--0001-000034-gt.png	3.6 MB
gt_sharp--0001-000035-gt.png	3.7 MB
gt_sharp--0001-000036-gt.png	3.5 MB
gt_sharp--0001-000037-gt.png	3.7 MB

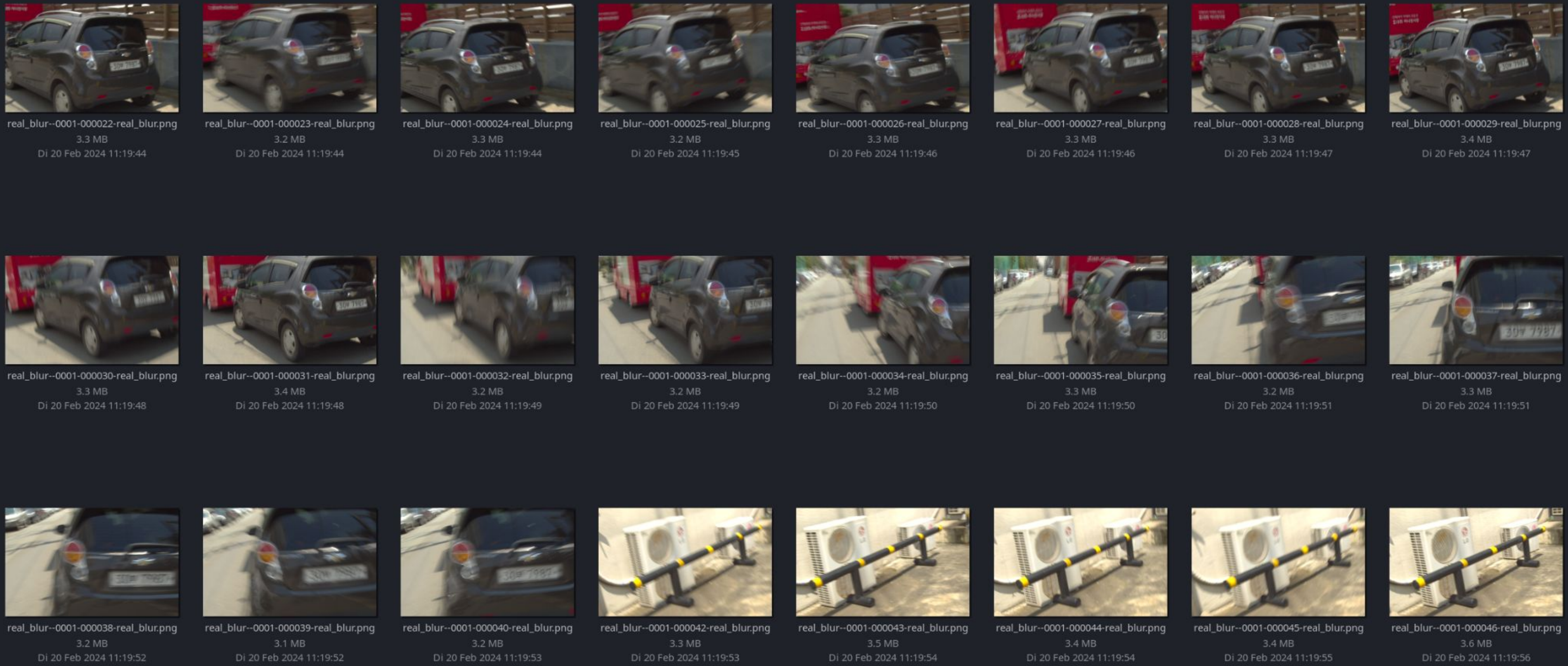
13’358 items, Free space: 341.5 GB



/phhofm/SSD/RSBlur_lr

Name	Size	Type
real_blur--0001-000022-real_blur.png	3.3 MB	Image
real_blur--0001-000023-real_blur.png	3.2 MB	Image
real_blur--0001-000024-real_blur.png	3.3 MB	Image
real_blur--0001-000025-real_blur.png	3.2 MB	Image
real_blur--0001-000026-real_blur.png	3.3 MB	Image
real_blur--0001-000027-real_blur.png	3.3 MB	Image
real_blur--0001-000028-real_blur.png	3.3 MB	Image
real_blur--0001-000029-real_blur.png	3.4 MB	Image
real_blur--0001-000030-real_blur.png	3.3 MB	Image
real_blur--0001-000031-real_blur.png	3.4 MB	Image
real_blur--0001-000032-real_blur.png	3.2 MB	Image
real_blur--0001-000033-real_blur.png	3.2 MB	Image
real_blur--0001-000034-real_blur.png	3.2 MB	Image
real_blur--0001-000035-real_blur.png	3.3 MB	Image
real_blur--0001-000036-real_blur.png	3.2 MB	Image
real_blur--0001-000037-real_blur.png	3.3 MB	Image

13’358 items, Free space: 341.5 GB



real_blur--0001-000022-real_blur.png
3.3 MB
Di 20 Feb 2024 11:19:44

real_blur--0001-000023-real_blur.png
3.2 MB
Di 20 Feb 2024 11:19:44

real_blur--0001-000024-real_blur.png
3.3 MB
Di 20 Feb 2024 11:19:44

real_blur--0001-000025-real_blur.png
3.2 MB
Di 20 Feb 2024 11:19:45

real_blur--0001-000026-real_blur.png
3.3 MB
Di 20 Feb 2024 11:19:46

real_blur--0001-000027-real_blur.png
3.3 MB
Di 20 Feb 2024 11:19:46

real_blur--0001-000028-real_blur.png
3.3 MB
Di 20 Feb 2024 11:19:47

real_blur--0001-000029-real_blur.png
3.4 MB
Di 20 Feb 2024 11:19:47

real_blur--0001-000030-real_blur.png
3.3 MB
Di 20 Feb 2024 11:19:48

real_blur--0001-000031-real_blur.png
3.4 MB
Di 20 Feb 2024 11:19:48

real_blur--0001-000032-real_blur.png
3.2 MB
Di 20 Feb 2024 11:19:49

real_blur--0001-000033-real_blur.png
3.2 MB
Di 20 Feb 2024 11:19:49

real_blur--0001-000034-real_blur.png
3.2 MB
Di 20 Feb 2024 11:19:50

real_blur--0001-000035-real_blur.png
3.3 MB
Di 20 Feb 2024 11:19:50

real_blur--0001-000036-real_blur.png
3.2 MB
Di 20 Feb 2024 11:19:51

real_blur--0001-000037-real_blur.png
3.3 MB
Di 20 Feb 2024 11:19:51

real_blur--0001-000038-real_blur.png
3.2 MB
Di 20 Feb 2024 11:19:52

real_blur--0001-000039-real_blur.png
3.1 MB
Di 20 Feb 2024 11:19:52

real_blur--0001-000040-real_blur.png
3.2 MB
Di 20 Feb 2024 11:19:53

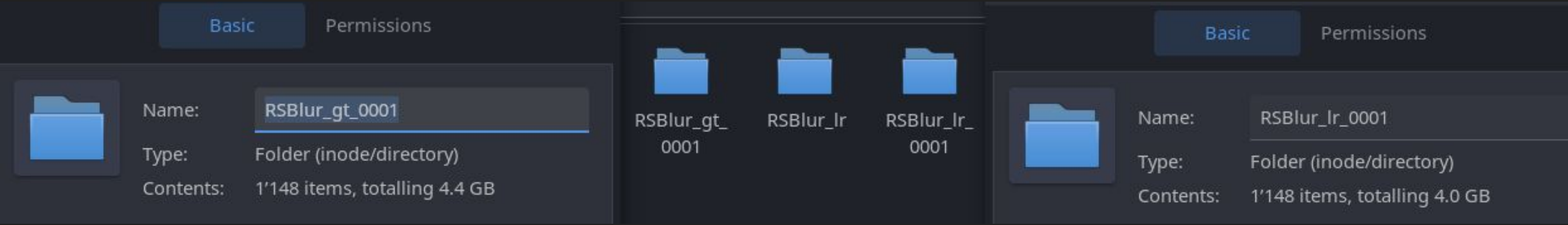
real_blur--0001-000042-real_blur.png
3.3 MB
Di 20 Feb 2024 11:19:53

real_blur--0001-000043-real_blur.png
3.5 MB
Di 20 Feb 2024 11:19:54

real_blur--0001-000044-real_blur.png
3.4 MB
Di 20 Feb 2024 11:19:54

real_blur--0001-000045-real_blur.png
3.4 MB
Di 20 Feb 2024 11:19:55

real_blur--0001-000046-real_blur.png
3.6 MB
Di 20 Feb 2024 11:19:56



Name: RSBlur_gt_0001
Type: Folder (inode/directory)
Contents: 1’148 items, totalling 4.4 GB

Name: RSBlur_lr_0001
Type: Folder (inode/directory)
Contents: 1’148 items, totalling 4.0 GB

More manageable size wise to continue processing
~59 Takes
But upon inspection was lacking in diversity. There were cars and structures. But for example no people / humans.
So I started handpicking Takes to get diversity: People, Cars, Structures, Plants, different Lightnings, and so forth, but still reducing dataset size for processing.
(Also mentioning, I could have taken just one image per take to have diversity of takes. But they have differences of motion blurs in each take (or object in motion), so leaving them as takes has value I think)

Handpicked Szenes:

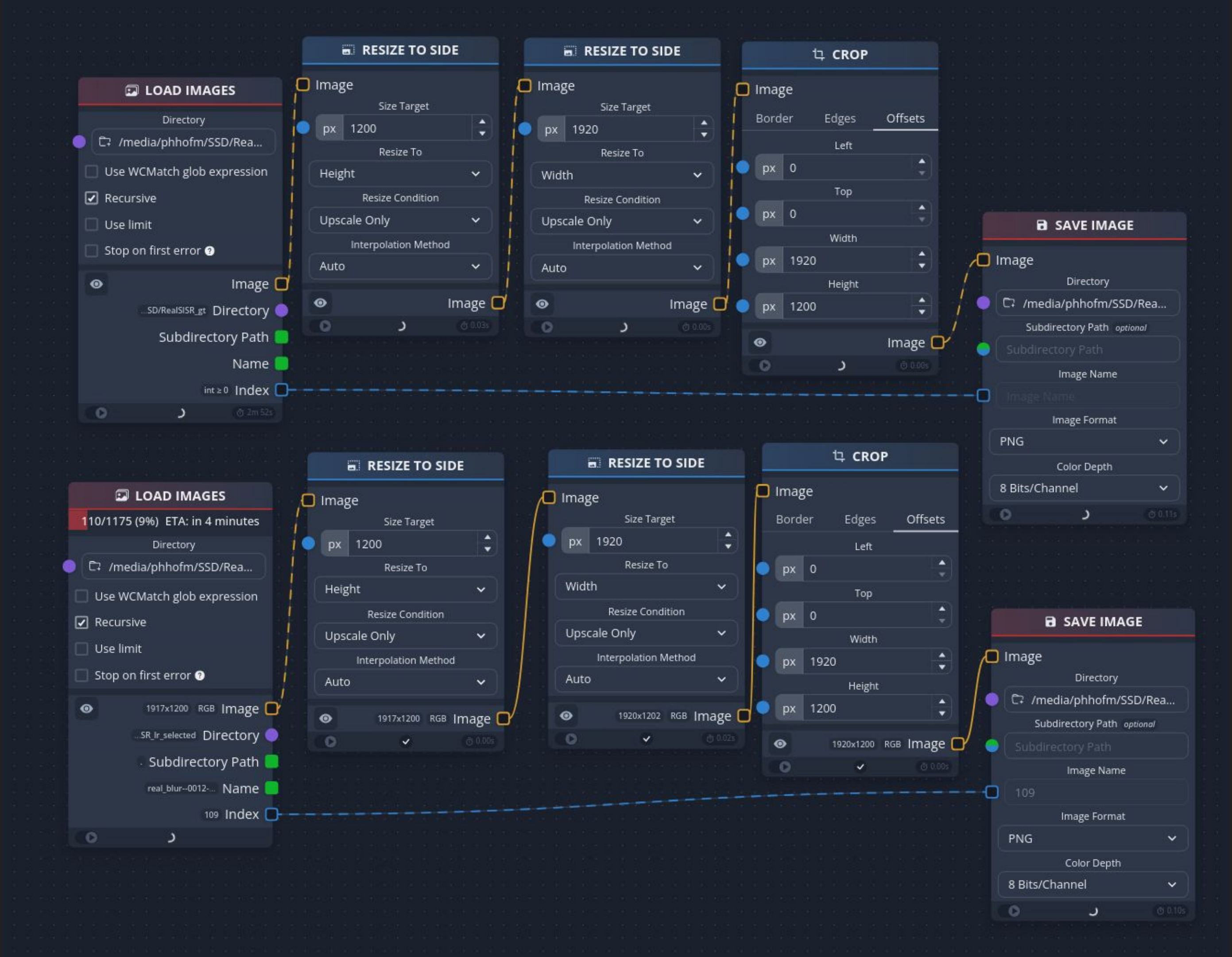
Dynamic scenes so lots of changes between shots / single images and also static ones with only camera shaking. People in motion. Food. Text. Cars. Buildings. Fence. Streets. Walls. Sky. Bright Sunlight. Sunlight and Shadows. Plants. Water. Glass.

New: Selection of around 61 scenes. 1’175 Images, gt is currently 4.5GB

I also selected 4 val images i will process in the same way (that are not used in the training dataset)

Now the image resolutions were fluctuating. Width 1918, 1917 (most) and 1916 and height 1199 (most), 1198 and 1197. These are some weird resolutions for doing a 4x scale paired dataset.

I normalized the dimensions to 1920x1200. (So lr’s after processing would be 480x300). Plus I also normalized the filenames to integers.



LOAD IMAGES

Directory: /media/phhofm/SSD/Rea...

□ Use WCMatch glob expression
☑ Recursive
□ Use limit
□ Stop on first error

Image
...SD/RealSISR_gt Directory
Subdirectory Path
Name
int ≥ 0 Index

RESIZE TO SIDE

Image
Size Target: px 1200
Resize To: Height
Resize Condition: Upscale Only
Interpolation Method: Auto
Image

RESIZE TO SIDE

Image
Size Target: px 1920
Resize To: Width
Resize Condition: Upscale Only
Interpolation Method: Auto
Image

CROP

Image
Border | Edges | Offsets
Left: px 0
Top: px 0
Width: px 1920
Height: px 1200
Image

SAVE IMAGE

Image
Directory: /media/phhofm/SSD/Rea...
Subdirectory Path *optional*: Subdirectory Path
Image Name: Image Name
Image Format: PNG
Color Depth: 8 Bits/Channel

LOAD IMAGES

110/1175 (9%) ETA: in 4 minutes

Directory: /media/phhofm/SSD/Rea...

□ Use WCMatch glob expression
☑ Recursive
□ Use limit
□ Stop on first error

1917x1200 RGB Image
...SR_lr_selected Directory
Subdirectory Path
real_blur--0012-... Name
109 Index

RESIZE TO SIDE

Image
Size Target: px 1200
Resize To: Height
Resize Condition: Upscale Only
Interpolation Method: Auto
1917x1200 RGB Image

RESIZE TO SIDE

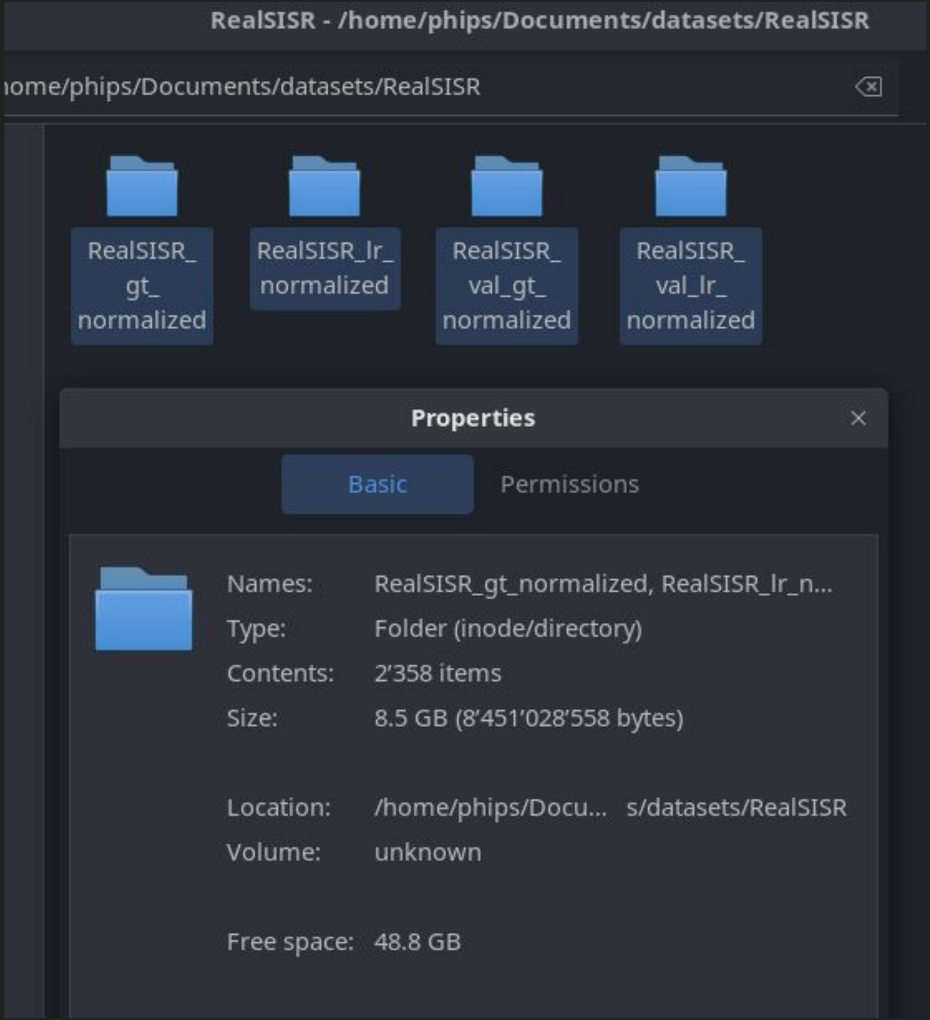
Image
Size Target: px 1920
Resize To: Width
Resize Condition: Upscale Only
Interpolation Method: Auto
1920x1202 RGB Image

CROP

Image
Border | Edges | Offsets
Left: px 0
Top: px 0
Width: px 1920
Height: px 1200
1920x1200 RGB Image

SAVE IMAGE

Image
Directory: /media/phhofm/SSD/Rea...
Subdirectory Path *optional*: Subdirectory Path
Image Name: 109
Image Format: PNG
Color Depth: 8 Bits/Channel


RealSISR - /home/phips/Documents/datasets/RealSISR
iome/phips/Documents/datasets/RealSISR
RealSISR_gt_normalized
RealSISR_lr_normalized
RealSISR_val_gt_normalized
RealSISR_val_lr_normalized
Properties
Basic
Permissions

Names: RealSISR_gt_normalized, RealSISR_lr_n...

Type: Folder (inode/directory)

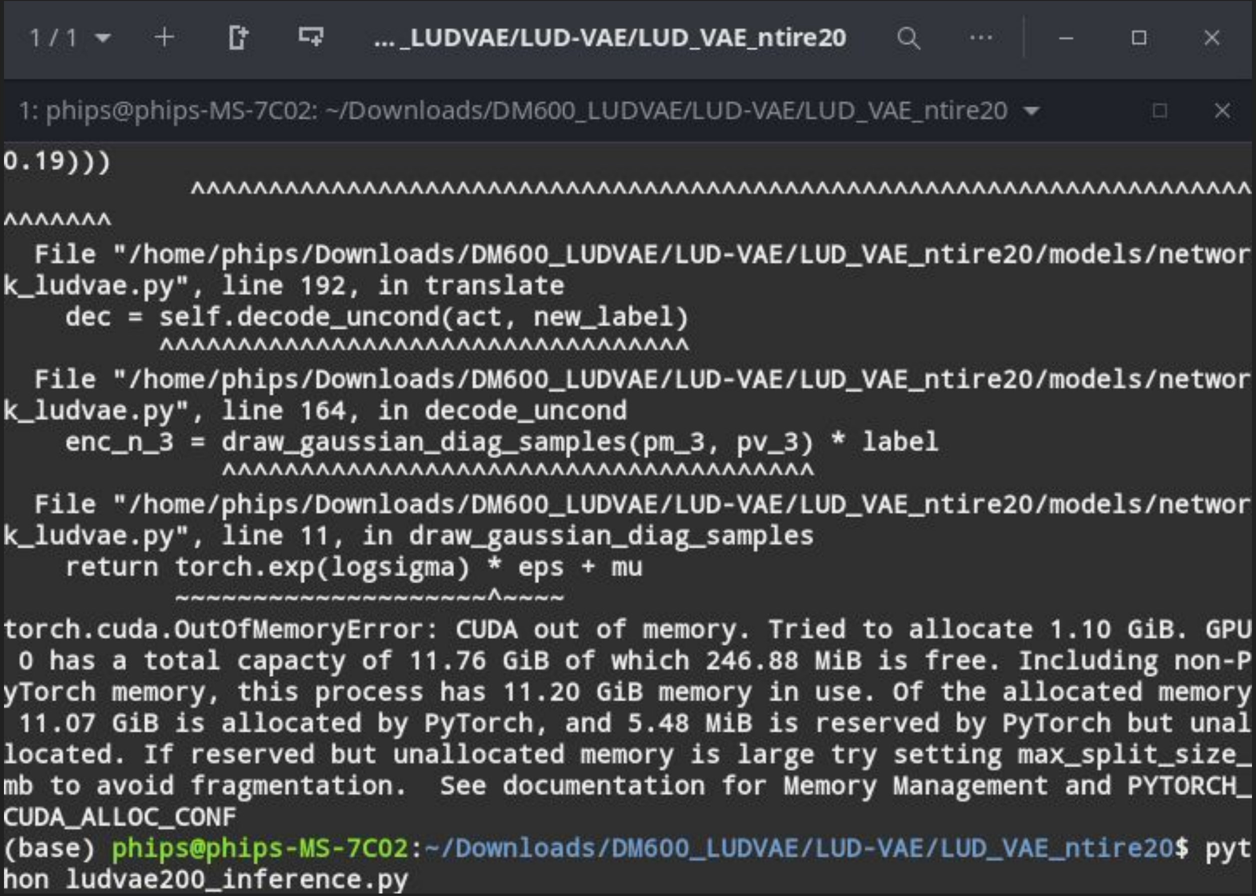
Contents: 2’358 items

Size: 8.5 GB (8’451’028’558 bytes)

Location: /home/phips/Docu... s/datasets/RealSISR

Volume: unknown

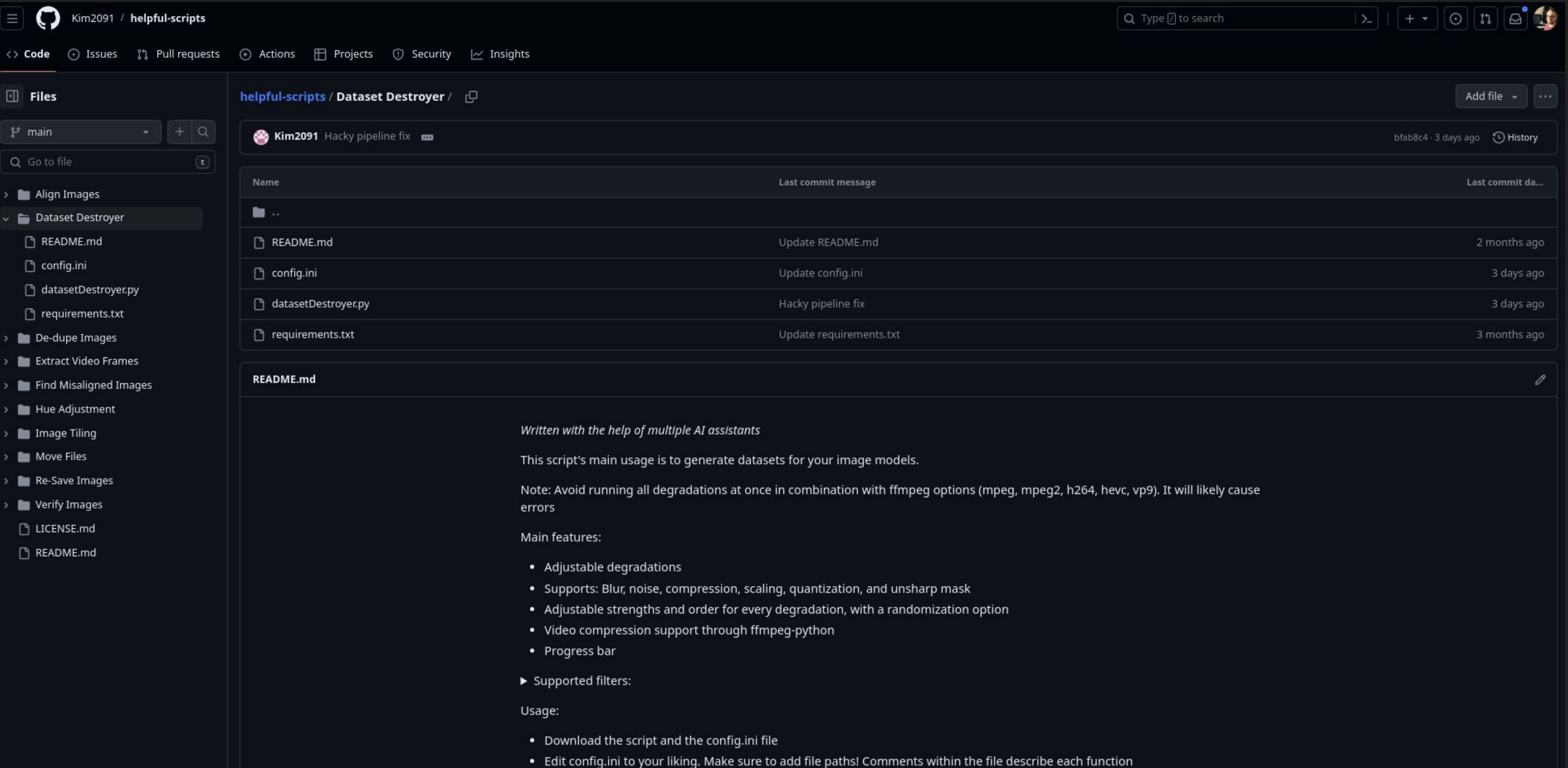
Free space: 48.8 GB


1 / 1
... _LUDVAE/LUD-VAE/LUD_VAE_ntire20
1: phips@phips-MS-7C02: ~/Downloads/DM600_LUDVAE/LUD-VAE/LUD_VAE_ntire20

```
0.19)))
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^
  File "/home/phips/Downloads/DM600_LUDVAE/LUD-VAE/LUD_VAE_ntire20/models/networ
k_ludvae.py", line 192, in translate
    dec = self.decode_uncond(act, new_label)
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/phips/Downloads/DM600_LUDVAE/LUD-VAE/LUD_VAE_ntire20/models/networ
k_ludvae.py", line 164, in decode_uncond
    enc_n_3 = draw_gaussian_diag_samples(pm_3, pv_3) * label
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/phips/Downloads/DM600_LUDVAE/LUD-VAE/LUD_VAE_ntire20/models/networ
k_ludvae.py", line 11, in draw_gaussian_diag_samples
    return torch.exp(logsigma) * eps + mu
           ~~~~~~~~~~~~~~~~~~~~^~~~~
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 1.10 GiB. GPU
 0 has a total capacity of 11.76 GiB of which 246.88 MiB is free. Including non-P
yTorch memory, this process has 11.20 GiB memory in use. Of the allocated memory
 11.07 GiB is allocated by PyTorch, and 5.48 MiB is reserved by PyTorch but unal
located. If reserved but unallocated memory is large try setting max_split_size_
mb to avoid fragmentation.  See documentation for Memory Management and PYTORCH_
CUDA_ALLOC_CONF
(base) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VAE/LUD_VAE_ntire20$ pyt
hon ludvae200_inference.py
```

Trying to apply my ludvae model, but running out of vram. I will need to mode the first half downsample step before this to not run out of vram

Kim’s Dataset Destroyer



Kim2091 Hacky pipeline fix

bfab8c4 · 3 days ago

Name	Last commit message	Last commit da...
..		
README.md	Update README.md	2 months ago
config.ini	Update config.ini	3 days ago
datasetDestroyer.py	Hacky pipeline fix	3 days ago
requirements.txt	Update requirements.txt	3 months ago

README.md

Written with the help of multiple AI assistants

This script's main usage is to generate datasets for your image models.

Note: Avoid running all degradations at once in combination with ffmpeg options (mpeg, mpeg2, h264, hevc, vp9). It will likely cause errors

Main features:

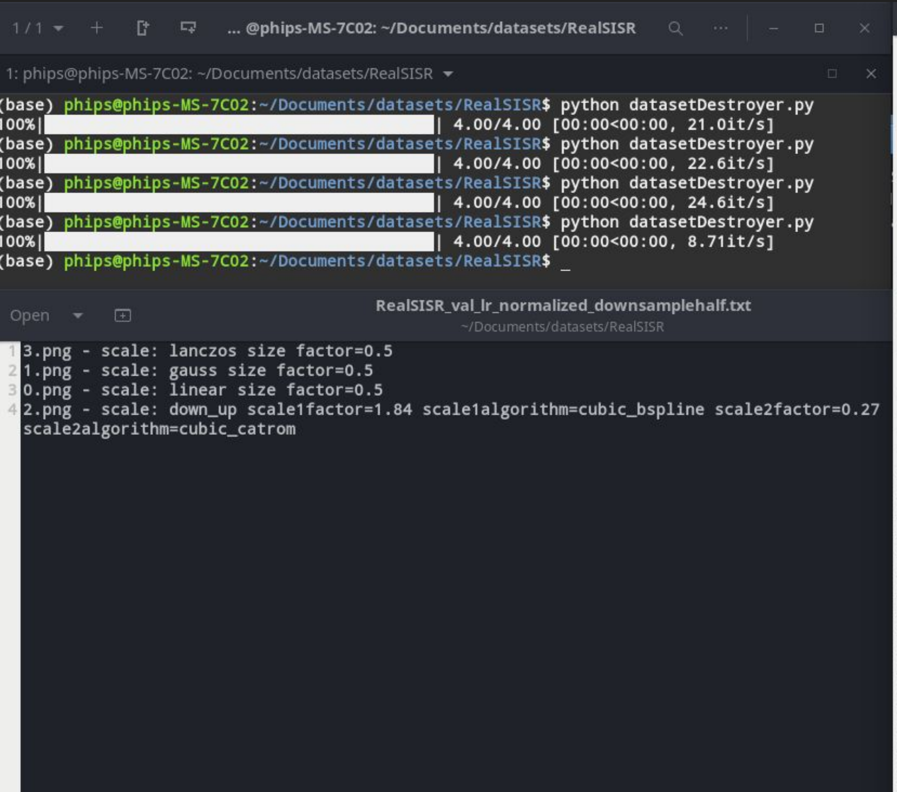
- Adjustable degradations
- Supports: Blur, noise, compression, scaling, quantization, and unsharp mask
- Adjustable strengths and order for every degradation, with a randomization option
- Video compression support through ffmpeg-python
- Progress bar

▶ Supported filters:

Usage:

- Download the script and the config.ini file
- Edit config.ini to your liking. Make sure to add file paths! Comments within the file describe each function

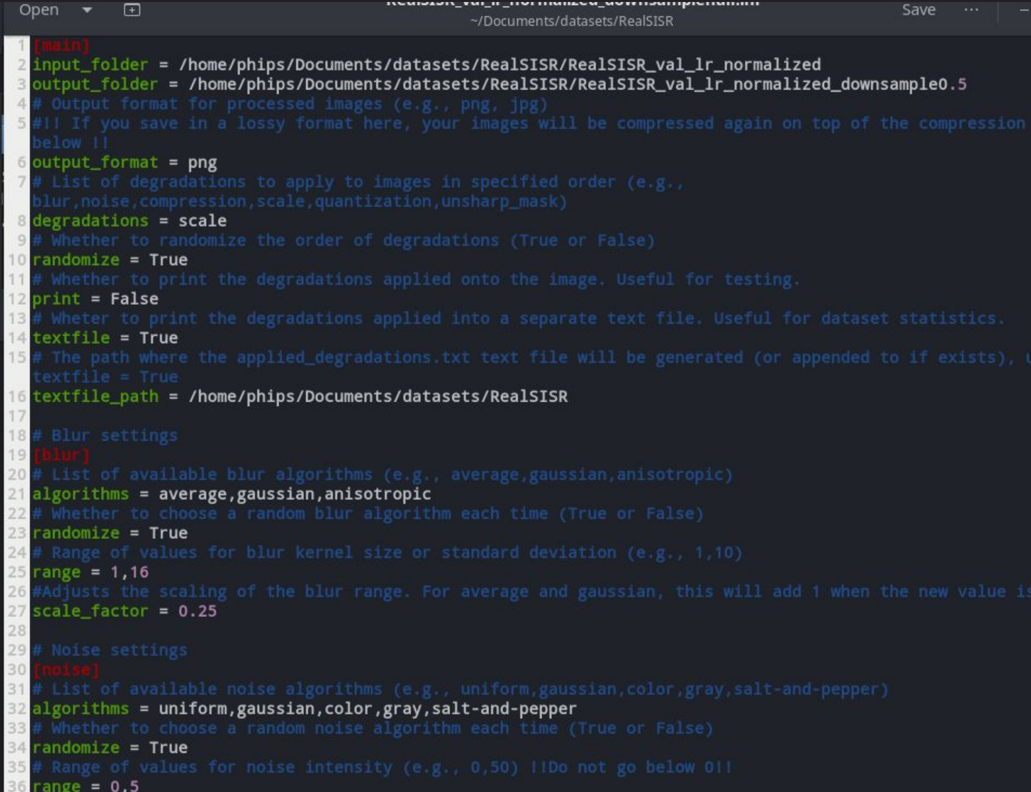
Half downsample with scale only, textfile log to check variety, all sampling methods provided enabled, val images test


1 / 1
... @phips-MS-7C02: ~/Documents/datasets/RealSISR
1: phips@phips-MS-7C02: ~/Documents/datasets/RealSISR

```
(base) phips@phips-MS-7C02:~/Documents/datasets/RealSISR$ python datasetDestroyer.py
100%|██████████████████████████████████████████| 4.00/4.00 [00:00<00:00, 21.0it/s]
(base) phips@phips-MS-7C02:~/Documents/datasets/RealSISR$ python datasetDestroyer.py
100%|██████████████████████████████████████████| 4.00/4.00 [00:00<00:00, 22.6it/s]
(base) phips@phips-MS-7C02:~/Documents/datasets/RealSISR$ python datasetDestroyer.py
100%|██████████████████████████████████████████| 4.00/4.00 [00:00<00:00, 24.6it/s]
(base) phips@phips-MS-7C02:~/Documents/datasets/RealSISR$ python datasetDestroyer.py
100%|██████████████████████████████████████████| 4.00/4.00 [00:00<00:00, 8.71it/s]
(base) phips@phips-MS-7C02:~/Documents/datasets/RealSISR$ _
```

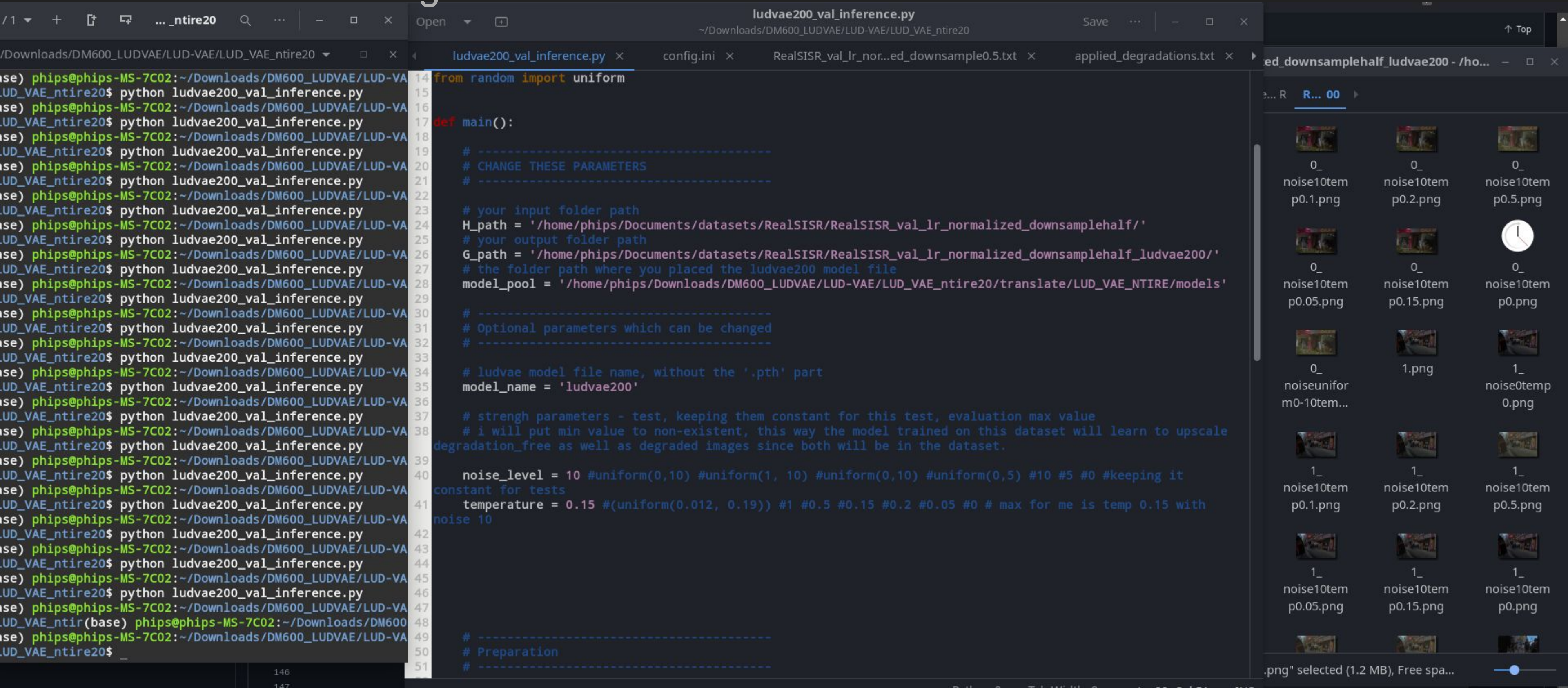
Open
RealSISR_val_lr_normalized_downsamplehalf.txt
~/Documents/datasets/RealSISR

```
3.png - scale: lanczos size factor=0.5
1.png - scale: gauss size factor=0.5
0.png - scale: linear size factor=0.5
2.png - scale: down_up scale1factor=1.84 scale1algorithm=cubic_bspline scale2factor=0.27 scale2algorithm=cubic_catrom
```



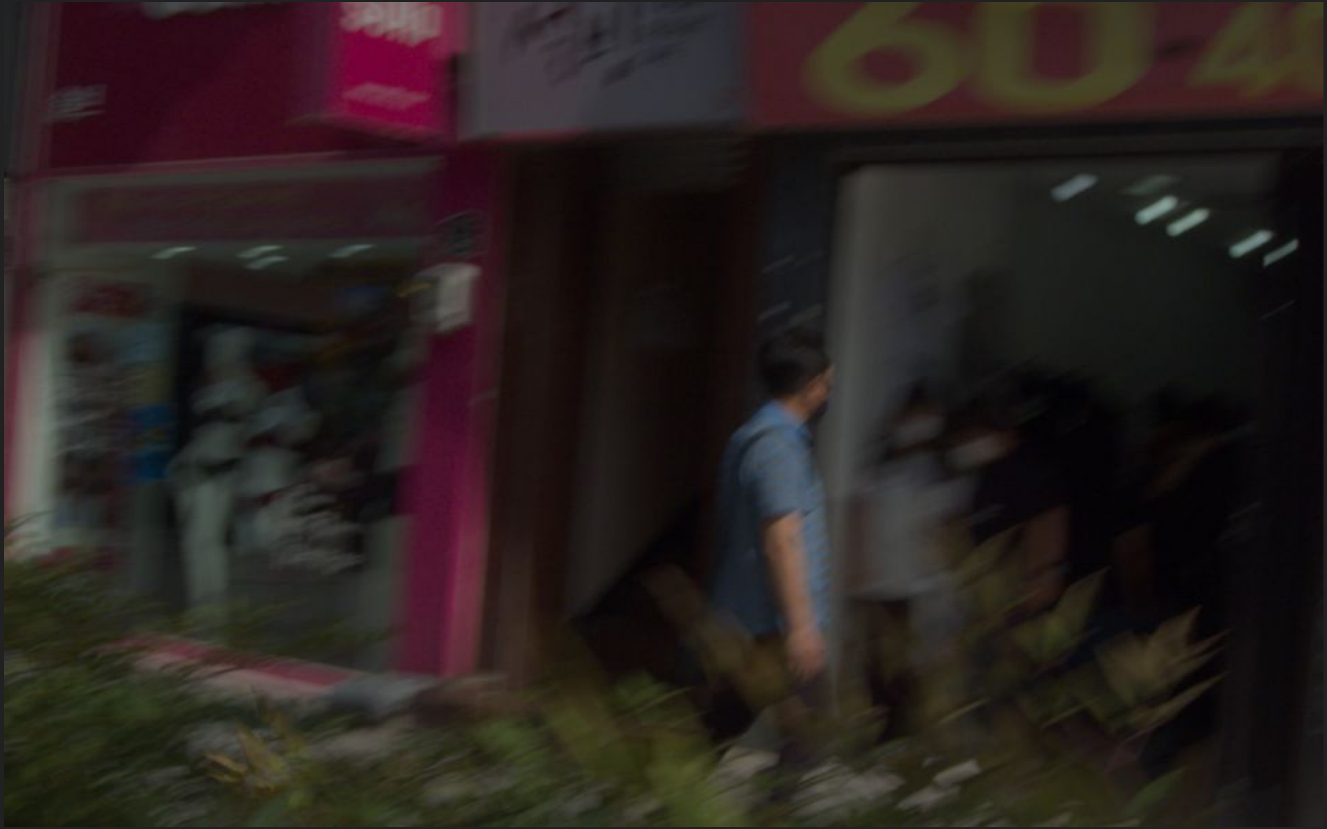
```
[main]
input_folder = /home/phips/Documents/datasets/RealSISR/RealSISR_val_lr_normalized
output_folder = /home/phips/Documents/datasets/RealSISR/RealSISR_val_lr_normalized_downsample0.5
# Output format for processed images (e.g., png, jpg)
#!! If you save in a lossy format here, your images will be compressed again on top of the compression below !!
output_format = png
# List of degradations to apply to images in specified order (e.g., blur,noise,compression,scale,quantization,unsharp_mask)
degradations = scale
# Whether to randomize the order of degradations (True or False)
randomize = True
# Whether to print the degradations applied onto the image. Useful for testing.
print = False
# Wheter to print the degradations applied into a separate text file. Useful for dataset statistics.
textfile = True
# The path where the applied_degradations.txt text file will be generated (or appended to if exists), textfile = True
textfile_path = /home/phips/Documents/datasets/RealSISR

# Blur settings
[blur]
# List of available blur algorithms (e.g., average,gaussian,anisotropic)
algorithms = average,gaussian,anisotropic
# Whether to choose a random blur algorithm each time (True or False)
randomize = True
# Range of values for blur kernel size or standard deviation (e.g., 1,10)
range = 1,16
#Adjusts the scaling of the blur range. For average and gaussian, this will add 1 when the new value is
scale_factor = 0.25

# Noise settings
[noise]
# List of available noise algorithms (e.g., uniform,gaussian,color,gray,salt-and-pepper)
algorithms = uniform,gaussian,color,gray,salt-and-pepper
# Whether to choose a random noise algorithm each time (True or False)
randomize = True
# Range of values for noise intensity (e.g., 0,50) !!Do not go below 0!!
range = 0,5
```

Worked (not out of vram anymore). But visually inspecting the results, degradations seemed too strong. So i started adjusting the strength settings in my inference script for this model, with constant values, what is the strongest setting i think okay with this model. Then final setting would be uniform down to 0 for the model to learn.



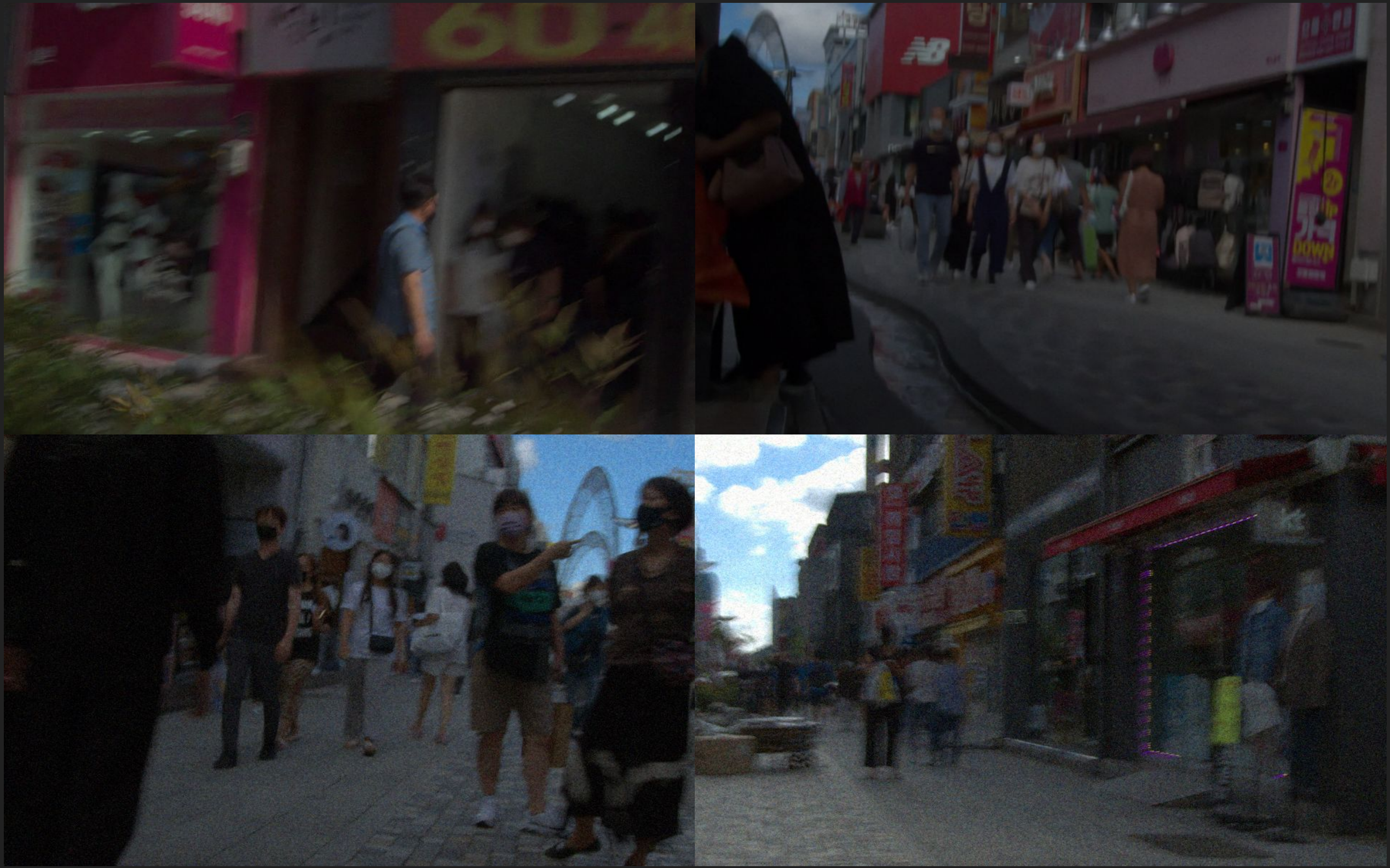
```
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ python ludvae200_val_inference.py
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntir(base) phips@phips-MS-7C02:~/Downloads/DM600
se) phips@phips-MS-7C02:~/Downloads/DM600_LUDVAE/LUD-VA
LUD_VAE_ntire20$ _
```

```python
from random import uniform


def main():

    # ----------------------------------------
    # CHANGE THESE PARAMETERS
    # ----------------------------------------

    # your input folder path
    H_path = '/home/phips/Documents/datasets/RealSISR/RealSISR_val_lr_normalized_downsamplehalf/'
    # your output folder path
    G_path = '/home/phips/Documents/datasets/RealSISR/RealSISR_val_lr_normalized_downsamplehalf_ludvae200/'
    # the folder path where you placed the ludvae200 model file
    model_pool = '/home/phips/Downloads/DM600_LUDVAE/LUD-VAE/LUD_VAE_ntire20/translate/LUD_VAE_NTIRE/models'

    # ----------------------------------------
    # Optional parameters which can be changed
    # ----------------------------------------

    # ludvae model file name, without the '.pth' part
    model_name = 'ludvae200'

    # strengh parameters - test, keeping them constant for this test, evaluation max value
    # i will put min value to non-existent, this way the model trained on this dataset will learn to upscale degradation_free as well as degraded images since both will be in the dataset.

    noise_level = 10 #uniform(0,10) #uniform(1, 10) #uniform(0,10) #uniform(0,5) #10 #5 #0 #keeping it constant for tests
    temperature = 0.15 #(uniform(0.012, 0.19)) #1 #0.5 #0.15 #0.2 #0.05 #0 # max for me is temp 0.15 with noise 10







    # ----------------------------------------
    # Preparation
    # ----------------------------------------
```

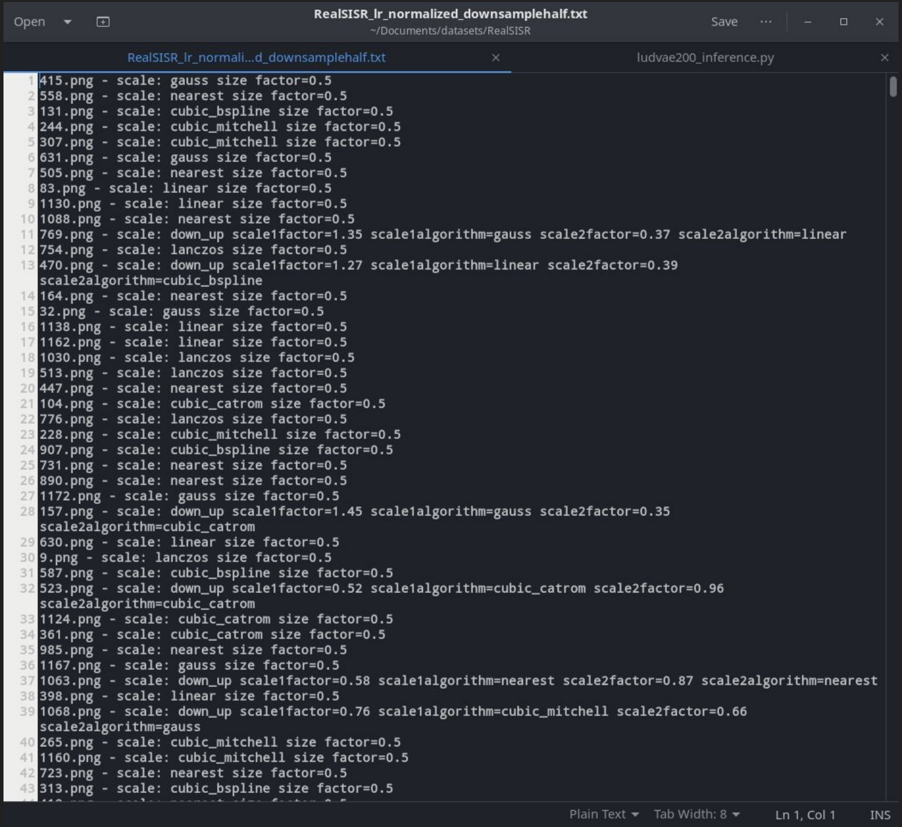


Input example (validation image)

Ludave200 inferred example, max degraded

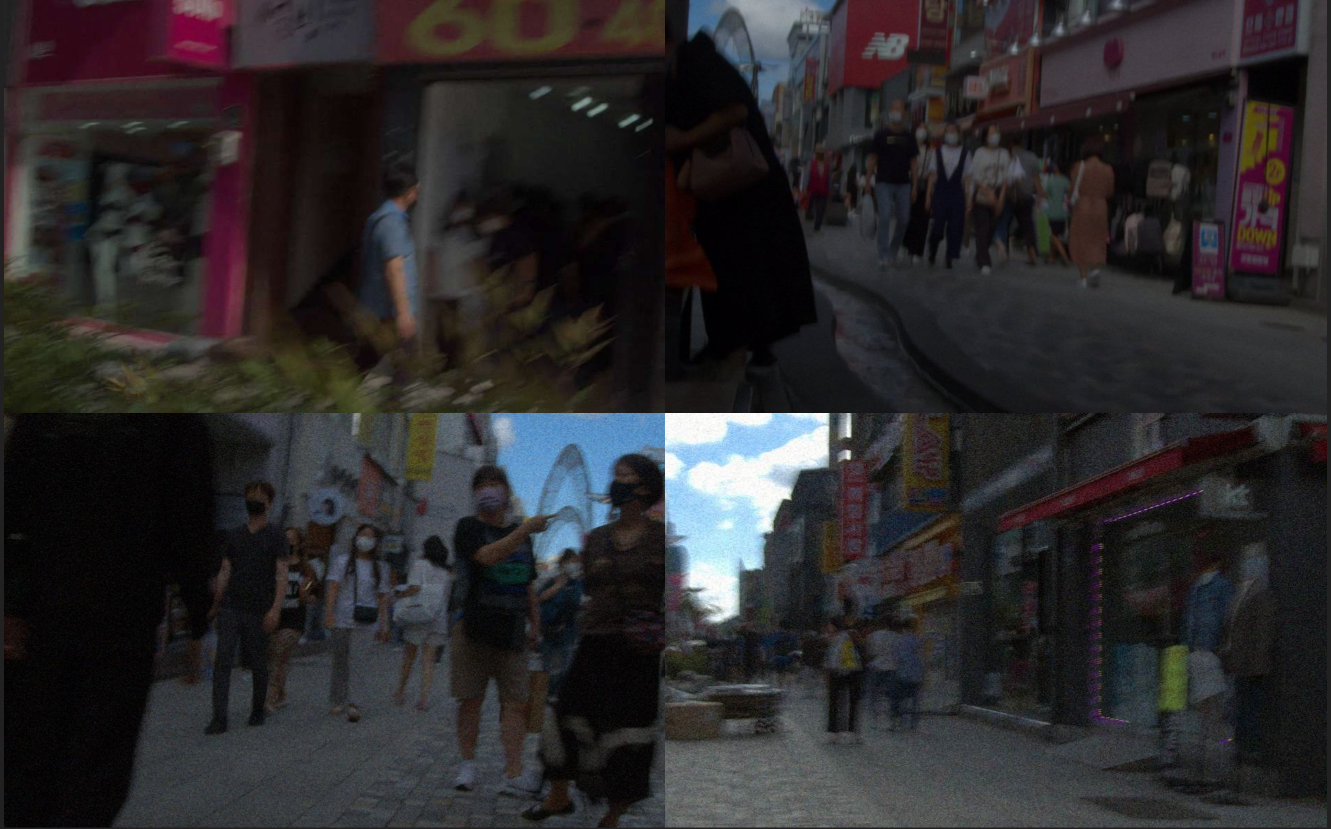


Val images example degraded with adjusted values for my model (these are only 4 images, the random strength distribution will be way better on the training dataset)

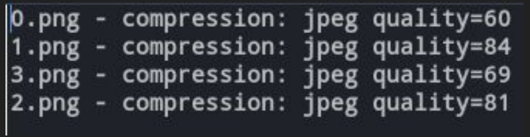

Open
Save
RealSISR_lr_normalized_downsamplehalf.txt
~/Documents/datasets/RealSISR
RealSISR_lr_normali...d_downsamplehalf.txt
ludvae200_inference.py
Plain Text
Tab Width: 8
Ln 1, Col 1
INS

```
415.png - scale: gauss size factor=0.5
558.png - scale: nearest size factor=0.5
131.png - scale: cubic_bspline size factor=0.5
244.png - scale: cubic_mitchell size factor=0.5
307.png - scale: cubic_mitchell size factor=0.5
631.png - scale: gauss size factor=0.5
505.png - scale: nearest size factor=0.5
83.png - scale: linear size factor=0.5
1130.png - scale: linear size factor=0.5
1088.png - scale: nearest size factor=0.5
769.png - scale: down_up scale1factor=1.35 scale1algorithm=gauss scale2factor=0.37 scale2algorithm=linear
754.png - scale: lanczos size factor=0.5
470.png - scale: down_up scale1factor=1.27 scale1algorithm=linear scale2factor=0.39
scale2algorithm=cubic_bspline
164.png - scale: nearest size factor=0.5
32.png - scale: gauss size factor=0.5
1138.png - scale: linear size factor=0.5
1162.png - scale: linear size factor=0.5
1030.png - scale: lanczos size factor=0.5
513.png - scale: lanczos size factor=0.5
447.png - scale: nearest size factor=0.5
104.png - scale: cubic_catrom size factor=0.5
776.png - scale: lanczos size factor=0.5
228.png - scale: cubic_mitchell size factor=0.5
907.png - scale: cubic_bspline size factor=0.5
731.png - scale: nearest size factor=0.5
890.png - scale: nearest size factor=0.5
1172.png - scale: gauss size factor=0.5
157.png - scale: down_up scale1factor=1.45 scale1algorithm=gauss scale2factor=0.35
scale2algorithm=cubic_catrom
630.png - scale: linear size factor=0.5
9.png - scale: lanczos size factor=0.5
587.png - scale: cubic_bspline size factor=0.5
523.png - scale: down_up scale1factor=0.52 scale1algorithm=cubic_catrom scale2factor=0.96
scale2algorithm=cubic_catrom
1124.png - scale: cubic_catrom size factor=0.5
361.png - scale: cubic_catrom size factor=0.5
985.png - scale: nearest size factor=0.5
1167.png - scale: gauss size factor=0.5
1063.png - scale: down_up scale1factor=0.58 scale1algorithm=nearest scale2factor=0.87 scale2algorithm=nearest
398.png - scale: linear size factor=0.5
1068.png - scale: down_up scale1factor=0.76 scale1algorithm=cubic_mitchell scale2factor=0.66
scale2algorithm=gauss
265.png - scale: cubic_mitchell size factor=0.5
1160.png - scale: cubic_mitchell size factor=0.5
723.png - scale: nearest size factor=0.5
313.png - scale: cubic_bspline size factor=0.5
```

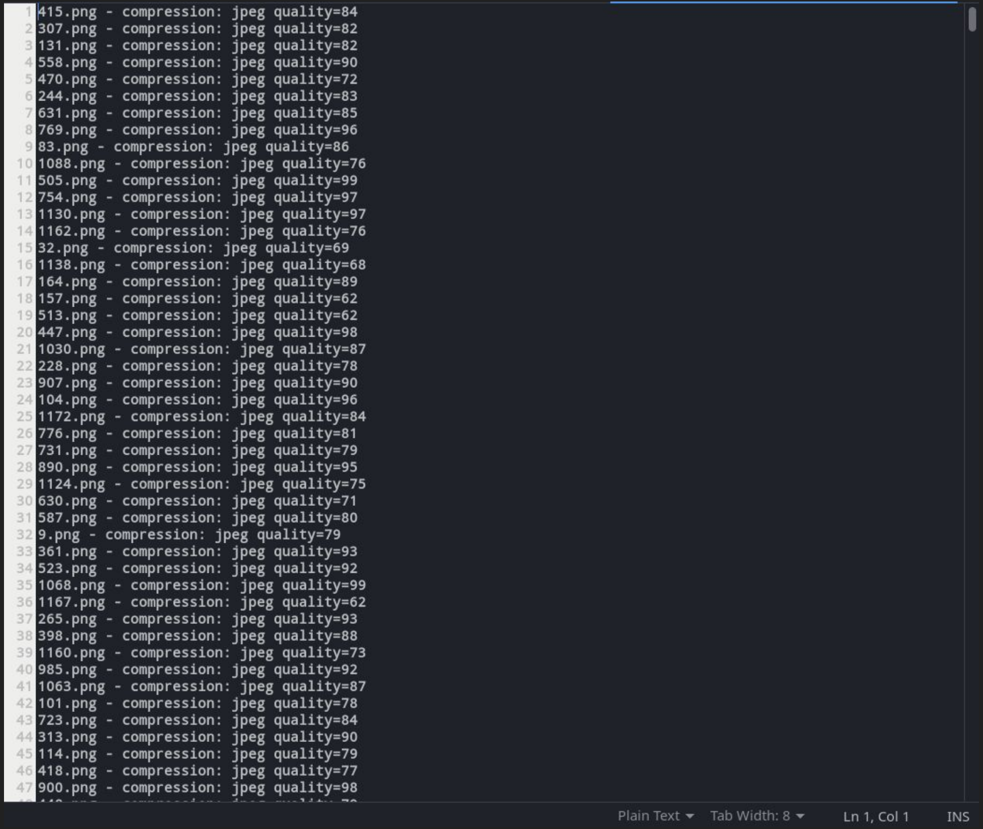
Same thing for the lr folder - downsample then ludvae200



Adding jpg compression with 60-100, google search preview images are 71. This simulates a person downsizing and compressing low quality (noise and blurry) photos to the web.

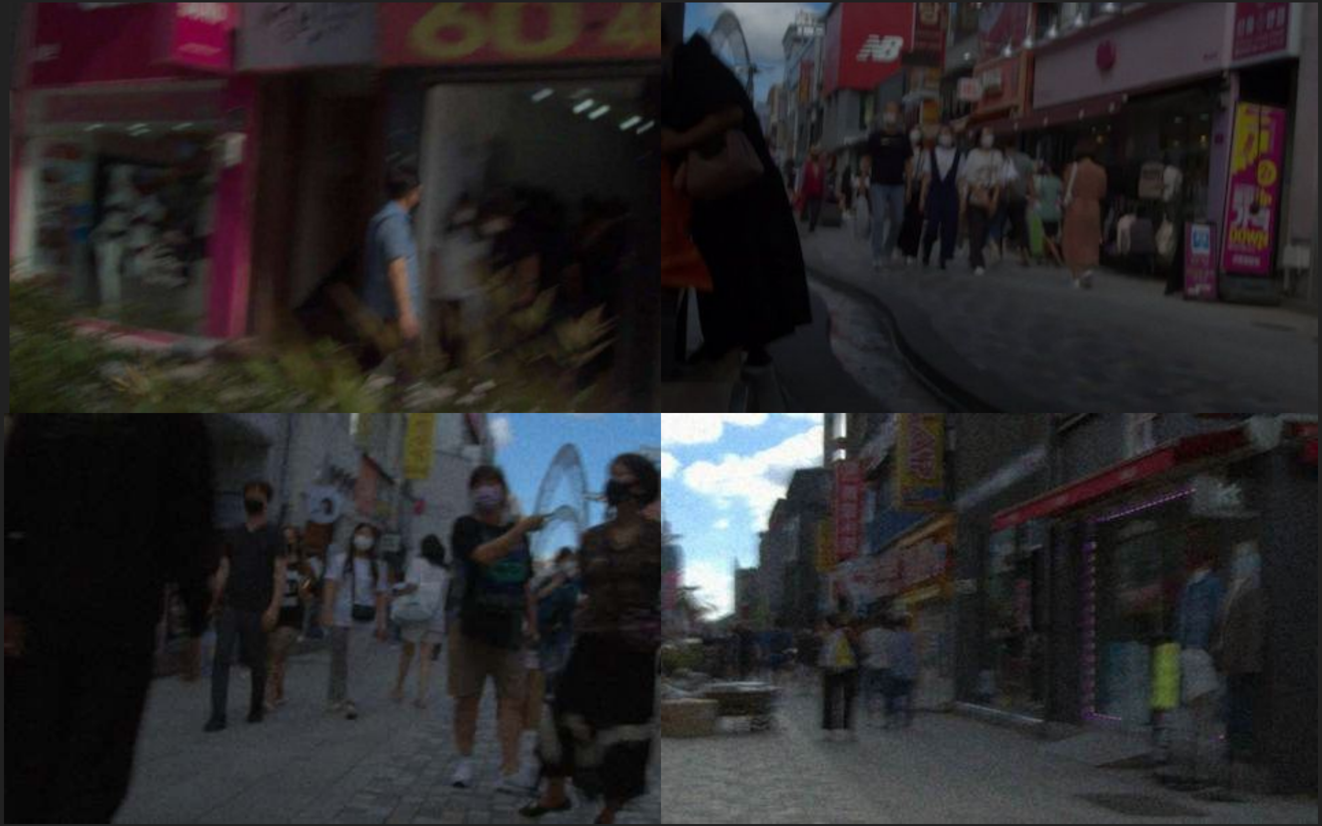


```
0.png - compression: jpeg quality=60
1.png - compression: jpeg quality=84
3.png - compression: jpeg quality=69
2.png - compression: jpeg quality=81
```



```
415.png - compression: jpeg quality=84
307.png - compression: jpeg quality=82
131.png - compression: jpeg quality=82
558.png - compression: jpeg quality=90
470.png - compression: jpeg quality=72
244.png - compression: jpeg quality=83
631.png - compression: jpeg quality=85
769.png - compression: jpeg quality=96
83.png - compression: jpeg quality=86
1088.png - compression: jpeg quality=76
505.png - compression: jpeg quality=99
754.png - compression: jpeg quality=97
1130.png - compression: jpeg quality=97
1162.png - compression: jpeg quality=76
32.png - compression: jpeg quality=69
1138.png - compression: jpeg quality=68
164.png - compression: jpeg quality=89
157.png - compression: jpeg quality=62
513.png - compression: jpeg quality=62
447.png - compression: jpeg quality=98
1030.png - compression: jpeg quality=87
228.png - compression: jpeg quality=78
907.png - compression: jpeg quality=90
104.png - compression: jpeg quality=96
1172.png - compression: jpeg quality=84
776.png - compression: jpeg quality=81
731.png - compression: jpeg quality=79
890.png - compression: jpeg quality=95
1124.png - compression: jpeg quality=75
630.png - compression: jpeg quality=71
587.png - compression: jpeg quality=80
9.png - compression: jpeg quality=79
361.png - compression: jpeg quality=93
523.png - compression: jpeg quality=92
1068.png - compression: jpeg quality=99
1167.png - compression: jpeg quality=62
265.png - compression: jpeg quality=93
398.png - compression: jpeg quality=88
1160.png - compression: jpeg quality=73
985.png - compression: jpeg quality=92
1063.png - compression: jpeg quality=87
101.png - compression: jpeg quality=78
723.png - compression: jpeg quality=84
313.png - compression: jpeg quality=90
114.png - compression: jpeg quality=79
418.png - compression: jpeg quality=77
900.png - compression: jpeg quality=98
```

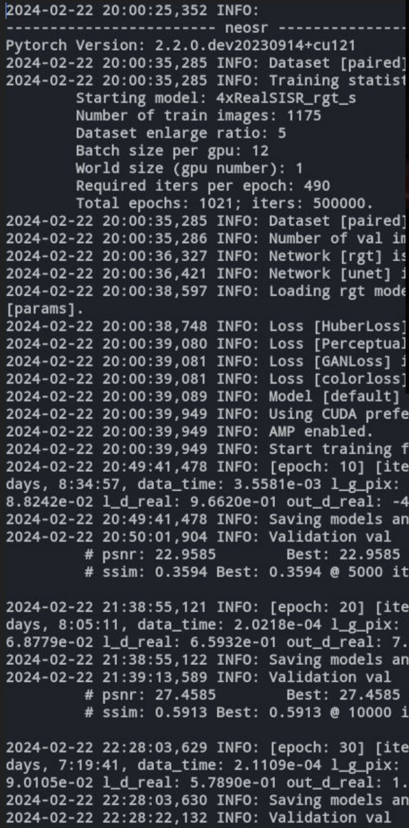
Same to lr folder



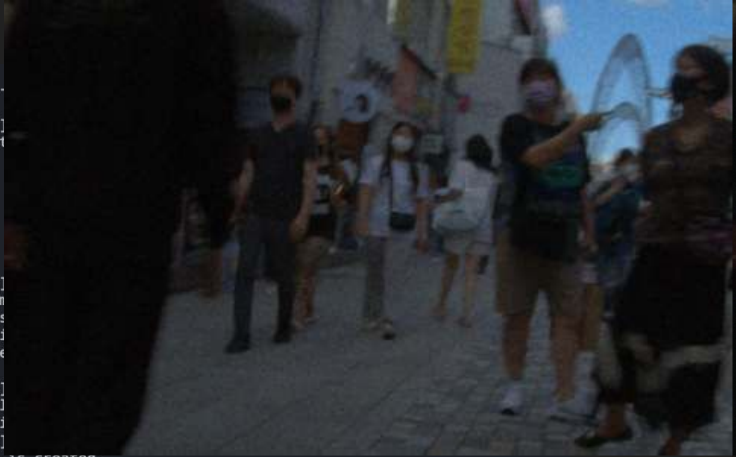
Then again applying scale, and then jpg compression to the lr images. This is how the final lr's might look like (the val images)

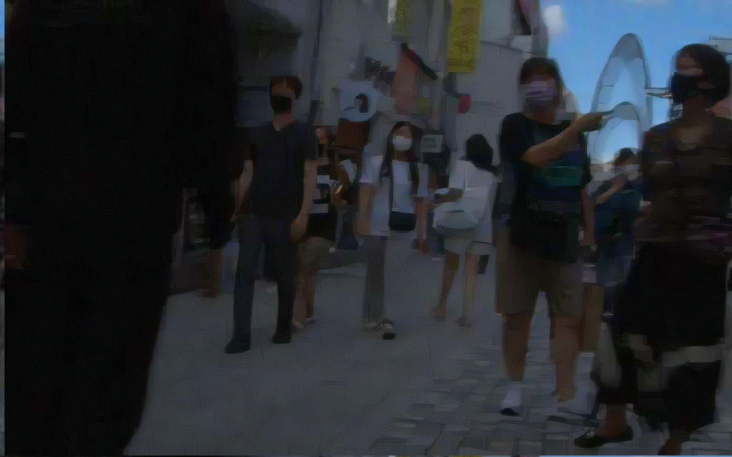
Here all the degradation steps on the example of 0.png, which is an already blurry image so no realistic blur needs to be added:

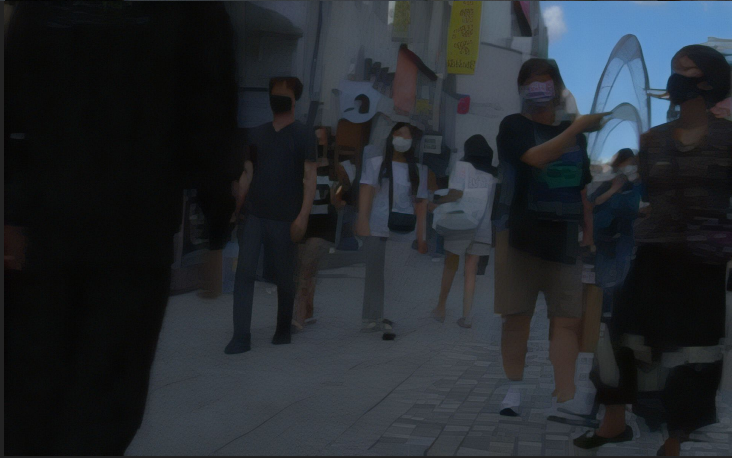
1. linear 0.5
2. ludvae200
3. jpg 60
4. down_up scale1factor=1.99 scale1algorithm=cubic_catrom scale2factor=0.25 scale2algorithm=gauss
5. jpg 85



```
2024-02-22 20:00:25,352 INFO:
------------------------ neosr --------------
Pytorch Version: 2.2.0.dev20230914+cu121
2024-02-22 20:00:35,285 INFO: Dataset [paired]
2024-02-22 20:00:35,285 INFO: Training statist
        Starting model: 4xRealSISR_rgt_s
        Number of train images: 1175
        Dataset enlarge ratio: 5
        Batch size per gpu: 12
        World size (gpu number): 1
        Required iters per epoch: 490
        Total epochs: 1021; iters: 500000.
2024-02-22 20:00:35,285 INFO: Dataset [paired]
2024-02-22 20:00:35,286 INFO: Number of val im
2024-02-22 20:00:36,327 INFO: Network [rgt] is
2024-02-22 20:00:36,421 INFO: Network [unet] i
2024-02-22 20:00:38,597 INFO: Loading rgt mode
[params].
2024-02-22 20:00:38,748 INFO: Loss [HuberLoss]
2024-02-22 20:00:39,080 INFO: Loss [Perceptual
2024-02-22 20:00:39,081 INFO: Loss [GANLoss] i
2024-02-22 20:00:39,081 INFO: Loss [colorloss]
2024-02-22 20:00:39,089 INFO: Model [default] is created.
2024-02-22 20:00:39,949 INFO: Using CUDA prefetch dataloader.
2024-02-22 20:00:39,949 INFO: AMP enabled.
2024-02-22 20:00:39,949 INFO: Start training from epoch: 0, iter: 0
2024-02-22 20:49:41,478 INFO: [epoch: 10] [iter:   5,000] [performance: 1.697 it/s] [lr:(7.143e-05)] [eta: 3
days, 8:34:57, data_time: 3.5581e-03 l_g_pix: 3.1343e-03 l_percep: 6.0649e+00 l_g_color: 9.3354e-04 l_g_gan:
8.8242e-02 l_d_real: 9.6620e-01 out_d_real: -4.4727e-01 l_d_fake: 5.6470e-01 out_d_fake: -3.1836e-01
2024-02-22 20:49:41,478 INFO: Saving models and training states.
2024-02-22 20:50:01,904 INFO: Validation val
         # psnr: 22.9585        Best: 22.9585 @ 5000 iter
         # ssim: 0.3594 Best: 0.3594 @ 5000 iter

2024-02-22 21:38:55,121 INFO: [epoch: 20] [iter:  10,000] [performance: 1.711 it/s] [lr:(1.429e-04)] [eta: 3
days, 8:05:11, data_time: 2.0218e-04 l_g_pix: 1.1371e-03 l_percep: 5.5915e+00 l_g_color: 2.2461e-04 l_g_gan:
6.8779e-02 l_d_real: 6.5932e-01 out_d_real: 7.2754e-02 l_d_fake: 7.0201e-01 out_d_fake: 1.4099e-02
2024-02-22 21:38:55,122 INFO: Saving models and training states.
2024-02-22 21:39:13,589 INFO: Validation val
         # psnr: 27.4585        Best: 27.4585 @ 10000 iter
         # ssim: 0.5913 Best: 0.5913 @ 10000 iter

2024-02-22 22:28:03,629 INFO: [epoch: 30] [iter:  15,000] [performance: 1.707 it/s] [lr:(2.143e-04)] [eta: 3
days, 7:19:41, data_time: 2.1109e-04 l_g_pix: 1.3216e-03 l_percep: 5.0609e+00 l_g_color: 2.0179e-04 l_g_gan:
9.0105e-02 l_d_real: 5.7890e-01 out_d_real: 1.2422e+00 l_d_fake: 6.7330e-01 out_d_fake: -2.2949e-01
2024-02-22 22:28:03,630 INFO: Saving models and training states.
2024-02-22 22:28:22,132 INFO: Validation val
```







Trained 2 experiment models with it, compact for 40k and rgt_s for 70k. Problem: Motion blur in dataset is too strong. Need another realistic blur dataset.
Better: Find a method to degrade with realistic blur with adjustable strengths instead of using pre-blurred dataset I cannot adjust