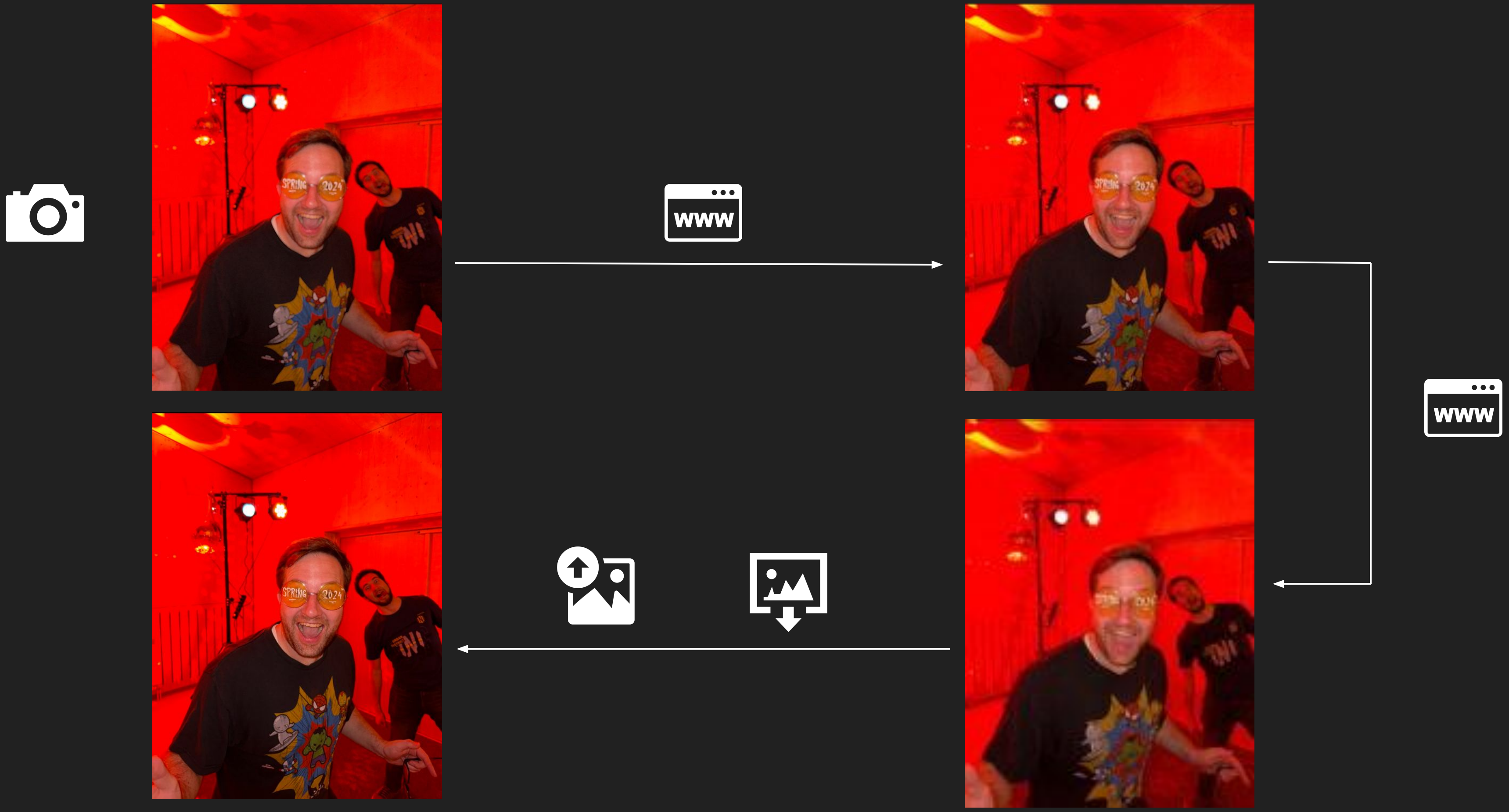
4xNomosWebPhoto

Dataset Creation Workflow
By Philip Hofmann

Use Case

4x upscaling an image that was downloaded from the web

Images from the web range from lossless good quality to downsampled and (re) compressed images for faster client web load speeds, while photography additionally could already be blurry and noisy before uploaded. This degradation workflow tries to simulate all of these cases so an upscaling model trained with it might be able to handle these use cases.



Use Case Visualization - A photo taken of me and a friend, uploaded and re-uploaded on the web (each time downscaled and compressed by the service provider). An upscale would ideally reach the original state again.

Dataset used: Nomos-v2

For this I use the Nomos-v2 dataset as released by musl on neosr and simple degrade it.

The Nomos-v2 dataset contains 6000 images of 512x512px each.

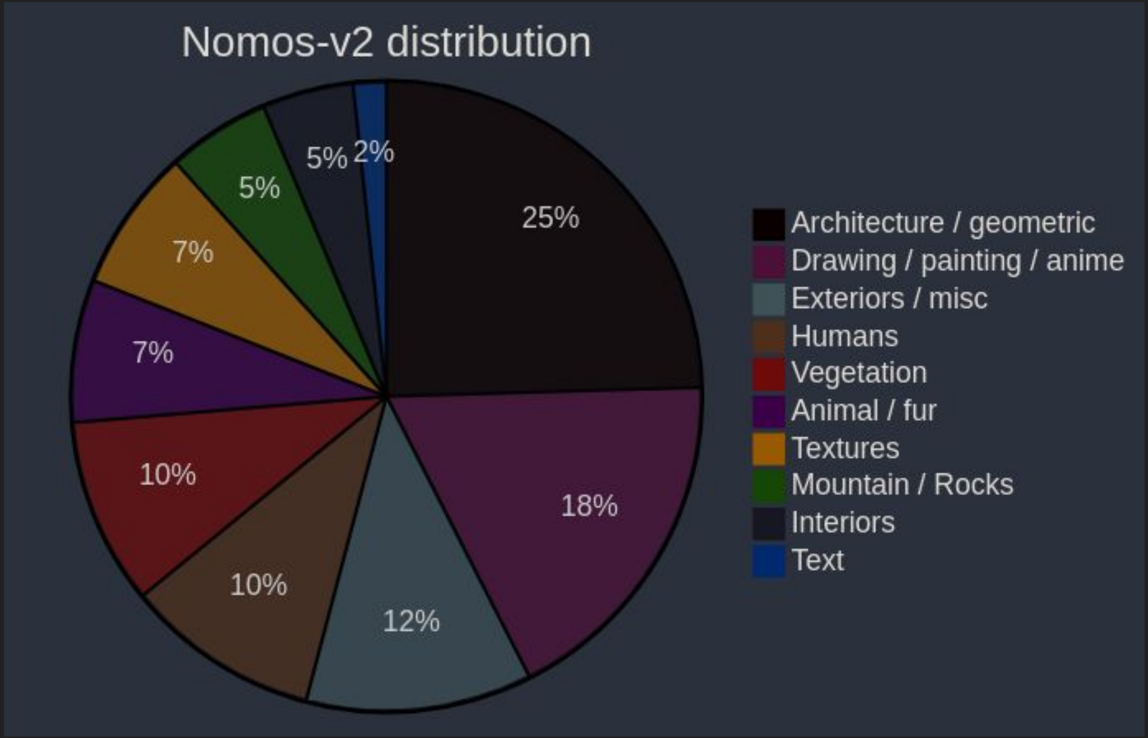
The purpose of this dataset is to distill only the best images from the academic and community datasets. A total of 14 datasets were manually reviewed, including: Adobe-MIT-5k, RAISE, LSDIR, LIU4k-v2, KONIQ, Nikon Low-Light RAW Image Dataset, DIV8k, FFHQ, Flickr2k, ModernAnimation1080, Rawsamples, SignatureEdits, Hasselblad raw samples and Unsplash.

Raw images were processed on rawtherapee using prebayer deconvolution, AMaZe and CAM16 on AP1 color space.

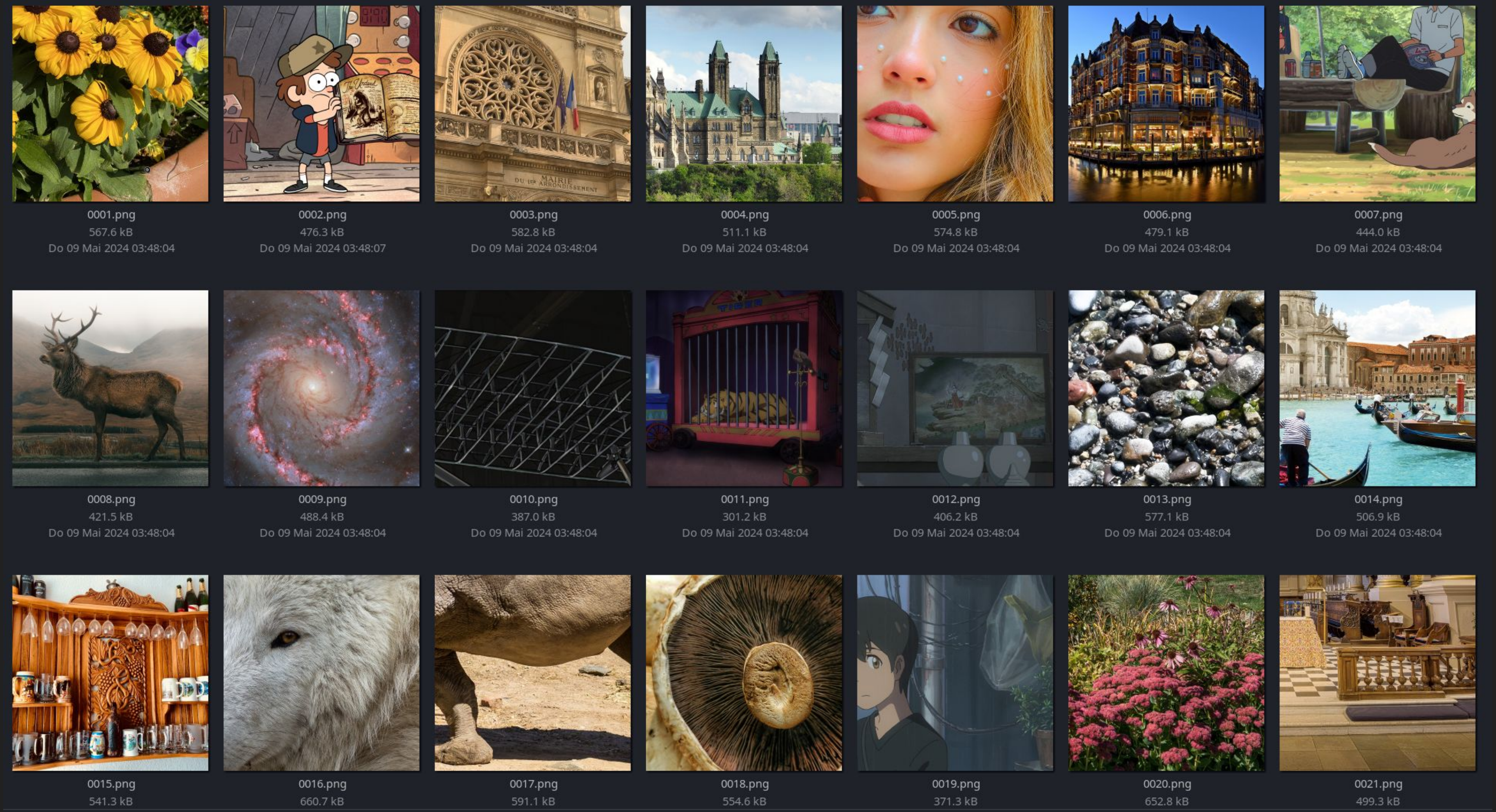
Downsampling was done using Mitchell interpolation and post RL deconvolution.

The criteria for selection were:

- High signal-to-noise ratio (low noise)
- Diverse
- Sharp (no motion blur, shallow DOF allowed)
- Contains mixed and complex textures/shapes that cover most part of the image



Nomos-v2 distribution



Example of Nomos-v2 tiles / images (First 21 out of 6000)

Adding Blur

I am adding lens blur with either radius 2 or 3 as visualized below

Adding Noise

To add noise im using my self-trained Ludvae200 degradation model as released.

The noise im using is from 1 to 5 and the temperature is from 0.06 to 1.4

Below a visualization for the min and max, it applies the noise in a spectrum, Noise 3 Temp 0.1 is just an example of a median noise it could apply in the dataset

Original Ludvae200 - Noise 1 Temp 0.06 Ludvae200 - Noise 3 Temp 0.1 Ludvae 200 - Noise 5 Temp 0.14

Adding Compression

Since this is images from the web, im adding jpg and webp compression again in a randomized manner, below a visualization of the max compression.

For jpg i went with quality 70 as max, which I thought should suffice, since previews in the Google search for pictures are compressed with quality 74. And for webp I chose 72 to result in the (almost) same compressed file size.

I think this should suffice for most images (we normally get better quality if training on less degradation so i am trying to keep the max reasonable) unless they went extreme on compression strength. We are also going to add re-compression to the dataset which would help with more compressed images.

Visualization of Max Compression

Adding Re-Compression

To simulate the download and re-upload of an image from the web to a service provider, I add re-compression to the dataset with the same max strengths

12 Variants

- Unaltered upload
  - Downscale
  - Blur, downscale
  - Noise, downscale
  - Blur, noise, downscale
- Upload
  - Downscale, compression
  - Blur, downscale, compression
  - Noise, downscale, compression
  - Blur, noise, downscale, compression
- Re-Upload
  - Downscale, compression, recompression
  - Blur, downscale, compression, recompression
  - Noise, downscale, compression, recompression
  - Blur, noise, downscale, compression, recompression

Name	Size	Type
nomosv2_lq_4x	6'000 items	Folder
nomosv2_lq_4x_comp	6'000 items	Folder
nomosv2_lq_4x_lens	6'000 items	Folder
nomosv2_lq_4x_lens_comp	6'000 items	Folder
nomosv2_lq_4x_lens_ludvae200	6'000 items	Folder
nomosv2_lq_4x_lens_ludvae200_comp	6'000 items	Folder
nomosv2_lq_4x_lens_ludvae200_recomp	6'000 items	Folder
nomosv2_lq_4x_lens_recomp	6'000 items	Folder
nomosv2_lq_4x_ludvae200	6'000 items	Folder
nomosv2_lq_4x_ludvae200_comp	6'000 items	Folder
nomosv2_lq_4x_ludvae200_recomp	6'000 items	Folder
nomosv2_lq_4x_recomp	6'000 items	Folder

Why Variants?

We are working with different degraded lr folders here because I want to simulate the different use cases.

The benefit here is that since we are using non-degraded inputs a trained model at the end should do also well on non-degraded, meaning good quality, input. Then blur only, then noise only, blur and noise, blur and noise and compressed, blur and compressed, and so forth. Basically ensuring that it gives a good output on images still that do not have less degradations occurring, until fully degraded.

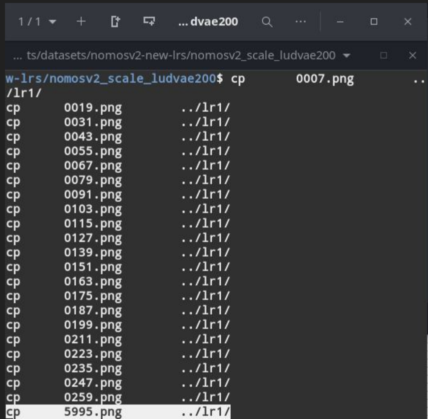
Visual example of all variants of a specific train lr image

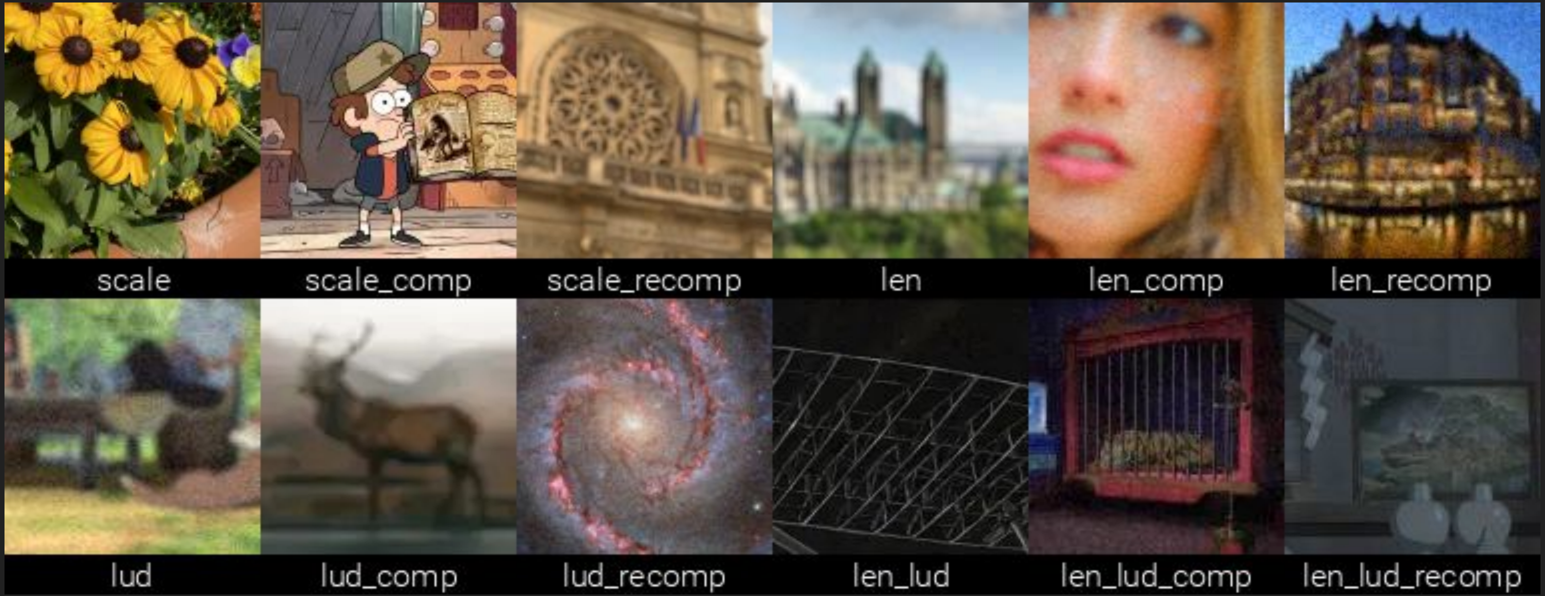
Created a mixed lr1 folder

Basically mixing together all the variants in the same folder, in a repetitive manner.

With batch size 12 now, each variation will appear within each batch, so this mixed lr1 folder kinda recommends that batch, at least for early stages of training.

Better would be in a randomized order. Meaning if the training software itself, one could in the config give an array of lr folders, and for each image (like ‘0001.png’) it randomly gets it from one of the lr folders. So with longer training it would see more variance of degradation settings, also batch would matter less since randomized.





Mixed lr1 folder

Training

I trained a RealPLKSR model on this dataset (a modification by musl on PLKSR) with neosr since im testing out that arch currently.

I did not do any fancy method but pretty barebones, I set batch to 12 to match the variants in the mixed lr1 folder, left gt_size at 128, used a gan pretrain, and let it train for 270’000 iterations without changing the configuration during training.

I will show the configuration on the next slide, and then validation outputs. Some validation inputs are pretty strongly degraded, that is for me to more clearly see the denoising, decompression etc effects of the trained model and to draw conclusions from it.

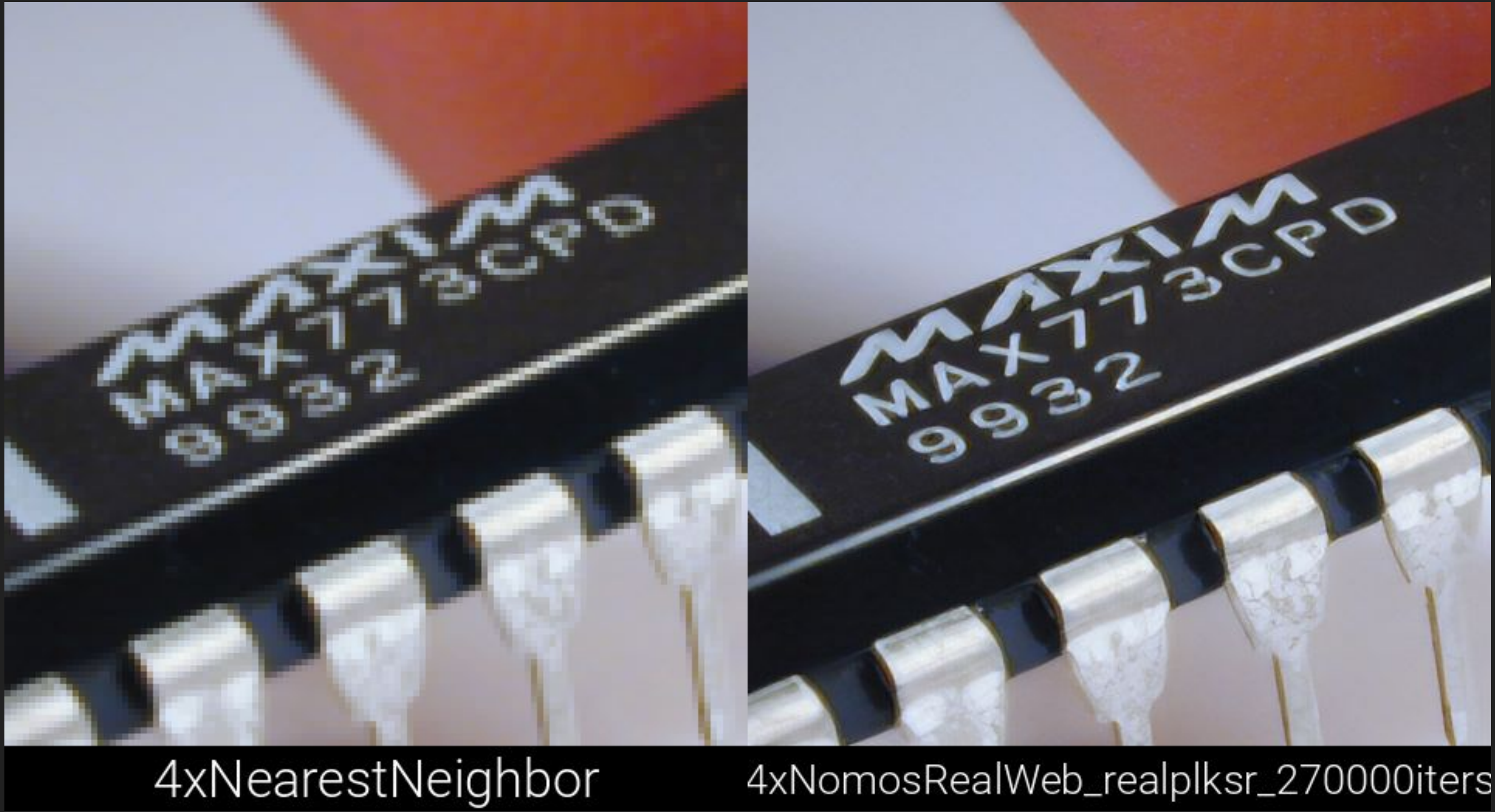
```yaml
name: 4xRealWebImage_realplksr
model_type: default
scale: 4
use_amp: true
bfloat16: true
fast_matmul: true
compile: false

datasets:
  train:
    type: paired
    dataroot_gt: '/home/phips/Documents/datasets/nomosv2/'
    dataroot_lq: '/home/phips/Documents/datasets/nomosv2_lr/nomosv2_lr1/'
    io_backend:
      type: disk

    gt_size: 128
    batch_size: 12 # the number of variants in my dataset
    accumulate: 1
    dataset_enlarge_ratio: 1

    use_hflip: true
    use_rot: true
    augmentation: ['none', 'mixup', 'cutmix', 'resizemix'] #['cutblur']
    aug_prob: [0.2, 0.3, 0.2, 0.5] #[0.7]

  val:
    name: val_deg
    type: single
    dataroot_lq: '/home/phips/Documents/datasets/RealLR7_256/'
    io_backend:
      type: disk

val:
  val_freq: 5000
  save_img: true
  tile: -1 # 200

path:
  pretrain_network_g: '/home/phips/Downloads/4x_realplksr_gan_pretrain.pth'
  resume_state: ~

network_g:
  type: realplksr

network_d:
  type: unet

train:
  optim_g:
    type: adamw
    lr: !!float 1e-4
    weight_decay: 0
    betas: [0.9, 0.99]
  optim_d:
    type: adamw
    lr: !!float 1e-4
    weight_decay: 0
    betas: [0.9, 0.99]

  scheduler:
    type: cosineannealing
    T_max: 360000
    eta_min: !!float 5e-5

  total_iter: 360000
  warmup_iter: 20000

  # losses
  wavelet_guided: "off" # "disc", "on"
  mssim_opt:
    type: mssim
    loss_weight: 1.0
  perceptual_opt:
    type: PerceptualLoss
    layer_weights:
      'conv1_2': 0.1
      'conv2_2': 0.1
      'conv3_4': 1
      'conv4_4': 1
      'conv5_4': 1
    perceptual_weight: 1.0
    criterion: huber
  gan_opt:
    type: GANLoss
    gan_type: vanilla
    loss_weight: 0.1

  match_lq: false
  color_opt:
    type: colorloss
    loss_weight: 1.0
    criterion: huber
  luma_opt:
    type: lumaloss
    loss_weight: 1.0
    criterion: huber

logger:
  print_freq: 100
  save_checkpoint_freq: 5000
  use_tb_logger: true
```

The training config

Validation Image 1

Validation Image 2

Validation Image 3



Validation Image 4

4xNearestNeighbor | 4xNomosRealWeb_realplksr_270000iters

Validation Image 5

4xNearestNeighbor

4xNomosRealWeb_realplksr_270000iters

Validation Image 6

Validation Image 7

Validation Image 8

Validation Image 9

Observations from model training outputs on dataset

- Lens blur radius 3 is too strong and needs to be reduced
- Noise degradation settings / strength is too strong and needs to be reduced
- Maybe adding down_up with multiple downsampling algos might help with some of these outputs

Rework 1 - Rebuilding Dataset

Using multiple downsampling algos and down_up now as base lr folder

```
# Scale settings
[scale]
# List of available scale algorithms (e.g., down_up,linear,cubic_catrom,cubic_mitchell,(
algorithms = down_up,linear,cubic_mitchell,lanczos,gauss,box
# List of available scale algorithms when applying down_up
down_up_algorithms = linear,cubic_mitchell,lanczos,gauss,box
# Whether to choose a random scale algorithm each time (True or False)
randomize = True
# Factor to scale your images to (e.g., 0.25, 0.50, 0.75) (0.25 = 25%, 0.50 = 50%)
size_factor = 0.25
# Range of values for down_up (e.g., 0.5,2.0) (0.5 = 50%, 2.0 = 200%)
range = 0.15,1.5
```

```
4918.png - scale: gauss size factor=0.25
2161.png - scale: cubic_mitchell size factor=0.25
0476.png - scale: linear size factor=0.25
1999.png - scale: lanczos size factor=0.25
4512.png - scale: cubic_mitchell size factor=0.25
0553.png - scale: cubic_mitchell size factor=0.25
3088.png - scale: down_up scale1factor=1.23 scale1algorithm=linear scale2factor=0.20 scale2algorithm=gauss
2750.png - scale: down_up scale1factor=1.30 scale1algorithm=lanczos scale2factor=0.19 scale2algorithm=lanczos
5043.png - scale: gauss size factor=0.25
2773.png - scale: gauss size factor=0.25
2178.png - scale: down_up scale1factor=0.31 scale1algorithm=lanczos scale2factor=0.82 scale2algorithm=linear
4496.png - scale: lanczos size factor=0.25
0440.png - scale: linear size factor=0.25
3454.png - scale: lanczos size factor=0.25
3527.png - scale: box size factor=0.25
2717.png - scale: lanczos size factor=0.25
2582.png - scale: lanczos size factor=0.25
4434.png - scale: gauss size factor=0.25
5950.png - scale: lanczos size factor=0.25
4672.png - scale: box size factor=0.25
3640.png - scale: box size factor=0.25
3176.png - scale: cubic_mitchell size factor=0.25
1468.png - scale: box size factor=0.25
1162.png - scale: down_up scale1factor=1.06 scale1algorithm=cubic_mitchell scale2factor=0.24 scale2algorithm=lanczos
0974.png - scale: lanczos size factor=0.25
0046.png - scale: lanczos size factor=0.25
4631.png - scale: lanczos size factor=0.25
5835.png - scale: down_up scale1factor=0.67 scale1algorithm=lanczos scale2factor=0.37 scale2algorithm=linear
0192.png - scale: box size factor=0.25
3893.png - scale: gauss size factor=0.25
1497.png - scale: box size factor=0.25
4475.png - scale: down_up scale1factor=0.75 scale1algorithm=gauss scale2factor=0.34 scale2algorithm=lanczos
3488.png - scale: cubic_mitchell size factor=0.25
3480.png - scale: box size factor=0.25
3310.png - scale: linear size factor=0.25
1244.png - scale: lanczos size factor=0.25
0769.png - scale: down_up scale1factor=1.27 scale1algorithm=box scale2factor=0.20 scale2algorithm=linear
1017.png - scale: down_up scale1factor=0.28 scale1algorithm=gauss scale2factor=0.89 scale2algorithm=lanczos
5396.png - scale: down_up scale1factor=0.75 scale1algorithm=cubic_mitchell scale2factor=0.33 scale2algorithm=cubic_mitchell
2199.png - scale: lanczos size factor=0.25
0837.png - scale: gauss size factor=0.25
3963.png - scale: cubic_mitchell size factor=0.25
2600.png - scale: cubic_mitchell size factor=0.25
3868.png - scale: cubic_mitchell size factor=0.25
2253.png - scale: lanczos size factor=0.25
0751.png - scale: down_up scale1factor=1.42 scale1algorithm=linear scale2factor=0.18 scale2algorithm=linear
1339.png - scale: down_up scale1factor=0.83 scale1algorithm=box scale2factor=0.30 scale2algorithm=linear
5359.png - scale: cubic_mitchell size factor=0.25
5293.png - scale: down_up scale1factor=0.29 scale1algorithm=linear scale2factor=0.88 scale2algorithm=box
0921.png - scale: gauss size factor=0.25
3943.png - scale: cubic_mitchell size factor=0.25
4866.png - scale: lanczos size factor=0.25
2900.png - scale: cubic_mitchell size factor=0.25
3885.png - scale: cubic_mitchell size factor=0.25
1614.png - scale: cubic_mitchell size factor=0.25
4832.png - scale: lanczos size factor=0.25
0348.png - scale: lanczos size factor=0.25
0601.png - scale: down_up scale1factor=0.83 scale1algorithm=box scale2factor=0.30 scale2algorithm=gauss
3447.png - scale: cubic_mitchell size factor=0.25
3936.png - scale: linear size factor=0.25
3384.png - scale: lanczos size factor=0.25
5463.png - scale: lanczos size factor=0.25
0027.png - scale: cubic_mitchell size factor=0.25
1569.png - scale: cubic_mitchell size factor=0.25
```

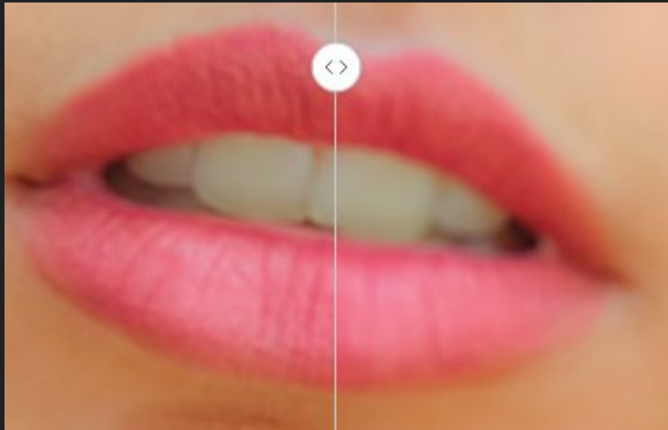
Lens Blur strength reduced and distribution increased

Radius, components and gamma random for each image within certain range

Drastically reduced, min is now radius 1 components 2, max is radius 2 components 4. I think blurs in general (lens, gaussian, …) need to be used very carefully, they get very strong very fast, even in their lowest settings.

Left img: difference original and new min. Right img: difference new and old max.

```
# selecting a random lens blur radius to adjust the
random_lens_blur_radius = random.uniform(1.0, 2.0)

# random components
random_lens_blur_components = random.randint(2, 4)

# random gamma
random_lens_blur_gamma = random.randint(1, 4)
```



Noise reduction - new min (new, old) (noise1tmp0.04, noise1tmp0.06)

Noise reduction - new max (new, old) (noise5tmp0.08, noise5tmp0.14)

Max lens&noise degraded new, max lens&noise degraded old

Compression and recompression settings kept the same

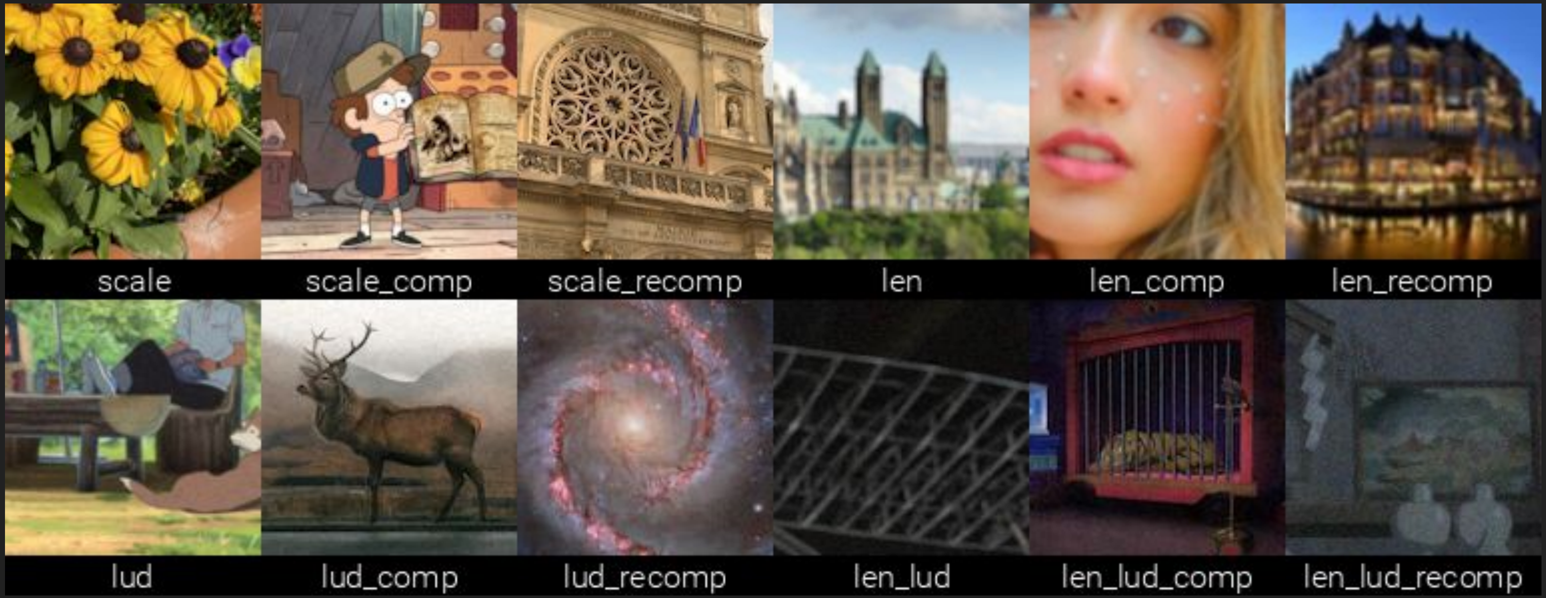
```
# Compression settings
[compression]
# List of available compressio
# Using more intensive codecs
algorithms = jpeg,webp
# Whether to choose a random a
randomize = True
# JPEG Quality Levels
jpeg_quality_range = 70,100
# WebP Quality levels
webp_quality_range = 72,100
```

```
2750.png - scale: box size factor=0.25, compression: jpeg quality=82
0476.png - scale: cubic_mitchell size factor=0.25, compression: jpeg quality=73
4512.png - scale: linear size factor=0.25, compression: jpeg quality=92
1999.png - scale: lanczos size factor=0.25, compression: jpeg quality=73
4918.png - scale: gauss size factor=0.25, compression: jpeg quality=90
3088.png - scale: cubic_mitchell size factor=0.25, compression: jpeg quality=87
0553.png - scale: down_up scale1factor=1.15 scale1algorithm=linear scale2factor=0.
2161.png - scale: down_up scale1factor=1.23 scale1algorithm=gauss scale2factor=0.2
2773.png - scale: lanczos size factor=0.25, compression: jpeg quality=97
5043.png - scale: cubic_mitchell size factor=0.25, compression: jpeg quality=74
0440.png - scale: lanczos size factor=0.25, compression: jpeg quality=81
3527.png - scale: linear size factor=0.25, compression: jpeg quality=79
3454.png - scale: gauss size factor=0.25, compression: jpeg quality=75
4496.png - scale: gauss size factor=0.25, compression: jpeg quality=80
2178.png - scale: cubic_mitchell size factor=0.25, compression: jpeg quality=87
1162.png - scale: linear size factor=0.25, compression: jpeg quality=77
3176.png - scale: box size factor=0.25, compression: jpeg quality=88
2582.png - scale: gauss size factor=0.25, compression: jpeg quality=73
4434.png - scale: linear size factor=0.25, compression: webp quality=77
2717.png - scale: cubic_mitchell size factor=0.25, compression: jpeg quality=72
4672.png - scale: cubic_mitchell size factor=0.25, compression: jpeg quality=81
3640.png - scale: cubic_mitchell size factor=0.25, compression: jpeg quality=98
5950.png - scale: gauss size factor=0.25, compression: webp quality=94
0974.png - scale: down_up scale1factor=0.23 scale1algorithm=cubic_mitchell scale2f
4475.png - scale: cubic_mitchell size factor=0.25, compression: jpeg quality=71
0046.png - scale: cubic_mitchell size factor=0.25, compression: webp quality=84
```

The order is still important of course, first blur, then noise, then compression

Max degraded image in new dataset, original, then lens,noise,jpg,jpg

Max degraded image in old dataset, original, then lens,noise,jpg,jpg



Mixed lr1 folder

Training

I trained again a RealPLKSR model on this dataset with the same settings as previously.

This one is trained for less iterations, but it is enough to see the difference.

Though I got some conclusions from it again, looking at outputs (for example tree val - too many details get lost) and at the lr’s, the lens blur still seems too strong.

Ill show some outputs of the new model


4xNearestNeighbor
4xNomosRealWeb_realplksr_110000iters

Rework Val 1

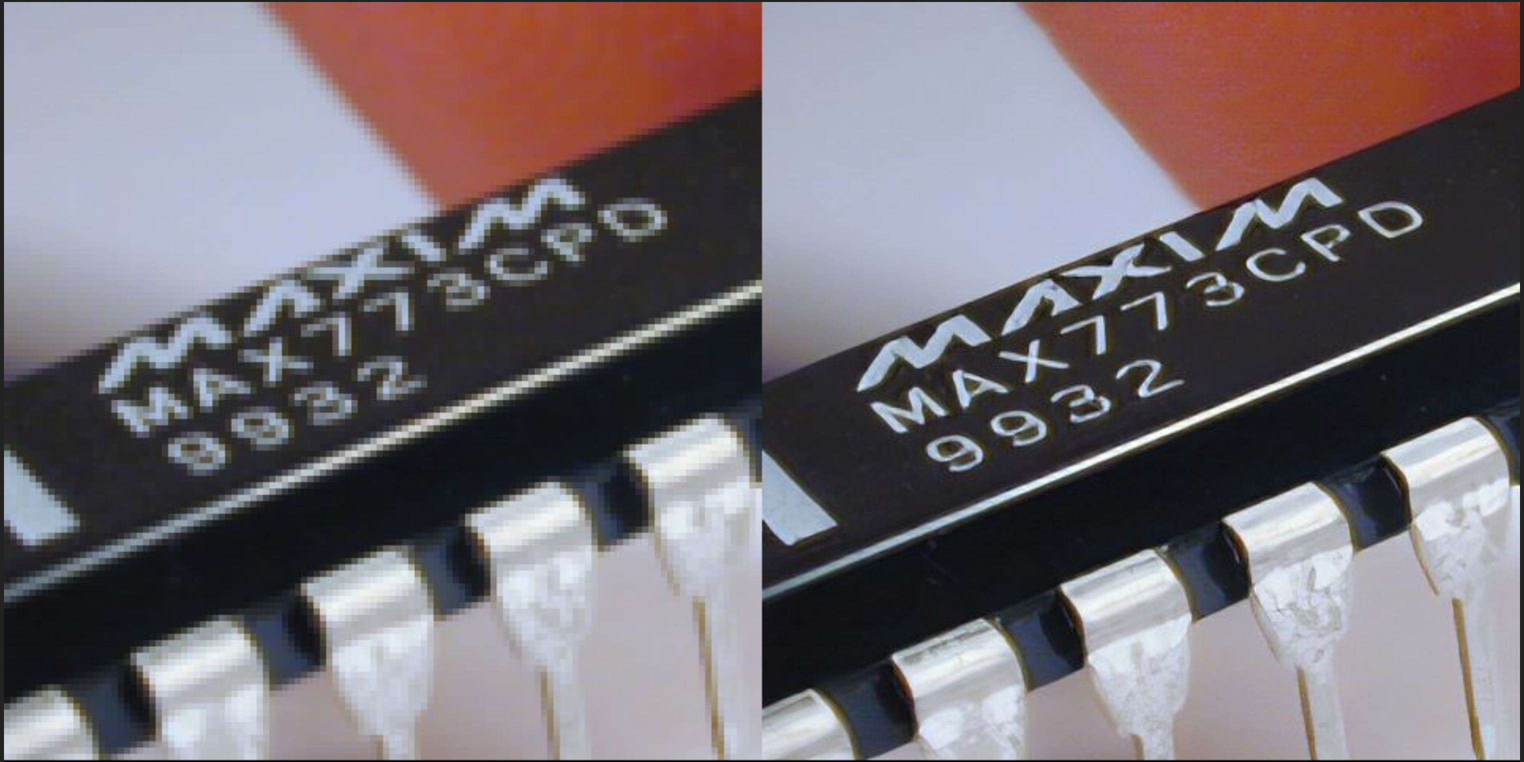

4xNearestNeighbor
4xNomosRealWeb_realplksr_110000iters

Rework Val 2

4xNearestNeighbor

4xNomosRealWeb_realplksr_110000iters

Rework Val 3



4xNearestNeighbor | 4xNomosRealWeb_realplksr_110000iters

Rework Val 4

4xNearestNeighbor

4xNomosRealWeb_realplksr_110000iters

Rework Val 5

4xNearestNeighbor

4xNomosRealWeb_realplksr_110000iters

Rework Val 6


4xNearestNeighbor


4xNomosRealWeb_realplksr_110000iters

Rework Val 7

4xNearestNeighbor

4xNomosRealWeb_realplksr_110000iters

Rework Val 8

4xNearestNeighbor 4xNomosRealWeb_realplksr_110000iters

Rework Val 9

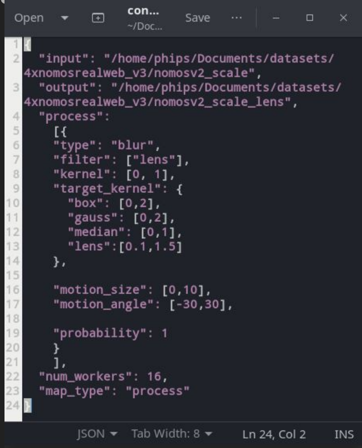
Lens Float Rework

New lens blur

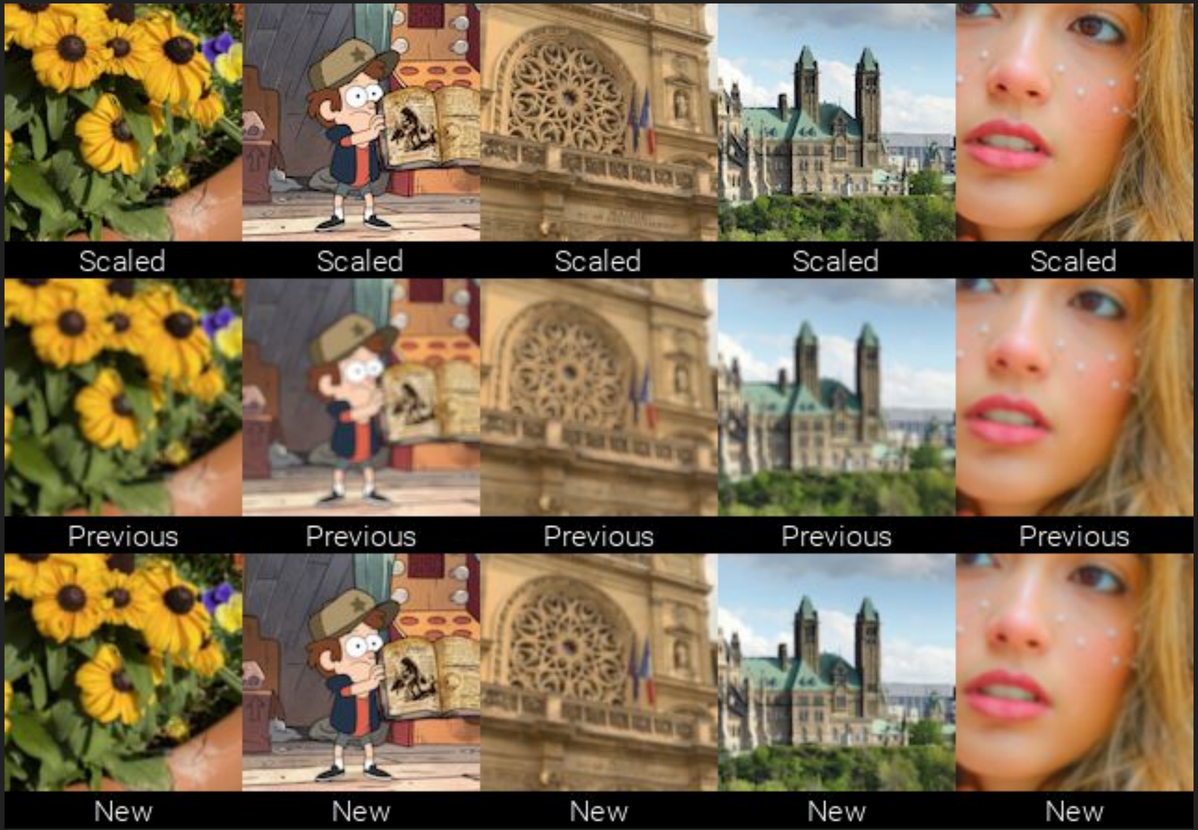
Float implementation by umzi

Way more fine grained control for me. 0.1 steps is barely visible to me. 0.01 steps are not visibly discernible to me, but with an image diff checker i can see that there are slight differences.

I redid the blur parts of the dataset, with 0.1 as min and 1.5 as max after inspecting the lr images.

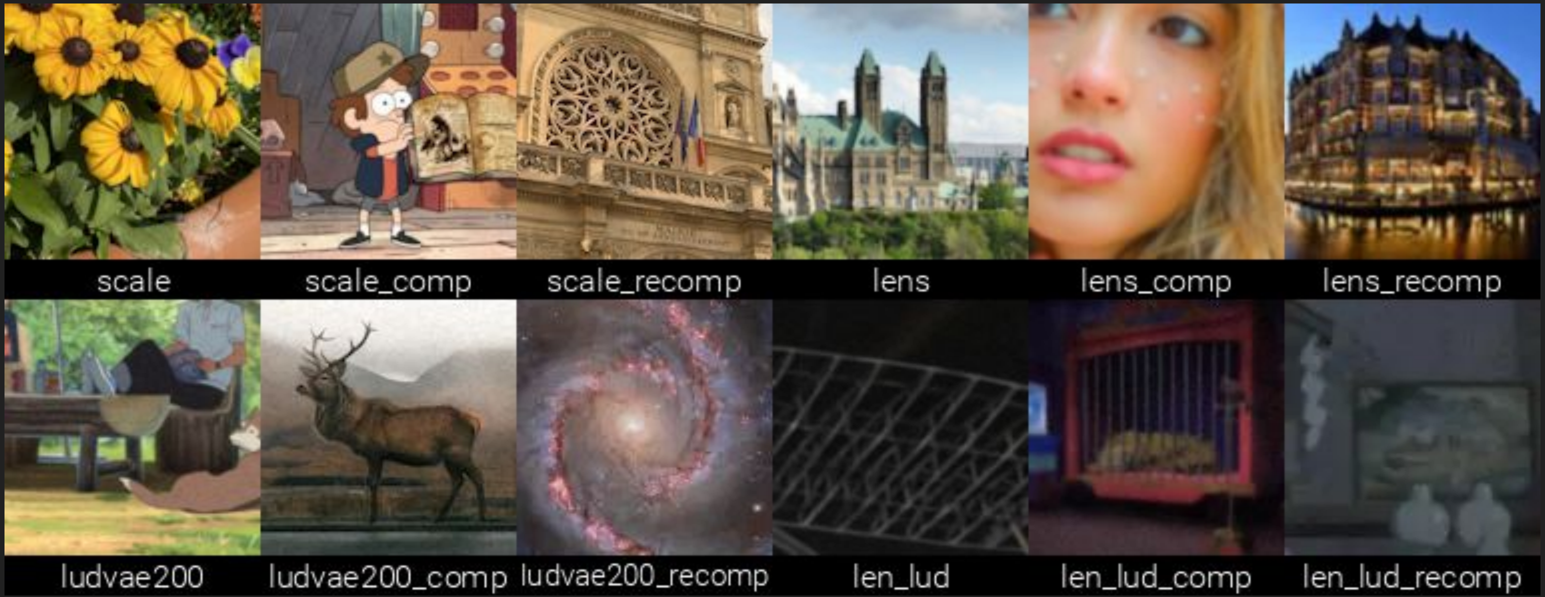


New lens strengths



The lr’s with new lens strengths (meaning randomized manner)

LR’s from the fully degraded folders, first row previous, second row new



Mixed lr1 folder, lens float rework

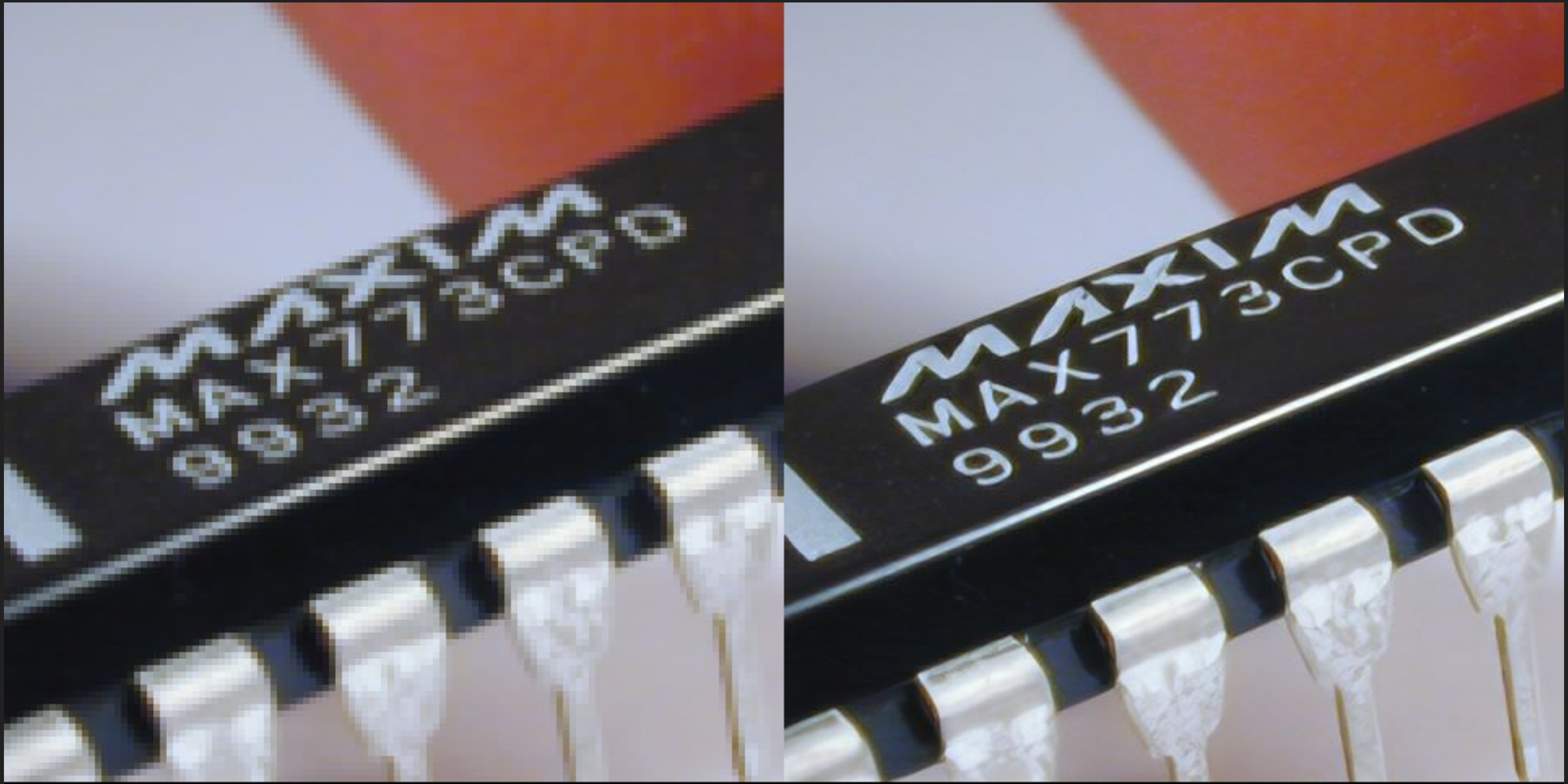
Training

I then again trained a realplksr model.

The outputs looked better to the point that I was satisfied that this version would be release worthy.

So I upped gt size to 256, enabled additional losses like ldl, dists and ff, upped gt size to 512. After testing around I interpolated two of the checkpoints, and released the model as 4xNomosWebPhoto_RealPLKSR on my models github repo.

Following outputs I prepared for the release



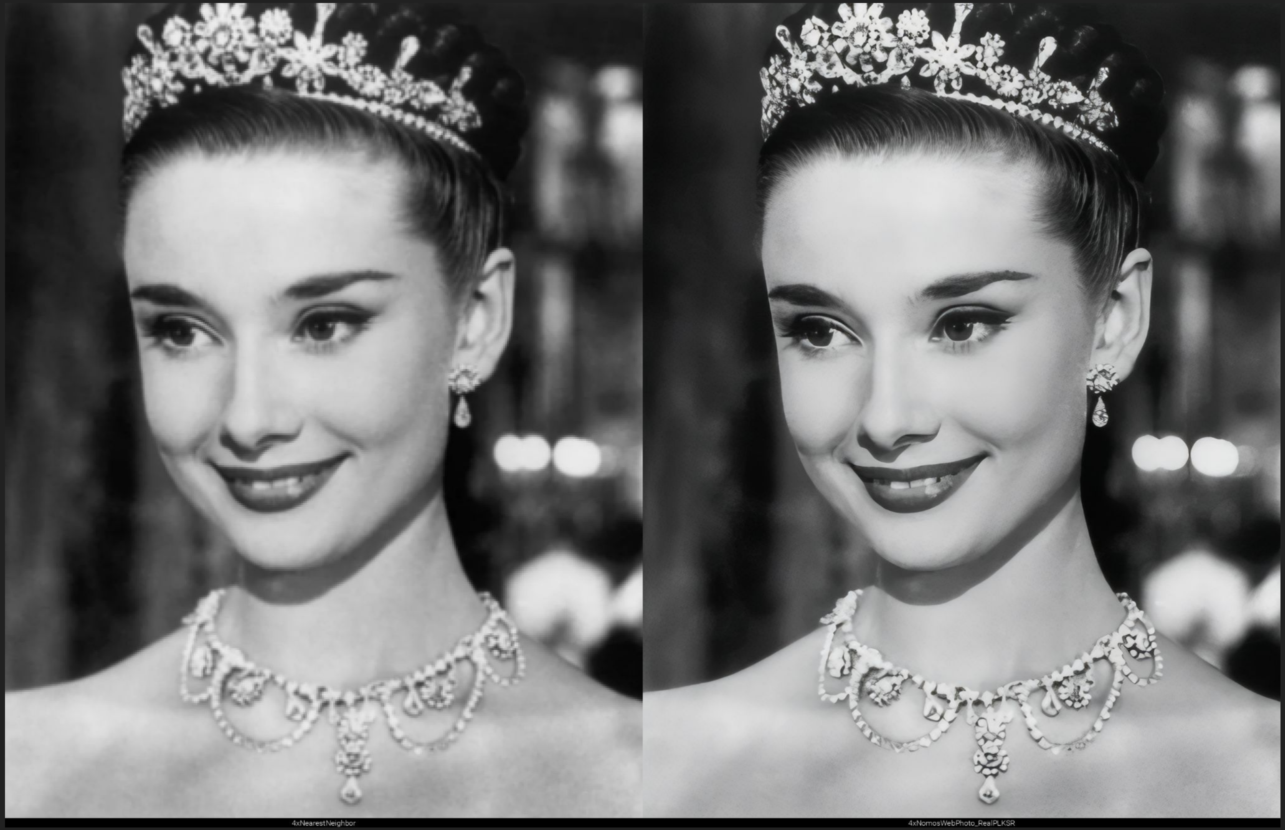
4xNearestNeighbor

4xNomosWebPhoto_RealPLKSR

Example 1

4xNearestNeighbor 4xNomosWebPhoto_RealPLKSR

Example 2



Example 3

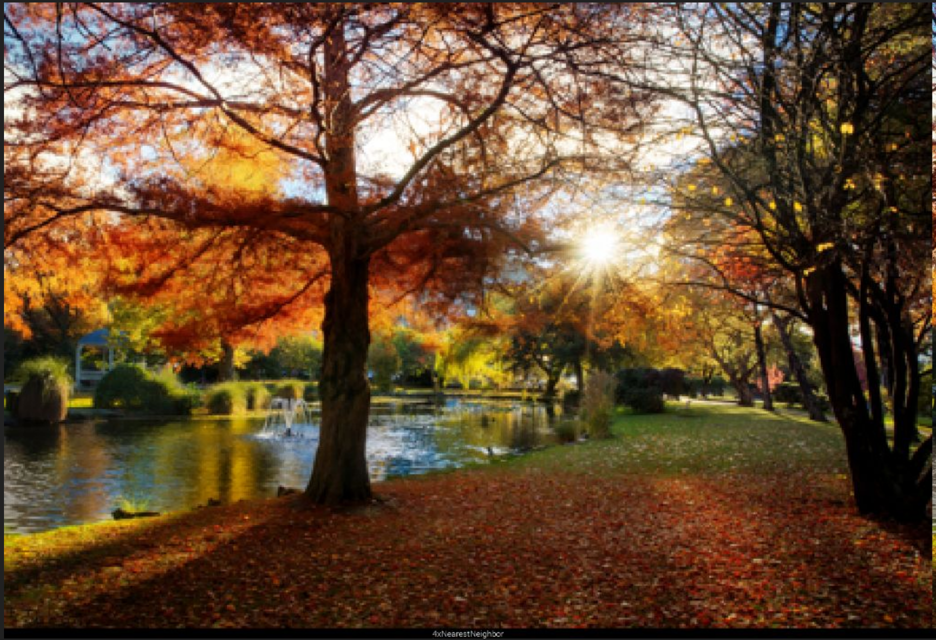
4xNearestNeighbor

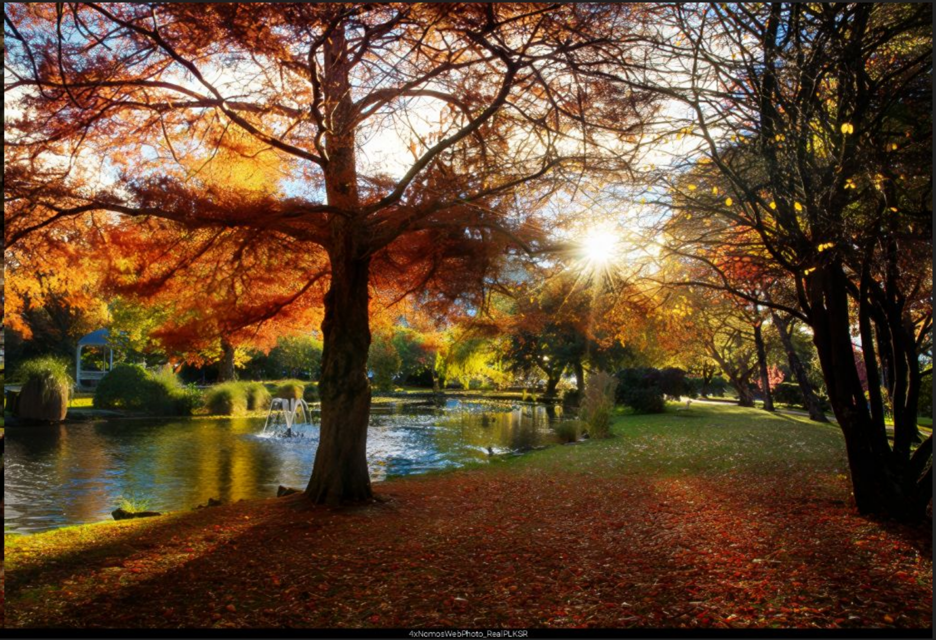
4xNomosWebPhoto_RealPLKSR

Example 4

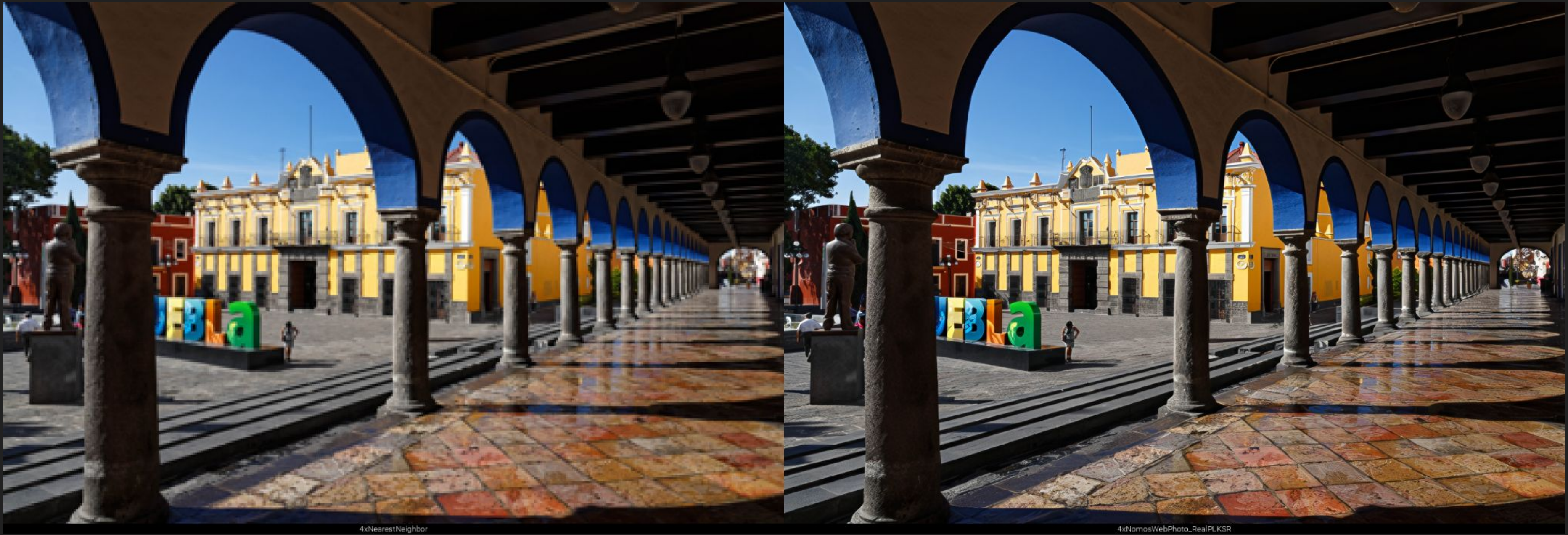
Example 5

Example 6





Example 7



Example 8

Example 9

4xNearestNeighbor | 4xNomosWebPhoto_RealPLKSR

Example 10