Original DLSR code to work for 2x/3x/4x/8x resolution

Jami

GitHub Repo Link: https://github.com/Jamijunky/DL-SR

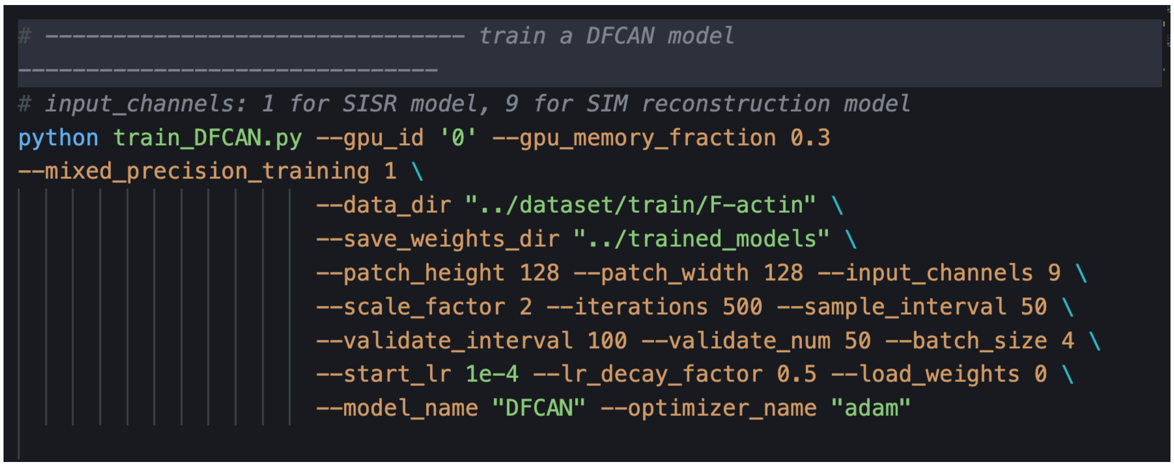
Running BioSR Dataset on MATLAB

I run the BioSR dataset on MATLAB and it gave LR-HR pair dataset and stored them in DLSR/dataset/train folder. I worked on F-actin and since the code is already set for 2x/3x scale so the generated pair is in 2x/3x resolution .

Training the model

Trained the model by using "bash demo_train.sh" while being in src folder but since the code is old it made a lot of different error on my Mac so I had to make a lot of tweaks like changing eras to tensorflow.keras and many more small changes albeit hectic ones.

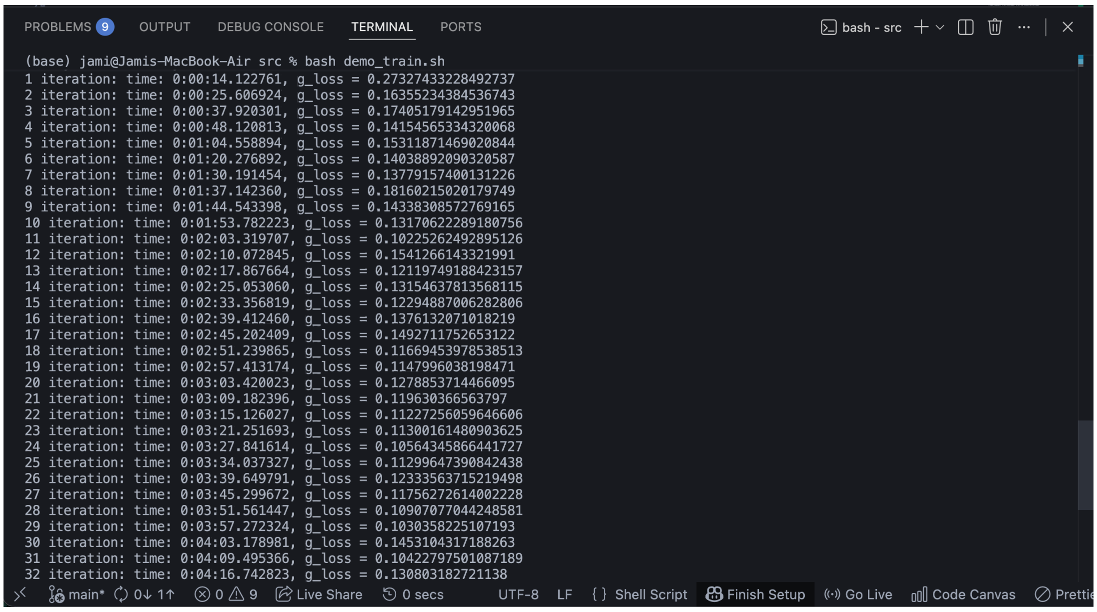
Here is the demo_train.sh:



```
# --------------------------------- train a DFCAN model
--------------------------------
# input_channels: 1 for SISR model, 9 for SIM reconstruction model
python train_DFCAN.py --gpu_id '0' --gpu_memory_fraction 0.3
--mixed_precision_training 1 \
                      --data_dir "../dataset/train/F-actin" \
                      --save_weights_dir "../trained_models" \
                      --patch_height 128 --patch_width 128 --input_channels 9 \
                      --scale_factor 2 --iterations 500 --sample_interval 50 \
                      --validate_interval 100 --validate_num 50 --batch_size 4 \
                      --start_lr 1e-4 --lr_decay_factor 0.5 --load_weights 0 \
                      --model_name "DFCAN" --optimizer_name "adam"
```

Changed the iterations to 500, sample interval to 50, validate interval to 100 so that we get weights and validate numbers to 50. Just changed them arbitrarily on my own and as you can see input channel is 9 so we are training for SIM reconstruction ie information recovery and not hallucinations (in case of WF).

Here is the first few iterations:



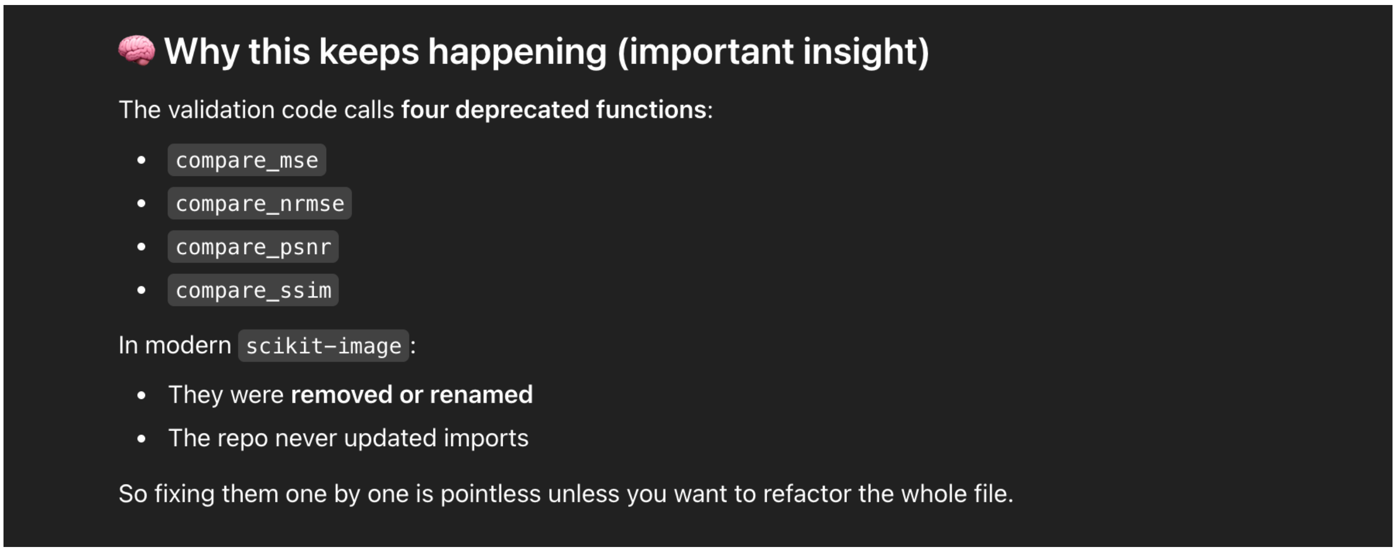
```
(base) jami@Jamis-MacBook-Air src % bash demo_train.sh
1 iteration: time: 0:00:14.122761, g_loss = 0.27327433228492737
2 iteration: time: 0:00:25.606924, g_loss = 0.16355234384536743
3 iteration: time: 0:00:37.920301, g_loss = 0.17405179142951965
4 iteration: time: 0:00:48.120813, g_loss = 0.14154565334320068
5 iteration: time: 0:01:04.558894, g_loss = 0.15311871469020844
6 iteration: time: 0:01:20.276892, g_loss = 0.14038892090320587
7 iteration: time: 0:01:30.191454, g_loss = 0.13779157400131226
8 iteration: time: 0:01:37.142360, g_loss = 0.18160215020179749
9 iteration: time: 0:01:44.543398, g_loss = 0.14338308572769165
10 iteration: time: 0:01:53.782223, g_loss = 0.13170622289180756
11 iteration: time: 0:02:03.319707, g_loss = 0.10225262492895126
12 iteration: time: 0:02:10.072845, g_loss = 0.1541266143321991
13 iteration: time: 0:02:17.867664, g_loss = 0.12119749188423157
14 iteration: time: 0:02:25.053060, g_loss = 0.13154637813568115
15 iteration: time: 0:02:33.356819, g_loss = 0.12294887006282806
16 iteration: time: 0:02:39.412460, g_loss = 0.1376132071018219
17 iteration: time: 0:02:45.202409, g_loss = 0.1492711752653122
18 iteration: time: 0:02:51.239865, g_loss = 0.11669453978538513
19 iteration: time: 0:02:57.413174, g_loss = 0.1147996038198471
20 iteration: time: 0:03:03.420023, g_loss = 0.1278853714466095
21 iteration: time: 0:03:09.182396, g_loss = 0.119630366563797
22 iteration: time: 0:03:15.126027, g_loss = 0.11227256059646606
23 iteration: time: 0:03:21.251693, g_loss = 0.11300161480903625
24 iteration: time: 0:03:27.841614, g_loss = 0.10564345866441727
25 iteration: time: 0:03:34.037327, g_loss = 0.11299647390842438
26 iteration: time: 0:03:39.649791, g_loss = 0.12333563715219498
27 iteration: time: 0:03:45.299672, g_loss = 0.11756272614002228
28 iteration: time: 0:03:51.561447, g_loss = 0.10907077044248581
29 iteration: time: 0:03:57.272324, g_loss = 0.1030358225107193
30 iteration: time: 0:04:03.178981, g_loss = 0.1453104317188263
31 iteration: time: 0:04:09.495366, g_loss = 0.10422797501087189
32 iteration: time: 0:04:16.742823, g_loss = 0.13080318272113
```

main* 0↓ 1↑ 0 9 Live Share 0 secs UTF-8 LF { } Shell Script Finish Setup Go Live Code Canvas Prettie

Had a error where I missed “**from skimage.metrics**”, “**import peak_signal_noise_ratio as compare_psnr**”, “**from skimage.metrics import structural_similarity as compare_ssim” in utils/utils.py.**

The error was shown after 50 iterations because sample_interval is set to 50 😅.

It happened again and chatGPT said this so.......aaahhhhhhhh



🧠 Why this keeps happening (important insight)

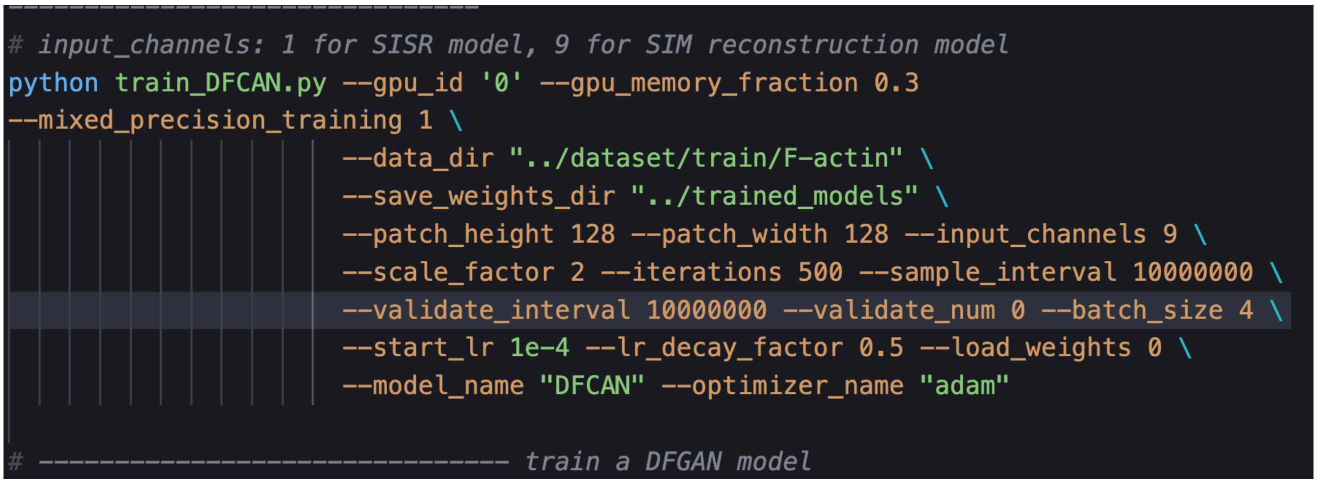
The validation code calls **four deprecated functions**:

- `compare_mse`
- `compare_nrmse`
- `compare_psnr`
- `compare_ssim`

In modern `scikit-image`:

- They were **removed or renamed**
- The repo never updated imports

So fixing them one by one is pointless unless you want to refactor the whole file.



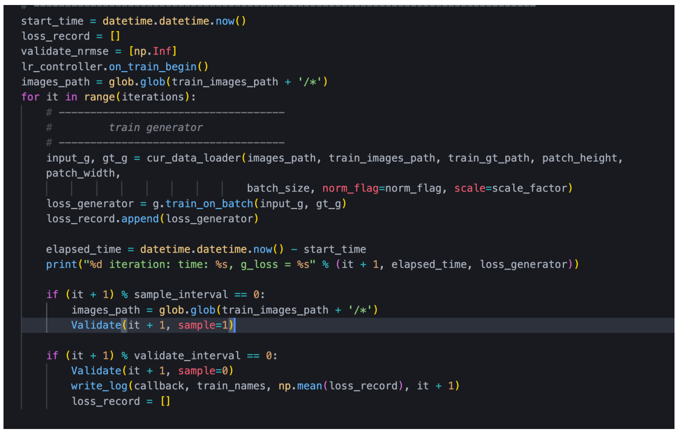
```
# input_channels: 1 for SISR model, 9 for SIM reconstruction model
python train_DFCAN.py --gpu_id '0' --gpu_memory_fraction 0.3
--mixed_precision_training 1 \
                    --data_dir "../dataset/train/F-actin" \
                    --save_weights_dir "../trained_models" \
                    --patch_height 128 --patch_width 128 --input_channels 9 \
                    --scale_factor 2 --iterations 500 --sample_interval 10000000 \
                    --validate_interval 10000000 --validate_num 0 --batch_size 4 \
                    --start_lr 1e-4 --lr_decay_factor 0.5 --load_weights 0 \
                    --model_name "DFCAN" --optimizer_name "adam"

# ------------------------------ train a DFGAN model
```

I changed sample interval to 10^7 ,validate interval as well to 10^7 and validate_num to 0. The model completed the training but weights where no where to be found so to save weights I ran again by changing iterations to 50 and sample_interval to 10.

Still nothing so we did add few blocks of codes which are:

From:



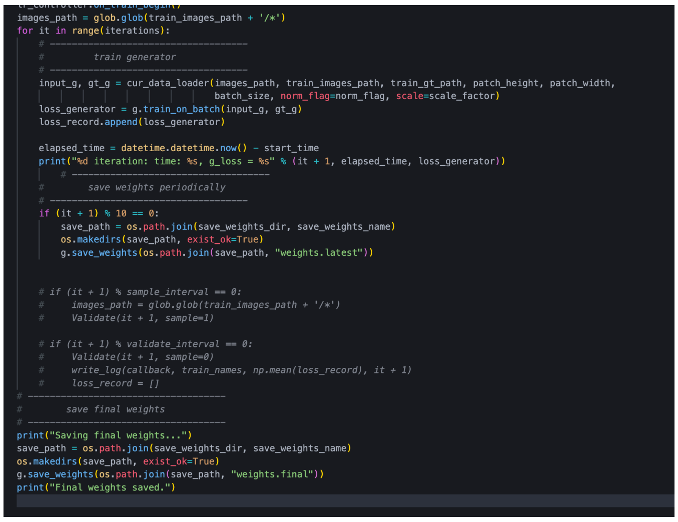
```
start_time = datetime.datetime.now()
loss_record = []
validate_nrmse = [np.Inf]
lr_controller.on_train_begin()
images_path = glob.glob(train_images_path + '/*')
for it in range(iterations):
    # ------------------------------------
    #          train generator
    # ------------------------------------
    input_g, gt_g = cur_data_loader(images_path, train_images_path, train_gt_path, patch_height,
    patch_width,
                                    batch_size, norm_flag=norm_flag, scale=scale_factor)
    loss_generator = g.train_on_batch(input_g, gt_g)
    loss_record.append(loss_generator)

    elapsed_time = datetime.datetime.now() - start_time
    print("%d iteration: time: %s, g_loss = %s" % (it + 1, elapsed_time, loss_generator))

    if (it + 1) % sample_interval == 0:
        images_path = glob.glob(train_images_path + '/*')
        Validate(it + 1, sample=1)

    if (it + 1) % validate_interval == 0:
        Validate(it + 1, sample=0)
        write_log(callback, train_names, np.mean(loss_record), it + 1)
        loss_record = []
```

To:



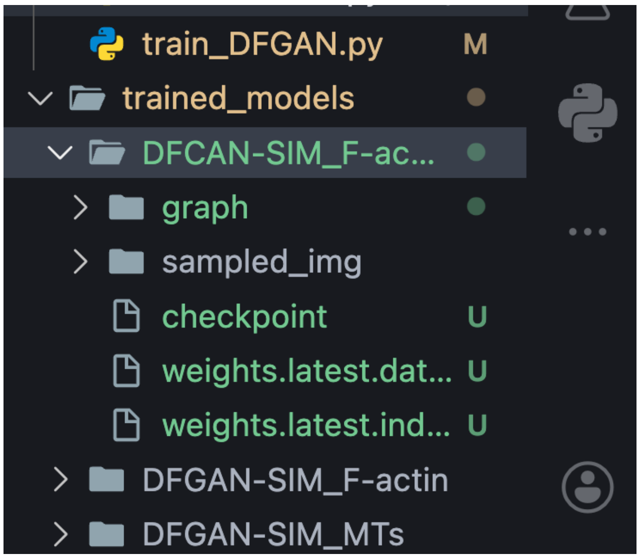
```
images_path = glob.glob(train_images_path + '/*')
for it in range(iterations):
    # ------------------------------------
    #          train generator
    # ------------------------------------
    input_g, gt_g = cur_data_loader(images_path, train_images_path, train_gt_path, patch_height, patch_width,
                                    batch_size, norm_flag=norm_flag, scale=scale_factor)
    loss_generator = g.train_on_batch(input_g, gt_g)
    loss_record.append(loss_generator)

    elapsed_time = datetime.datetime.now() - start_time
    print("%d iteration: time: %s, g_loss = %s" % (it + 1, elapsed_time, loss_generator))
        # ------------------------------------
    #         save weights periodically
    # ------------------------------------
    if (it + 1) % 10 == 0:
        save_path = os.path.join(save_weights_dir, save_weights_name)
        os.makedirs(save_path, exist_ok=True)
        g.save_weights(os.path.join(save_path, "weights.latest"))


    # if (it + 1) % sample_interval == 0:
    #     images_path = glob.glob(train_images_path + '/*')
    #     Validate(it + 1, sample=1)

    # if (it + 1) % validate_interval == 0:
    #     Validate(it + 1, sample=0)
    #     write_log(callback, train_names, np.mean(loss_record), it + 1)
    #     loss_record = []
# ------------------------------------
#         save final weights
# ------------------------------------
print("Saving final weights...")
save_path = os.path.join(save_weights_dir, save_weights_name)
os.makedirs(save_path, exist_ok=True)
g.save_weights(os.path.join(save_path, "weights.final"))
print("Final weights saved.")
```

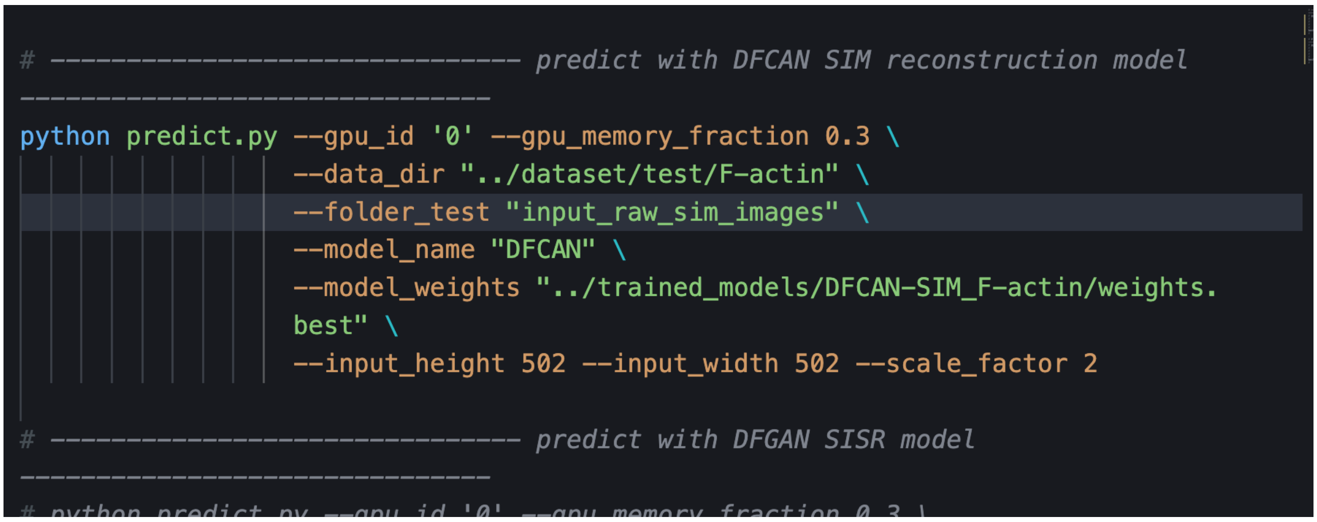
Folder Structure:



```
    train_DFGAN.py            M
  trained_models
    DFCAN-SIM_F-ac...
      graph
      sampled_img
      checkpoint              U
      weights.latest.dat...   U
      weights.latest.ind...   U
    DFGAN-SIM_F-actin
    DFGAN-SIM_MTs
```

Anyways, after training the model we have weights and graph stored in trained_models folder. Now I run “bash demo_predict.sh”

<u>This is demo_predict.sh:</u>



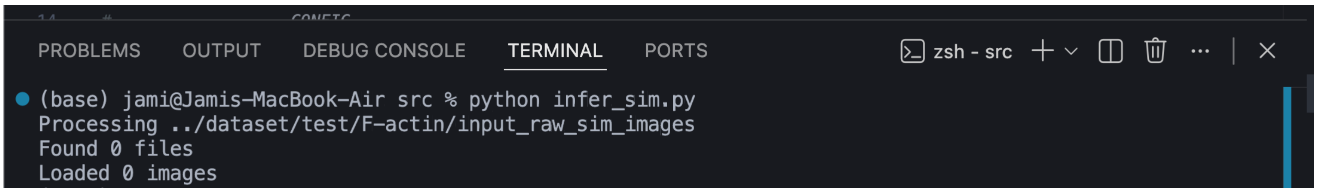
```
# ------------------------------- predict with DFCAN SIM reconstruction model
-------------------------------
python predict.py --gpu_id '0' --gpu_memory_fraction 0.3 \
                  --data_dir "../dataset/test/F-actin" \
                  --folder_test "input_raw_sim_images" \
                  --model_name "DFCAN" \
                  --model_weights "../trained_models/DFCAN-SIM_F-actin/weights.
                  best" \
                  --input_height 502 --input_width 502 --scale_factor 2

# ------------------------------- predict with DFGAN SISR model
-------------------------------
# python predict.py --gpu_id '0' --gpu_memory_fraction 0.3 \
```

Commented out rest of the snippets ie SISR DFCAN and SISR & SIM DFGAN.

Note: changed weights.best to weights.latest

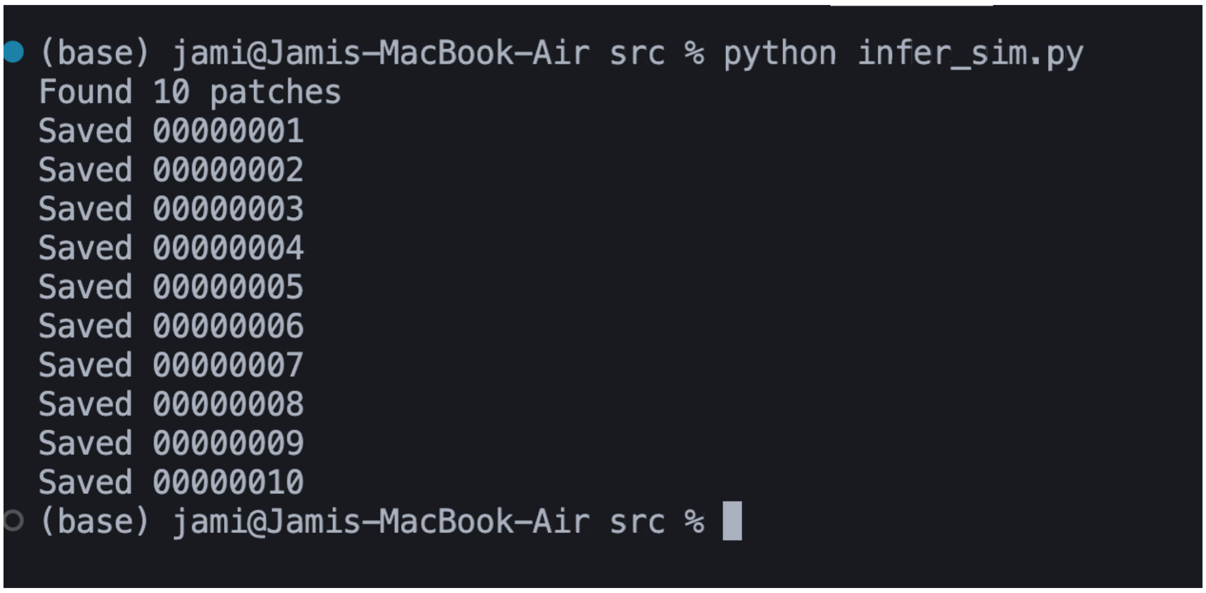
We did have one more problem which is that our python code was not saving images ie dataset/test folder was empty so I added a file named "infer_sim.py" to save the images but still we were not able to get our images.



```
(base) jami@Jamis-MacBook-Air src % python infer_sim.py
Processing ../dataset/test/F-actin/input_raw_sim_images
Found 0 files
Loaded 0 images
```

So we added some of our validation images in train/validation folder to test/F-actin/input/raw_sim_images folder

Still we were able to find and load images but not save them so in our "infer_sim.py" file we made major changes basically everything and.............Eureka!!!!

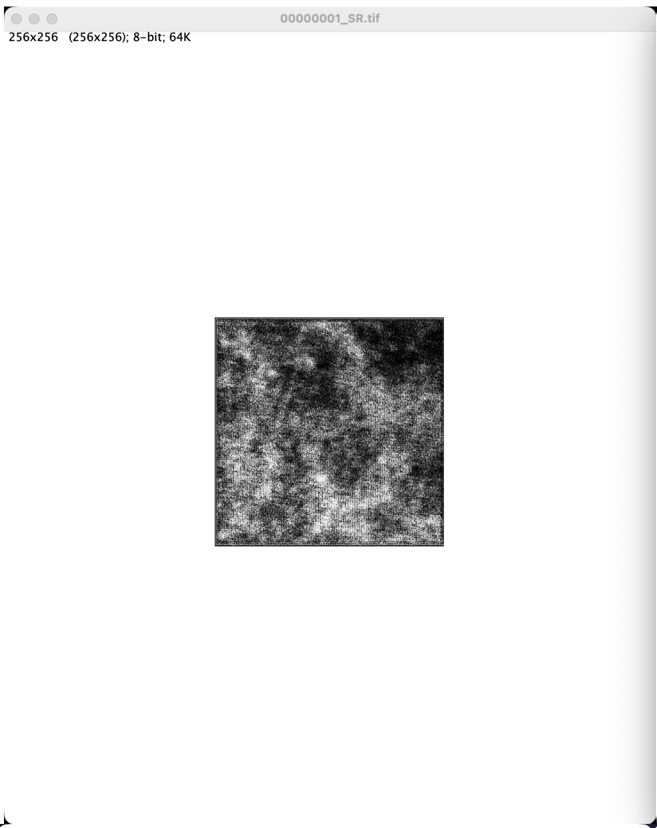


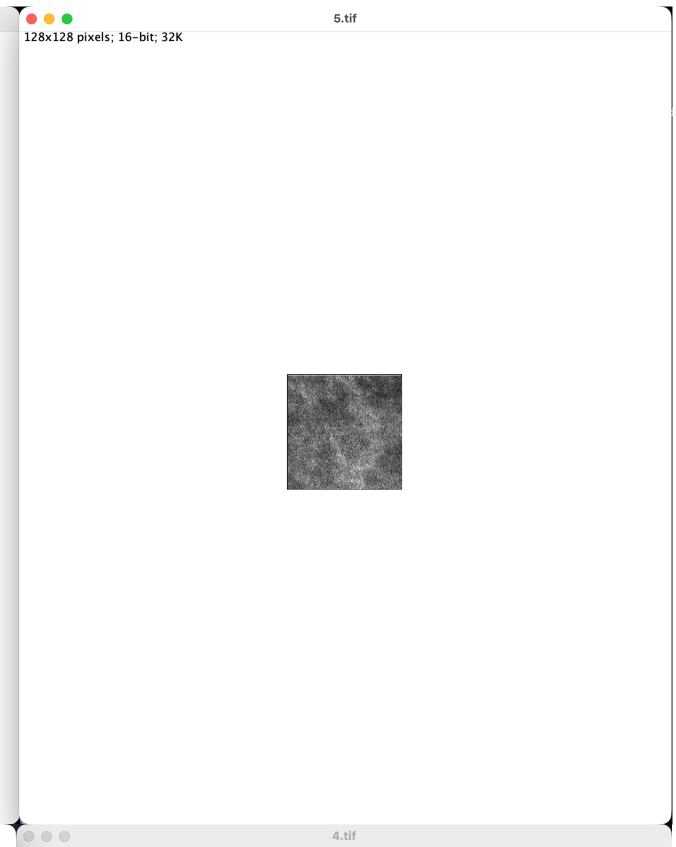
```
(base) jami@Jamis-MacBook-Air src % python infer_sim.py
Found 10 patches
Saved 00000001
Saved 00000002
Saved 00000003
Saved 00000004
Saved 00000005
Saved 00000006
Saved 00000007
Saved 00000008
Saved 00000009
Saved 00000010
(base) jami@Jamis-MacBook-Air src %
```

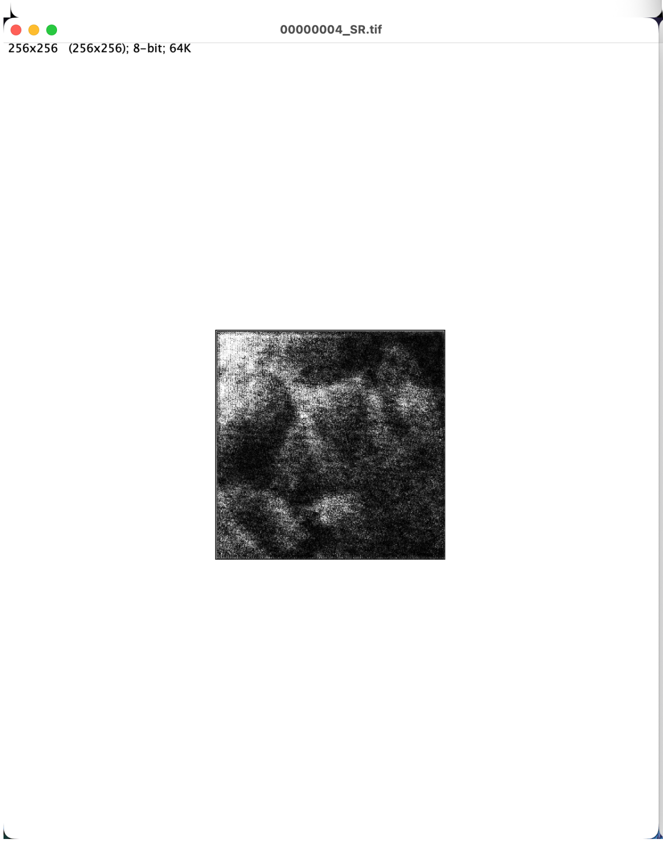
We generated 2x scaled images ie (128 x 128) -> (256 x 256). We have 10 images stored in dataset/test/F-actin/SR-results

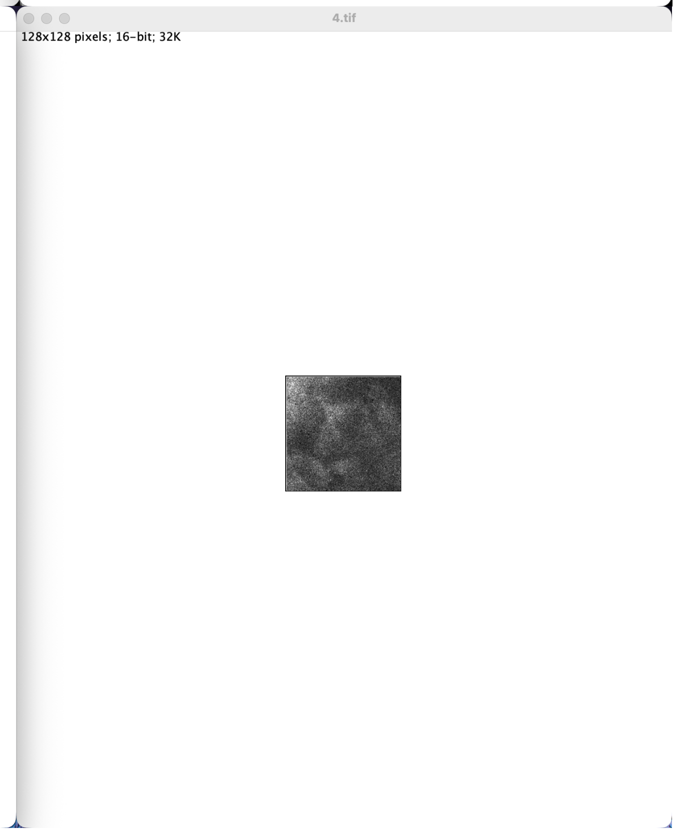
Output

Input


00000001_SR.tif
256x256 (256x256); 8-bit; 64K


5.tif
128x128 pixels; 16-bit; 32K


00000004_SR.tif
256x256 (256x256); 8-bit; 64K


4.tif
128x128 pixels; 16-bit; 32K

Now lets do it for 4x/8x

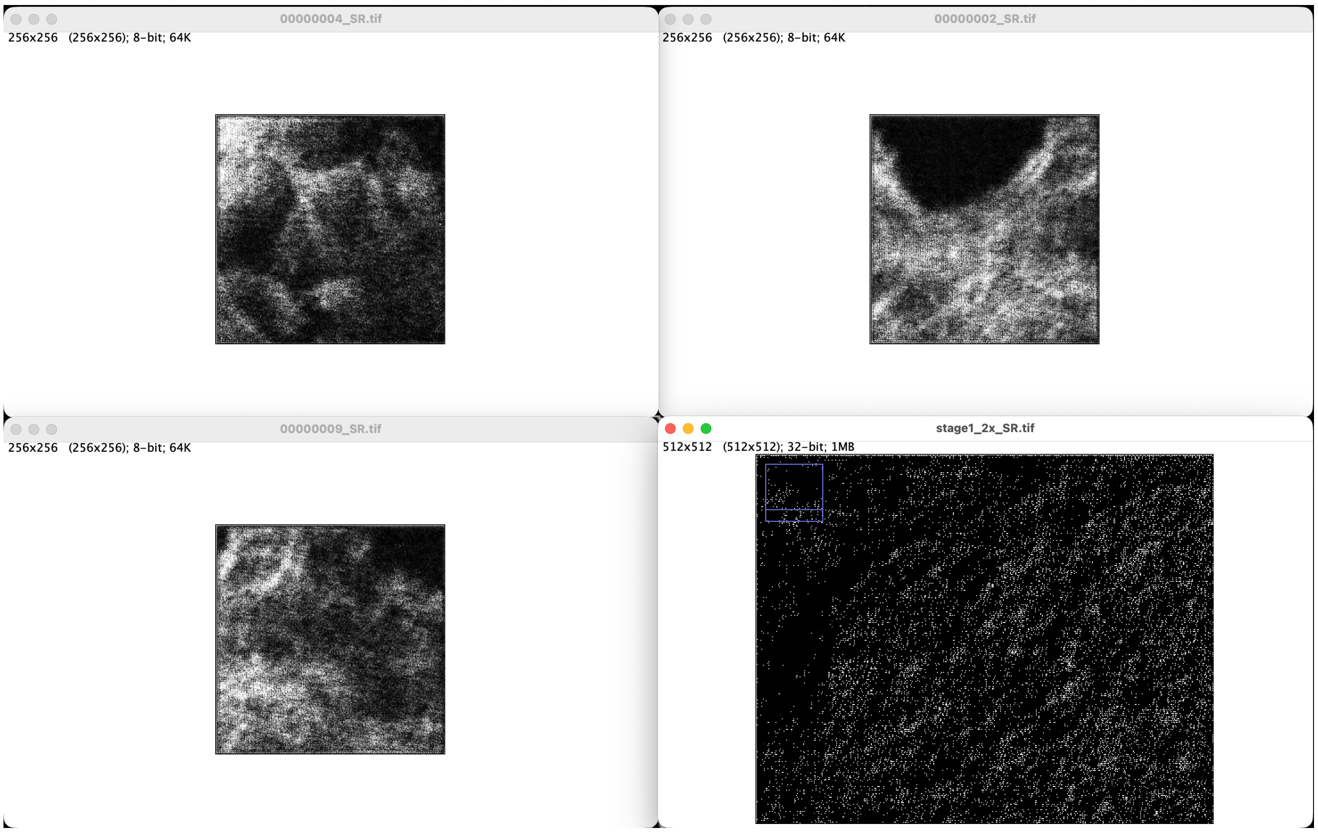
We can't directly change scale_factor to 4x or 8x without changing MATLAB generation.Why?? Because the model weights were trained to hallucinate at 2x frequency bands and at 4x it invents structures it was never constrained against. Technically model failure.

So, there are two ways either having physical consistency or using synthetic scaling, albeit it's hectic I am going with hybrid method because I want to see how it works in either way.

STAGE 1: Recursive Inference (2x → 4x → 8x)

So our predict.py folder was messed up and I had a lot of problems so I chatted with chatGPT for hours and he came to a conclusion that we can't do it this way but me being determined looked at the predict.py file again and made few changes. Most important being changing channel from 9 to 1 and yes it did produce a single 4x image from our 10 SR results ie 2x resolution ones.

Here are they:


00000004_SR.tif
256x256 (256x256); 8-bit; 64K
00000002_SR.tif
256x256 (256x256); 8-bit; 64K
00000009_SR.tif
256x256 (256x256); 8-bit; 64K
stage1_2x_SR.tif
512x512 (512x512); 32-bit; 1MB