Progressive GAN-aided High-Resolution Microscopy for Live Cell Dynamics Study

Tianyao Wu, Tianchi Liu
`{twu31, kitliu5}@stanford.edu`

Introduction

One of the interesting application of Generative Adversarial Networks (GANs) is image super-resolution. It could be a useful technique to enhance various type of microscopy images, thereby providing additional scientific value in different research fields. Currently there exist several super-resolution microscopy techniques to study small cellular structures such as structured illumination microscopy (SIM). However, they often require slow frame rate, unhealthy environment for live cells and complicated optical setup.

In this work, we attempt to utilize GAN to enhance resolution of fluorescence microscopy images, which could aid the study of the dynamics of live cells. In particular, we will use progressive GANs [1] because both efficiency and stability are important to the large-size noisy microscopy images.

Due to the long training time of the progressive GAN model, and the difficulty to evaluate our results due to lack of references, we planned to first implement a traditional non-GAN approach using U-Net and set its results as the baseline of this super-resolution task. Then we attempt to implement the progressive growing of GAN layers outlined in *Karras et al.*

Although progressive GANs architecture succeeded on tasks like face-recognition, microscopy images differ from ordinary images greatly, thereby requiring algorithmic modification for it to work. In general we want to exploit the fact that a large portion of microscopy image is background and extra emphasis on the high resolution layers is needed, because we need the high resolution sharp enough for us to study things like curvature of small cellular structures.

Problem Statement

We used the imaging system in Kural lab at the Ohio State University to obtain fluorescence microscopy images. The imaging system consists of an Eclipse TI-E microscope (Nikon) equipped with a temperature controlled chamber, a CSU-W1 spinning disk confocal unit (Yokogawa Electric Corporation), a 100 objective lens (Nikon CFI Plan-Apochromat Lambda, NA 1.45) and an EMCCD camera (iXon DU897 Ultra, Andor Technology). We acquired 3 types of fluorescence labels, including RedNile Beads, AP2 labelled in SUM159 cells, and single molecules. Each movie was taken in a way that two exposure times were used and a low exposure time acquisition is followed by a high exposure time acquisition, and vise versa. Using this method, even though the fluorescence labels in the movie are constantly changing, the adjacent pairs can be considered as matched image pairs. An example of these pairs is shown in Fig.1.

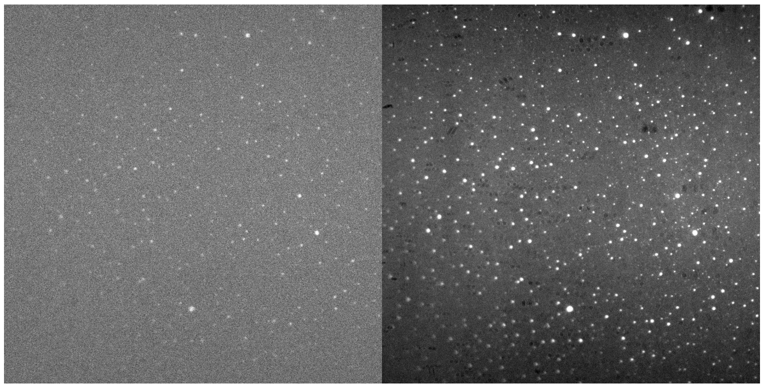


Figure 1: Sample image taken at low exposure (left) and high exposure (right)

We expect reasonable results from the U-Net. As the most used network in biomedical data, it should give a good result on a macroscopic scale. That is, the generated images should be indistinguishable by human without adjusting brightness and contrast or quantitative analysis. However, if we do look closely, we expect U-Net predicted images to lack detailed structure of each label and perform poorly on signal-to-noise ratio. This is when GAN steps in. Fluorescence microscopy images suffer from the fact that it requires high resolution to maintain its conceptual identity. We hope that with progressively growing of GAN layers, it can achieve the sharpness we need to identify details in each label on a high scale of resolution.

Evaluation could be tricky because there are no similar published study, which is why we decided to implement U-Net as a baseline for GAN to compare to. We will have to rely heavily on the traditional analysis in individual fluorescence labels. Each fluorescence point will be hand-labeled from a group of randomly selected images and the accuracy and signal-to-noise ratio of these labels will be measured. Their point spread function (PSF) will be studied in the predicted images from GAN.

Technical Approach

In need of a standard for our final GAN model to compare to, we used U-Net, the dominant network used in the field of biomedical images, as a baseline. The architecture is shown in Fig.7. Pairs of low exposure (LP) and high exposure (HP) images are cropped from 1200x1200 original size to 256x256 size and are then fed into the model. A total of 3200 pairs of single molecule images acquired in 4 different sessions are used as training data and 400 pairs of images in 2 other different sessions are used as test data. Since this is not our primary focus and no extra modification is applied, pseudocode is omitted and loss function will be briefly covered in the next section.

For the first iteration of implementing progressive GAN, we followed *Karras et al*[1] closely by incrementally adding layers to generator and discriminator pairs that are mirror images of each other, so that finer details of image distribution could be gradually learned. The low exposure images are fed as input while the high exposure images are treated as real images in the network. The layers added will be smoothly faded with residual-block nature to avoid sudden shock to existing layers. This procedure is illustrated by the simplified model architecture in Fig.3 . The detailed architecture is shown in Fig. 8 . For better stabilization, the Wasserstein GAN with gradient penalty (WGAN-GP) shown in Fig.2 is used as the loss function. Nearest neighbor filtering and average pooling are used to double and halve resolution of the image, respectively. We also included additional techniques used by *Karras et al*[1] such as minibatch standard deviation and equalized learning rate. We did not implement pixel normalization as it is not applicable to fluorescence microscopy images. We expect style-transfer type of trained images using this first iteration of our model. Next, we will combine pixel-wise mae loss with the currently used WGAN-GP loss into a joint loss similar to the methodology used in SRGAN [2] which can be formulated as shown below, with l_{MAE} as content loss and λ as weight factor.

$$l_{Joint} = l_{MAE} + \lambda \cdot l_{GAN}$$

$$L = \underbrace{\underset{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g}{\mathbb{E}} [D(\tilde{\boldsymbol{x}})] - \underset{\boldsymbol{x} \sim \mathbb{P}_r}{\mathbb{E}} [D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \underset{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}}{\mathbb{E}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right]}_{\text{Our gradient penalty}} .$$

Figure 2: WGAN-GP loss

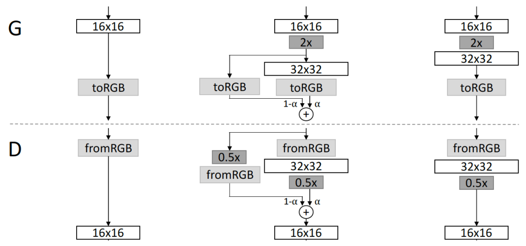


Figure 3: Progressive GAN Architecture

Preliminary Results

Overall, U-Net provides reasonable images with good accuracy of label identification. However, it lacks the precision needed in order to further study the details in each label and has quite high background fluctuation. Initially, we used binary crossentropy as our loss function, consistent with the original U-Net. Results show largely widened point spread function (PSF). Since we will want to incorporate mae loss into our GAN model and we encountered this PSF widening problem, we tried using mae as loss. This ended up being more challenging than we thought it to be. The training process became very unstable after we only changed its loss function from binary crossentropy to mae. We solved this problem by adding a random integer from 0 to 200 to each pair (each pair gets the same value). Examples of trained samples using 2 different loss functions are shown in Fig.4 and Fig.5.

We implemented the same strategy of adding random constant to pairs and used the same training dataset in our progressive growing GAN. Fig.6 shows some of the early stage results. The training process is taking much more computation power than expected. The sample trained images, while unsatisfying, are still improving slowly as of now. Moving forward, considering our current training taking too long, we plan to continue the training on our 2 local gpus, while testing some simplified versions of progressively growing GAN with emphasis on the last few growing layers using cloud services, and start the incorporation of content mae loss into our network in parallel.

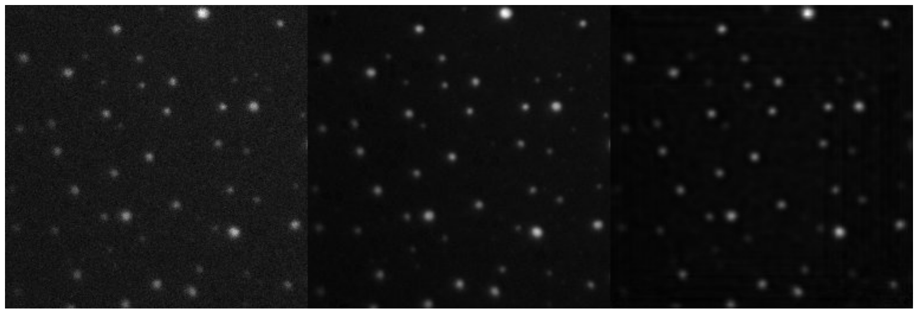


Figure 4: U-Net training using binary crossentropy loss. From left to right: LP, HP, predicted

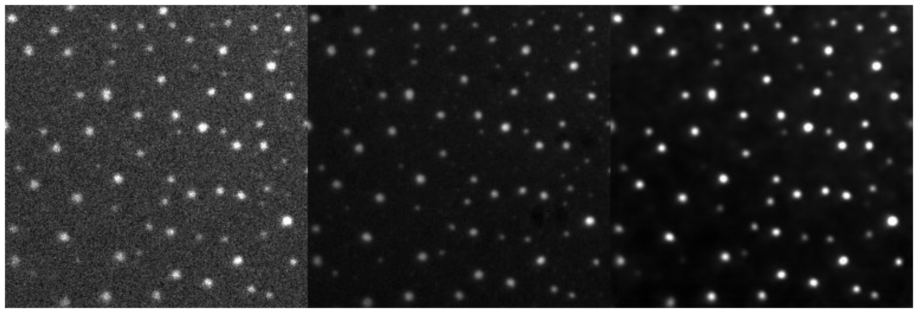


Figure 5: U-Net training using mae loss. From left to right: LP, HP, predicted

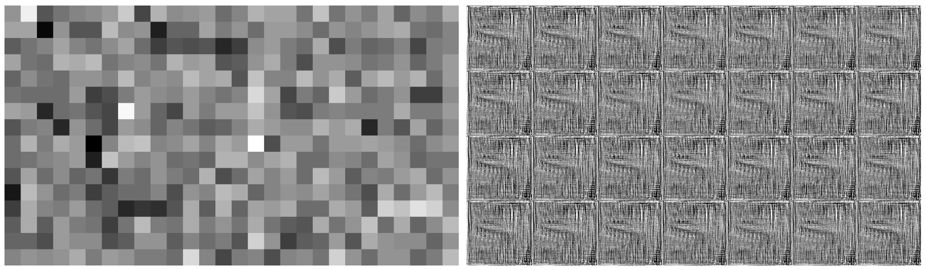


Figure 6: 4x7 grid of predicted sample images. Left: early stages of training with size 4x4; Right: 2 days of training with size 128x128

References

[1] Tero Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. In: *arXiv e-prints*, arXiv:1710.10196 (Oct. 2017), arXiv:1710.10196. arXiv: `1710.10196 [cs.NE]`.

[2] Christian Ledig et al. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”. In: *CoRR* abs/1609.04802 (2016). arXiv: `1609.04802`. URL: `http://arxiv.org/abs/1609.04802`.

Appendix - Additional Figures

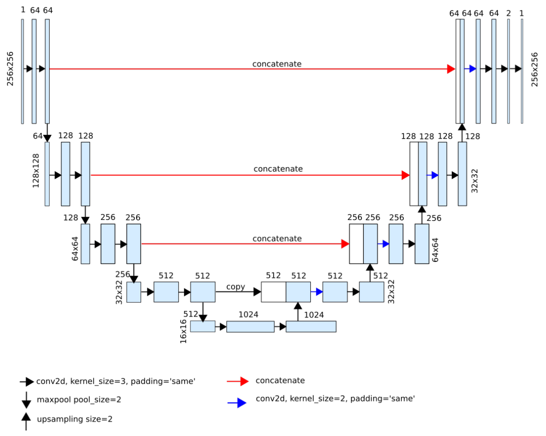
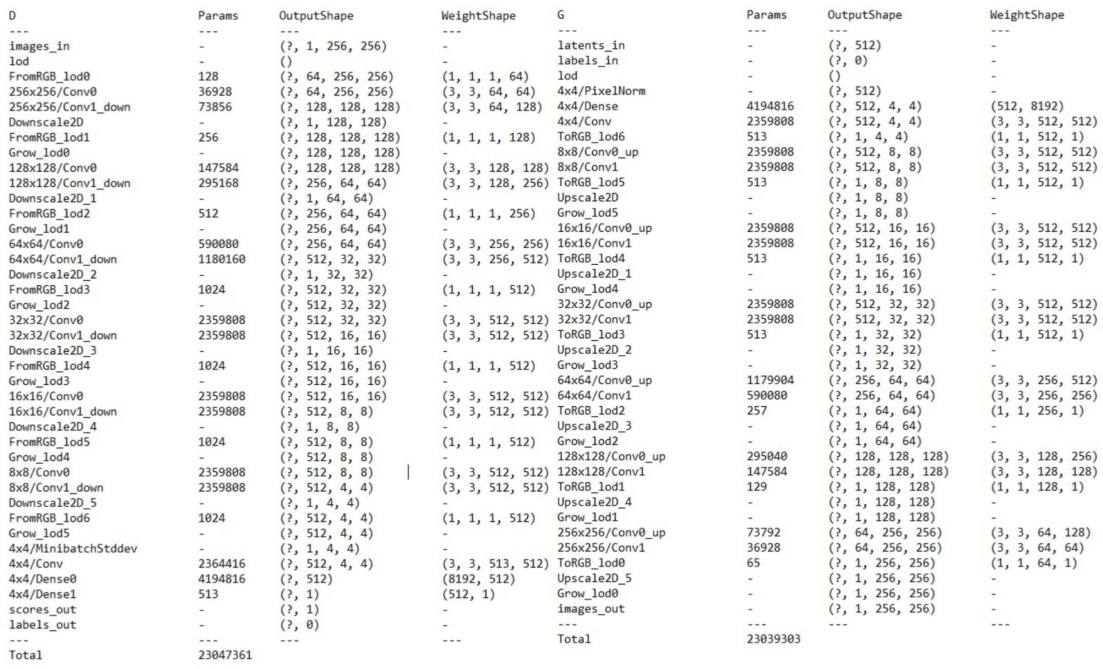


Figure 7: U-Net Architecture



```
D                     Params    OutputShape          WeightShape      G                   Params    OutputShape          WeightShape
---                   ---       ---                  ---              ---                 ---       ---                  ---
images_in             -         (?, 1, 256, 256)     -                latents_in          -         (?, 512)             -
lod                   -         ()                   -                labels_in           -         (?, 0)               -
FromRGB_lod0          128       (?, 64, 256, 256)    (1, 1, 1, 64)    lod                 -         ()                   -
256x256/Conv0         36928     (?, 64, 256, 256)    (3, 3, 64, 64)   4x4/PixelNorm       -         (?, 512)             -
256x256/Conv1_down    73856     (?, 128, 128, 128)   (3, 3, 64, 128)  4x4/Dense           4194816   (?, 512, 4, 4)       (512, 8192)
Downscale2D           -         (?, 1, 128, 128)     -                4x4/Conv            2359808   (?, 512, 4, 4)       (3, 3, 512, 512)
FromRGB_lod1          256       (?, 128, 128, 128)   (1, 1, 1, 128)   ToRGB_lod6          513       (?, 1, 4, 4)         (1, 1, 512, 1)
Grow_lod0             -         (?, 128, 128, 128)   -                8x8/Conv0_up        2359808   (?, 512, 8, 8)       (3, 3, 512, 512)
128x128/Conv0         147584    (?, 128, 128, 128)   (3, 3, 128, 128) 8x8/Conv1           2359808   (?, 512, 8, 8)       (3, 3, 512, 512)
128x128/Conv1_down    295168    (?, 256, 64, 64)     (3, 3, 128, 256) ToRGB_lod5          513       (?, 1, 8, 8)         (1, 1, 512, 1)
Downscale2D_1         -         (?, 1, 64, 64)       -                Upscale2D           -         (?, 1, 8, 8)         -
FromRGB_lod2          512       (?, 256, 64, 64)     (1, 1, 1, 256)   Grow_lod5           -         (?, 1, 8, 8)         -
Grow_lod1             -         (?, 256, 64, 64)     -                16x16/Conv0_up      2359808   (?, 512, 16, 16)     (3, 3, 512, 512)
64x64/Conv0           590080    (?, 256, 64, 64)     (3, 3, 256, 256) 16x16/Conv1         2359808   (?, 512, 16, 16)     (3, 3, 512, 512)
64x64/Conv1_down      1180160   (?, 512, 32, 32)     (3, 3, 256, 512) ToRGB_lod4          513       (?, 1, 16, 16)       (1, 1, 512, 1)
Downscale2D_2         -         (?, 1, 32, 32)       -                Upscale2D_1         -         (?, 1, 16, 16)       -
FromRGB_lod3          1024      (?, 512, 32, 32)     (1, 1, 1, 512)   Grow_lod4           -         (?, 1, 16, 16)       -
Grow_lod2             -         (?, 512, 32, 32)     -                32x32/Conv0_up      2359808   (?, 512, 32, 32)     (3, 3, 512, 512)
32x32/Conv0           2359808   (?, 512, 32, 32)     (3, 3, 512, 512) 32x32/Conv1         2359808   (?, 512, 32, 32)     (3, 3, 512, 512)
32x32/Conv1_down      2359808   (?, 512, 16, 16)     (3, 3, 512, 512) ToRGB_lod3          513       (?, 1, 32, 32)       (1, 1, 512, 1)
Downscale2D_3         -         (?, 1, 16, 16)       -                Upscale2D_2         -         (?, 1, 32, 32)       -
FromRGB_lod4          1024      (?, 512, 16, 16)     (1, 1, 1, 512)   Grow_lod3           -         (?, 1, 32, 32)       -
Grow_lod3             -         (?, 512, 16, 16)     -                64x64/Conv0_up      1179904   (?, 256, 64, 64)     (3, 3, 256, 512)
16x16/Conv0           2359808   (?, 512, 16, 16)     (3, 3, 512, 512) 64x64/Conv1         590080    (?, 256, 64, 64)     (3, 3, 256, 256)
16x16/Conv1_down      2359808   (?, 512, 8, 8)       (3, 3, 512, 512) ToRGB_lod2          257       (?, 1, 64, 64)       (1, 1, 256, 1)
Downscale2D_4         -         (?, 1, 8, 8)         -                Upscale2D_3         -         (?, 1, 64, 64)       -
FromRGB_lod5          1024      (?, 512, 8, 8)       (1, 1, 1, 512)   Grow_lod2           -         (?, 1, 64, 64)       -
Grow_lod4             -         (?, 512, 8, 8)       -                128x128/Conv0_up    295040    (?, 128, 128, 128)   (3, 3, 128, 256)
8x8/Conv0             2359808   (?, 512, 8, 8)       (3, 3, 512, 512) 128x128/Conv1       147584    (?, 128, 128, 128)   (3, 3, 128, 128)
8x8/Conv1_down        2359808   (?, 512, 4, 4)       (3, 3, 512, 512) ToRGB_lod1          129       (?, 1, 128, 128)     (1, 1, 128, 1)
Downscale2D_5         -         (?, 1, 4, 4)         -                Upscale2D_4         -         (?, 1, 128, 128)     -
FromRGB_lod6          1024      (?, 512, 4, 4)       (1, 1, 1, 512)   Grow_lod1           -         (?, 1, 128, 128)     -
Grow_lod5             -         (?, 512, 4, 4)       -                256x256/Conv0_up    73792     (?, 64, 256, 256)    (3, 3, 64, 128)
4x4/MinibatchStddev   -         (?, 1, 4, 4)         -                256x256/Conv1       36928     (?, 64, 256, 256)    (3, 3, 64, 64)
4x4/Conv              2364416   (?, 512, 4, 4)       (3, 3, 513, 512) ToRGB_lod0          65        (?, 1, 256, 256)     (1, 1, 64, 1)
4x4/Dense0            4194816   (?, 512)             (8192, 512)      Upscale2D_5         -         (?, 1, 256, 256)     -
4x4/Dense1            513       (?, 1)               (512, 1)         Grow_lod0           -         (?, 1, 256, 256)     -
scores_out            -         (?, 1)               -                images_out          -         (?, 1, 256, 256)     -
labels_out            -         (?, 0)               -                ---                 ---       ---                  ---
---                   ---       ---                  ---              Total               23039303
Total                 23047361
```

Figure 8: Fully Grown Discriminator and Generator of Progressive GAN