

# Anime Super Resolution Using GANs

Jin Huang (huang86)

CS230 - Deep Learning, Stanford University

## Introduction

- Anime images and videos are at most 1080p. The old classic ones are even at 480p.
- Anime resides in a lower dimension in terms of color and feature, usually with low frequency content. It's a different and unique field to focus on. Most importantly, I'm an anime fan.
- The best modern approach for super resolution problem is SRGAN. New solutions show up to improve the training of GAN, such as WGAN with gradient penalty.

## Data

- Only images larger than 512 x 512.
- 11328 random images from kaggle dataset <https://www.kaggle.com/mylesoneill/tagged-anime-illustrations/home>.
- Train/Dev/Test set sizes are 10816/256/256.



Fig 1. Dataset sample

## Preprocessing

- CenterCrop to 384 x 384
- RandomCrop to 128 x 128 (SRGAN) or 96 x 96 (SRWGAN-GP) - Original HR
- Bicubic downsample to 32 x 32 (SRGAN) or 24 x 24 (SRWGAN-GP) - LR input

## Method

### SRGAN Discriminator

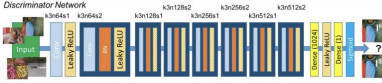


Fig 4. SRGAN Discriminator Network

```

(0) Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1) LeakyReLU(negative_slope=0.2)
(2) Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(3) BatchNorm2d(64, eps=1e-05, momentum=0.1)
(4) LeakyReLU(negative_slope=0.2)
(5) Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(6) BatchNorm2d(128, eps=1e-05, momentum=0.1)
(7) LeakyReLU(negative_slope=0.2)
(8) Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(9) BatchNorm2d(128, eps=1e-05, momentum=0.1)
(10) LeakyReLU(negative_slope=0.2)
(11) Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(12) BatchNorm2d(256, eps=1e-05, momentum=0.1)
(13) LeakyReLU(negative_slope=0.2)
(14) Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(15) BatchNorm2d(512, eps=1e-05, momentum=0.1)
(16) LeakyReLU(negative_slope=0.2)
(17) Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(18) BatchNorm2d(512, eps=1e-05, momentum=0.1)
(19) LeakyReLU(negative_slope=0.2)
(20) Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(21) BatchNorm2d(512, eps=1e-05, momentum=0.1)
(22) LeakyReLU(negative_slope=0.2)
(23) AdaptiveAvgPool2d(output_size=1)
(24) Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), padding=(0, 0))
(25) LeakyReLU(negative_slope=0.2)
(26) Conv2d(1024, 1, kernel_size=(1, 1), stride=(1, 1), padding=(0, 0))
(27) Sigmoid()
    
```

Fig 6. SRGAN Discriminator Network Details

- Input & output pixel values [0, 1].
- SRWGAN-GP has no BatchNorm layers.
- SRWGAN-GP has no sigmoid.

\* Batch normalization creates correlation between samples in the same batch. It impacts the effectiveness of the gradient penalty.

### SRGAN Discriminator Loss

$$-\frac{1}{M} \sum_{m=1}^M \log(D(x^m)) - \frac{1}{M} \sum_{m=1}^M (1 - y^m) \log(1 - D(G(z^m)))$$

### SRWGAN-GP Discriminator Loss

$$\frac{1}{M} \sum_{m=1}^M [D(x^m)] - \frac{1}{M} \sum_{m=1}^M [D(x^m)] + \lambda \frac{1}{M} \sum_{m=1}^M \left( \left\| \nabla_{x^m} D(x^m) \right\|_2 - 1 \right)^2$$

### SRWGAN-GP Discriminator

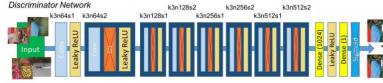


Fig 5. SRWGAN-GP Discriminator Network

```

(0) Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1) LeakyReLU(negative_slope=0.2)
(2) Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(3) LeakyReLU(negative_slope=0.2)
(4) Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(5) LeakyReLU(negative_slope=0.2)
(6) Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(7) LeakyReLU(negative_slope=0.2)
(8) Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(9) LeakyReLU(negative_slope=0.2)
(10) Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(11) LeakyReLU(negative_slope=0.2)
(12) Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(13) LeakyReLU(negative_slope=0.2)
(14) Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(15) LeakyReLU(negative_slope=0.2)
(16) AdaptiveAvgPool2d(output_size=1)
(17) Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), padding=(0, 0))
(18) LeakyReLU(negative_slope=0.2)
(19) Conv2d(1024, 1, kernel_size=(1, 1), stride=(1, 1), padding=(0, 0))
    
```

Fig 7. SRWGAN-GP Discriminator Network Details

## Method

### Generator

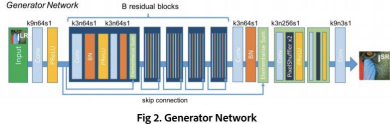


Fig 2. Generator Network

```

(0) Conv2d(3, 64, kernel_size=(9, 9), stride=(1, 1), padding=(4, 4))
(1) PReLU(num_parameters=1)
(2-9) [8x] ResidualBlock(
  Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  BatchNorm2d(64, eps=1e-05, momentum=0.1)
  PReLU(num_parameters=1)
  Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  BatchNorm2d(64, eps=1e-05, momentum=0.1)
)
(10) Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11) PReLU(num_parameters=1)
(12-13) [2x] UpsampleBicubic(
  Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  PReLU(num_parameters=1)
)
(14) Conv2d(512, 3, kernel_size=(9, 9), stride=(1, 1), padding=(4, 4))
(15) tanh()
    
```

- Input & output pixel values [0, 1].
- Tanh as the last layer.
- Using 8 residual blocks, not 16.

**Generator Loss**  
Loss = MSE\_loss +  $\lambda$  \* adversarial\_loss  
\* pre-trained model NOT used

### Generator Adversarial Loss

$$\text{SRGAN} \quad -\frac{1}{M} \sum_{m=1}^M \log(D(G(z^m)))$$

$$\text{SRWGAN-GP} \quad -\frac{1}{M} \sum_{m=1}^M D(G(z^m))$$

Fig 3. Generator Network Details

## Training

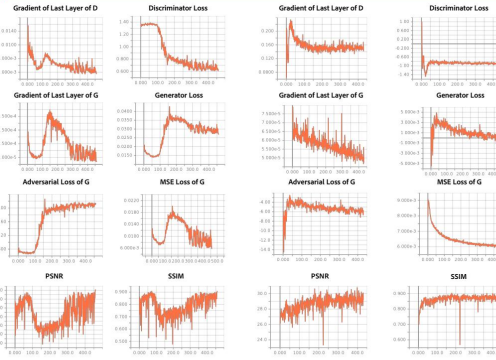


Fig 8. SRGAN Training

Fig 9. SRWGAN-GP Training

### HyperParameters

- 500 epochs.
- Default initializations from PyTorch.
- ADAM, betas=(0.9, 0.999), eps=1e-8.
- SRGAN uses learning rate 1e-3, SRWGAN-GP uses 1e-4.
- SRGAN uses  $\lambda = 1e-2$  in Generator loss, SRWGAN-GP uses 1e-3.
- SRGAN has label smoothing, Real (0.85, 1.10), Fake (0, 0.15), and noisy labels.
- SRGAN has input 32 x 32 and batch size 64, SRWGAN-GP has 24 x 24 and 32.

### Findings

- SRWGAN-GP's MSE loss decreases smoothly.
- SRWGAN-GP has stable gradients.
- SRWGAN-GP converges faster.
- SRGAN gets higher PSNR and SSIM at peak.

## Results

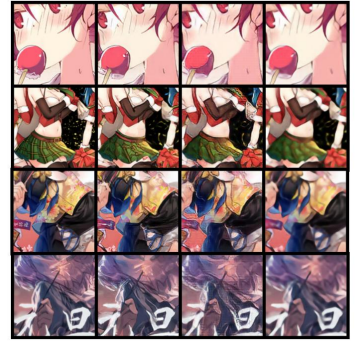


Fig 10. Original HR / SRWGAN-GP / SRGAN / Deep ResNet  
\* input LR images are 64x downsampled from original HR ones. SR images are similar with original HR images when the model is working.

- Deep ResNet gets very blurry results with high PSNR and SSIM.
- SRGAN results are more blurry than SRWGAN-GP ones by average.
- SRGAN fails in more cases. SRWGAN-GP produces good results even when SRGAN fails. See the last row in Fig 10.

## Conclusions

- Anime and realistic photos are on different distributions. Anime Super Resolution is a different problem from realistic photo Super Resolution.
- SRGAN is working on Anime Super Resolution without features from per-trained models.
- SRWGAN-GP overall outperforms SRGAN on Anime Super Resolution.

## Future Work

- Explore more GAN training tricks.
- Experiment with other GANs.
- Work on Anime video Super Resolution. Use RNN to get consistent video frames.