

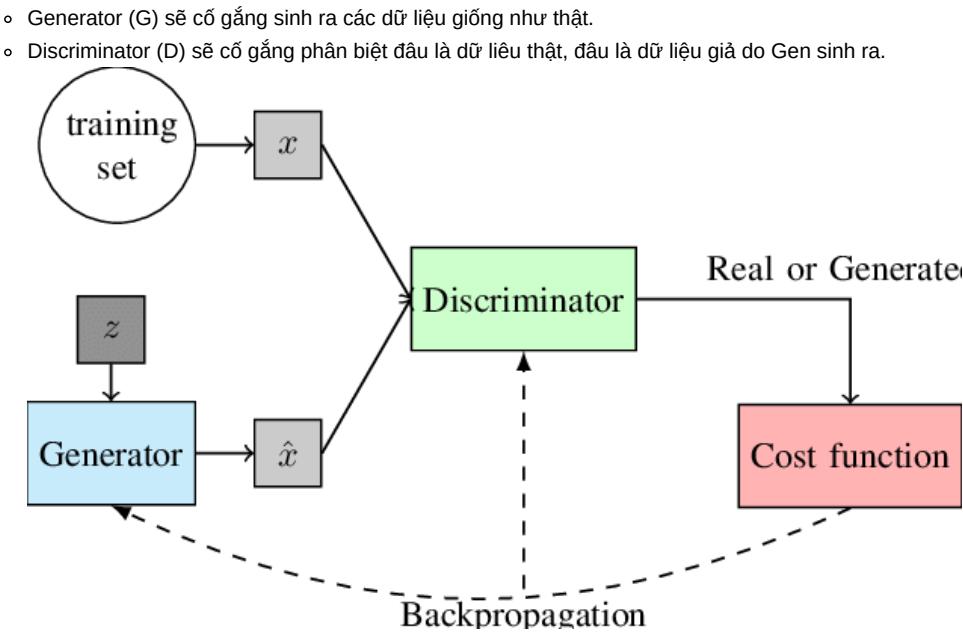
Documents Deblur-Gan

Generative Adversarial Networks (GAN)

Tìm hiểu về [Deblur-Gan](#)

- Khái niệm: GAN thuộc nhóm generative model. Với Generative nghĩ là có khả năng sinh dữ liệu , Network mạng (mô hình) và Adversarial đối nghịch (nghĩa là GAN sẽ có 2 mạng đối nghịch nhau).

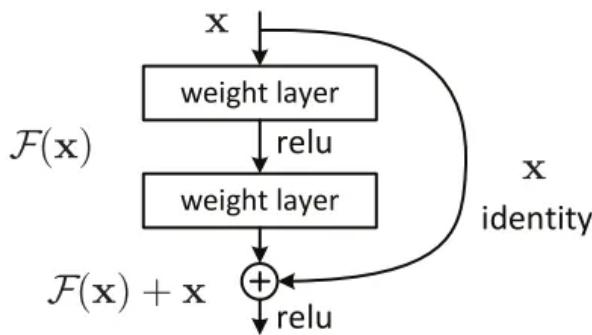
2. Generator và Discriminator



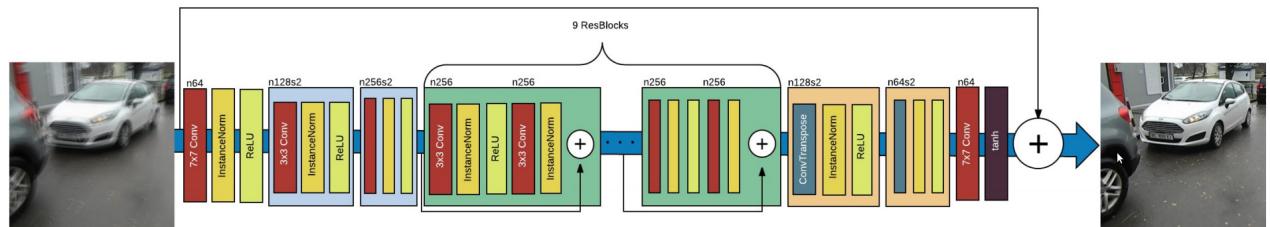
- Generator (G) sẽ cố gắng sinh ra các dữ liệu giống như thật.
- Discriminator (D) sẽ cố gắng phân biệt đâu là dữ liệu thật, đâu là dữ liệu giả do Gen sinh ra.

3. The Generator

- Ứng dụng GAN vào khôi phục ảnh mờ thì ở đây việc của Generator sẽ là tái tạo lại ảnh sắc nét nhất có thể với đầu vào là một bức ảnh bị mờ.
- Vì là mạng học sâu nên việc xảy ra vanishing/exploding gradient là khó có thể tránh khỏi. Để hạn chế việc đó thì ở đây ta có dùng ResNet blocks (9 block)
- Resnet.**
 - Ý tưởng chung của Resnet là việc có thể skip qua 1 hay nhiều lớp, và việc tính đạo hàm cũng dễ dàng hơn do.



- Cấu trúc mạng DeblurGAN Generator



4. The Discriminator

- Đối nghịch lại với Generator thì ở đây việc của Discriminator sẽ cố gắng phân biệt ảnh thật (ảnh gốc không bị mờ) và ảnh giả (ảnh giả được khôi phục từ ảnh mờ)

- Discriminator Architecture

Name	Filter	Kernel	Stride	Input	Output
Conv2D	64	(4, 4)	2	(256, 256, 3)	(128, 128, 64)
<hr/>					
LeakyReLU					
<hr/>					
Conv2D	64	(4, 4)	2	(128, 128, 64)	(64, 64, 64)
<hr/>					
BN+LeakyReLU					
<hr/>					
Conv2D	128	(4, 4)	2	(64, 64, 64)	(32, 32, 128)
<hr/>					
BN+LeakyReLU					
<hr/>					
Conv2D	256	(4, 4)	2	(32, 32, 128)	(16, 16, 256)
<hr/>					
BN+LeakyReLU					
<hr/>					
Conv2D	512	(4, 4)	1	(16, 16, 256)	(16, 16, 512)
<hr/>					
BN+LeakyReLU					
<hr/>					
Conv2D	1	(4, 4)	1	(16, 16, 256)	(16, 16, 1)
<hr/>					
Flatten					
<hr/>					
Dense(1024, 'tanh')					
<hr/>					
Dense(1, 'sigmoid')					

5. Hàm Loss và Optimizer

$$\mathcal{L} = \underbrace{\mathcal{L}_{GAN}}_{adv\ loss} + \underbrace{\lambda \cdot \mathcal{L}_X}_{content\ loss} \underbrace{\quad}_{total\ loss}$$

- Loss Function

- L1 (MAE)
- Perceptual Loss ?

$$\mathcal{L}_X = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^S)_{x,y} - \phi_{i,j}(G_{\theta_G}(I^B))_{x,y})^2$$

```
def perceptual_loss(y_true, y_pred):
    vgg = VGG16(include_top=False, weights='imagenet', input_shape=image_shape)
    loss_model = Model(inputs=vgg.input, outputs=vgg.get_layer('block3_conv3').output)
    loss_model.trainable = False
    return K.mean(K.square(loss_model(y_true) - loss_model(y_pred)))
```

- Wasserstein Loss

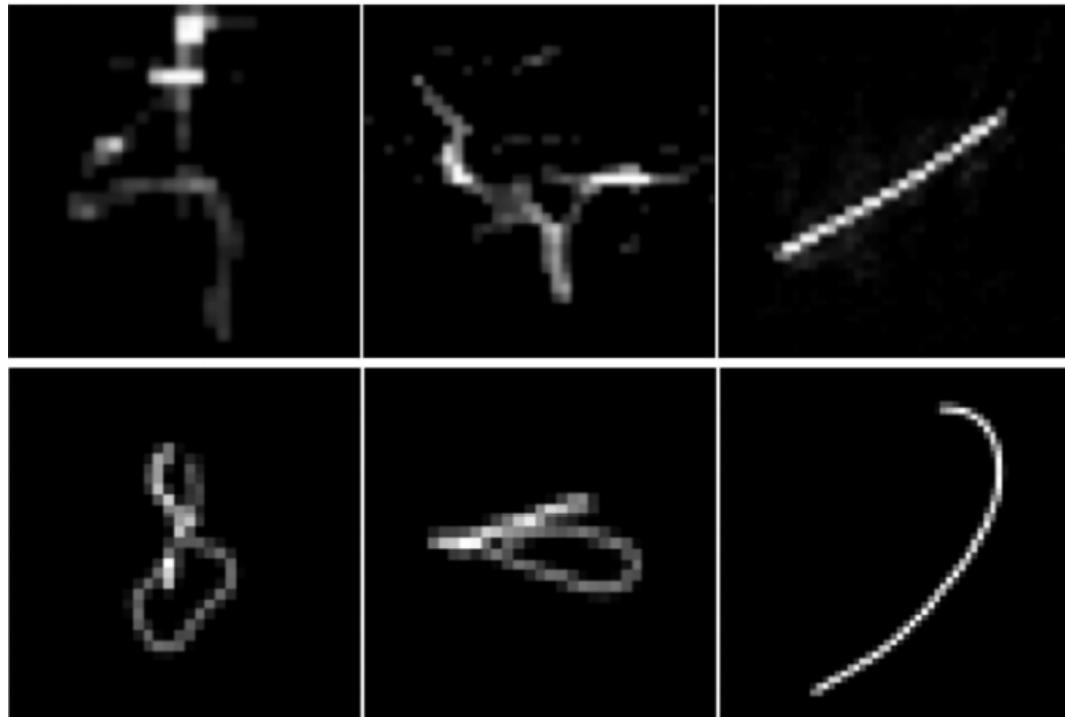
$$\mathcal{L}_{GAN} = \sum_{n=1}^N -D_{\theta_D}(G_{\theta_G}(I^B))$$

```
def wasserstein_loss(y_true, y_pred):
    return K.mean(y_true*y_pred)
```

- Optimizer Adam

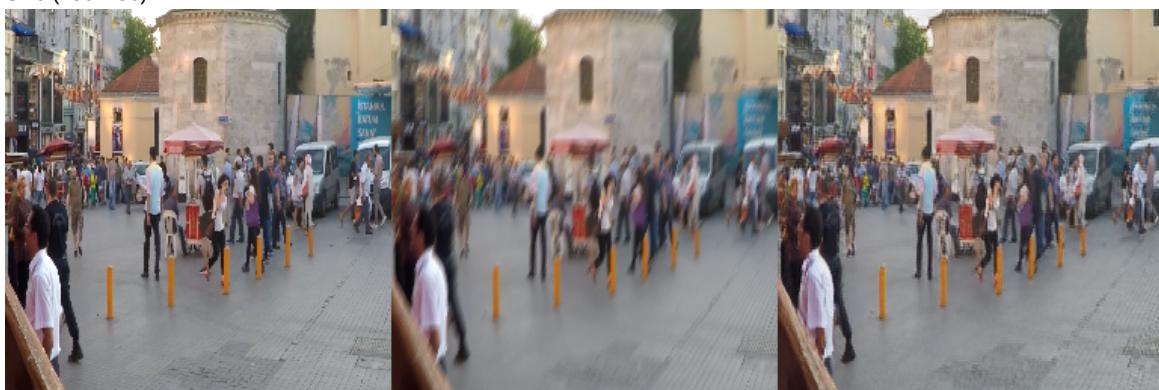
6. Dataset

- Sharp image
 - GoProAll
 - Kadid
 - GoPro DataSet
- Blur image
 - Convolution with kernel same same



7. Statistics

- Result
- Size (256x256)



- Size (1280x720)





STT	Epoch	Name Model Gen	Sample Train	Bath Size	Critic Updates	File Sample Test	File Sample Out	Average Score Sharp	Average Score Deblur	Ratio
1	50	generator_49_298.h5	2100	2	5	100	(256x256)	3162.7987	1816.9127	57.4463%
2	100	generator_99_243.h5	2100	2	5	100	(256x256)	3162.7987	1849.6389	58.4811%
3	?	generator.h5	?	?	?	100	(256x256)	3162.7987	1965.7959	62.1536%
4	50	generator_49_298.h5	2100	2	5	100	(1280x720)	196.3081	208.8575	106.3927%
5	100	generator_99_243.h5	2100	2	5	100	(1280x720)	196.3081	224.3599	114.2897%
6	?	generator.h5	?	?	?	100	(1280x720)	196.3081	28267.7345	143.9968%