

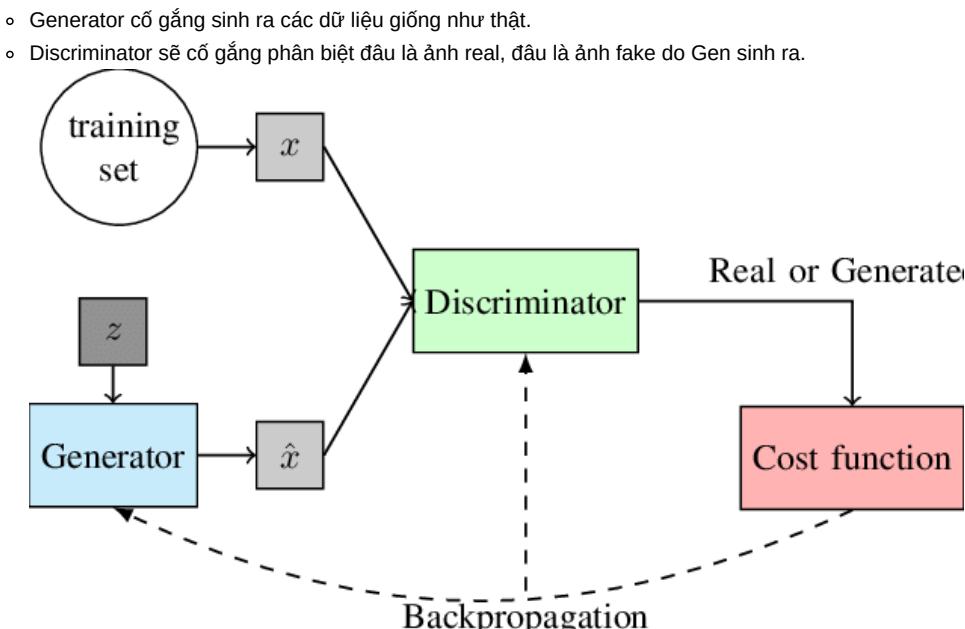
Documents Deblur-Gan

Generative Adversarial Networks (GAN)

Tìm hiểu về [Deblur-Gan](#)

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

2. Generator vs Discriminator



- Việc Gen và Dis qua lại như vậy sẽ là 2 bên tự giúp nhau học giúp nhau cải tiến lên.
- Mục tiêu là maximize D(x) (giá trị dự đoán với ảnh x) và minimize D(G(z)) (giá trị dự đoán với ảnh z được sinh ra từ G)

3. The Generator

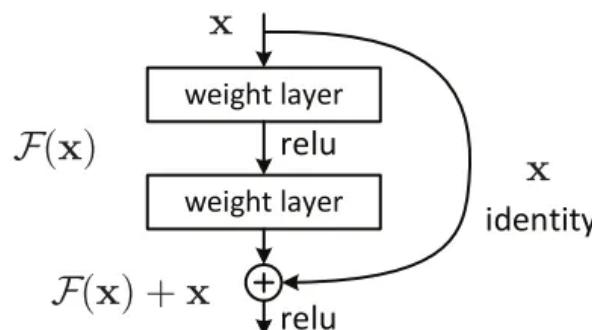
- Ở đây việc của Generator sẽ là tái tạo lại ảnh sắc nét nhất có thể.
- Vì là mạng học sâu nên việc xảy ra vanishing/exploding gradient là khó có thể tránh khỏi.
 - Trước tiên là nói về Backpropagation Algorithm là kỹ thuật thường được dùng trong quá trình training Deep Neural Network (DNNs). Ý tưởng chung của thuật toán là sẽ đi từ output layer đến input layer và tính toán gradient của cost function tương ứng cho từng parameter (weight) của network. Sau đó, Gradient Descent, sẽ được sử dụng để cập nhật các parameter đó. Quá trình trên sẽ được lặp lại cho tới khi các parameter của network hội tụ. Thường thì sẽ có 1 hyperparameter sẽ thể hiện số lượng vòng lặp để thực hiện quá trình trên (Epoch) nếu epoch nhỏ thì KQ sẽ không tốt còn quá lớn sẽ training quá lâu.
 - Và gradient sẽ có giá trị nhỏ dần khi nó xuống các layer thấp hơn, dẫn đến việc cập nhật của gradient descent không làm thay đổi nhiều weight của các layer đó, khiến chung không thể hội tụ và DNNs sẽ không thu được kết quả tốt. Hiện tượng này được gọi là Vanishing Gradients.
 - Ngược lại đôi khi gradient sẽ có giá trị khá lớn khiến cho việc Backpropagation phình to ra nó cũng dẫn đến DNN sẽ không có được kết quả như mong muốn.
- Để hạn chế hai vấn đề đó thì chúng ta sẽ sử dụng Batch Normalization (BN)
 - Giống như việc ta normalization dữ liệu đầu vào thì ta cũng nên normalization hidden layer, "also BN allows each layer of a network to learn by itself a little bit more independently of other layer". BN giúp cho việc tăng tốc độ training, bằng việc có thể tăng learning rate because it make sure that there's no activation that's gone really high or really low.
 - It reduces overfitting because it has a slight regularization effects. It adds some noise to each hidden layer's activations. Therefore, if we use batch normalization, we will use less dropout, which is a good thing because we are not going to lose a lot of information. However, we should not depend only on batch normalization for regularization, we should better use it together with dropout.
- Generator Architecture

Name	Filter	Kernel	Stride	Input	Output
------	--------	--------	--------	-------	--------

Name	Filter	Kernel	Stride	Input	OutPut
Padding	-	[[3, 3],[3, 3]]	-	(256, 256, 3)	(262, 262, 3)
Conv2D	64	(7, 7)	-	(262, 262, 3)	(256, 256, 64)
BN+'relu'					
Conv2D	128	(3, 3)	2	(256, 256, 64)	(128, 128, 128)
BN+'relu'					
Conv2D	256	(3, 3)	2	(128, 128, 128)	(64, 64, 256)
BN+'relu'					
ResBlock (9)	-	-	-	-	-
Padding	-	[[1, 1],[1, 1]]	-	(64, 64, 256)	(66, 66, 256)
Conv2D	256	(3, 3)	(1, 1)	(66, 66, 256)	(64, 64, 256)
BN+'relu'					
Dropout(0.5)					
Padding	-	[[1, 1],[1, 1]]	-	(64, 64, 256)	(66, 66, 256)
Conv2D	256	(3, 3)	(1, 1)	(66, 66, 256)	(64, 64, 256)
BN+Add()					
UpSampling2D	-	-	-	(64, 64, 256)	(128, 128, 256)
Conv2D	128	(3, 3)	-	(128, 128, 256)	(128, 128, 128)
BN+'relu'					
UpSampling2D	-	-	-	(128, 128, 128)	(256, 256, 128)
Conv2D	64	(3, 3)	-	(256, 256, 128)	(256, 256, 64)
BN+'relu'					
Padding	-	[[3, 3],[3, 3]]	-	(256, 256, 64)	(262, 262, 64)
Conv2D	3	(7, 7)	-	(262, 262, 3)	(256, 256, 3)
'tanh'+Add					

- 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.



4. The Discriminator

- Ở đây việc của Discriminator sẽ cố gắng phân biệt ảnh real và fake -> binary classification -> last activation sigmoid
- Discriminator Architecture

Name	Filter	Kernel	Stride	Input	OutPut
------	--------	--------	--------	-------	--------

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

5. Loss + Optimizer

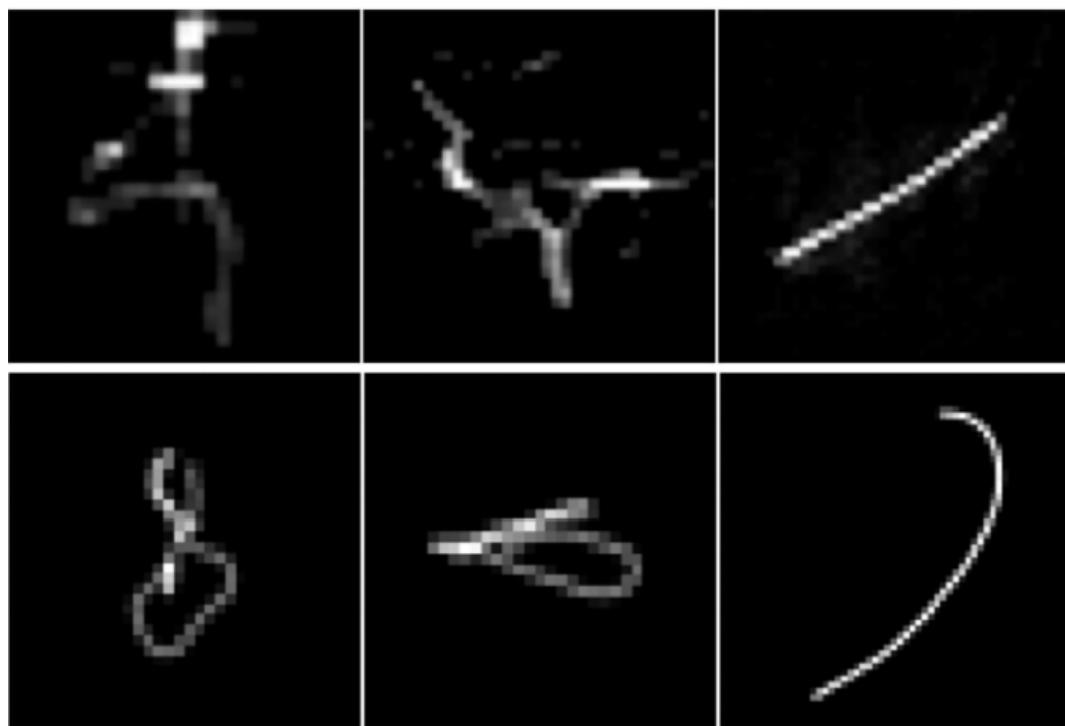
- [L1 \(MAE\)](#)
- Perceptual Loss ?


```
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


```
def wasserstein_loss(y_true, y_pred):
    return K.mean(y_true*y_pred)
```
- [Adam](#)

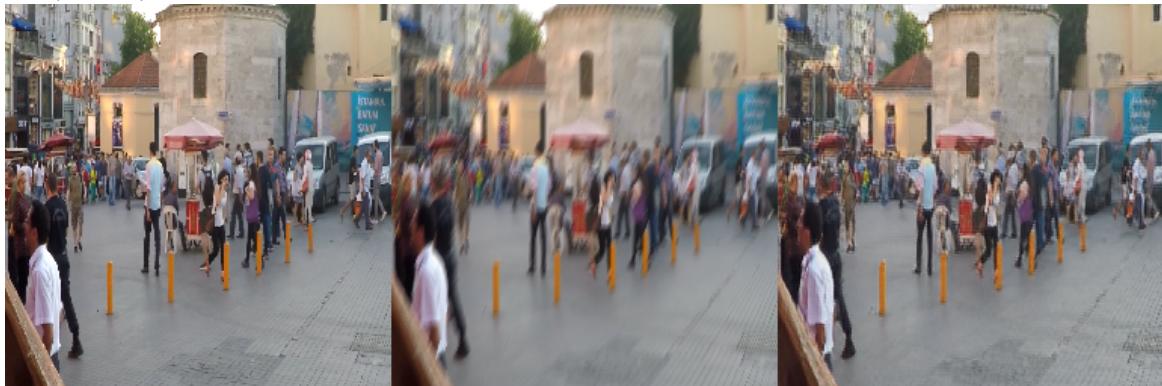
6. Dataset

- Sharp image
 - [GoProAll](#)
 - [Kadid](#)
 - [GoPro DataSet](#)
- Blur image
 - Convolution with kernel same same



7. Statistics

- Result
- Size (256x256)



- Size (1280x1280)





STT	Epoch	Name 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%