Bài 49 - Pix2Pix GAN

13 Nov 2020 - phamdinhkhanh

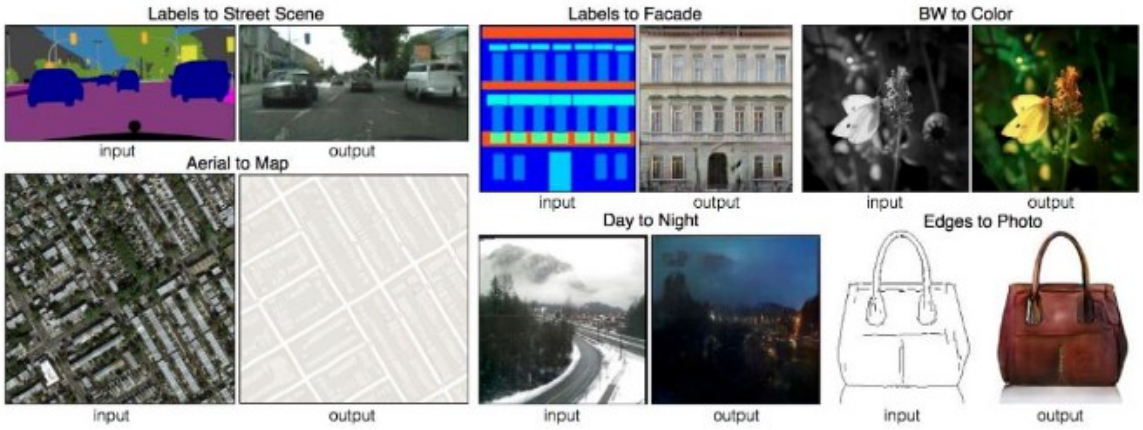
Menu

- 1. Giới thiệu chung
- 2. Kiến trúc của Pix2Pix
- 3. Loss function
- 4. Kiến trúc Pix2Pix dựa trên PatchGAN
- 5. Thực hành
    - Discriminator
    - Generator
    - Loss function GAN
    - Huấn luyện mô hình
- 6. Kết luận
- 7. Tài liệu

1. Giới thiệu chung

Gần đây trong công việc của mình tôi đã sử dụng các mô hình GAN cho các tác vụ sinh ảnh nhằm bổ sung thêm dữ liệu cho các mô hình. Kết quả khá lạc quan là độ chính xác của mô hình đã được cải thiện đáng kể. Tôi nghĩ rằng GAN là lớp mô hình rất hay mà nếu bạn biết áp dụng một cách khéo léo vào data augumentation, nó có thể giúp tạo ra những mô hình chất lượng hơn với ít dữ liệu hơn. Đó cũng chính là lý do và động lực để tôi dành thời gian viết bài chia sẻ về mô hình Pix2Pix, một lớp mô hình chuyên biệt cho các tác vụ image-to-image translation.

Pix2Pix có thể tạo ra vô số các ứng dụng liên quan đến ảnh bên cạnh data augumentation, tiêu biểu như:



- Chuyển đổi màu sắc giữa các bức ảnh: Từ ảnh màu sang ảnh thường và ngược lại.
- Tái tạo phong cách hội họa của những họa sĩ nổi tiếng đã qua đời như Van Gogh, Picasso, Monet,… dựa trên những bức ảnh họ đã để lại.
- Chuyển đổi màu sắc các bức ảnh.
- Thêm màu sắc cho các nét vẽ để tạo thành ảnh màu.
- Chuyển trời tối sang trời sáng.
- Tạo các ảnh segmentation từ ảnh gốc.
- Hay một ứng dụng tuyệt vời nhất mà các bạn sử dụng hàng ngày, đó chính là chuyển đổi bức ảnh chụp từ vệ tinh sang google map.

Top

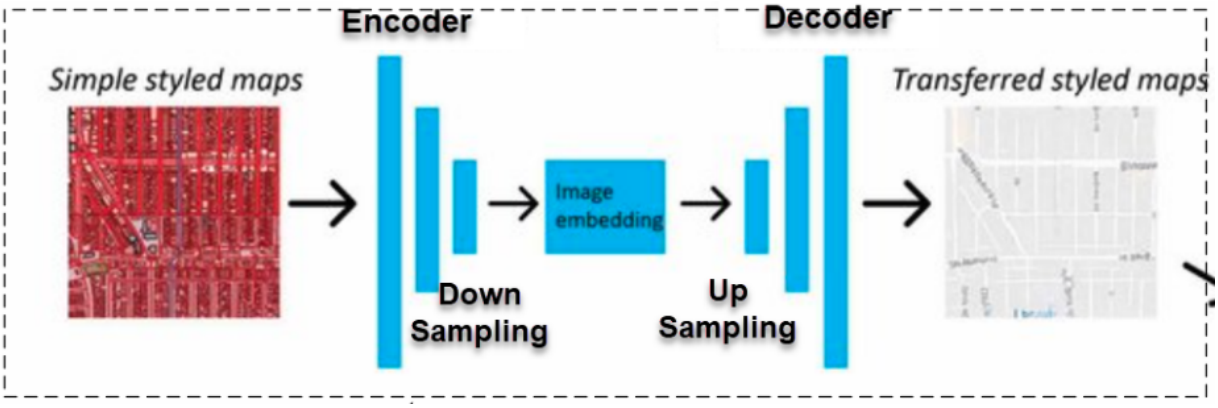
Để xem các ứng dụng này cụ thể hơn, các bạn có thể vào trang của tác giả Pix2Pix (https://phillipi.github.io/pix2pix/).

Thế nào là một conditional GAN?

Pix2Pix (https://arxiv.org/abs/1611.07004) là một mô hình cGAN (conditional GAN) được giới thiệu tại CVPR-2017 với tiêu đề `Image-to-Image Translation with Conditional Adversarial Networks` . Về cGAN thì mình đã giới thiệu ở Bài 45 - conditional GAN (https://phamdinhkhanh.github.io/2020/08/09/ConditionalGAN.html). Điểm khác biệt chính giữa cGAN và mô hình GAN đó là nó yêu cầu chúng ta phải có điều kiện cho ảnh đầu vào. Điều kiện ở đây có thể là nhãn của ảnh cần được tạo ra hoặc chính là một bức ảnh. Do đó chúng ta có thể kiểm soát được đầu ra dựa trên điều kiện truyền vào. Còn trong GAN chúng ta chỉ khởi tạo một véc tơ đầu vào ngẫu nhiên và hoàn toàn không kiểm soát được ảnh đầu ra là gì.

2. Kiến trúc của Pix2Pix

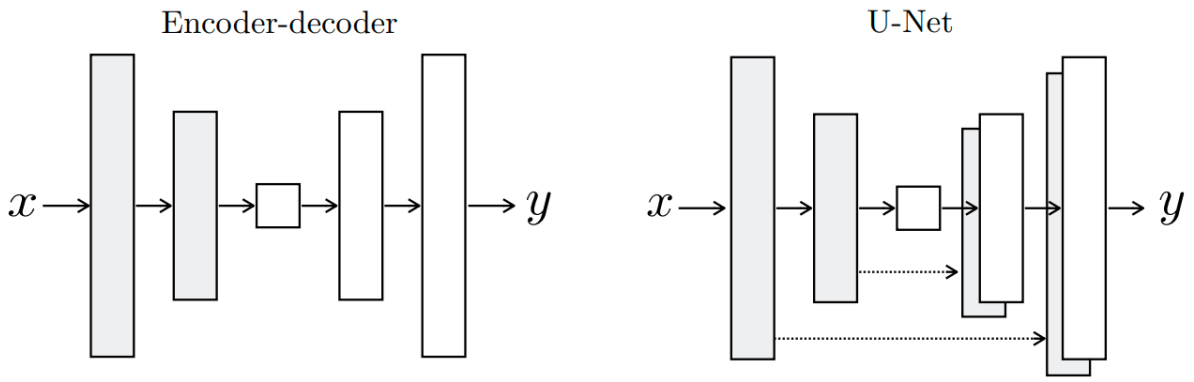
Kiến trúc của Pix2Pix cũng bao gồm 2 mô hình là generator và discriminator. Generator có tác dụng sinh ảnh fake để đánh lừa discriminator và discriminator có mục tiêu phân biệt giữa ảnh real và ảnh fake.



Hình 1: Kiến trúc down sampling và upsampling trong bài toán image-to-image translation. Đây cũng chính là kiến trúc chung của generator trong Pix2Pix GAN.

- **Generator**: Đầu vào của generator là một bức ảnh nguồn (source image), như trong hình là ảnh chụp vệ tinh có màu. Sau đó bức ảnh được truyền vào một kiến trúc mạng tích chập để trích lọc đặc trưng thông qua quá trình down sampling. Ở quá trình này, output ở các layer sẽ có kích thước giảm dần thông qua tích chập 2 chiều. Trong khi đó mạng giải chập (Deconvolution) làm nhiệm vụ ngược lại là biến đổi đặc trưng thành ảnh đích (target image) thông qua quá trình up sampling bằng các layers chuyên biệt như `transposed Convolution, Dilation Convolution, Upsampling 2D` . Quá trình up sampling sẽ gia tăng dần kích thước output và kết quả trả ra là một bức ảnh đích mang nhãn fake. Về quá trình giải chập các bạn có thể xem thêm tại tại Bài 40 - mạng giải chập (https://phamdinhkhanh.github.io/2020/06/10/ImageSegmention.html#5-m%E1%BA%A1ng-gi%E1%BA%A3i-ch%E1%BA%ADp-deconvolutional-neural-network).

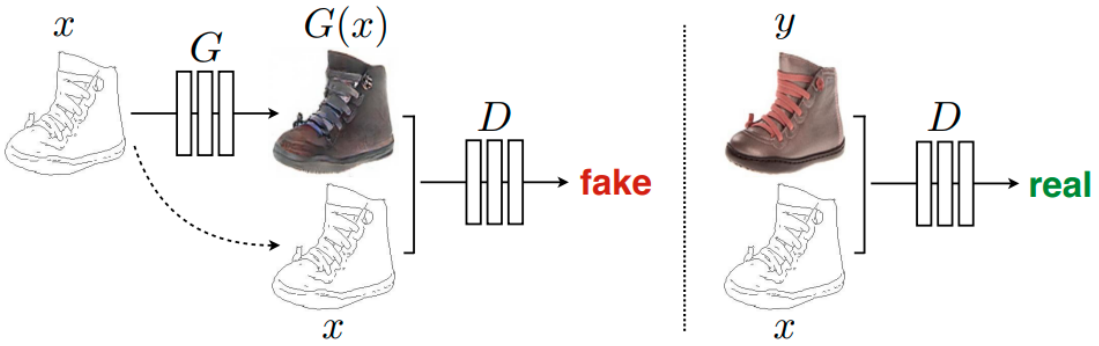
Top



Hình 2: Kiến trúc Unet được sử dụng trong generator của Pix2Pix.

Trong bài báo gốc về Pix2Pix thì tác giả dựa trên kiến trúc Unet, một kiến trúc mạng nổi tiếng trong xử lý ảnh y tế (biomedical image segmentation), để khởi tạo generator. Các layers ở nhánh bên trái (phần encoder) sẽ được concatenate trực tiếp vào các layers ở nhánh bên phải (phần decoder) có cùng kích thước. Kết nối này được gọi là kết nối tắt (skip connection) nhằm bổ sung thêm thông tin và gia tăng độ chính xác.

Tiếp theo là discriminator.



- **Discriminator**: Đầu vào của discriminator là một cặp ảnh x, y. Trong đó x là ảnh source đầu vào và y là ảnh target được sinh ra từ generator hoặc được lấy từ tập train. Hai ảnh x, y có cùng kích thước và sẽ được concatenate với nhau trước khi truyền vào mô hình discriminator. Cặp ảnh sẽ có nhãn là real nếu ảnh y được lấy từ tập train và trái lại khi y được sinh ra từ generator thì cặp ảnh x, y nó sẽ mang nhãn fake.

3. Loss function

Loss function của Pix2Pix là một hàm dạng binary cross entropy có dạng như sau:

$$\mathcal{L}_{cGAN}(G, D) = \underbrace{\mathbb{E}_{x,y}[\log D(x, y)]}_{\text{log-probability that D predict } \{x, y\} \text{ is real}} + \underbrace{\mathbb{E}_{x,z}[\log(1 - D(x, G(x, z)))]}_{\text{log-probability D predicts G(x, z) is fake}} \quad (1)$$

Bạn có thấy công thức này quen thuộc không ? Nó hoàn toàn giống với loss function của GAN (https://phamdinhkhanh.github.io/2020/07/13/GAN.html#34-h%C3%A0m-loss-function) mà mình đã giới thiệu ở bài trước đó. Chỉ khác ở đầu vào là một cặp ảnh x, y thay vì chỉ là x.

Qúa trình huấn luyện của chúng ta sẽ là một quá trình huấn luyện đồng thời trên cả Generator và Discriminator. Discriminator sẽ tìm cách phân loại ảnh real và fake chuẩn xác nhất nên giá trị loss function của nó là hàm cross entropy phải càng nhỏ càng tốt. Tức là chúng ta cần tìm $\max_D \mathcal{L}_{cGAN}(G, D)$.

Top

Bên cạnh đó, trong trường hợp ảnh là fake thì chúng ta cần Generator sinh ra nó giống với thật nhất, tức là giá trị xác suất $D(x, G(x, z))$ phải gần 1 nhất có thể. Do đó chúng ta cần tìm $\min_G \mathcal{L}_{cGAN}(G, D)$.

Kết hợp cả hai bài toán tối ưu giữa Generator và Discriminator chúng ta thu được bài toán tối ưu:

$$\min_G \max_D \mathcal{L}_{cGAN}(G, D)$$

Để kiểm tra tầm quan trọng của điều kiện trong việc phân loại, chúng ta có một phiên bản không điều kiện bằng cách loại bỏ x khỏi mô hình discriminator. Khi đó loss function của GAN sẽ trở thành:

$$\mathcal{L}_{GAN}(G, D) = \underbrace{\mathbb{E}_y[\log D(y)]}_{\text{log-probability that D predict } y \text{ is real}} + \underbrace{\mathbb{E}_{x,z}[\log(1 - D(G(x, z)))]}_{\text{log-probability D predicts G(x, z) is fake}}$$

Bạn có thể hình dung phương pháp này tương đương với việc ta không truyền vào discriminator một cặp x, y mà chỉ truyền vào y.

Ngoài ra tác giả còn nhận thấy có một lợi ích nếu kết hợp giữa loss function của GAN với một loss function truyền thống như norm chuẩn bậc 1 (L_1) hoặc bậc 2 (L_2). Khi đó vai trò phân biệt real/fake của discriminator không đổi, còn generator ngoài đánh lừa được discriminator thì nó còn phải làm sao sinh ảnh giống với ground truth nhất thông qua tối thiểu hóa L_1 hoặc L_2. Bên dưới là thành phần norm chuẩn bậc L_1 trong loss function:

$$\mathcal{L}_{L_1}(G) = \mathbb{E}_{x,y,z}[||y - G(x, z)||_1]$$

Hàm loss function mới:

$$\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L_1}(G)$$

Ở đây λ là một hệ số của loss function theo norm chuẩn bậc 1, thường được thiết lập rất nhỏ.

Ở phần thực hành bên dưới các bạn có thể thử nghiệm hàm loss function mới này để kiểm nghiệm xem liệu việc kết hợp với các loss function truyền thống như norm chuẩn bậc 1 hoặc bậc 2 sẽ giúp cải thiện chất lượng bức ảnh được sinh ra hay không ?

4. Kiến trúc Pix2Pix dựa trên PatchGAN

PatchGAN là một kiến trúc mà có Discriminator dựa trên các vùng nhận thức (`receptive field`). Nó sẽ maping mỗi một pixel ở output từ một vùng diện tích hình vuông nằm trên input (hay còn gọi là patch).

Một PatchGAN với kích thước `70 x 70` có nghĩa rằng mỗi một output sẽ được map tới một patch kích thước `70 x 70` . Mô hình sẽ phân loại các patch `70 x 70` là real hay fake. Xin trích dẫn:

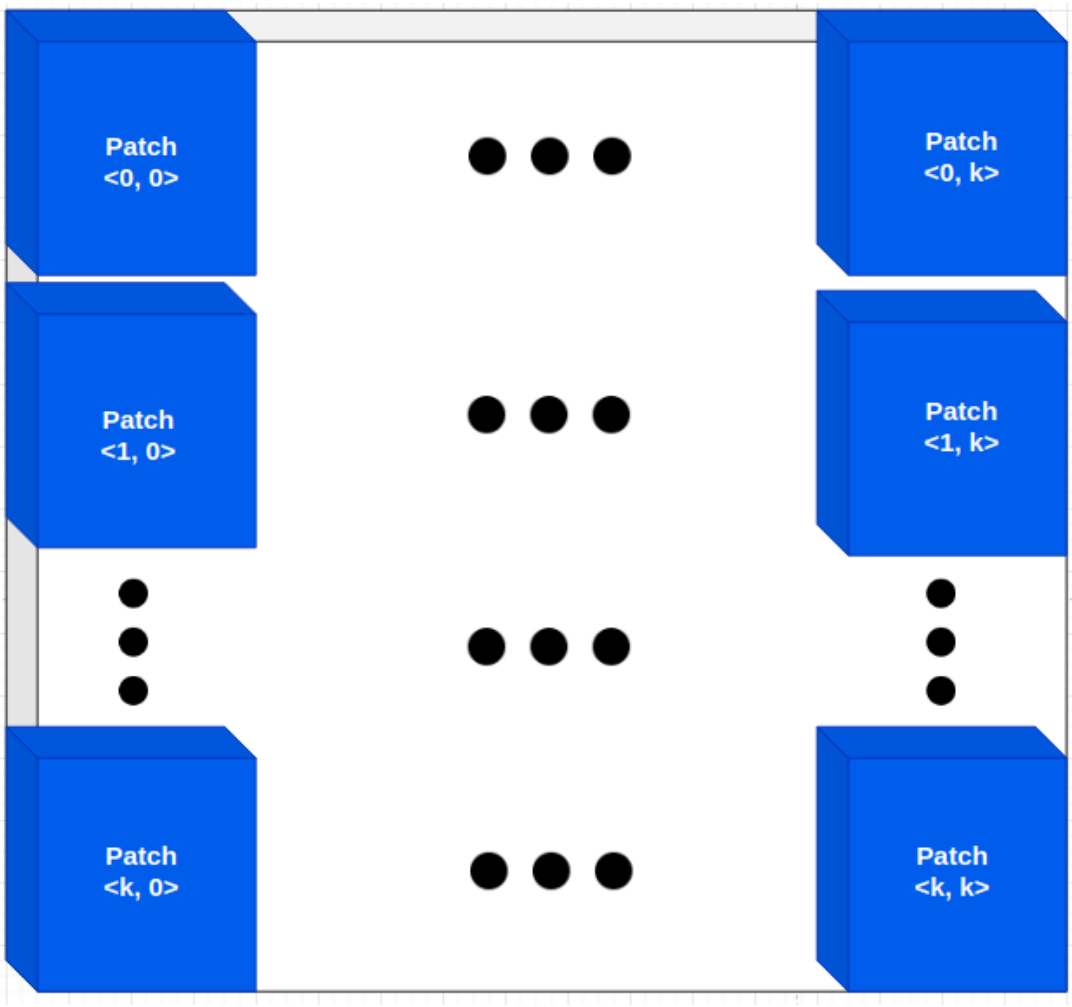
` we design a discriminator architecture – which we term a PatchGAN – that only penalizes structure at the scale of patches. This discriminator tries to classify if each NxN patch in an image is real or fake. We run this discriminator convolutionally across the image, averaging all responses to provide the ultimate output of D. `

Source: Image-to-Image Translation with Conditional Adversarial Networks (https://arxiv.org/abs/1611.07004).

Top

Kết quả của PatchGAN sẽ có output là một feature map hình vuông gồm tập hợp các xác suất mà mỗi một xác suất tương ứng với khả năng một patch kích thước `70 x 70` rơi vào real hoặc fake. Xác suất cho toàn bộ hình ảnh input là real hoặc fake có thể được tính bằng trung bình cộng của toàn bộ các xác suất trên toàn bộ patches.

Về bản chất của PatchGAN thì vẫn là một kiến trúc mạng CNN gồm nhiều layers CNN liên tiếp nhau, nhưng chúng ta không thực hiện flatten ở gần cuối để truyền qua các fully connected layers. Mà thay vào đó tính toán ra feature map xác suất trên từng patch như đã nêu ở trên. Cách tính như vậy sẽ mang lại hiệu quả nếu áp dụng trên các patch có kích thước lớn vì có vùng nhận thức (receptive field) lớn hơn. Kết quả xác suất trung bình của nhiều patches cũng sẽ chuẩn xác hơn.



Hình 3: Giả sử hình ảnh được chia thành $k \times k$ patches, trên mỗi patch ta dự báo một xác suất ảnh rơi vào real. Xác suất để toàn bộ bức ảnh là real sẽ là trung bình cộng xác suất của toàn bộ các patches. Theo phương pháp này chúng ta sẽ không flatten và fully connected layers.

Tiếp theo ta sẽ cùng tìm hiểu kiến trúc CNN được áp dụng trên một patch.

Kiến trúc các layers của PatchGAN cũng bao gồm 4 layers CNN theo thứ tự: `I -> C1 -> C2 -> C3 -> C4 -> O` . Dựa vào kích thước output shape của một layer ta có thể xác định được kích thước input shape của layer liền trước nó. Thật vậy:

Theo công thức tính kích thước output của tích chập 2 chiều theo kích thước input W_{input}, kích thước filter F, stride S và padding P ta có:

$$W_{output} = \frac{W_{input} - F + 2P}{S} + 1$$

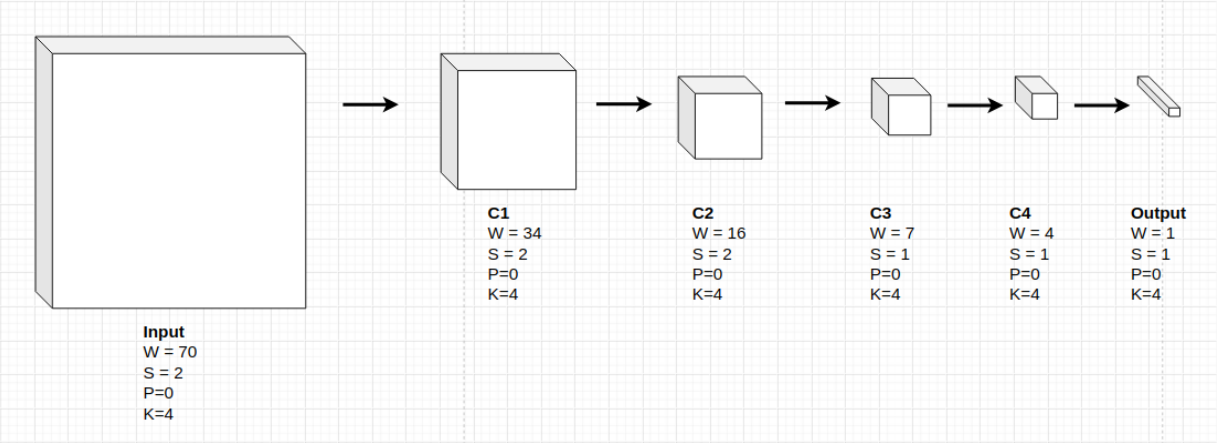
Top

Như vậy kích thước của receptive field W_{input} được tính dựa trên W_{output} sẽ là:

$$W_{input} = (W_{output} - 1) \times S - 2P + F$$

Chúng ta sẽ lần ngược theo công thức trên để suy ra kích thước của từng layer. Thứ tự lần ngược `O -> C4 -> C3 -> C2 -> C1 -> I`.

- Tại layer output `O`. $W_{output} = 1, S = 1, P = 0, F = 4$ Suy ra kích thước của `C4`: $W_{input} = (1 - 1) \times 1 - 2 \times 0 + 4 = 4$
- Tại layer `C4`. $W_{output} = 4, S = 1, P = 0, F = 4$ Suy ra kích thước của `C3`: $W_{input} = (4 - 1) \times 1 - 2 \times 0 + 4 = 7$
- Tại layer `C3`. $W_{output} = 7, S = 2, P = 0, F = 4$ Suy ra kích thước của `C2`: $W_{input} = (7 - 1) \times 2 - 2 \times 0 + 4 = 16$
- Tại layer `C2`. $W_{output} = 16, S = 2, P = 0, F = 4$ Suy ra kích thước của `C1`: $W_{input} = (16 - 1) \times 2 - 2 \times 0 + 4 = 34$
- Tại layer `C1`. $W_{output} = 34, S = 2, P = 0, F = 4$ Suy ra kích thước của `I`: $W_{input} = (34 - 1) \times 2 - 2 \times 0 + 4 = 70$



Hình 3: Sơ đồ tổng quát kiến trúc các layers CNN áp dụng trên một patch `70x70` của PatchGAN.

Trong bài viết này, để đơn giản hóa, ở phần thực hành mình sẽ chỉ giới thiệu tới các bạn một kiến trúc discriminator không áp dụng PatchGAN.

5. Thực hành

Trong phần thực hành chúng ta sẽ cùng xây dựng một mô hình Pix2Pix thuộc lớp bài toán image-to-image translation để tạo ra ứng dụng sinh ảnh google map từ ảnh chụp vệ tinh.

Đầu tiên chúng ta sẽ download dữ liệu:

```
from google.colab import drive
import os

drive.mount('/content/gdrive')
os.chdir("gdrive/My Drive/Colab Notebooks/Pix2PixModel")
```

```
!wget http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/maps.tar.gz
!tar -xvf maps.tar.gz
```

Top

Bộ dữ liệu của chúng ta sẽ bao gồm hai tập train và validation, trong đó mỗi tập đều có kích thước là 1096 ảnh.

```
import glob
import numpy as np
import matplotlib.pyplot as plt

img_trains = glob.glob("maps/train/*")
img_vals = glob.glob("maps/val/*")

fg, ax = plt.subplots(2, 3, figsize=(20, 8))
fg.suptitle('Google Map Plot')

for i in np.arange(2):
  for j in np.arange(3):
    img = plt.imread(img_trains[i + j + j*i])
    ax[i, j].imshow(img)
    ax[i, j].set_xlabel('Google Map '+str(i+j+j*i))
```

Như vậy một bức ảnh bao gồm một cặp ảnh `<source, target>` . Trong đó ảnh source là những ảnh vệ tinh có màu và target là ảnh bản đồ tương ứng. Ảnh source chiếm 50% bức ảnh phía bên trái và ảnh target chiếm 50% ảnh phía bên phải. Mỗi bức ảnh có kích thước là `600 x 600` . Trong bài này chúng ta sẽ resize image về kích thước `256 x 256` nhằm phù hợp với mạng CNN. Nhiệm vụ của chúng ta là xây dựng một mô hình `Pix2Pix GAN` để translate ảnh `source` sang `target` . Tiếp theo chúng ta sẽ viết hàm trả ra cặp ảnh `<source, target>` từ đường link.

```
import cv2

def _load_imgs(all_images):
  sources = []
  targets = []
  for image_path in all_images:
    img = plt.imread(image_path)
    src = img[:, :600]
    tar = img[:, 600:1200]
    src = cv2.resize(src, dsize=(256, 256), interpolation=cv2.INTER_L
    tar = cv2.resize(tar, dsize=(256, 256), interpolation=cv2.INTER_L
    sources.append(src)
    targets.append(tar)
  sources = np.array(sources)
  targets = np.array(targets)
  return sources, targets

train_src, train_tar = _load_imgs(img_trains)
val_src, val_tar = _load_imgs(img_vals)

print('train_src.shape: ', train_src.shape)
print('val_src.shape: ', val_src.shape)
```

Discriminator

Đầu vào của discriminator sẽ là một cặp ảnh `<source, target>` có cùng kích thước, ảnh source là ảnh màu chụp từ vệ tinh và ảnh target là ảnh google map.

Top

```
from tensorflow.keras.layers import Input, Conv2D, Conv2DTranspose, C
from tensorflow.keras.initializers import RandomNormal
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model

# Xác định discriminator
def _discriminator(image_shape):
        # khởi tạo weight theo phân phối chuẩn có phương sai là 0.02
        init = RandomNormal(stddev=0.02)
  # concatenate source và target image
        in_src_image = Input(shape=image_shape)
        in_target_image = Input(shape=image_shape)
        merged = Concatenate()([in_src_image, in_target_image])
        # C1: S=2, K=4, P=0, N_Fitler=64
        d = Conv2D(64, (4,4), strides=(2,2), padding='same', kernel_i
        d = LeakyReLU(alpha=0.2)(d)
        # C2: S=2, K=4, P=0, N_Filter=128
        d = Conv2D(128, (4,4), strides=(2,2), padding='same', kernel_
        d = BatchNormalization()(d)
        d = LeakyReLU(alpha=0.2)(d)
        # C3: S=2, K=4, P=same, N_Filter=256
        d = Conv2D(256, (4,4), strides=(2,2), padding='same', kernel_
        d = BatchNormalization()(d)
        d = LeakyReLU(alpha=0.2)(d)
        # C4: S=2, K=4, P=same, N_Filter=512
        d = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_
        d = BatchNormalization()(d)
        d = LeakyReLU(alpha=0.2)(d)
        # Out: S=1, K=4, P=same
        d = Conv2D(512, (4,4), padding='same', kernel_initializer=ini
        d = BatchNormalization()(d)
        d = LeakyReLU(alpha=0.2)(d)
        d = Conv2D(1, (4,4), padding='same', kernel_initializer=init)
        patch_out = Activation('sigmoid')(d)
        # model
        model = Model([in_src_image, in_target_image], patch_out)
        # compile model
        opt = Adam(lr=0.0002, beta_1=0.5)
        model.compile(loss='binary_crossentropy', optimizer=opt, loss_
        return model

dis = _discriminator((256, 256, 3))
plot_model(dis)
```

Generator

Generator là một mô hình `encoder-decoder` sử dụng kiến trúc U-net như được giải thích ở mục II. Điểm đặc biệt của kiến trúc này đó là chúng ta sẽ sử dụng các kết nối tắt tới những layer có cùng shape ở nhánh `down sampling` (dùng để encoder) và `up sampling` (dùng để decoder). Các nhánh encoder và decoder sẽ sử dụng các khối block chuẩn là kết hợp của Convolution, BatchNormalization, Dropout và Relu. Các layer BatchNormalization, Dropout, Relu sẽ không làm thay đổi output shape nhưng nó sẽ có từng tác dụng riêng đối với mô hình. Cụ thể là BatchNormalization chuẩn hóa dữ liệu theo từng batch, giúp mô hình huấn luyện nhanh hơn.

top

Dropout sẽ ngắt ngẫu nhiên một tỷ lệ các kết nối và giảm overfitting cho mô hình. Relu là một layer tạo ra các biểu diễn phi tuyến. Convolution là layer chính mà chúng ta sẽ áp dụng để trích lọc đặc trưng và thay đổi shape của output. Những khối block kết hợp các layer này được sử dụng lặp lại trong kiến trúc của encoder và decoder. Vì vậy khi đọc các source code bạn thường thấy chúng được wrap vào một hàm block. Cụ thể như bên dưới.

Khởi tạo một block trong encoder của generator

```
def _encoder_block(layer_in, n_filters, batchnorm=True):
        # khởi tạo weight
        init = RandomNormal(stddev=0.02)
        # thêm downsampling layer
        g = Conv2D(n_filters, (4,4), strides=(2,2), padding='same', k
  # thêm tùy chọn batch normalization
        if batchnorm:
                g = BatchNormalization()(g, training=True)
        # activation
        g = LeakyReLU(alpha=0.2)(g)
        return g
```

Khởi tạo một block trong decoder của generator

```
def _decoder_block(layer_in, skip_in, n_filters, dropout=True):
        # khởi tạo weight
        init = RandomNormal(stddev=0.02)
        # thêm upsampling layer
        g = Conv2DTranspose(n_filters, (4,4), strides=(2,2), padding=
        # thêm batch normalization
        g = BatchNormalization()(g, training=True)
        # thêm tùy chọn dropout
        if dropout:
                g = Dropout(0.5)(g, training=True)
        # Kết nối tắt vói layers skip connection
        g = Concatenate()([g, skip_in])
        # activation
        g = Activation('relu')(g)
        return g
```

Ta thấy trong decoder chúng ta thay thế layer `Conv2D` bằng `Conv2DTranspose` để upsampling. Thêm vào đó layer `Concatenate` cũng được sử dụng ở decoder để thực hiện kết nối tắt.

Tiếp theo chúng ta sẽ khởi tạo kiến trúc generator.

Khởi tạo generator

Top

```
def _generator(image_shape=(256, 256, 3)):
        # Khởi tạo weight ngẫu nhiên
        init = RandomNormal(stddev=0.02)
        # input image
        in_image = Input(shape=image_shape)
        # nhánh encoder
        e1 = _encoder_block(in_image, 64, batchnorm=False)
        e2 = _encoder_block(e1, 128)
        e3 = _encoder_block(e2, 256)
        e4 = _encoder_block(e3, 512)
        e5 = _encoder_block(e4, 512)
        e6 = _encoder_block(e5, 512)
        e7 = _encoder_block(e6, 512)
        # bottleneck layer, không có batch norm và relu
        b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_
        b = Activation('relu')(b)
        # nhánh decoder
        d1 = _decoder_block(b, e7, 512)
        d2 = _decoder_block(d1, e6, 512)
        d3 = _decoder_block(d2, e5, 512)
        d4 = _decoder_block(d3, e4, 512, dropout=False)
        d5 = _decoder_block(d4, e3, 256, dropout=False)
        d6 = _decoder_block(d5, e2, 128, dropout=False)
        d7 = _decoder_block(d6, e1, 64, dropout=False)
        # output
        g = Conv2DTranspose(3, (4,4), strides=(2,2), padding='same',
        out_image = Activation('tanh')(g)
        # model
        model = Model(in_image, out_image)
        return model

gen = _generator(image_shape=(256, 256, 3))
plot_model(gen)
```

Discriminator sẽ được huấn luyện trên cặp ảnh source và target. Trong đó ảnh target là ảnh được sinh ra từ generator hoặc ảnh thật được lấy từ tập train. Model generator sẽ được huấn luyện thông qua discriminator. Quá trình huấn luyện sẽ diễn ra xen kẽ.

Loss function GAN

Một lưu ý khác là loss function là tổng cuả adversarial loss $\mathcal{L}_{cGAN}$ của cGAN cộng với $\mathcal{L}_{L_1}$. Thành phần loss L_1 là norm chuẩn bậc 1 tính khoảng cách giữa được sinh ra từ generator và ảnh target nên nó sẽ có tác dụng làm cho bức ảnh sinh ra giống với thật hơn khi khoảng cách này là nhỏ hơn. Hệ số λ của $\mathcal{L}_{L_1}$ thường được lựa chọn bằng 100.

Top

```
def _gan(g_model, d_model, image_shape):
  # đóng băng discriminator (không train)
        d_model.trainable = False
  # Truyền source image vào generator model
        in_src = Input(shape=image_shape)
        gen_out = gen(in_src)
  # concatenate source input vào generator output và truyền qua discr
        dis_out = dis([in_src, gen_out])
  # source image như là input, gan model sẽ sinh ảnh và phân loại out
        model = Model(in_src, [dis_out, gen_out])
        # compile model
        opt = Adam(lr=0.0002, beta_1=0.5)
  # sử dụng loss function là tổng của binary crossentropy và mae với
        model.compile(loss=['binary_crossentropy', 'mae'], optimizer=
        return model

gan = _gan(gen, dis, (256, 256, 3))
gan.summary()
```

Huấn luyện mô hình

Tiếp theo chúng ta sẽ chuẩn hóa dữ liệu image về khoảng $[-1, 1]$ cho toàn bộ các bức ảnh. Đây là một bước xử lý rất quan trọng và cơ bản ở hầu hết các mô hình computer vision.

```
def _standardize(src, tar):
        X1, X2 = src, tar
        # chuẩn hóa từ [0,255] về [-1,1]
        X1 = (X1 - 127.5) / 127.5
        X2 = (X2 - 127.5) / 127.5
        return [X1, X2]

train_src_norm, train_tar_norm = _standardize(train_src, train_tar)
val_src_norm, val_tar_norm = _standardize(val_src, val_tar)

print(train_src_norm.shape, train_tar_norm.shape)
print(val_src_norm.shape, val_tar_norm.shape)
```

Mô hình discriminator sẽ nhận 50% là ảnh real và 50% là ảnh fake nên tiếp theo sẽ cần phải tạo các hàm lấy ảnh real và fake từ mô hình.

```
# Lựa chọn một batch của mẫu ngẫu nhiên và trả về sources và targets
def _generate_real_samples(dataset, i_batch, n_samples, patch_shape):
        trainA, trainB = dataset
  # Khôi phục lại ảnh gốc
        X1, X2 = trainA[i_batch], trainB[i_batch]
  # Gán nhãn real = 1
        y = np.ones((n_samples, patch_shape, patch_shape, 1))
        return [X1, X2], y
```

Top

```
# Lựa chọn các mẫu fake và trả về sources và targets.
def _generate_fake_samples(g_model, samples, patch_shape):
        # Khởi tạo các mẫu fake từ generator
        X = g_model.predict(samples)
        # Gán nhãn fake = 0
        y = np.zeros((len(X), patch_shape, patch_shape, 1))
        return X, y
```

Để biết kết quả huấn luyện như thế nào sau một số bước hữu hạn, chúng ta cần sử dụng một hàm visualize kết quả. Hàm này sẽ sinh ảnh fake từ mô hình generator tại thời điểm hàm được gọi và so sánh kết quả mô hình giữa ảnh fake với real image.

```
# Thống kê mô hình performance
def _summarize_performance(g_model, dataset, n_samples=3):
  # Lựa chọn một mẫu của input images
        [X_realA, X_realB], _ = _generate_real_samples(dataset, i_bat
  # Tạo ra một batch của các fake samples
        X_fakeB, _ = _generate_fake_samples(g_model, X_realA, 1)
  # Visualize hình ảnh của source images
        for i in range(n_samples):
                plt.subplot(3, n_samples, 1 + i)
                plt.axis('off')
                plt.imshow(X_realA[i])
  # Visualize hình ảnh của target images
        for i in range(n_samples):
                plt.subplot(3, n_samples, 1 + n_samples + i)
                plt.axis('off')
                plt.imshow(X_fakeB[i])
  # Visualize target image
        for i in range(n_samples):
                plt.subplot(3, n_samples, 1 + n_samples*2 + i)
                plt.axis('off')
                plt.imshow(X_realB[i])
```

Cuối cùng chúng ta sẽ huẩn luyện mô hình GAN qua các epochs. Ở mỗi một batch huấn luyện chúng ta sẽ khởi tạo một tập ảnh real và một tập ảnh fake. Huấn luyện discriminator trên cả tập real và fake. Mô hình generator sau đó được huấn luyện thông qua loss function của GAN.

Top

```
# Huấn luyện pix2pix model
def _train(d_model, g_model, gan_model, dataset, n_epochs=100, n_batch
  # Xác định output square shape của discriminator
  n_patch = d_model.output_shape[1]
  trainA, trainB = dataset
  # Tính toán số lượng các batches trên một training epoch
  bat_per_epo = int(len(trainA) / n_batch)
  # Khởi tạo list indices thứ tự các quan sát
  idxs = np.arange(len(trainA))
  for epoch in range(n_epochs):
    # Shuffle lại idxs sau mỗi epoch
    np.random.shuffle(idxs)
    # Huấn luyện trên từng epoch
    for i in range(bat_per_epo):
      # Khởi tạo một batch của real samples
      i_batch = idxs[(i*n_batch):((i+1)*n_batch)]
      [X_realA, X_realB], y_real = _generate_real_samples(dataset, i_
      # Khởi tạo một batch của fake samples
      X_fakeB, y_fake = _generate_fake_samples(g_model, X_realA, n_pa
      # Cập nhật discriminator cho real samples
      d_loss1 = d_model.train_on_batch([X_realA, X_realB], y_real)
      # Cập nhật discriminator cho generated samples
      d_loss2 = d_model.train_on_batch([X_realA, X_fakeB], y_fake)
      # Cập nhật generator
      g_loss, _, _ = gan_model.train_on_batch(X_realA, [y_real, X_rea
      # Thống kê performance
      print('>%d %d d1[%.3f] d2[%.3f] g[%.3f]' % (epoch+1, i+1, d_los
      # Thống kê model performance
      if (i+1) % (bat_per_epo * 10) == 0:
        _summarize_performance(i, g_model, dataset)
    # Lưu lại model generator sau mỗi epoch
    filename = 'model_%06d.h5' % (epoch+1)
    g_model.save(filename)
    print('>Saved: %s' % (filename))
```

```
image_shape = train_src[0].shape
# xác định GAN model
gan_model = _gan(gen, dis, image_shape)
# train model
_train(dis, gen, gan_model, (train_src, train_tar), n_epochs=100, n_ba
```

Quá trình huấn luyện sẽ khá lâu, chúng ta sẽ có nghiệm tốt sau khoảng 50 epochs. Sau khi kết thúc quá trình huấn luyện thì sẽ sử dụng generator để sinh ảnh như sau:

```
gen.predict(val_src[0])
```

6. Kết luận

Như vậy qua bài viết này mình đã giới thiệu với các bạn ý tưởng chính của mô hình Pix2PixGAN. Đây là một mô hình conditional GAN và thuộc lớp bài toán image-to-image translation có nhiều ứng dụng trong thực tiễn như chuyển đổi màu sắc ảnh, tạo tranh vẽ theo phong cách hội họa,

Top

image segmentation,…. Việc kết hợp loss function giữa adversarial loss và norm chuẩn bậc 2 đã giúp cho mô hình Pix2Pix cải thiện được chất lượng output so với các phương pháp của cGAN trước đây. Và cuối cùng không thể thiếu là các tài liệu tham khảo mà mình đã tổng hợp cho bài viết này.

7. Tài liệu

1. Image-to-Image Translation with Conditional Adversarial Networks (https://arxiv.org/abs/1611.07004)
2. pix2pix - phillipi - github (https://github.com/phillipi/pix2pix)
3. pix2pix - phillipi - web (https://phillipi.github.io/pix2pix/)
4. pix2pix - tensorflow tutorial (https://www.tensorflow.org/tutorials/generative/pix2pix)
5. How to Develop a Pix2Pix GAN for Image-to-Image Translation (https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation/)
6. How to Implement Pix2Pix GAN Models From Scratch With Keras (https://machinelearningmastery.com/how-to-implement-pix2pix-gan-models-from-scratch-with-keras/)
7. Pix2pix model towardsdatascience (https://towardsdatascience.com/pix2pix-869c17900998)

Top