

Bài 40 - Image Segmentation

10 Jun 2020 - phamdinhkhanh

Menu

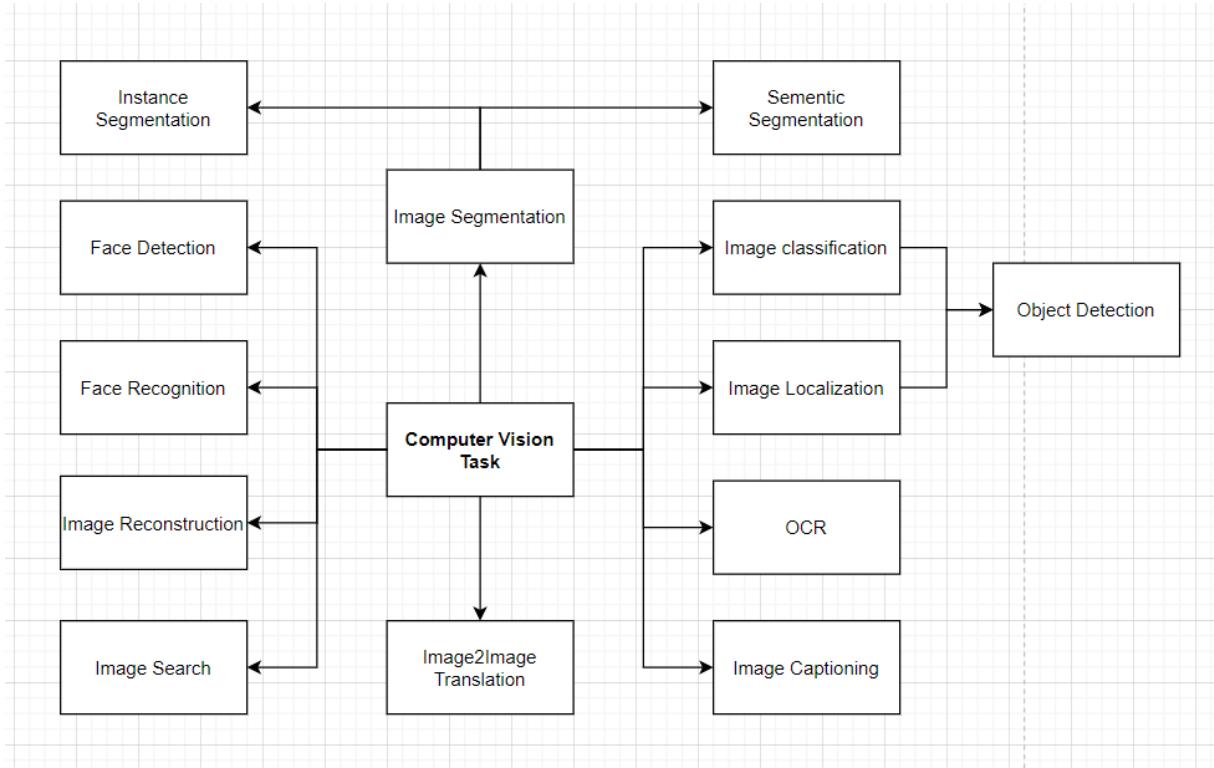
- 1. Khái quát các bài toán trong Computer Vision
 - 1.1. Các dạng bài toán trong Computer Vision
 - 1.2. Bài toán Image Segmentation
 - 1.3. Input và output của bài toán Image Segmentation
- 2. Các dạng image segmentation khác nhau
- 3. Các ứng dụng của Image Segmentation
- 4. Thuật ngữ:
- 5. Mạng giải chập (Deconvolutional Neural Network)
- 6. Upsampling 2D Layer
- 7. Tích chập chuyển vị (Transposed Convolution)
- 8. Tích chập giãn nở (Dilation Convolution)
- 9. Các thuật toán image segmentation
 - 9.1. Các phương pháp cổ điển.
 - 9.1.1. Sử dụng bộ lọc binary threshold
 - 9.1.2. k-mean clustering
 - 9.1.3. Expectation Maximization Clustering
 - 9.1.4. Mean shift Clustering
- 10. Thuật toán Mask - CNN
- 11. Unet (2012)
 - 11.1. Khởi tạo kiến trúc mạng Unet
- 12. FCN (2015)
 - 12.1. Khởi tạo Mô hình FCN trên tensorflow
 - 13. Tổng kết
- 14. Tài liệu tham khảo

1. Khái quát các bài toán trong Computer Vision

1.1. Các dạng bài toán trong Computer Vision

Các bài toán và ứng dụng của computer vision

Top



Hình 1: Sơ đồ các tác vụ của computer vision trong AI.

Các bài toán trong computer vision khá đa dạng. Image classification là lớp bài toán phổ biến nhất giúp chúng ta phân loại ảnh. Object detection thì không chỉ phân loại ảnh mà còn xác định vị trí của vật thể trong ảnh. Image Captioning kết hợp giữa ảnh và NLP để đưa ra một câu chú thích có nội dung phù hợp với bức ảnh. Nếu bạn là người yêu thích các ứng dụng của ảnh như chuyển nam thành nữ, nữ thành nam, chuyển da màu thành da đen, chuyển ảnh thành tranh vẽ thì có thể tìm hiểu về lớp bài toán Image-to-Image Translation. Những bài toán liên quan đến mặt người như Face Detection và Face Recognition là những bài toán có ứng dụng cao dùng trong chấm công và quản lý doanh nghiệp. Các hệ thống search ảnh của các search engine hoặc các trang thương mại điện tử thì sẽ cần các bài toán về trích suất nội dung ảnh như Image Search, Image Retrieval,... OCR là lớp bài toán nhận diện chữ quang học, đóng vai trò rất lớn trong việc số hóa văn bản và tài liệu. Để thiết kế được những hệ thống hiểu sâu về nội dung bức ảnh hơn là object detection và object classification thì ta sẽ cần tới các thuật toán Image Segmentation.

Dựa trên những hiểu biết còn hạn chế của mình thì các bạn đã hình dung được ứng dụng của computer vision rồi chứ? Sơ đồ trên có lẽ sẽ còn phải cập nhật thường xuyên vì khi một nhu cầu mới phát sinh thì AI lại sản sinh ra những lớp bài toán mới và luôn phát triển chứ không ngừng nghỉ.

Đằng sau những lớp bài toán trên là một câu chuyện dài của các nhà khoa học trong tiến trình khai phá học máy mà đòi hỏi phải có ý tưởng về mạng và về thu thập và gán nhãn dữ liệu.

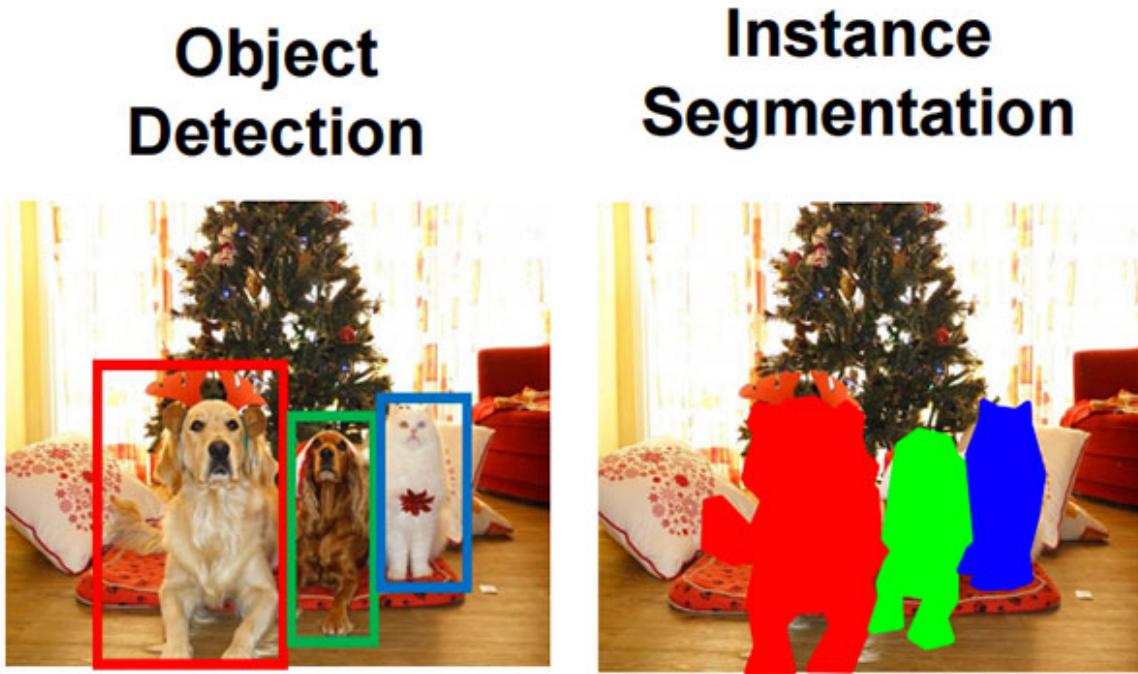
Tổ chức dữ liệu cho các bài toán

Mỗi một bài toán sẽ có một cách thức thiết kế mô hình và định dạng dữ liệu input/output chuyên biệt. Chẳng hạn như lớp bài toán phổ biến nhất trong computer vision là **Image Classification** Bài 8 - Convolutional Neural Network (<https://phamduinhkhanh.github.io/2019/08/22/convolutional-neural-network.html>) và Bài 38 - Các kiến trúc CNN hiện đại (<https://phamduinhkhanh.github.io/2020/05/31/CNNHistory.html>) sẽ chỉ cần ảnh và nhãn của ảnh. Tuy nhiên một số bức ảnh có nhiều vật thể xuất hiện thì chúng ta vừa phải tìm nhãn cho vật thể, vừa phải khoanh vùng vị trí của vật thể trên ảnh thông qua bounding box. Do đó lớp bài toán **Object Detection** ở Bài 12 - Các thuật toán Object Detection (<https://phamduinhkhanh.github.io/2019/09/29/OverviewObjectDetection.html>), bài 13 model SSD (<https://phamduinhkhanh.github.io/2019/10/05/SSDModelObjectDetection.html>), bài 25 model YOLO (<https://phamduinhkhanh.github.io/2020/03/09/DarknetAlgorithm.html>) thì ngoài nhãn còn cần thêm tọa độ bounding box.

Đối với các bài toán Image Segmentation thì mục tiêu của chúng ta là tìm ra vùng ảnh chứa vật thể nên chúng ta sẽ cần phải gán nhãn cho từng pixel.

1.2. Bài toán Image Segmentation

Tên của lớp bài toán là Image Segmentation có nghĩa là phân khúc hình ảnh, hàm ý rằng bài toán sẽ phân chia một hình ảnh thành nhiều vùng ảnh khác nhau. Image Segmentation cũng có chung mục tiêu như object detection là phát hiện ra vùng ảnh chứa vật thể và gán nhãn phù hợp cho chúng. Tuy nhiên tiêu chuẩn về độ chính xác của Image Segmentation ở mức cao hơn so với Object Detection khi nó yêu cầu nhãn dự báo đúng tới từng pixel.



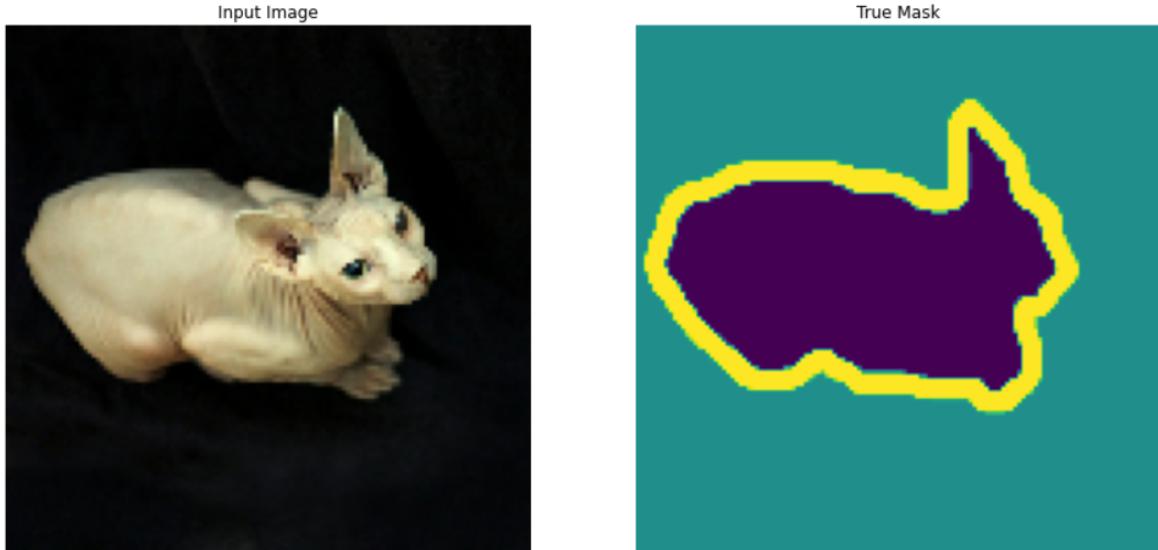
Hình 2: Phân biệt giữa thuật toán Object Detection và Instance Segmentation. Source : cs231n.stanford.edu

Mặc dù Image Segmentation yêu cầu về mức độ chi tiết cao hơn nhưng bù lại thuật toán giúp ta hiểu được nội dung của một bức ảnh ở mức độ sâu hơn khi chúng ta biết được đồng thời: **Vị trí** của vật thể trong ảnh, **hình dạng** của vật thể và **từng pixel nào thuộc về vật thể nào**.

1.3. Input và output của bài toán Image Segmentation

Image Segmentation nếu được huấn luyện theo bài toán học có giám sát trong thị giác máy tính thì sẽ yêu cầu gán nhãn cho ảnh. Input của bài toán là một bức ảnh và output là một ma trận mask mà giá trị của từng pixel đã được gán nhãn trên đó.

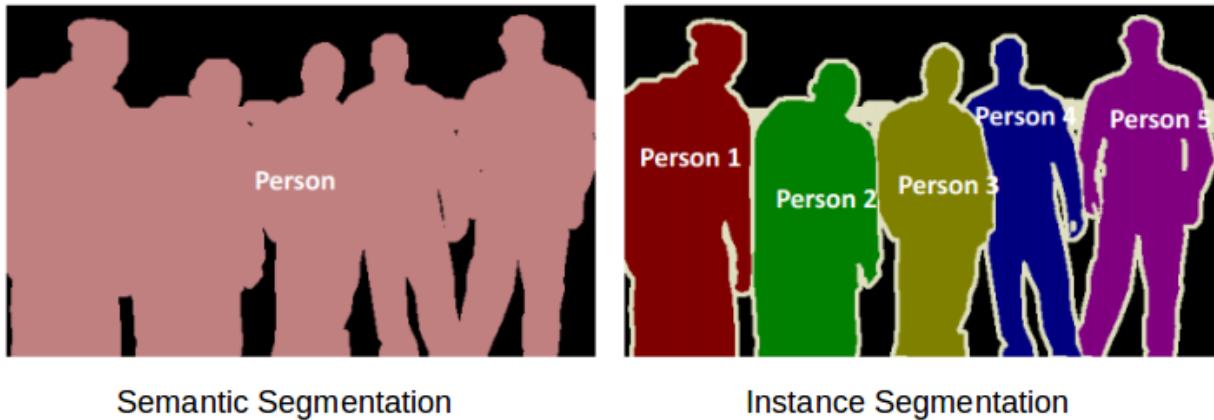
Top



Hình 3: Input (bên trái) và Output (bên phải) của mô hình Image Segmentation đối với bài toán một đối tượng. Mỗi một nhãn segment sẽ được thể hiện bởi một màu sắc khác nhau. Màu xám là nền, màu vàng là đường viền của ảnh và màu tím là nǎm bên trong vật thể.

Note: Bổ sung cụ thể một số dạng (Input, Output) của Image Segmentation trong một số bộ dữ liệu phổ biến

2. Các dạng image segmentation khác nhau



Có 2 bài toán image segmentation chính:

- Semantic segmentation: Chúng ta phân đoạn (segment) các vùng ảnh theo những nhãn khác nhau mà không phân biệt sự khác nhau giữa các đối tượng trong từng nhãn. Ví dụ trong hình ảnh bên trái chúng ta phân biệt được pixel nào thuộc về người và pixel nào thuộc về background. Tuy nhiên trong bức ảnh xuất hiện 5 người, mức độ phân chia sẽ không xác định từng pixel thuộc về người nào.
- Instance segmentation: Chúng ta phân đoạn các vùng ảnh chi tiết đến từng đối tượng trong mỗi nhãn. Ví dụ: ở hình ảnh bên phải đối với nhãn người sẽ được phân chia chi tiết tới từng người 1, 2, ..., 5.

3. Các ứng dụng của Image Segmentation

Image Segmentation có rất nhiều các ứng dụng trong y học, xe tự hành, xử lý ảnh vệ tinh.

- Y học: Trong y học, thuật toán Image Segmentation có thể hỗ trợ bác sĩ chuẩn đoán khối từ ảnh x-quang. Ưu điểm của Image Segmentation đó là không chỉ cho chúng ta biết vị trí của các

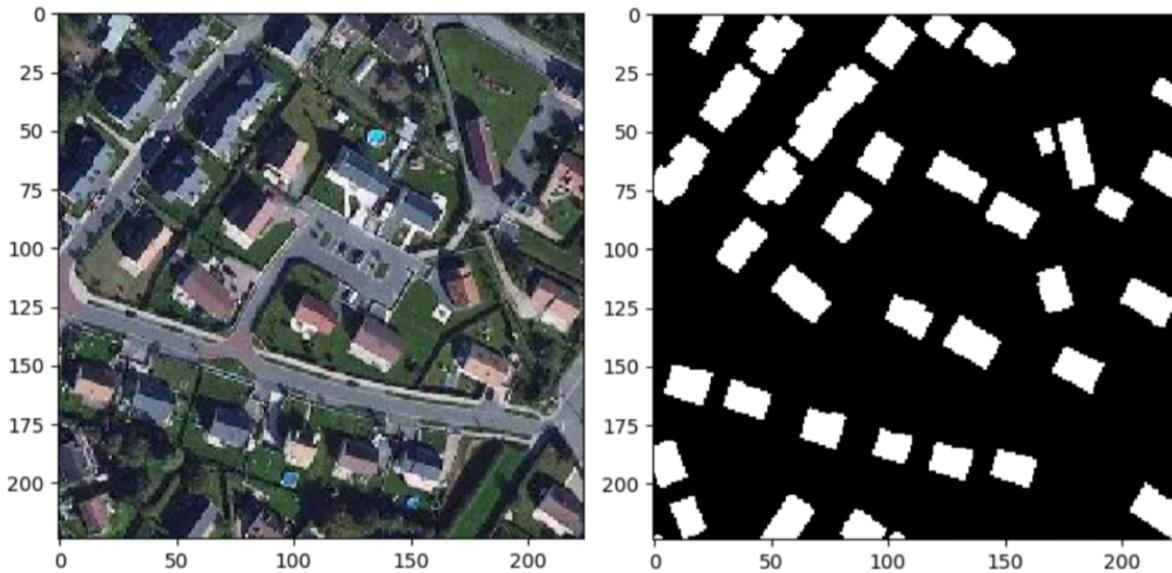
khối u trong ảnh mà còn cho chúng ta biết được hình dạng của chúng.



- Xe tự hành: Xe tự hành đòi hỏi phải liên tục nhận thức, xử lý và lên kế hoạch trong một môi trường phát triển liên tục. Vì yêu cầu an toàn tuyệt đối và độ chính xác cao trong mọi quyết định nên một hệ thống xe tự hành cần phải xác định chính xác các vật thể xuất hiện khi tham gia giao thông như người, đèn tín hiệu, biển báo, vạch kẻ đường, xe cộ.



- Xử lý ảnh vệ tinh: Các vệ tinh quay quanh trái đất sẽ liên tục thu thập hình ảnh bề mặt trái đất ở những vùng khác nhau. Từ các bức ảnh chụp vệ tinh, mô hình Image Segmentation sẽ phân đoạn hình ảnh thành tuyến đường, khu phố, biển cả, cây cối,....



- Ứng dụng trong nông nghiệp: Chúng ta có thể tiết kiệm được một lượng lớn thuốc trừ sâu trong nông nghiệp nhờ sử dụng hệ thống phun thuốc trừ sâu tự động có khả năng phân biệt được diện tích cỏ và cây trồng dựa trên thuật toán Image Segmentation. Khi diện tích cỏ lấn át so với cây trồng thì hệ thống sẽ tự động kích hoạt.
- Cảnh báo cháy rừng: Những hệ thống kiểm soát cháy rừng có thể segment được chính xác vị trí phát sinh các đám cháy từ ảnh chụp vệ tinh. Từ đó đưa ra cảnh báo về qui mô và mức độ lây lan của các đám cháy trên diện rộng.

Trên đây là một vài ứng dụng tiêu biểu của Image Segmentation. Còn rất nhiều các ứng dụng tiềm năng khác của thuật toán Image Segmentation đang được khai thác.

4. Thuật ngữ:

Trước khi tìm hiểu về các kiến trúc Image Segmentation tiêu biểu, mình sẽ giải thích một số thuật ngữ sử dụng trong bài:

- **Upsampling:** Các kỹ thuật giúp tăng kích thước output trong mạng CNN.
- **Tích chập chuyển vị** (Conv2DTranspose hoặc Transposed Conv2D): Một dạng tích chập giúp gia tăng kích thước của mạng CNN.
- **Mạng giải chập** (DeconvNet hoặc Deconvolutional Neural Network): Là một mạng áp dụng liên tiếp quá trình Upsampling để giải mã ngược trở lại từ đặc trưng sang ảnh. Thường áp dụng trong các bài toán Image2Image.
- **Vùng nhận thức** (Receptive Field): Là những vùng ảnh được áp dụng tích lũy với bộ lọc để tạo ra các pixel ở output.

Top

- **Ma trận giãn nở** (Dilation Matrix): Là ma trận được tạo thành từ ma trận gốc bằng cách padding xen kẽ các dòng và cột bằng giá trị 0.

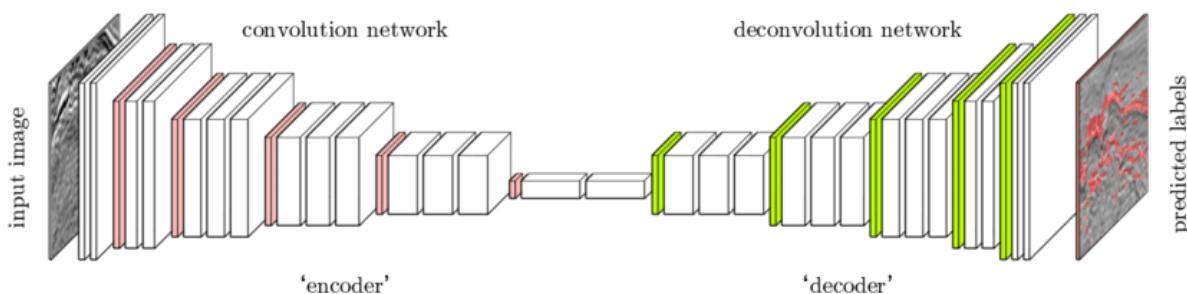
5. Mạng giải chập (Deconvolutional Neural Network)

Bạn hình dung các mạng CNN thông thường sẽ có kích thước giảm dần qua các layers để cuối cùng chúng ta thu được những đặc trưng bậc cao (high-level). Chức năng chính của CNN là chuyển từ ảnh sang đặc trưng. Vậy nếu muốn chuyển từ đặc trưng sang ảnh chúng ta sẽ cần thực hiện như thế nào?

Mạng giải chập sẽ giúp chúng ta thực hiện điều đó. Một mạng giải chập sẽ có kiến trúc chung là shape của các layers tăng dần. Qua từng layer mạng sẽ giải mã các khối đặc trưng thành những thông tin không gian của từng điểm ảnh và tạo thành một bức ảnh mới ở output.

Quá trình gia tăng kích thước tại các layers của mạng giải chập còn được gọi là Upsampling. Trong các bài toán classification và object detection đường như là chúng ta không sử dụng mạng giải chập bởi output của những bài toán này là xác định nhãn hoặc vị trí. Thế nhưng đối với các bài toán Image2Image (đầu vào là ảnh và trả ra output cũng là ảnh), chúng ta thường xuyên bắt gặp mạng giải chập và các layer có tác dụng Upsampling như Transposed Convolution, Dilation Convolution, Upsampling 2D. Bên dưới là một số ví dụ:

Ứng dụng trong các bài toán Image Segmentation



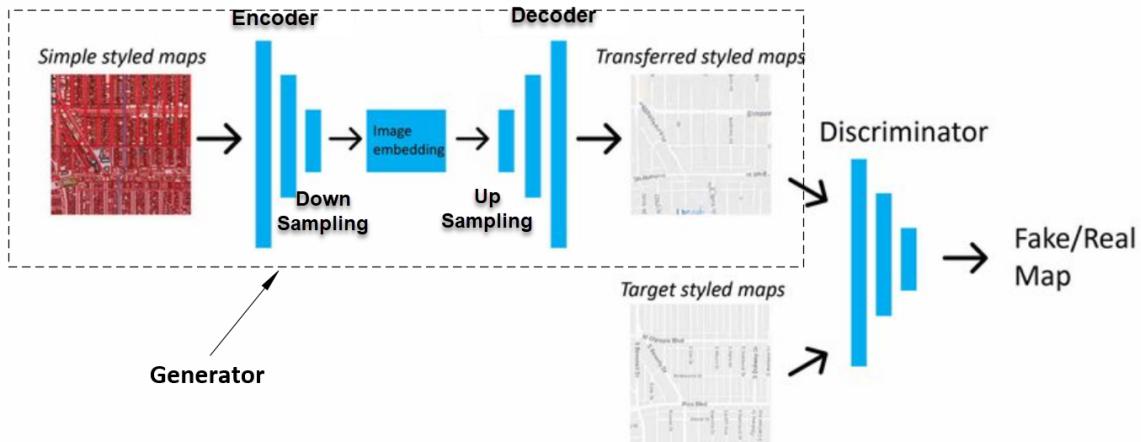
Hình 4: Mạng tích chập (Convolutional Neural Network) kết hợp với giải chập (Deconvolutonal Neural Network) trong các bài toán ImageSegmentation.

Nhánh bên trái của một mạng Segmentation là một CNN có nhiệm vụ như một encoder để tạo đặc trưng. Nhánh bên phải làm nhiệm vụ như một decoder giúp giải mã các features sang ảnh dự báo. Như vậy decoder sẽ làm nhiệm vụ ngược lại của encoder.

Ứng dụng của Upsampling trong GAN

Mạng giải chập cũng thường được áp dụng trong các kiến trúc của GAN ở lớp bài toán Image2Image translation. Hãy cùng phân tích kiến trúc pix2pix Conditional GAN để hiểu hơn điều này.

Top



Hình 5: Kiến trúc pix2pix.

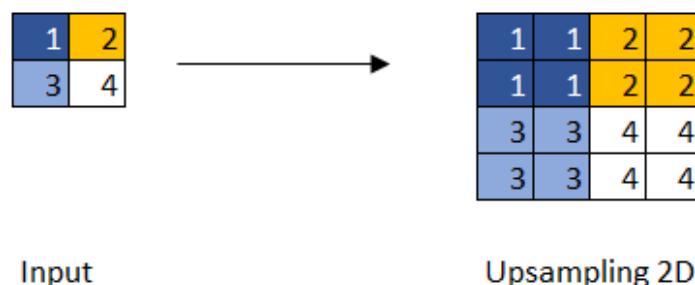
Giống như kiến trúc của một mô hình GAN thông thường, pix2pix cũng bao gồm 2 mô hình Generator và Discriminator. Mô hình Generator có tác dụng sinh ra ảnh fake gần giống với ảnh thật nhất, Discriminator sẽ làm nhiệm vụ phân loại ảnh fake và ảnh thật.

Quan sát kỹ hơn ở mô hình Generator bạn sẽ thấy có 2 nhánh đối lập là Encoder và Decoder. Trong đó nhánh Encoder là một mạng CNN thông thường đóng vai trò tạo đặc trưng cho ảnh input. Trên nhánh Encoder kích thước output giảm dần qua các layer. Nhánh Decoder đối xứng với Encoder có tác dụng mapping từ output của Encoder sang một ảnh mới có kích thước lớn hơn nó.

Vậy làm cách nào ta có thể biến đổi từ một khối output có kích thước nhỏ hơn sang một khối lớn hơn? Đó là nhờ các layer Upsampling mà chúng ta sẽ cùng tìm hiểu bên dưới.

6. Upsampling 2D Layer

Để dễ hình dung thì chức năng của Upsampling 2D layer cũng tương tự như hàm resize với kích thước lớn hơn ảnh input trong opencv bằng cách copy các giá trị pixel liền kề theo các window size.



Trên tensorflow bạn đọc có thể áp dụng UpSampling 2D thông qua layer UpSampling2D như sau:

Top

```

1 import tensorflow as tf
2 import numpy as np
3
4 input_shape = (1, 2, 2, 1)
5 x = np.arange(1, 5, 1).reshape(input_shape)
6 print('x \n', x)
7 upLayer = tf.keras.layers.UpSampling2D(size=(2, 2))
8 y = upLayer(x)
9 print('y: \n', y)

1 x
2 [[[1]
3 [2]]
4
5 [[3]
6 [4]]]
7 y:
8 tf.Tensor(
9 [[[1]
10 [1]
11 [2]
12 [2]]
13
14 [[1]
15 [1]
16 [2]
17 [2]]
18
19 [[3]
20 [3]
21 [4]
22 [4]]]
23
24 [[3]
25 [3]
26 [4]
27 [4]]]], shape=(1, 4, 4, 1), dtype=int64)

```

Ngoài phương pháp upsampling bằng copy giá trị của pixel còn có những phương pháp khác như Bilinear Interpolation, Max-Unpooling, Bed of Nails. Ý tưởng của các phương pháp này khá đơn giản và bạn đọc cũng không cần phải nhớ chúng.

7. Tích chập chuyển vị (Transposed Convolution)

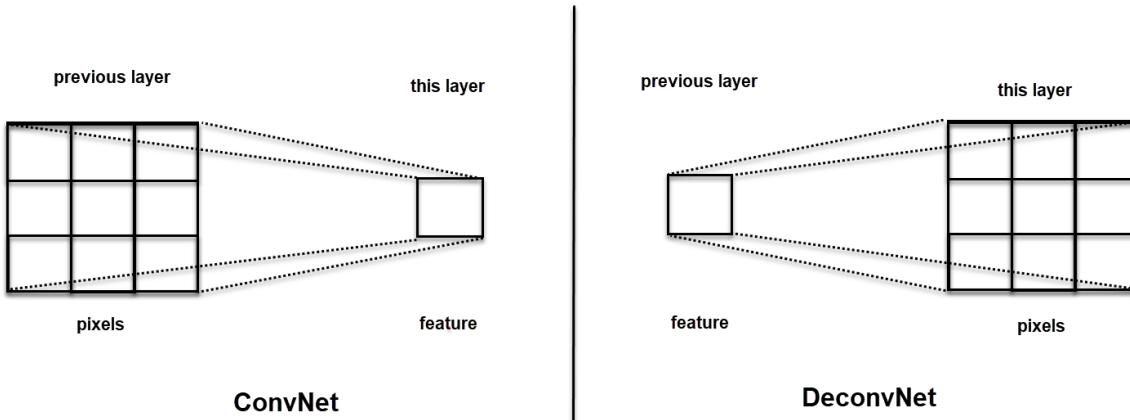
Ngoài gia tăng kích thước thông qua Upsampling, chúng ta có thể thực hiện theo cách phức tạp hơn thông qua tích chập chuyển vị (Transposed Convolution hoặc Conv2DTranspose). Vai trò của tích chập chuyển vị xuất phát từ nhu cầu biến đổi theo quá trình ngược lại của mạng tích chập thông thường hay còn gọi là giải chập (Deconvolutional Neural Network). Giả sử từ ma trận đầu vào có kích thước (w_1, h_1) sau khi áp dụng phép tích chập thông thường ta thu được kích thước (w_2, h_2) . Tích chập chuyển vị sẽ biến đổi từ một ma trận có kích thước (w_2, h_2) của output sang ma trận có kích thước (w_1, h_1) của input trong khi vẫn duy trì được các kiểu kết nối phù hợp với tích chập. Xin trích dẫn:

Top

The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution

A Guide To Convolution Arithmetic For Deep Learning, 2016 (<https://arxiv.org/abs/1603.07285>).

Trên thực tế thì có thể coi tích chập chuyển vị là một quá trình ngược của tích chập thông thường khi mỗi một đặc trưng (feature) được mapping sang các pixels ảnh thay vì ngược lại từ các pixels sang đặc trưng (feature).

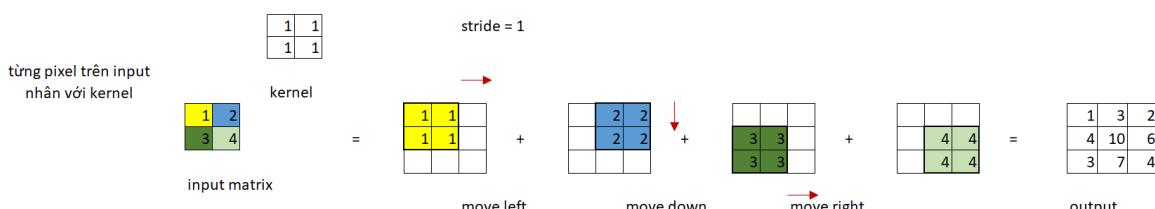


Xin trích dẫn:

A deconvnet can be thought of as a convnet model that uses the same components (filtering, pooling) but in reverse, so instead of mapping pixels to features does the opposite.

Visualizing and Understanding Convolutional Networks, 2013 (<https://arxiv.org/abs/1311.2901>).

Bạn đọc sẽ dễ dàng hình dung hơn về tích chập chuyển vị qua ví dụ minh họa bên dưới:



Cách tính tích chập:

Ta di chuyển các pixel của ma trận đầu vào từ trái qua phải và từ trên xuống dưới. Sau đó lấy giá trị của pixel nhân với ma trận bộ lọc sẽ thu được ma trận output có kích thước tương đương. Tùy vào stride qui định là bao nhiêu mà ta sẽ di chuyển kết quả của mỗi lần nhân pixel với bộ lọc sang bấy nhiêu đơn vị. Sau cùng ta tính tổng các vị trí tương ứng của các ma trận kết quả để thu được ma trận chuyển vị. Trong trường hợp stride không bằng kích thước kernel thì ma trận kết quả tích chập sẽ overlapping lén nhau. Khi đó ta sẽ cộng dồn chúng.

Tính kích thước cho output:

Có 3 tham số chính ảnh hưởng đến kích thước của output trong phép nhân tích chập chuyển vị bao gồm số bước nhảy S , kích thước bộ lọc F . Công thức chung được tính như sau:

$$W_{out} = (W - 1) \times S + F \quad (1)$$

Top

Trong trường hợp có thêm padding P vào output thì W_{out} có giá trị:

$$W_{out} = (W - 1) \times S + F + P \quad (2)$$

Ví dụ: $W = 2, S = 3, P = 2, F = 2$ thì output có kích thước là:

$$W_{out} = (2 - 1) \times 3 + 2 + 2 = 7$$

Từ công thức W_{out} suy ngược ra W :

$$W = \frac{W_{out} - F - P + S}{S} \quad (3)$$

Việc chứng minh các công thức trên khá dễ dàng và xin dành cho bạn đọc.

Trên tensorflow ta thực hiện tích chập chuyển vị như sau:

```

1 import tensorflow as tf
2 import numpy as np
3
4 W=2
5 S=3
6 P=2
7 F=2
8
9 input_shape = (1, W, W, 1)
10 x = np.random.uniform(size = input_shape).astype(np.float32)
11 print('x: \n', x)
12 con2DTran = tf.keras.layers.Conv2DTranspose(filters = 1,
13                                              kernel_size=F,
14                                              strides=S,
15                                              output_padding = P,
16                                              input_shape=(2, 2, 1))
17 y = con2DTran(x)
18 print('y: \n', y.shape)

1 x:
2 [[[[0.5444585
3 [0.24255177]
4
5 [[0.07139957
6 [0.4786547 ]]]
7 y:
8 (1, 7, 7, 1)

```

Một tính chất thú vị: Nếu chúng ta truyền ma trận X vào một layer tích chập chuyển vị để thu được $Y = f(X)$ và tạo ra một layer tích chập g có tham số như f để biến đổi ngược lại từ Y thì kết quả $g(Y)$ thu được có shape bằng với X .

```

1 conv2D_y = tf.keras.layers.Conv2D(filters=1, kernel_size=F, strides=S)(y)
2 conv2D_y.shape == x.shape

```

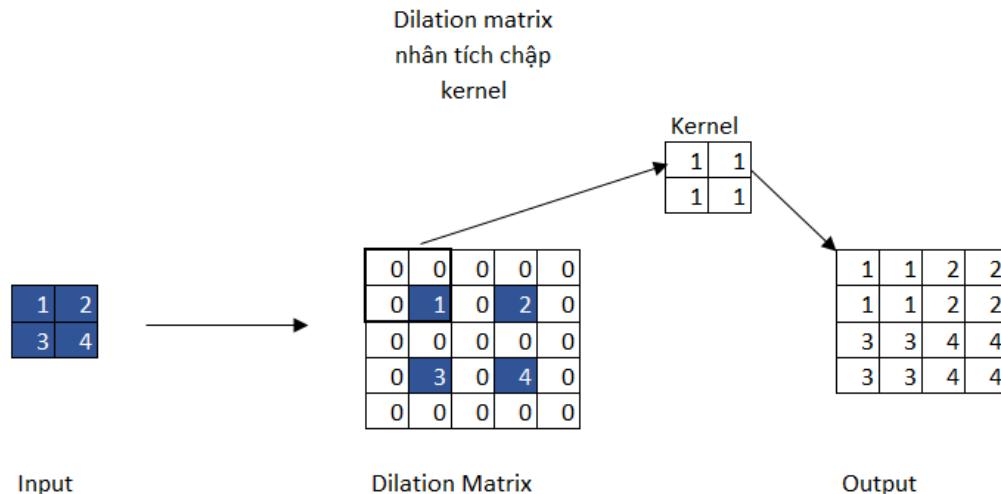
1 True

Đó là bởi kết quả của tích chập chuyển vị là quá trình biến đổi ngược lại của tích chập.

Top

8. Tích chập giãn nở (Dilation Convolution)

Tích chập giãn nở (Dilation Convolution) cũng là một phương pháp thường được áp dụng trong các mạng giải chập.



Hình 6: Từ ma trận input gốc bên trái, ta padding xen kẽ các dòng và cột 0 vào ma trận input và thu được ma trận dilation (ở giữa trong hình). Phép tích chập được thực hiện trên ma trận dilation. Kích thước ma trận sau tích chập tăng từ 2x2 lên 4x4 .

Mỗi pixel ở ma trận output được tính theo tích element-wise product giữa vùng nhận thức (receptive field) với bộ lọc.

$$o_{ij} = \sum_{\text{sum element}} \left(\begin{bmatrix} 1, 0 \\ 0, 0 \end{bmatrix} \otimes \begin{bmatrix} 1, 1 \\ 1, 1 \end{bmatrix} \right) = \sum_{\text{sum element}} \begin{bmatrix} 1, 0 \\ 0, 0 \end{bmatrix} = 1 + 0 + 0 + 0 = 1$$

Cách thực hiện tương tự như đối với phép tích chập CNN tại Bài 8 - Convolutional Neural Network (<https://phamduinhkhanh.github.io/2019/08/22/convolutional-neural-network.html>).

9. Các thuật toán image segmentation

9.1. Các phương pháp cổ điển.

Hầu hết các phương pháp image segmentation cổ điển đều là những phương pháp học không giám sát. Chúng ta không cần phải xác định trước nhãn cho từng pixel thuộc về đối tượng nào. Do đó dẫn tới hạn chế là các giá trị segment của ảnh khá ngẫu nhiên và không định nghĩa được các nhãn cần segment. Tiếp theo ta sẽ tìm hiểu một số phương pháp segment ảnh theo phương pháp cổ điển.

9.1.1. Sử dụng bộ lọc binary threshold

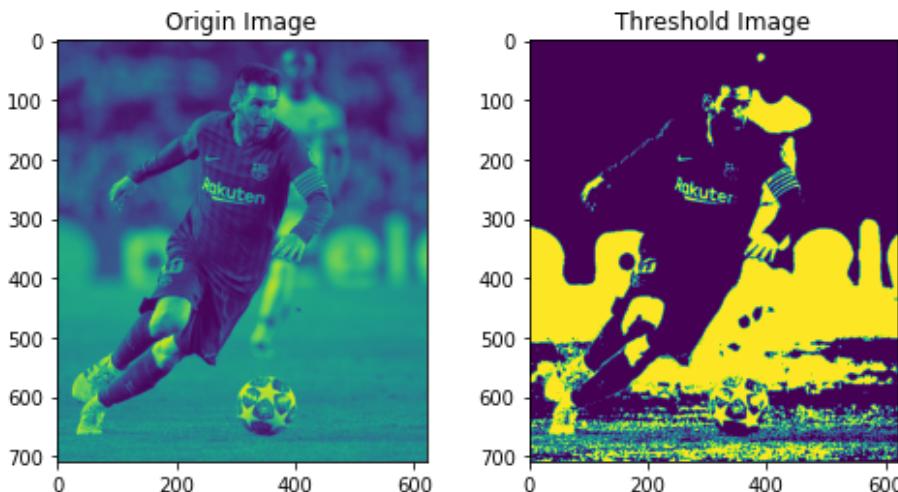
Các vùng ảnh sẽ được chuyển về dạng đen trắng dựa trên cường độ sáng của chúng lớn hoặc nhỏ hơn một ngưỡng cố định. Phương pháp này cho kết quả biến đổi theo cường độ của ngưỡng và thường không chuẩn xác đối với ảnh không có phân vùng màu sắc rõ ràng. Bạn đọc có thể xem thêm

Opencv Image Thresholding (http://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html)

```

1   import cv2
2   import numpy as np
3   from PIL import Image
4   import requests
5   from io import BytesIO
6   import matplotlib.pyplot as plt
7
8   url = 'https://i.imgur.com/1vzDG2J.jpg'
9   def _downloadImage(url):
10      resp = requests.get(url)
11      img = np.asarray(bytearray(resp.content), dtype="uint8")
12      img = cv2.imdecode(img, cv2.IMREAD_COLOR)
13      return img
14
15      # Download image
16      img = _downloadImage(url)
17      # Gray convert
18      img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
19
20      # Filter threshold according to minimum is 127
21      ret, th_img = cv2.threshold(img_gray, 127, 255, cv2.THRESH_BINARY)
22
23      # Visualize
24      fg, ax = plt.subplots(1, 2, figsize = (8, 4))
25      for i, image in enumerate([img_gray, th_img]):
26          ax[i].imshow(image)
27          if i == 0:
28              ax[i].set_title('Origin Image')
29          else:
30              ax[i].set_title('Threshold Image')

```



9.1.2. k-mean clustering

Thuật toán k-mean clustering sẽ phân cụm cường độ của các pixels trên ảnh thành k clusters. Sau đó giá trị của mỗi pixels sẽ được thay thế bởi centroids của chúng để segment hình ảnh.

Top



Hình 7: Các phân vùng của ảnh có ranh giới bị chồng lấn. Ví dụ như một phần rìa của gương mặt, vùng mắt và môi cũng bị lẫn sang màu đỏ của khăn. Nếu chúng ta muốn xác định vùng đối tượng cần segment là toàn bộ gương mặt không phân biệt mắt, môi và rìa thì thuật toán thường như không chuẩn xác.

Chúng ta segmentation hình ảnh bằng thuật toán k-mean clustering trên sklearn như sau:

```

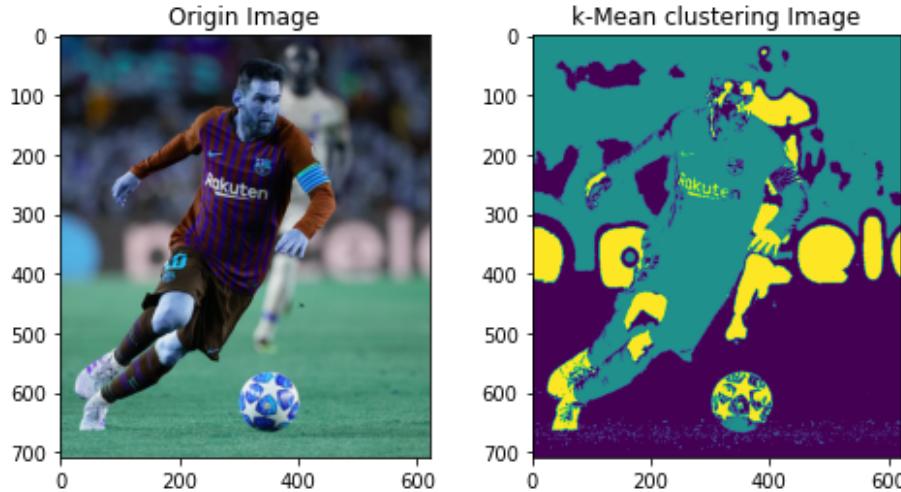
1   import cv2
2   import numpy as np
3   from PIL import Image
4   import requests
5   from io import BytesIO
6   import matplotlib.pyplot as plt
7
8   img = _downloadImage(url)

1   from sklearn.cluster import KMeans
2
3   # Reshape X into tensor2D: (width x height, n_channels)
4   X = img.reshape((-1, 3))
5   # Kmeans clustering with 3 clusters
6   kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
7   print('Centers found by scikit-learn:')
8   print(kmeans.cluster_centers_)
9   pred_label = kmeans.predict(X)

10  # Reshape pred_label
11  X_img = pred_label.reshape(img.shape[:2])
12
13
14  # Display image clustering
15  fg, ax = plt.subplots(1, 2, figsize = (8, 4))
16  for i, image in enumerate([img, X_img]):
17      ax[i].imshow(image)
18      if i == 0:
19          ax[i].set_title('Origin Image')
20      else:
21          ax[i].set_title('k-Mean clustering Image')

1   Centers found by scikit-learn:
2   [[ 86.33225972 140.33544883 127.99777344]
3    [ 43.29972249 41.67322892 54.91830723]
4    [161.73683501 176.1473516 191.64734197]]
```

Top



hạn chế của k-mean clustering là tốn kém chi phí tính toán vì khi huấn luyện cần tính khoảng cách từ centroids tới toàn bộ các pixels. Khía cạnh hạn chế khác là không rõ nên chọn bao nhiêu cluster là phù hợp.

9.1.3. Expectation Maximization Clustering

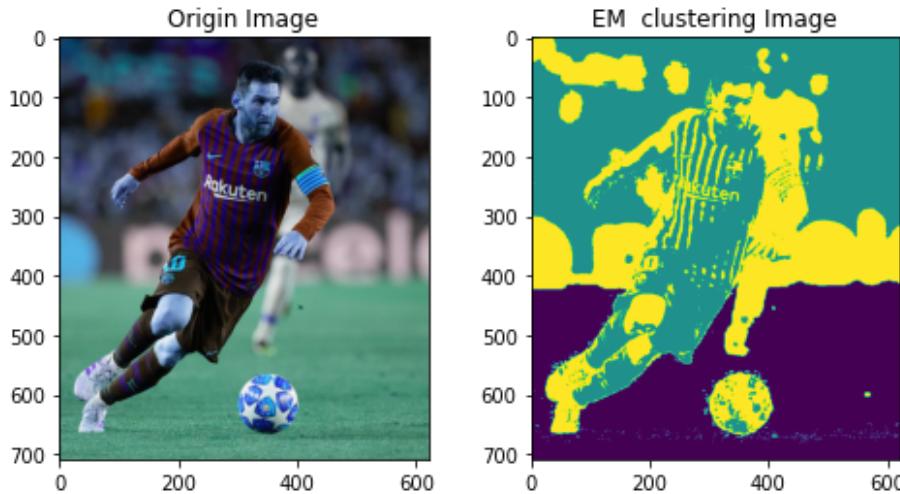
Là phương pháp phát triển hơn của k-Mean clustering. Nó không chỉ là thuật toán clustering thông thường mà còn tìm ra ước lượng hợp lý tối đa (maximum likelihood estimator) trong các parametric models. Khi đó các clusters được biểu diễn bởi phân phối xác suất thay cho trung bình.

Chúng ta segmentation hình ảnh bằng thuật toán Expectation Maximization Clustering như sau:

```

1   from sklearn.mixture import GaussianMixture
2
3   em = GaussianMixture(n_components=3,
4                       covariance_type='full', max_iter=20, random_state=0).fit(X
5
6   pred_label = em.predict(X)
7
8   # Reshape pred_label
9   X_img = pred_label.reshape(img.shape[:2])
10
11  # Display image clustering
12  fg, ax = plt.subplots(1, 2, figsize = (8, 4))
13  for i, image in enumerate([img, X_img]):
14      ax[i].imshow(image)
15      if i == 0:
16          ax[i].set_title('Origin Image')
17      else:
18          ax[i].set_title('EM clustering Image')
```

Top



9.1.4. Mean shift Clustering

k-means và Expectation Maximization yêu cầu chúng ta phải xác định trước số lượng các clusters. Sẽ rất khó xác định chính xác số lượng clusters vì nó biến động tùy theo ảnh đầu vào.

Mean shift clustering khắc phục được nhược điểm này khi tự động tìm ra được số lượng các clusters là hợp lý nhất cho mỗi một bức ảnh.

Ý tưởng của Mean shift khá đơn giản và dựa trên phân phối histogram của bức ảnh. Nếu bạn đọc chưa biết về phân phối histogram của ảnh là gì vui lòng xem lại Bài 17 - Thuật toán HOG (<https://phamdinhkhanh.github.io/2019/11/22/HOG.html>).

Giải thích ngắn gọn thì phân phối histogram là một mã hóa phân phối theo các bins giữa độ lớn gradient theo phương gradient trên toàn bộ grid cells của ảnh. Mỗi một bức ảnh sẽ có một véc tơ histogram đặc trưng riêng.

Thuật toán Mean Shift:

Bạn đọc có thể bỏ qua phần này nếu không quan tâm tới toán.

Cho n điểm $\mathbf{x}_i \in \mathbb{R}^d$, một kernel mật độ xác suất \hat{f}_K đa biến được ước lượng dựa trên một hàm kernel đối xứng $K(\mathbf{x})$ (thông thường là Gaussian hoặc Epanechnikov) theo công thức:

$$\hat{f}_K = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \quad (1)$$

ở đây h (được gọi là tham số bandwidth) xác định bán kính của kernel. Hàm kernel đối xứng (radially symmetric kernel) được xác định như sau:

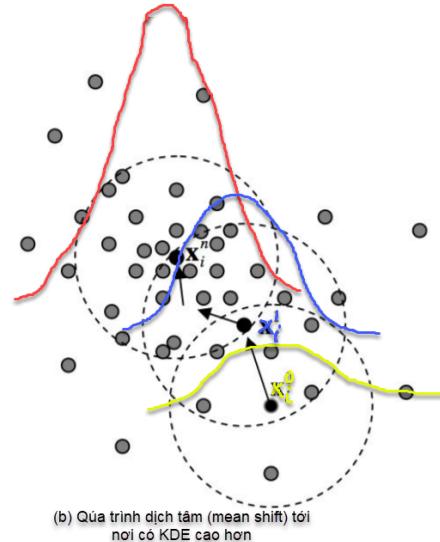
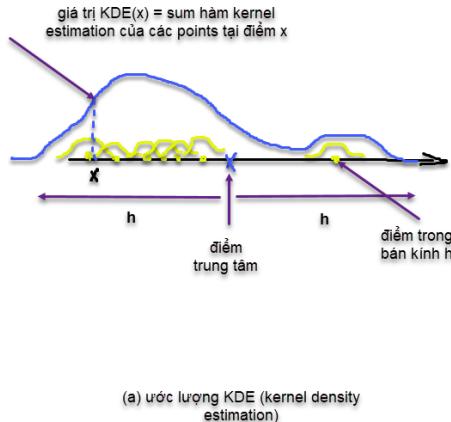
$$K(\mathbf{x}) = c_k k(||\mathbf{x}||^2)$$

với c_k là hằng số chuẩn hóa và $||\mathbf{x}||$ là norm chuẩn bậc 2 có công thức như sau:

$$||\mathbf{x}|| = \sqrt{x_1^2 + x_2^2 + \dots + x_d^2}$$

Ý nghĩa của công thức (1) chính là ước lượng kernel mật độ xác suất theo hàm kernel đối xứng của toàn bộ các điểm trong bán kính bandwidth. Quá trình ước lượng kernel mật độ xác suất KDE đã được tôi giới thiệu tại Bài 11 - Visualization trong python (<https://phamdinhkhanh.github.io/2019/09/16/VisualizationPython.html#21-density-plot>).

Top



Hình 8: Hình (a) Quá trình ước lượng KDE. Giá trị của KDE chính là \hat{f}_K ở công thức (1). Hình (b) mô tả quá trình dịch chuyển tâm thông qua tối đa hóa \hat{f}_K . Đường dịch chuyển của tâm sẽ từ \mathbf{x}_i^0 tới $\mathbf{x}_i^1, \dots, \mathbf{x}_i^n$.

Cập nhật gradient descent:

Để đơn giản hóa tính toán gradient descent, đặt

$$\mathbf{y} = \left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2$$

và $g(\mathbf{x}) = -k'(\mathbf{x})$. Khi đó:

$$\nabla_{\mathbf{x}} \hat{f}(\mathbf{x}) = \frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^n g(\mathbf{y}) \mathbf{x}_i - \sum_{i=1}^n g(\mathbf{y}) \mathbf{x} \right] = \frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^n g(\mathbf{y}) \right] \left[\frac{\mathbf{x}_i}{\sum_{i=1}^n g(\mathbf{y})} - \mathbf{x} \right]$$

Phân tích đạo hàm ta có: Phần tử đầu tiên $\frac{2c_{k,d}}{nh^{d+2}} [\sum_{i=1}^n g(\mathbf{y})]$ là tỷ trọng phân phối của hàm mật độ xác suất tại điểm \mathbf{x} . Phần tử thứ hai $\left[\frac{\mathbf{x}_i}{\sum_{i=1}^n g(\mathbf{y})} - \mathbf{x} \right]$ được gọi là mean shift vector, kí hiệu là \mathbf{m} mà di chuyển theo vector này sẽ giúp gia tăng tối đa mật độ xác suất.

Quá trình dịch chuyển tâm cụ thể như sau:

1. Tính toán vector mean shift \mathbf{x}_i^t .
2. Dịch chuyển vòng tròn (hoặc cửa sổ) ước lượng mật độ xác suất bằng cách dịch tâm: $\mathbf{x}^{(t+1)}_i = \mathbf{x}_i^t + \mathbf{m}(\mathbf{x}_i^t)$
3. Lặp lại quá trình 1 và 2 cho tới khi $\hat{f}(\mathbf{x})$ hội tụ.

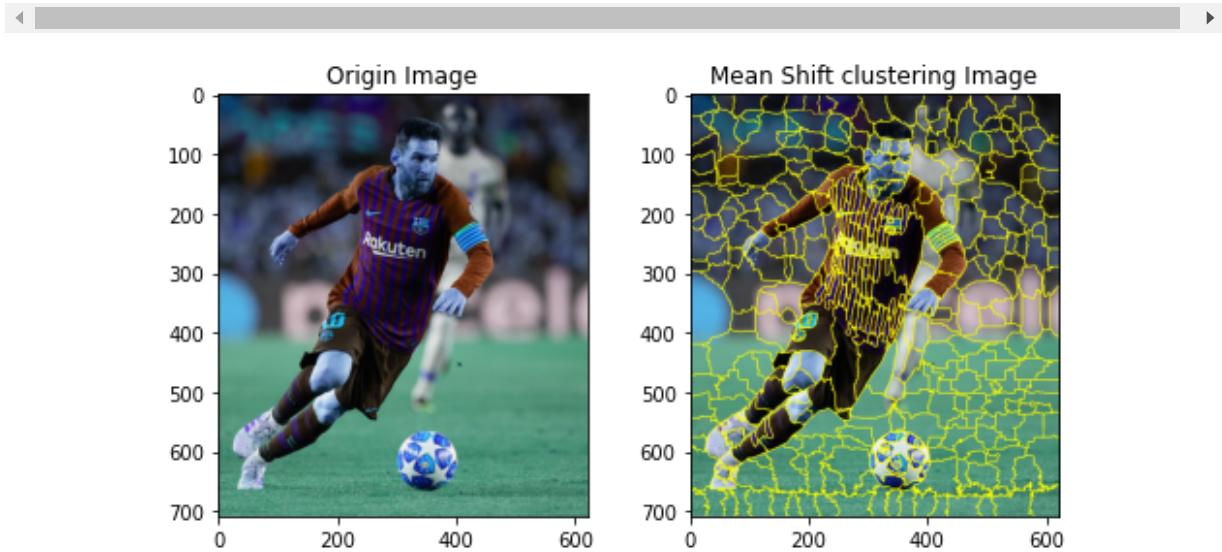
Chúng ta segmentation theo thuật toán Mean shift sử dụng skimage như sau:

Top

```

1  from skimage.segmentation import quickshift
2  from skimage.segmentation import mark_boundaries
3
4  segments_quick = quickshift(img, kernel_size=5, max_dist=10, ratio=0.5)
5
6  # Display image clustering
7  fg, ax = plt.subplots(1, 2, figsize = (8, 4))
8
9  ax[0].imshow(img)
10 ax[0].set_title('Origin Image')
11 ax[1].imshow(mark_boundaries(img, segments_quick))
12 ax[1].set_title('Mean Shift clustering Image')

```



Tiếp theo chúng ta sẽ tìm hiểu tới những kiến trúc hiện đại hơn của deep learning trong bài toán Image Segmentation.

10. Thuật toán Mask - CNN

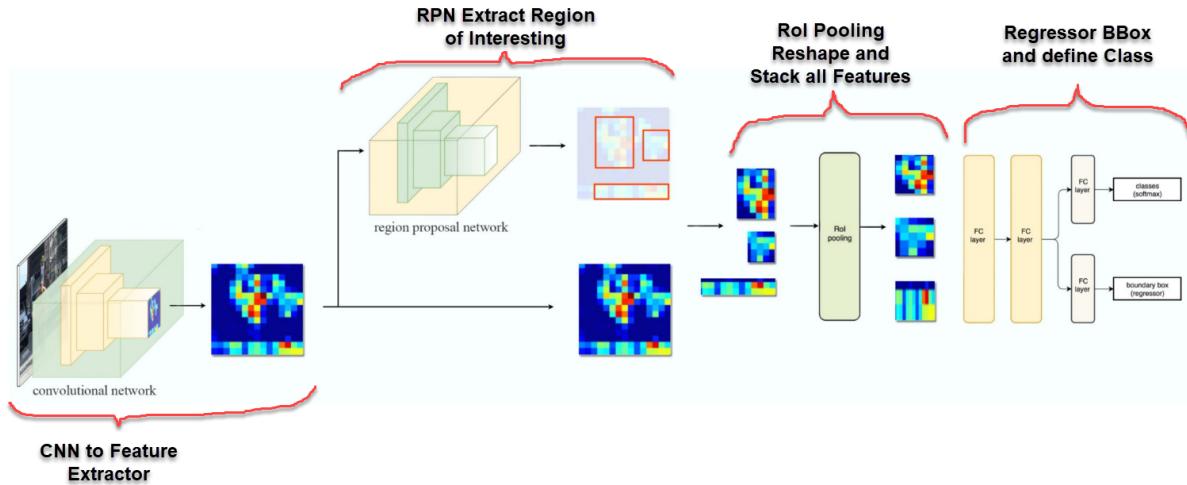
Paper: Mask R-CNN (<https://arxiv.org/pdf/1703.06870.pdf>)

Author: Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick - Facebook AI Research

Code: pytorch-mask-rcnn (<https://github.com/multimodallearning/pytorch-mask-rcnn>)

Như chúng ta đã tìm hiểu ở Bài 12 - Các thuật toán Object Detection (<https://phamduinhkhanh.github.io/2019/09/29/OverviewObjectDetection.html#43-faster-r-cnn-2016>) thì Faster R-CNN là một trong những thuật toán thuộc họ R-CNN. Mình sẽ tóm tắt lại các bước của thuật toán này một cách ngắn gọn:

Top



Hình 9: Kiến trúc của pipeline trong mạng Faster R-CNN. Source Jonathan-Hui (https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272)

Từ hình vẽ các bạn hình dung ra toàn bộ các xử lý của Faster R-CNN rồi chứ?

- Đầu tiên một base network (hoặc backbone) là một mạng CNN có tác dụng trích lọc đặc trưng ảnh. Chúng ta sẽ truncate output của mạng CNN và chỉ lấy feature map tại layer cuối.
- Feature map tiếp tục được huấn luyện qua một mạng RPN để tìm ra các RoI (là những vùng có khả năng cao chứa vật thể).
- RoI sẽ được crop từ feature map và sử dụng RoI pooling layer để reshape và stack các RoI thành một khối cùng kích thước.
- Khối RoI tiếp tục được truyền qua các layer fully connected và tách làm 2 nhánh, một nhánh dự báo BBox (bounding box) và một nhánh dự báo nhãn.

Nguyên lý hoạt động khá đơn giản phải không các bạn? Mask R-CNN có kiến trúc kế thừa lại Faster R-CNN nhưng add thêm nhánh mask có nhiệm vụ dự báo nhãn trên từng pixel.



Hình 10: Kiến trúc Mask R-CNN. Source Jonathan-Hui (https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272)

Một nhánh mới được add thêm ngay sau RoI Pooling để dự đoán mask cho ảnh.

Kích thước của từng kernel size và layer được thể hiện cụ thể hơn trong bài báo gốc như sau:



Hình 11: Bên trái là mô hình Faster R-CNN mà tác giả áp dụng đối với base network là mạng ResNet và bên phải là mạng FPN. Đối với kiến trúc FPN thì ở nhánh mask tác giả Upsampling bằng một mạng giải chấp để tăng kích thước từ 14x14 lên 28x28 .

11. Unet (2012)

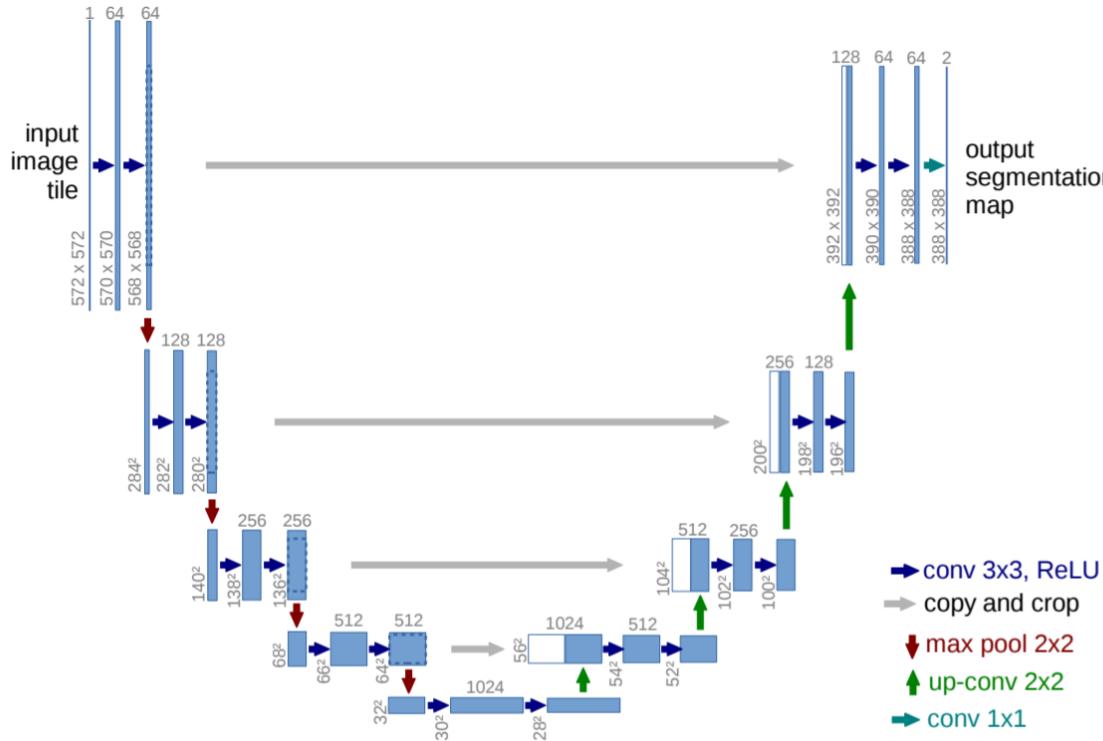
Paper: U-Net: Convolutional Networks for Biomedical Image Segmentation (<https://arxiv.org/abs/1505.04597>)

Author: Olaf Ronneberger, Philipp Fischer, and Thomas Brox - University of Freiburg, Germany

Code: unet - zhixuhao (<https://github.com/zhixuhao/unet>)

Unet là một kiến trúc được phát triển bởi Olaf Ronneberger và các cộng sự phát triển nhằm phân vùng các cấu trúc não ron thần kinh trong não người. Kiến trúc này lần đầu áp dụng đã dành **Top 1** chiến thắng trong cuộc thi EM segmentation challenge at ISBI 2012

(http://brainiac2.mit.edu/isbi_challenge/home). Bí kíp nào đã giúp Unet làm nên thành công trong cuộc thi này? Hãy phân tích kiến trúc của chúng:



Hình 12: Kiến trúc mô hình Unet. Mỗi một thanh chữ nhật màu xanh là một feature map đa kênh. Kích thước width x height được kí hiệu góc trái bên dưới của thanh chữ nhật và số lượng channels được kí hiệu trên đỉnh của feature map. Các thanh chữ nhật màu trắng bên nhánh phải của hình chữ U được copy từ nhánh bên trái và concatenate vào nhánh bên phải. Mỗi một mũi tên có màu sắc khác nhau tương ứng với một phép biến đổi khác nhau như chúng ta có thể thấy trong mô tả của mạng.

Mạng Unet bao gồm 2 nhánh đối xứng nhau hình chữ U nên được gọi là Unet.

Kiến trúc mạng Unet bao gồm 2 phần là **phần thu hẹp (contraction)** ở bên trái và **phần mở rộng (expansion)** ở bên phải. Mỗi phần sẽ thực hiện một nhiệm vụ riêng như sau:

- **Phần thu hẹp:** Làm nhiệm vụ trích lọc đặc trưng để tìm ra bối cảnh của hình ảnh. Vai trò của phần thu hẹp tương tự như một Encoder. Một mạng Deep CNN sẽ đóng vai trò trích lọc đặc trưng. Lý do nhánh được gọi là thu hẹp vì kích thước dài và rộng của các layers giảm dần. Từ input kích thước 572×572 chỉ còn 32×32 . Đồng thời độ sâu cũng tăng dần từ 3 lên 512.
- **Phần mở rộng:** Gồm các layer đối xứng tương ứng với các layer của nhánh thu hẹp. Quá trình Upsampling được áp dụng giúp cho kích thước layer tăng dần lên. Sau cùng ta thu được một ảnh mask đánh dấu nhãn dự báo của từng pixel.

Đặc trưng riêng trong cấu trúc của Unet đó là áp dụng kết nối tắt đối xứng giữa layer bên trái với layer bên phải.

Mặc dù có độ chính xác khá cao nhưng Unet có tốc độ thấp. Với kiến trúc Unet cho input 572×572 như bài báo gốc có tốc độ là 5 fps. Do đó nó không phù hợp để áp dụng vào các tác vụ yêu cầu realtime như xe tự hành. Tuy nhiên, Unet lại thường được sử dụng khá phổ biến trong các tác vụ không đòi hỏi realtime vì accuracy của nó cũng không tồi và kiến trúc dễ implement.

11.1. Khởi tạo kiến trúc mạng Unet

Khởi tạo mạng Unet khá dễ dàng nếu nhận ra qui luật của chúng.

Top

Chúng ta sẽ chia mạng Unet thành những block module CNN có tác dụng downsample ở phần thu hẹp và upsample ở phần mở rộng và có thể viết thành hàm để khởi tạo chúng một cách khái quát như sau:

Hàm khởi tạo downsample cnn block

```

1   import tensorflow as tf
2
3   def _downsample_cnn_block(block_input, channel, is_first = False):
4       if is_first:
5           conv1 = tf.keras.layers.Conv2D(filters=channel, kernel_size=3, strid
6               conv2 = tf.keras.layers.Conv2D(filters=channel, kernel_size=3, strid
7                   return [block_input, conv1, conv2]
8   else:
9       maxpool = tf.keras.layers.MaxPool2D(pool_size=2)(block_input)
10      conv1 = tf.keras.layers.Conv2D(filters=channel, kernel_size=3, strid
11      conv2 = tf.keras.layers.Conv2D(filters=channel, kernel_size=3, strid
12      return [maxpool, conv1, conv2]
13
14 ds_block1 = _downsample_cnn_block(tf.keras.layers.Input(shape=(572, 572,
15 ds_block2 = _downsample_cnn_block(ds_block1[-1], channel=128)
16 ds_block3 = _downsample_cnn_block(ds_block2[-1], channel=256)
17 ds_block4 = _downsample_cnn_block(ds_block3[-1], channel=512)
18 ds_block5 = _downsample_cnn_block(ds_block4[-1], channel=1024)
```

Hàm khởi tạo upsample cnn block

```

1   def _upsample_cnn_block(block_input, block_counterpart, channel, is_last
2       # Upsampling block
3       uppool1 = tf.keras.layers.Convolution2DTranspose(channel, kernel_size=
4       # Crop block counterpart
5       shape_input = uppool1.shape[2]
6       shape_counterpart = block_counterpart.shape[2]
7       crop_size = int((shape_counterpart - shape_input)/2)
8       block_counterpart_crop = tf.keras.layers.Cropping2D(cropping=((crop_si
9       concat = tf.keras.layers.concatenate(axis=-1)([block_counterpart_crop,
10      conv1 = tf.keras.layers.Conv2D(filters=channel, kernel_size=3, strides
11      conv2 = tf.keras.layers.Conv2D(filters=channel, kernel_size=3, strides
12      if is_last:
13          conv3 = tf.keras.layers.Conv2D(filters=2, kernel_size=3, strides=1)(
14              return [concat, conv1, conv2, conv3]
15      return [uppool1, concat, conv1, conv2]
16
17 us_block4 = _upsample_cnn_block(ds_block5[-1], ds_block4[-1], channel=51
18 us_block3 = _upsample_cnn_block(us_block4[-1], ds_block3[-1], channel=25
19 us_block2 = _upsample_cnn_block(us_block3[-1], ds_block2[-1], channel=12
20 us_block1 = _upsample_cnn_block(us_block2[-1], ds_block1[-1], channel=64
```

```

1   model = tf.keras.models.Model(inputs = ds_block1[0], outputs = us_block1[
2   model.summary()
```

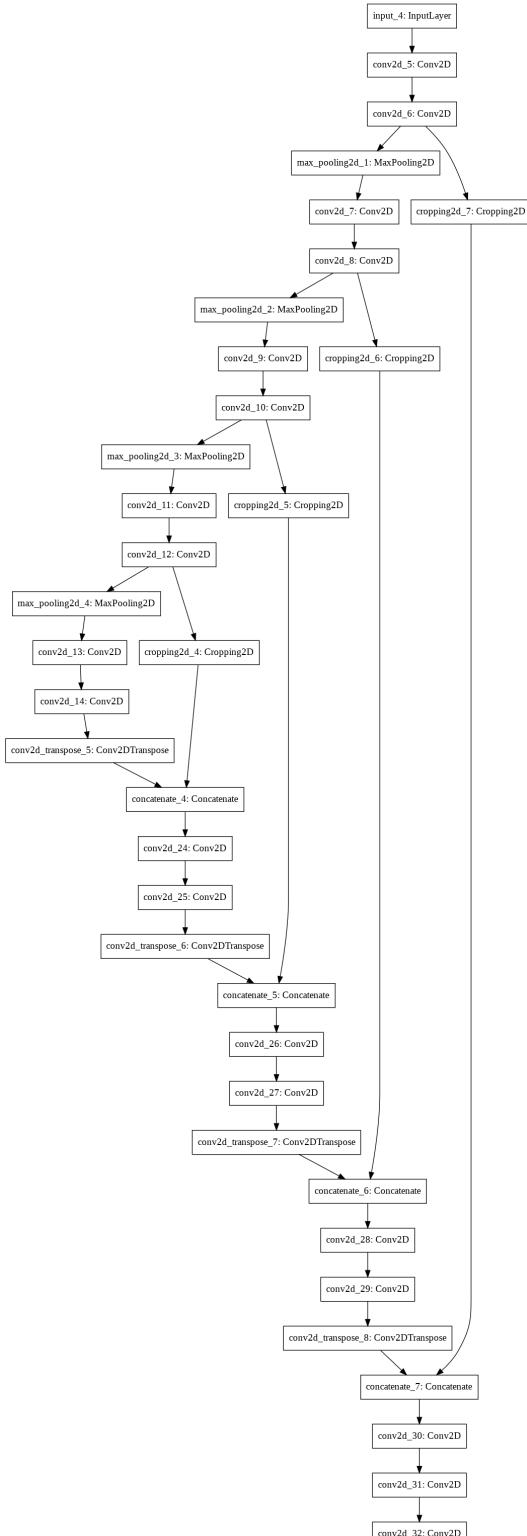
[Top](#)

```
1 Model: "model_4"
2
3 Layer (type)          Output Shape       Param #
4 =====
5 input_4 (InputLayer)   [(None, 572, 572, 1)] 0
6
7 conv2d_5 (Conv2D)      (None, 570, 570, 64) 640      input_4
8
9 conv2d_6 (Conv2D)      (None, 568, 568, 64) 36928     conv2d_
10
11 ...
12
13 conv2d_32 (Conv2D)    (None, 386, 386, 2) 1154     conv2d_
14 =====
15 Total params: 31,031,682
16 Trainable params: 31,031,682
17 Non-trainable params: 0
18
```

Sau khi khởi tạo xong model unet chúng ta có thể visualize sơ đồ các layers của mạng.

```
1 tf.keras.utils.plot_model(model)
```

Top



12. FCN (2015)

Paper: Fully Convolutional Networks for Semantic Segmentation (<https://arxiv.org/pdf/1411.4038.pdf>)

Author: Jonathan Long*, Evan Shelhamer*, Trevor Darrell - UC Berkeley

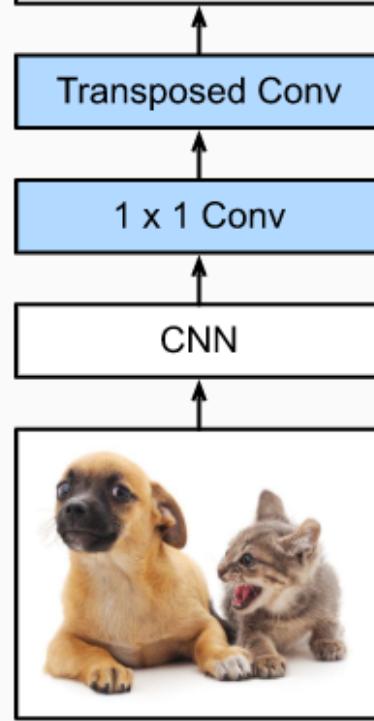
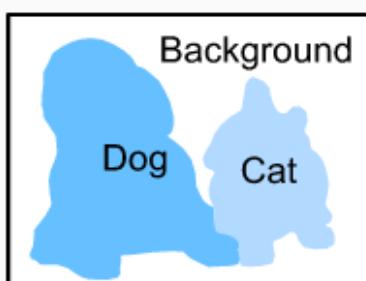
Code: FCN.tensorflow (<https://github.com/shekkizh/FCN.tensorflow>)

Hình 13: Hình minh họa kiến trúc FCN áp dụng base network là mạng VGG19.

FCN (Fully Convolutional Network) là một kiến trúc dựa trên mạng CNN nhằm mục đích segment một hình ảnh đầu vào bằng cách dự báo nhãn cho từng pixel trên ảnh input.

Top

Kiến trúc của FCN có thể khái quát thông qua hình mô tả bên dưới:



Hình 14: Kiến trúc khái quát của mạng FCN. Nguồn DeepLearning - Chapter 13 (https://d2l.ai/chapter_computer-vision/fcn.html)

Kiến trúc gồm 2 path có tác dụng khác nhau đó là:

- **Downsampling path:** Trích suất đặc trưng ngữ cảnh và nội dung của ảnh. Downampling path là kiến trúc thu hẹp kích thước layer. Đó chính là layer CNN ở vị trí đầu tiên, thông thường là những kiến trúc CNN cơ bản như AlexNet, VGG16, VGG19, ResNet đã được Truncate Fully Connected. Output của Downsampling path là Feature Map mang giá trị thông tin về vị trí và cường độ của các pixels. Tích chập kích thước 1×1 được áp dụng ngay sau block CNN với số lượng bộ lọc bằng với số lượng nhãn ở output. Trong bài báo gốc tác giả huấn luyện trên bộ dữ liệu Pascal VOC với số nhãn là 21 (20 class + 1 back ground). Sở dĩ chúng ta lựa chọn số bộ lọc như vậy là để trả ra phân phối xác suất nhãn cho mỗi pixel ở output.
- **Upsampling path:** Khôi phục thông tin không gian của ảnh theo các segment dự báo. Chúng ta áp dụng một layer Transposed Conv để giải chập thông tin thành segmentation map.

Toàn bộ quá trình này khá đơn giản.

12.1. Khởi tạo Mô hình FCN trên tensorflow

Load pretrain model VGG19 từ bộ dữ liệu imagenet.

Top

```

1 import tensorflow as tf
2 # Load pretrain model
3 pretrain_net = tf.keras.applications.VGG19(include_top=True, weights="ima
4
5 # In ra 5 layers shape cuối cùng
6 for layer in pretrain_net.layers[-5:]:
7     print('layer {}: {}'.format(layer.output.name, layer.output.shape))
8
9 print('pretrain_net output: ', pretrain_net.output)

```

```

1 Downloading data from https://storage.googleapis.com/tensorflow/keras-app
2 574717952/574710816 [=====] - 10s 0us/step
3 layer block5_pool/Identity:0: (None, 7, 7, 512)
4 layer flatten/Identity:0: (None, 25088)
5 layer fc1/Identity:0: (None, 4096)
6 layer fc2/Identity:0: (None, 4096)
7 layer predictions/Identity:0: (None, 1000)
8 pretrain_net output: Tensor("predictions/Identity:0", shape=(None, 1000))

```

Khởi tạo block CNN ở downsampling path. Chúng ta sẽ bỏ qua các fully connected layers và chỉ lấy đến layer Maxpooling ở cuối cùng có kích thước 7×7 .

```

1 net = tf.keras.models.Model(
2     inputs = pretrain_net.input,
3     outputs = pretrain_net.layers[-5].output
4 )

```

Tiếp theo ta sẽ áp dụng tích chập 1×1 để thay đổi độ sâu của khối output từ mạng CNN ở bước trên về độ sâu 21.

Đồng thời cần áp dụng tích chập chuyển vị (Transposed Conv) để giải chập khối 7×7 về kích thước bằng với kích thước input là 224×224 . Khi đó cần lựa chọn kích thước bộ lọc như thế nào? Áp dụng công thức (1) ta có:

$$W_{out} = (W - 1) \times S + F$$

Thế $W = 7$, $W_{out} = 224$, $S = 32$ suy ra bộ lọc có kích thước:

$$F = W_{out} - W(S - 1) = 224 - 7 \times (32 - 1) = 7$$

Top

```

1 num_classes=21
2 S=32
3 F=7
4
5 # Tích chập 1x1 trên feature map output
6 conv2D = tf.keras.layers.Conv2D(
7     num_classes, kernel_size=1)
8
9 # Tích chập chuyển vị
10 conv2DTran = tf.keras.layers.Conv2DTranspose(
11     num_classes, kernel_size=F, strides=S)
12
13 # Khởi tạo mô hình
14 model = tf.keras.models.Sequential([
15     net,
16     conv2D,
17     conv2DTran
18 ])
19
20 model.summary()

```

```

1 Model: "sequential"
2
3 Layer (type)          Output Shape         Param #
4 =====
5 model (Model)         (None, 7, 7, 512)      20024384
6
7 conv2d (Conv2D)        (None, 7, 7, 21)       10773
8
9 conv2d_transpose (Conv2DTran (None, 224, 224, 21)   21630
10
11 Total params: 20,056,787
12 Trainable params: 20,056,787
13 Non-trainable params: 0
14

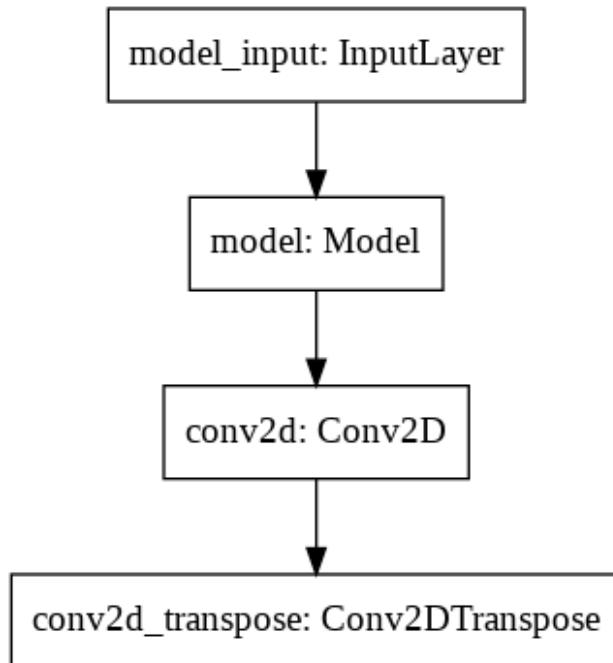
```

```

1 from tensorflow.keras.utils import plot_model
2 plot_model(model)

```

Top



13. Tổng kết

Image Segmentation là một trong những lớp bài toán có nhiều ứng dụng thực tiễn. Image Segmentation đòi hỏi mức độ chi tiết và chính xác hơn trong việc hiểu và biểu diễn nội dung của hình ảnh so với các thuật toán khác như Image Classification, Object Detection. Ở bài viết này chúng ta đã được tìm hiểu một số khái niệm mới như mạng giải chập và các tích chập đặc biệt có chức năng đặc biệt như tích chập chuyển vị, tích chập giãn nở.

Đồng thời chúng ta cũng làm quen với lý thuyết thuật toán, kiến trúc mô hình và thực hành xây dựng mô hình trên tensorflow của một số kiến trúc Image Segmentation cơ bản như Mask-CNN, Unet, FCN . Các mô hình trong Image Segmentation vẫn còn tiếp tục phát triển và còn rất nhiều các mô hình mà mình sẽ đề cập ở những bài tiếp theo.

14. Tài liệu tham khảo

1. image segmentation techniques (<https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/>)
2. 13. Computer Vision - Dive Into Deep Learning (http://d2l.ai/chapter_computer-vision/index.html)
3. transposed convolution (<https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba>)
4. lecture13 - cs131
(http://vision.stanford.edu/teaching/cs131_fall1617/lectures/lecture13_kmeans_mean_shift_cs131_2016)
5. Mask R-CNN - jonathan hui (https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272)
6. U-Net: Convolutional Networks for Biomedical Image Segmentation
(<https://arxiv.org/abs/1505.04597>)
7. Fully Convolutional Networks for Semantic Segmentation (<https://arxiv.org/abs/1411.4038>)
8. Image Segmentation in 2020 (<https://neptune.ai/blog/image-segmentation-in-2020>)

Top