

Projects in Mathematics and Applications

ỨNG DỤNG MẠNG CNN VÀ KIẾN TRÚC U-NET TRONG VIỆC PHÂN TÁCH NỀN VÀ XE

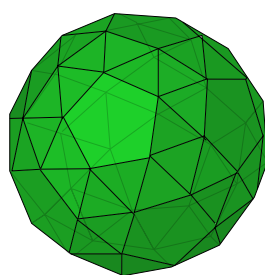
Ngày 29 tháng 8 năm 2019

Lê Duy Thức *

Lê Bảo Hiệp ‡

†Phạm Đình Anh Khoa

§Nguyễn Đại Nghĩa



*Trường THPT Chuyên Hùng Vương, Bình Dương

†Trường THPT Chuyên Lê Hồng Phong, TP.HCM

‡Trường Phổ thông Năng khiếu, ĐHQG-HCM

§Trường Phổ thông Năng khiếu, ĐHQG-HCM

Lời cảm ơn

Chúng em xin chân thành cảm ơn Ban tổ chức trại hè PiMA đã mang tới cho chúng em một trại hè vô cùng bổ ích và thú vị, đã tạo điều kiện cho mọi người có cơ hội học tập và nghiên cứu những phần thật sự thú vị trong toán học cũng như ứng dụng vào Deep Learning. Ngoài việc học toán, chúng em còn được gặp và tiếp xúc với nhiều người có cùng đam mê với Toán trên mọi miền đất nước, thậm chí là cả ngoài nước, đó là một cơ hội hiếm khi có được.

Với những thành công trong 4 năm của trại hè, mong rằng PiMA có thể duy trì và phát triển hơn nữa, để có thể đem niềm yêu thích toán đến với nhiều học sinh hơn nữa.

Chúng em xin cảm ơn các diễn giả đã tới và có những buổi nói chuyện rất thân mật, nhưng cũng đầy tính trí tuệ, qua đó đã giúp chúng em hiểu hơn cũng như được truyền lửa đam mê trong việc học sau này.

Chân thành cảm ơn Ban Giám hiệu Đại học Khoa học Tự nhiên TP.HCM và các nhà tài trợ đã giúp chúng em có đủ điều kiện để tham gia trại hè một cách tốt nhất.

Cuối cùng, xin cảm ơn anh Nguyễn Nguyễn và anh Phạm Nguyễn Hoàng Duy đã theo sát nhóm chúng em trong quá trình hoàn thành dự án này, cảm ơn sự góp ý của anh Trần Hoàng Bảo Linh để cho bài báo cáo này được hoàn thiện hơn.

Giới thiệu

Deep Learning (học sâu) là một nhánh của Machine Learning (học máy) nhằm mô phỏng hoạt động nhận thức của con người bằng những công cụ máy móc và có ứng dụng rộng rãi trong các lĩnh vực như nhận diện hình ảnh, xử lý ngôn ngữ tự nhiên, phân tích dữ liệu, v.v. Dù thời gian tồn tại chưa lâu nhưng Machine Learning nói chung và Deep Learning nói riêng đã đem lại những thành tựu to lớn giúp cách mạng hóa nền văn minh trí thức của con người.

Mở đầu cho mạng học sâu là sự xuất hiện của Neural Network (NN) với 2 lớp đầu vào là một vector đặc tính của vật và đầu ra là xác suất sự vật thuộc lớp nào. Sau nhiều cải tiến mạng NN được thêm vào các layer để xử lý dữ liệu nhiều hơn, phân tích được nhiều đặc điểm của vật hơn vì thế được gọi là Deep Neural Network (DNN). Những năm gần đây, ngành Computer Vision chứng kiến nhiều thành tựu đột phá nhờ áp dụng các mô hình học sâu thay cho các phương pháp xử lý ảnh truyền thống. Một trong những mô hình được sử dụng rộng rãi nhất là Convolutional Neural Network (CNN).

Trong phạm vi bài viết, chúng ta sẽ bàn về ứng dụng của phép tích chập (convolution) vào mạng học sâu. Từ đó bàn về kiến trúc cơ bản của CNN cho việc xử lý ảnh, đặc biệt là giải quyết bài toán Image Segmentation. Sau đó áp dụng mô hình U-Net để phân vùng xe trong tập dữ liệu Carvana Image Masking Challenge.

Mục lục

1	Convolutional Neural Network (CNN)	1
1.1	Giới thiệu	1
1.2	Kiến trúc cơ bản	1
2	Bài toán Image Segmentation	9
2.1	Giới thiệu	9
2.2	Hàm mục tiêu	10
3	Mạng U-Net	11
3.1	Cấu trúc mạng	11
3.2	Điểm khác biệt so với mạng CNN cổ điển	12
4	Áp dụng mô hình	16
4.1	Data Preprocessing	17
4.2	Initialization	19
5	Nhận xét và Đánh giá	21

1 Convolutional Neural Network (CNN)

1.1 Giới thiệu

Convolution Neural Network (ConvNet hoặc CNN) là một mô hình Neural Network với khả năng học các đặc tính của ảnh một cách hiệu quả đã khẳng định sự mạnh mẽ và vượt trội so với các phương pháp truyền thống trong các bài toán xử lý hình ảnh. CNN đã được ứng dụng trong nhiều lĩnh vực như nhận diện khuôn mặt, phát triển xe tự lái, v.v.

Lịch sử CNN bắt đầu khá sớm khi CNN đã được đề xuất từ những năm 90 bởi Yann LeCun để giải quyết bài toán nhận diện chữ cái. Tuy vậy, CNN (cũng như Deep Learning) chỉ thật sự phát triển sau bước đột phá vào năm 2012 với mô hình AlexNet cùng nhiều kỹ thuật mới như hàm ReLU, dropout và đặc biệt là việc ứng dụng GPU để huấn luyện mô hình.

1.2 Kiến trúc cơ bản

1.2.1 Convolution Layer

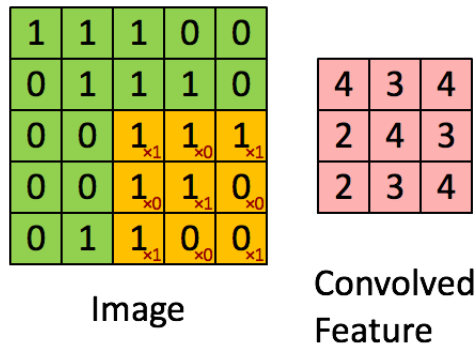
Khi xử lý hình ảnh, do lượng điểm ảnh lớn, nên việc sử dụng tất cả điểm ảnh làm dữ liệu đầu vào là không hiệu quả và không cần thiết, đặc biệt là khi áp dụng lên những lớp Fully Connected có thể làm lượng tham số trở nên khổng lồ. Vì mỗi neuron ở lớp này nối với mọi neuron của lớp kế tiếp, số lượng tham số tăng nhanh chóng và dễ gây ra hiện tượng Overfitting. Chính vì thế, việc tiền xử lý để giảm lượng dữ liệu nhưng vẫn giữ được mối quan hệ giữa các điểm ảnh lân cận (spatial relationship) của mô hình là cần thiết và có thể đạt được thông qua phép tích chập (convolution).

Định nghĩa 1.1

Phép tích chập là tổng của tích các phần tử tương ứng với nhau trên ma trận gốc và filter, có thể được tính bởi công thức:

$$y = \sum_{i,j} w_{i,j} x_{i,j},$$

với $w_{i,j}$, $x_{i,j}$ là các phần tử tương ứng của \mathbf{W} là ma trận trọng số của filter và \mathbf{X} là ma trận trích xuất của ma trận gốc.

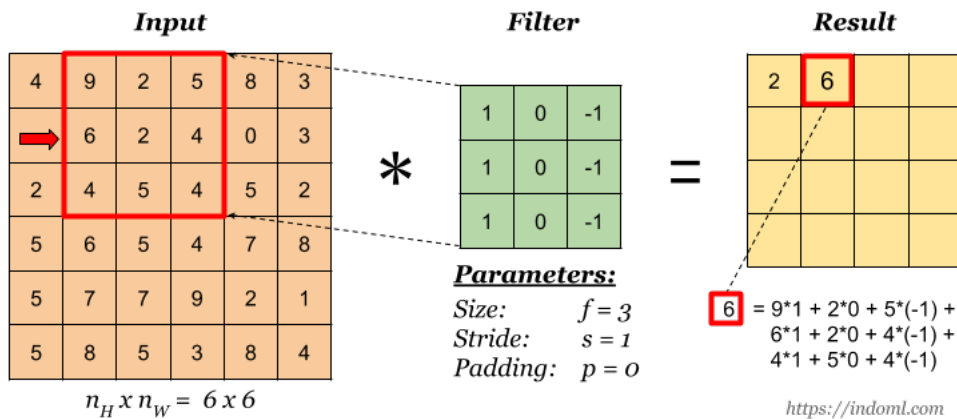


Hình 1: Minh họa phép tích chập trên ma trận gốc 5×5 với filter kích thước 3×3

Bằng phép tích chập, một lượng điểm ảnh lân cận được gom lại thành một điểm ảnh mới. Ý tưởng của phép tính là bằng một điểm ảnh vẫn tích hợp thông tin của những điểm lân cận và vẫn giữ được spatial relationship, từ đó giảm được số lượng tham số nhưng vẫn bảo toàn được thông tin.

Nhận thấy rằng kích thước ma trận đầu ra phải nhỏ hơn kích thước ma trận đầu vào vì ta đang giả sử filter phải nằm trọn vẹn trong ma trận đầu vào. Trong trường hợp cần giữ cho ma trận đầu ra có kích thước bằng ma trận đầu vào, ta có thể **thêm các giá trị 0 vào rìa của ma trận gốc**. Kỹ thuật này được gọi là **padding**.

Trong hình 1, filter được dịch sang phải một ô sau mỗi phép tính. Trong một số trường hợp, ta có thể **dịch filter đi nhiều ô sau mỗi phép tính**. Số lượng ô được dịch này được gọi là **stride**.

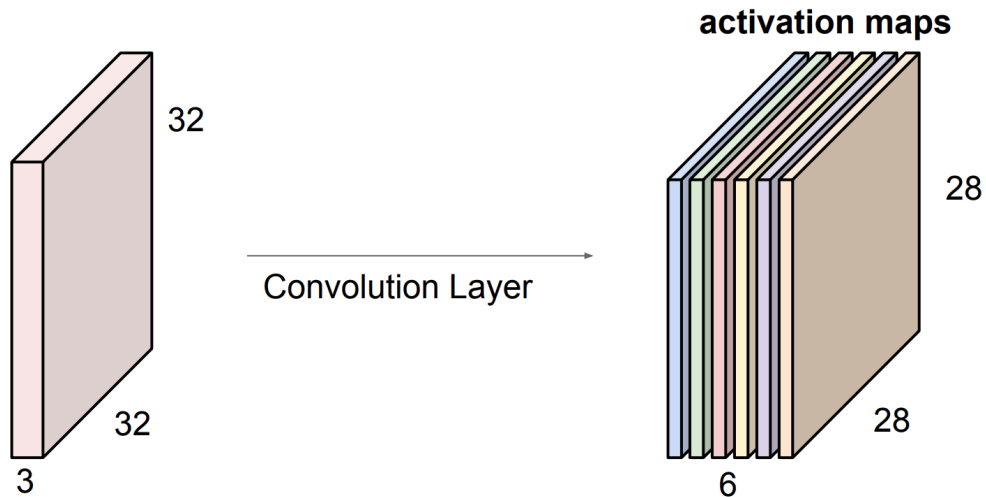


Với một ma trận đầu vào có kích thước $n_1 \times n_2$, filter có kích thước $k_1 \times k_2$, stride = s , padding = p , ma trận đầu ra sẽ có kích thước:

$$\left\lfloor \frac{n_1 + 2 \times p - k_1}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_2 + 2 \times p - k_2}{s} + 1 \right\rfloor$$

Hình biểu diễn phía trên thường được áp dụng cho các hình đơn sắc (monochrome), nhưng một bức hình thật thường có 3 kênh màu (R: đỏ, G: xanh lá, B: xanh dương) cùng với kích thước rộng và dài của bức hình tạo thành một ma trận 3 chiều làm đầu vào. Để thực hiện phép tích chập, filter cần có số kênh tương đương với số kênh của

hình. Đầu ra của phép nhân này sẽ là một mảng 2 chiều. Thoạt đầu ta có thể nghĩ lượng thông tin bị giảm đi, tuy nhiên, mỗi lớp Convolution không chỉ áp dụng 1 filter mà nhiều filter, mỗi filter chiết xuất 1 đặc tính của đối tượng. Những đặc tính này được chồng lên nhau thành một ma trận đầu ra có chiều dài và chiều rộng nhỏ hơn ma trận đầu vào, nhưng số kênh lại tăng lên và vẫn giữ được những đặc tính của đối tượng.

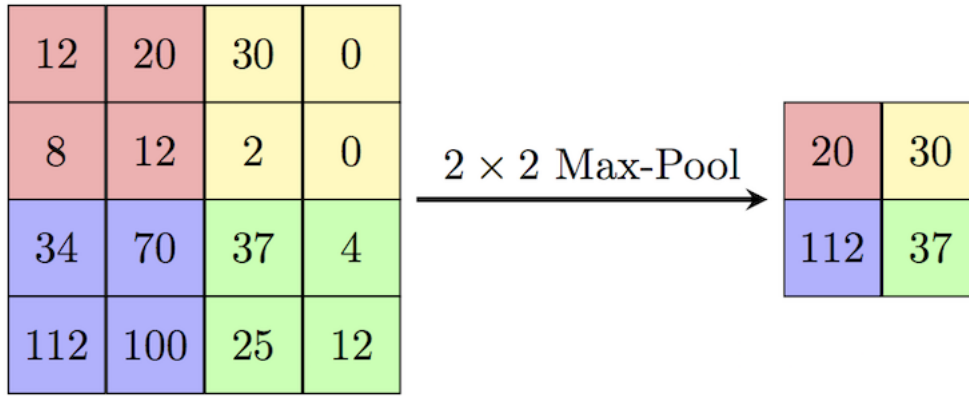


Hình 2: Mô phỏng cách lớp Convolution hoạt động trong việc tăng số kênh của bức ảnh

1.2.2 Pooling Layer

Mục đích chính của lớp Convolution không nhất thiết phải giảm dữ liệu của mẫu một cách triệt để mà là duy trì được spatial relationship giữa các pixel với nhau. Dữ liệu sau khi qua lớp Convolution thường sẽ được xử lý để giảm lượng dữ liệu nhờ lớp **Pooling** (còn gọi là Down-Sampling). Nhờ sự kết hợp giữa Convolution và Pooling, dữ liệu được giảm đáng kể nhưng vẫn giữ được các đặc tính cần thiết.

Pooling hoạt động bằng cách kết hợp những cụm dữ liệu lại với nhau, chẳng hạn như cụm 2×2 thành 1 pixel duy nhất. Một phương pháp thường dùng là **Max Pooling**, bằng cách chia nhỏ ma trận đầu vào thành các vùng nhỏ hơn có kích thước tương đương nhau và thường không trùng lặp, sau đó lấy giá trị lớn nhất của vùng đó để tạo thành ma trận mới với ít phần tử hơn.



Theo hình trên, với 4 phần tử mỗi vùng, chỉ giá trị lớn nhất được chọn để hình thành ma trận mới, giảm 75% số lượng phần tử trong 1 ma trận.

Ngoài **Max Pooling**, còn 2 loại pooling khá phổ biến khác là **Min Pooling** (lấy giá trị nhỏ nhất) và **Average Pooling** (lấy giá trị trung bình).

Ý tưởng đằng sau Pooling là vị trí tuyệt đối của 1 đặc tính ít quan trọng hơn quan hệ của nó với các đặc tính xung quanh, từ đó giảm đáng kể số lượng phần tử trong một ma trận, giúp giảm số lượng trọng số phải tính toán và hạn chế overfitting.

1.2.3 Fully Connected Layer

Lớp Fully Connected là lớp cơ bản nhất của một mô hình Máy học, mô phỏng lại hoạt động của neuron não người. Một mạng neural network chứa hoàn toàn những lớp Fully Connected thường được biết đến với tên gọi Multi Layer Perceptron.

Đặc điểm của lớp Fully Connected là mỗi neuron của lớp trước sẽ liên kết với mỗi neuron của lớp kế nó, tạo nên một mạng lưới liên kết hoàn toàn các neuron lại với nhau. Sau khi chiết xuất được các đặc tính của mẫu, lớp Fully Connected sẽ làm nhiệm vụ phân loại mẫu.

Đầu vào của lớp Convolution (hoặc Pooling) cuối cùng chứa đựng những thông tin về đặc tính của mẫu sẽ được chuyển thành vector 1D đầu vào (thông qua thao tác **flatten**). Vector này sẽ nhân với bộ trọng số của lớp Fully Connected, thông qua một hàm activation như sigmoid hay softmax chuyển thành những con số duy nhất (thường là các phân phối xác suất) phục vụ cho mục đích phân loại.

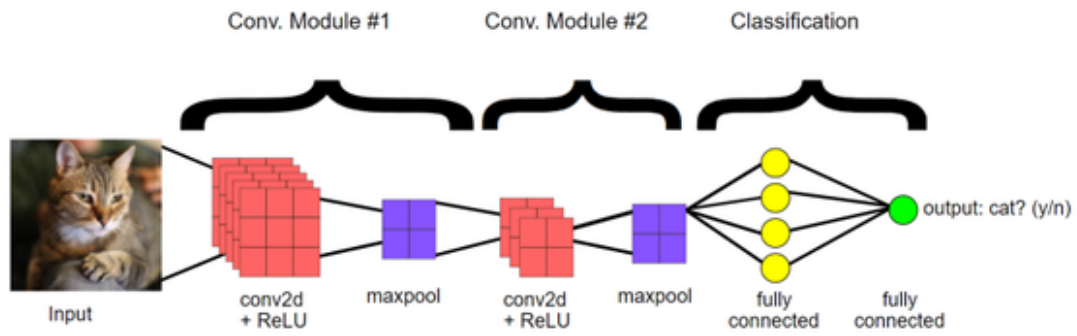


Figure 6. The CNN shown here contains two convolution modules (convolution + ReLU + pooling) for feature extraction, and two fully connected layers for classification. Other CNNs may contain larger or smaller numbers of convolutional modules, and greater or fewer fully connected layers. Engineers often experiment to figure out the configuration that produces the best results for their model.

Tuy nhiên, hạn chế của lớp Fully Connected là làm mất đi Spatial Relationship của các điểm ảnh nên lớp Fully Connected thường chỉ được dùng làm những lớp cuối của một mạng Convolutional Neural Network.

1.2.4 Activation

Hàm activation là một hàm phi tuyến mô phỏng hoạt động của neuron như một cổng tín hiệu. Khi có tín hiệu đầu vào, hàm sẽ quyết định việc tín hiệu đầu ra sẽ bị biến đổi như thế nào.

Thông qua hàm activation, mô hình sẽ có tính chất phi tuyến và nhờ đó có thể mô phỏng được những mô hình phức tạp hơn. Với tính chất phi tuyến, mô hình có thể xấp xỉ bất kì phương trình nào, đem lại cho mô hình khả năng học hỏi mạnh mẽ.

Có rất nhiều hàm activation phổ biến trong các mô hình Neural Network. Xin được giới thiệu 3 hàm activation thông dụng trong CNN là **sigmoid**, **softmax** và **ReLU**.

- **Sigmoid:**

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Đầu ra của hàm sigmoid là một số thực trong khoảng (0, 1), phù hợp nếu đầu ra là một xác suất phục vụ cho bài toán phân loại.

- **Softmax:** trong bài toán phân loại, có những trường hợp ta phải xác định nhiều vật khác nhau thì hàm sigmoid không thể đáp ứng được điều kiện đó, thay vào đó ta sử dụng hàm softmax.

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^C \exp(z_j)}, \quad \forall i = 1, 2, \dots, C.$$

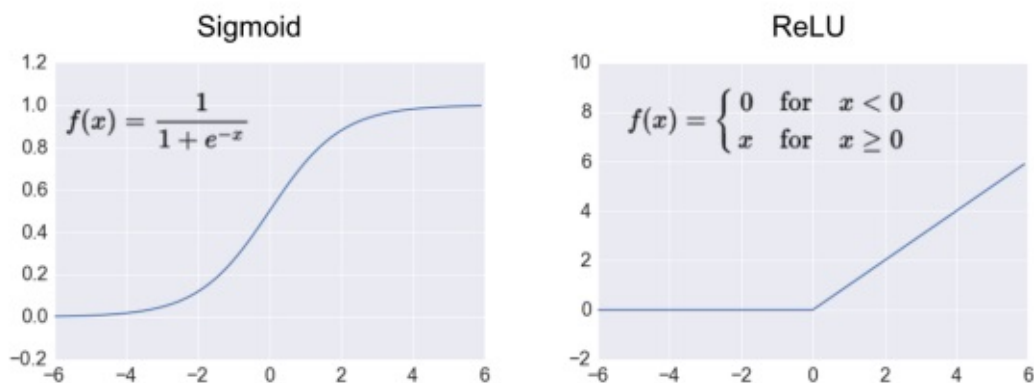
Đầu ra của hàm softmax là một phân phối xác suất mà tổng của chúng bằng 1. Dựa vào đó ta có thể phân loại nhiều vật thể.

- **ReLU (Rectifier Linear Unit):** hàm ReLU đưa mọi giá trị âm về bằng 0 theo công thức:

$$f(s) = \max(0, s).$$

Hàm ReLU cho phép kích hoạt chỉ một số neuron nhất định khi chuyển những dữ liệu có giá trị âm về 0. Hàm ReLU được tạo ra với mục đích bổ sung tính phi tuyến cho mô hình nhưng vẫn đơn giản và không yêu cầu những phép tính toán phức tạp, cho phép việc train mô hình diễn ra nhanh hơn.

Đồ thị các hàm **sigmoid** và **ReLU**:



1.2.5 Batch Normalization

Trong việc huấn luyện mạng Deep Learning, có hai vấn đề đáng chú ý là non-zero mean (hiện tượng dữ liệu không phân bố quanh giá trị 0, mà dữ liệu có phần nhiều giá trị lớn hơn 0 hoặc nhỏ hơn 0) và high variance (độ lệch chuẩn cao) khiến dữ liệu có nhiều thành phần rất lớn hoặc rất nhỏ. Tùy vào cách khởi tạo trọng số mà khi huấn luyện mạng càng sâu thì độ lệch càng nghiêm trọng.

Độ lệch lớn này buộc các layer phía sau phải học theo các thay đổi trong trọng số và bias của layer phía trước (vì output của layer trước là input của layer sau). Hậu quả là thời gian huấn luyện tăng nhưng độ ổn định lại giảm.

Ngoài ra vấn đề này còn ảnh hưởng đến các hàm activation phi tuyến tính như tanh, sigmoid, ReLU. Các hàm này đều có các vùng bão hoà (tức giá trị đầu ra không phụ thuộc giá trị đầu vào). Khi dữ liệu quá lớn hoặc quá nhỏ và rơi vào vùng bão hoà của hàm activation thì giá trị đầu ra rất giống nhau.

Phương pháp rất hiệu quả để giải quyết vấn đề này là chuẩn hoá dữ liệu về trạng thái zero mean (giá trị trung bình bằng 0) và unit variance (độ lệch chuẩn bằng 1). Phương pháp này được gọi là Batch Normalization.

Trong việc huấn luyện mạng Deep Learning, có việc thay đổi trọng số và bias của một lớp có thể dẫn đến sự thay đổi lớn ở lớp sau đó, khiến việc huấn luyện mất thời gian và không hiệu quả. Batch normalization ra đời nhằm khắc phục tình trạng này.

Trong quá trình huấn luyện, lớp Batch Normalization hoạt động như sau:

1. Tính giá trị trung bình và phương sai của dữ liệu đầu vào:

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$$

2. Chuẩn hoá dữ liệu đầu vào bằng các thông số trên:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma^2 + \epsilon}}$$

3. Tính dữ liệu đầu ra:

$$y_i = \gamma \hat{x}_i + \beta$$

Trong đó: x là dữ liệu đầu vào, \hat{x} là giá trị đã chuẩn hoá, μ_B là giá trị trung bình và σ^2 là phương sai. γ và β là các tham số học được.

Trong quá trình test, giá trị trung bình và độ lệch chuẩn được tính từ những giá trị tương ứng trong quá trình train:

$$E_x = \frac{1}{m} \sum_{i=1}^j \mu_B^{(i)}$$

$$Var_x = \left(\frac{m}{m-1} \right) \frac{1}{m} \sum_{i=1}^j \sigma_B^{2(i)}$$

$$y = \frac{\gamma}{\sqrt{Var_x + \epsilon}} x + \left(\beta + \frac{\gamma E_x}{\sqrt{Var_x + \epsilon}} \right)$$

Với m là số điểm dữ liệu trong mỗi batch, j là số batch.

Nhận xét. Các điểm dữ liệu được chọn ngẫu nhiên cho mỗi minibatch, do đó giá trị trung bình và độ lệch cũng khác nhau cho mỗi minibatch. Hai tham số này lại được thêm vào dữ liệu trong mỗi lần train. Do đó Batch Normalization còn có tác dụng như một Regularization giúp mô hình tránh bị overfitting.

Các ưu điểm của Batch Normalization gồm:

- Giúp các layer độc lập với nhau hơn (việc thay đổi tham số ở layer trước không ảnh hưởng lớn đến layer sau)
- Cho phép learning rate lớn
- Giảm sự phụ thuộc vào initialization
- Có vai trò như một regularization

2 Bài toán Image Segmentation

2.1 Giới thiệu

Trong bài toán Image Segmentation (ImS), có 2 nhánh cốt yếu mà chúng ta quan tâm là Phân vùng theo nghĩa (Semantic Segmentation) và Phân vùng theo cá thể (Instance Segmentation)

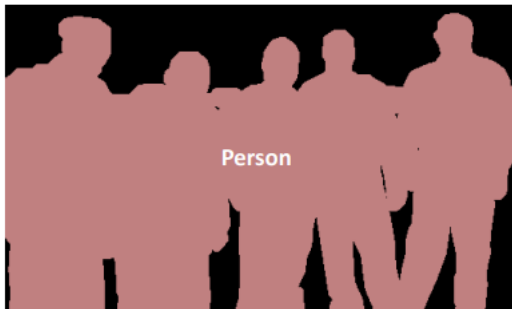
Định nghĩa 2.1 (Semantic Segmentation)

Với mỗi pixel ta cần xác định loại đối tượng mà nó thuộc vào (ví dụ như pixel đó là thuộc lớp xe hay cây cối, v.v)

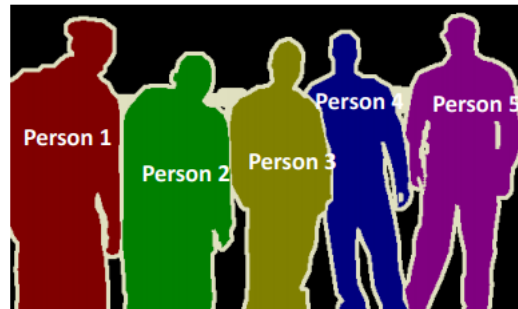
Định nghĩa 2.2 (Instance Segmentation)

Cũng giống như Semantic Segmentation, nhưng chi tiết hơn, mỗi pixel được phân loại theo đối tượng mà nó thuộc vào.

Bên dưới là mô tả sự khác biệt của 2 loại.



Semantic Segmentation



Instance Segmentation

Mô hình CNN cổ điển giải quyết rất tốt bài toán phân loại đối tượng trong hình ảnh, tuy nhiên lại không giải quyết được bài toán Image Segmentation. Nguyên nhân chính là hình ảnh sau khi đi qua mạng CNN sẽ bị giảm độ lớn, mất đi thông tin về vị trí của chủ thể trong ảnh. Trong khi đó, Image Segmentation yêu cầu mô hình mạng phải trả về output có cùng kích thước với input, mỗi pixel đều được phân loại riêng biệt. Vì vậy chúng ta cần nâng cấp mô hình mạng CNN cơ bản lên để có những công cụ mạnh mẽ hơn giúp ta giải quyết bài toán này. Có nhiều biến thể của mạng CNN được sử dụng trong bài toán Image segmentation như:

- **Region-based Convolutional Network (R-CNN):** tách hình ảnh thành nhiều vùng sau đó nhận diện từng thành phần trong các vùng nhỏ.
- **Fully Convolutional Network (FCN):** dùng các thao tác upsampling để tăng kích thước ảnh output về kích thước input.

2.2 Hàm mục tiêu

Với bài toán Image Segmentation, bởi vì output chúng ta là một hình ảnh, mỗi pixel sẽ mang một giá trị (hoặc một phân phối xác suất) tương ứng với vùng mà nó được phân chia vào (hoặc là xác suất mà nó được phân chia vào).

Cụ thể hơn, xét bài toán phân vùng chủ thể và ảnh nền, mỗi pixel sẽ trả về một phân phối xác suất $y = (y_0, y_1)$ tương trưng cho xác suất mà pixel đó thuộc ảnh nền (y_0) hoặc chủ thể (y_1) thỏa mãn $y_0 + y_1 = 1$. Trong quá trình training, giả sử rằng pixel hiện tại thuộc chủ thể, chúng ta cần phân phối xác suất đầu ra là $y = (0.0, 1.0)$, nghĩa là xác suất pixel thuộc chủ thể đạt max. Trong thực tế điều đó rất khó xảy ra. Do đó, nếu mô hình cho ra một phân phối xác suất $\hat{y} = (\hat{y}_0, \hat{y}_1)$ thì chúng ta cần tiến hành điều chỉnh các tham số sao cho \hat{y} tiến gần tới y nhất có thể. Vì thế chúng ta cần một hàm số để đánh giá sự khác biệt giữa y và \hat{y} .

Có rất nhiều cách để đánh giá sự khác biệt này, nhưng phổ biến nhất vẫn là hàm **Cross-Entropy** để đánh giá sự khác biệt giữa 2 phân phối xác suất.

Định nghĩa 2.3 (Cross Entropy)

Với 2 xác suất p và q rời rạc, hàm số:

$$H(p, q) = - \sum_{i=1}^C p_i \cdot \log(q_i),$$

được gọi là **hàm Cross-Entropy** của hai phân phối p và q .

Nhận xét 2.1. $H(p, q) \neq H(q, p)$ và $q_i \neq 0$ (vì hàm log không xác định tại 0).

Vì thế nên trong bài toán này, p nên là output thật sự (y) còn q là output dự đoán (\hat{y}), vì các giá trị của y đa số bằng 0, chỉ có 1 giá trị bằng 1, với \hat{y} thì các giá trị khó có thể đạt giá trị 0 tuyệt đối, vì vậy không lo ngại vấn đề log không xác định.

Nhận xét 2.2. Nếu phân phối xác suất chỉ có 2 giá trị (y và \hat{y} ở trên) thì hàm Cross Entropy có thể viết lại như sau:

$$\begin{aligned} H(y, \hat{y}) &= -(y_0 \cdot \log(\hat{y}_0) + y_1 \cdot \log(\hat{y}_1)) \\ &= -(y_0 \cdot \log(\hat{y}_0) + (1 - y_0) \cdot \log(1 - \hat{y}_0)). \end{aligned}$$

Đó còn gọi là **hàm Binary Cross Entropy** của hai phân phối y và \hat{y} .

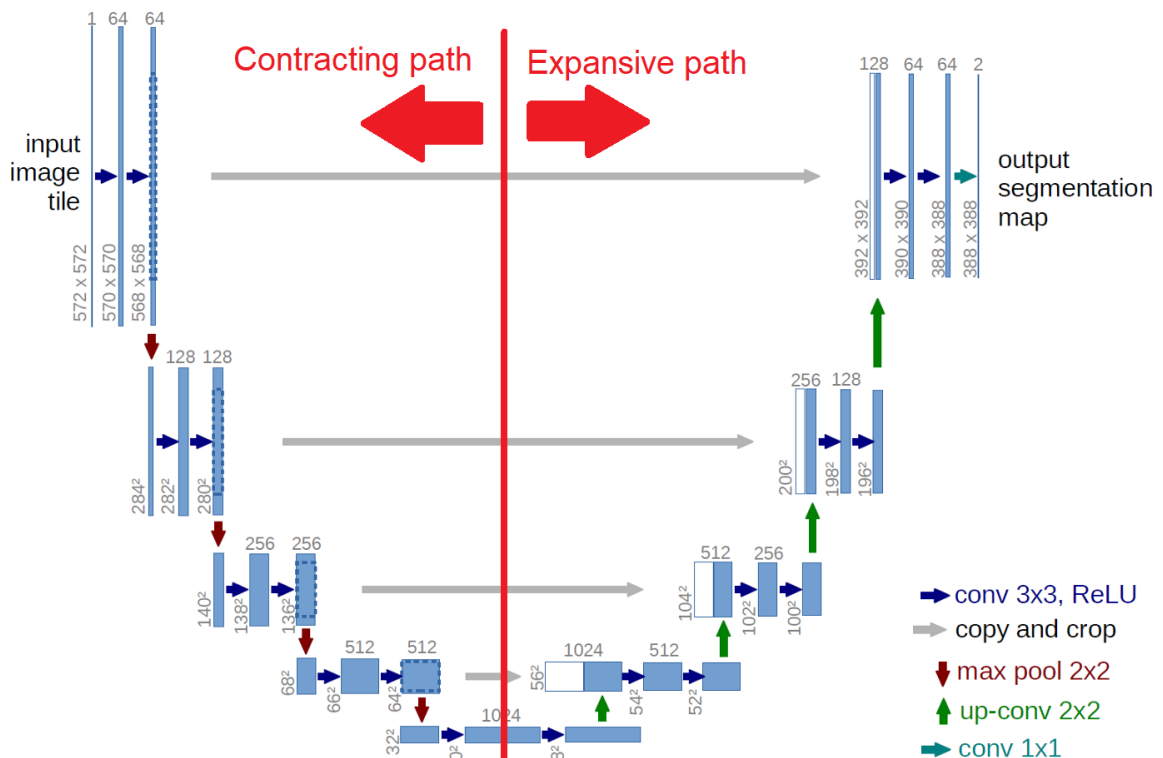
Nhận xét 2.3. Với bài toán phân vùng ảnh, ta có thể lấy tổng giá trị hàm Cross-Entropy (nếu có nhiều hơn 2 loại đối tượng cần phân chia) hoặc hàm Binary Cross-Entropy (nếu chỉ có 2 loại đối tượng cần phân chia) của mọi pixels để xem đó là hàm loss.

3 Mạng U-Net

3.1 Cấu trúc mạng

Bài toán Segmentation là một trong những bài toán cơ bản của Computer Vision, với mục tiêu là phải phân tách được các lớp vật thể với nhau. Như đã giới thiệu, bài toán Segmentation bao gồm 2 mảng: Semantic Segmentation và Instance Segmentation. Mạng U-Net được tạo ra để tập trung vào xử lý bài toán Semantic Segmentation.

Ý tưởng về hoạt động của U-Net là trong quá trình học, nếu mô hình đã có thể tự tìm ra những đặc điểm chung qua quá trình trích xuất đặc tính từ mẫu, thì mô hình cũng có thể học được quá trình kết hợp các đặc tính lại với nhau để tăng kích thước ảnh. U-Net được tạo ra nhằm cải tiến hạn chế giữa những mạng CNN trước đó khi phải chọn lựa giữa độ chính xác và thông tin vị trí của một vật. Sự cải tiến đáng kể này xuất phát từ việc U-Net không tồn tại lớp Fully Connected nào giúp đảm bảo thông tin không bị trộn lẫn trong quá trình xử lý. Thay vào đó U-Net sử dụng kiến trúc **encoder** - **decoder** nhằm tái xây dựng hình ảnh một cách hiệu quả. Về bản chất, **encoder** và **decoder** đều là các mạng CNN riêng biệt. Trong đó, **encoder** có cấu trúc tương tự CNN truyền thống giúp trích xuất các đặc tính của ảnh, còn **decoder** được thay các lớp convolution bằng transposed convolution để dựng ảnh đầu ra. Việc khôi phục kích thước ảnh đạt được thông qua việc khôi phục thông tin vị trí từ các ma trận đặc tính ở **encoder**.



Theo như mô hình, thông tin đi qua các lớp convolution không giảm đáng kể về mặt kích thước nhưng số chiều lại tăng lên rất nhiều, giúp đảm bảo những thông tin về vị trí tương quan giữa các đối tượng được bảo toàn. Các lớp pooling có nhiệm vụ chọn lọc ra những thông tin quan trọng thể hiện rõ đối tượng nhất và loại bỏ những thông

tin không cần thiết đi, giảm lượng thông tin cần phải xử lý. Tại lớp cuối cùng của encoder, các đặc tính đã được chiết xuất khỏi mẫu và được gửi qua decoder để tái tạo kích thước. Ở đây vì không dùng đến lớp Fully Connected nên hạn chế được sự mất mát thông tin về Spatial Relationship giữa các pixel và nhờ vậy, việc tái tạo diễn ra thuận lợi hơn. Tại decoder, các lớp convolution và pooling như encoder được thay thế bằng **transposed convolution** và **crop - concatenate**. **Transposed convolution** đảm nhiệm chính trong việc tăng kích thước ảnh, **crop - concatenate** sao chép một phần feature map từ encoder để tái tạo lại chính xác hơn vị trí của các đặc tính.

3.2 Điểm khác biệt so với mạng CNN cổ điển

3.2.1 Encoder

Encoder (Contracting Path): Thường giống mạng CNN cơ bản, chỉ gồm các lớp convolution và các lớp pooling. Encoder thực hiện nhiều lần các lớp convolution, mỗi lớp đều được thêm vào một hàm ReLU cùng một số lớp max poolings để giảm diện tích và tăng số channels (chiều sâu). Các channels chứa những features của ảnh được tổng hợp ở nhiều cấp độ.

3.2.2 Decoder

Decoder (Expanding Path): Trong mạng U-Net, decoder được dựng gần như đối xứng với encoder, cũng gồm các lớp convolutions và transposed convolutions dùng để tăng kích thước ảnh và giảm chiều sâu của ảnh. Điều đặc biệt, ngoài việc tăng kích thước ảnh, decoder còn kết nối đối xứng với encoder để lấy thông tin trực tiếp từ encoder bằng **Copy and Crop**, việc này làm giảm đi sự mất mát thông tin trong quá trình upsample. Decoder nhận vào một ma trận có chiều sâu (nhiều ma trận $m \times n$) và xuất ra mask của bức ảnh. **Mask là biểu đồ biểu hiện những pixels được đánh dấu thuộc cùng một tập dữ liệu** sau khi đã phân tích những feature bị tổng hợp nhiều cấp. Bên dưới là ví dụ của một mask.



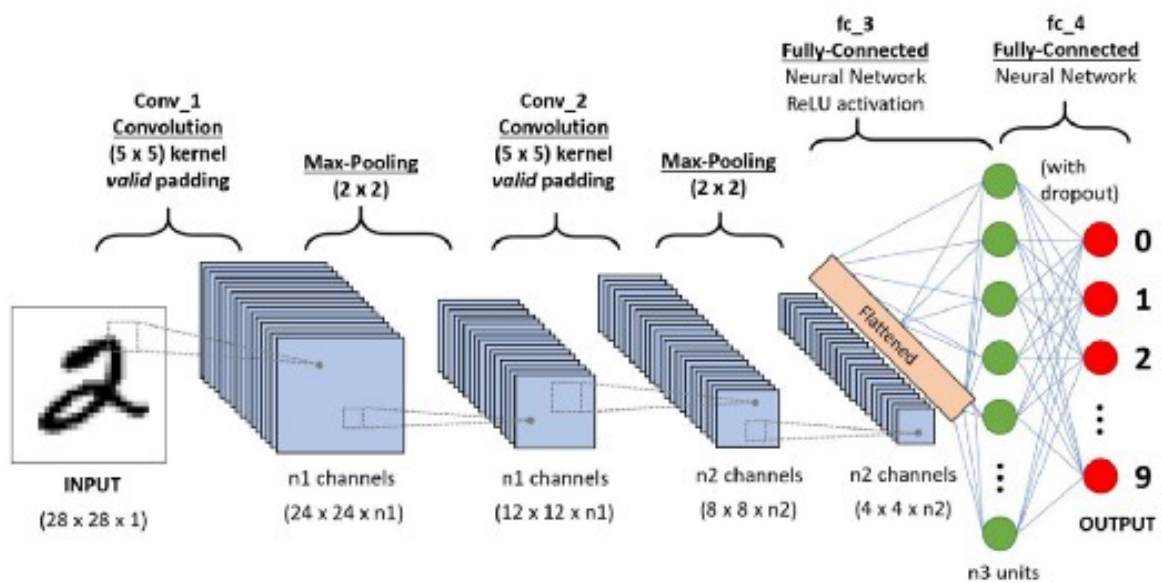
3.2.3 Copy and Crop

Copy and Crop (Skip Connections): Trong mạng U-Net, ở phần decoder có tính đối xứng với encoder. Mục đích của điều này là để khi upsample, các feature ở phần expanding path sẽ được cộng với đúng các feature ở vị trí đối xứng trong contracting path. Những kết nối này trong thực tế cho thấy đã tăng đáng kể hiệu quả trong việc khôi phục độ phân giải của ảnh ở đầu ra.

Trả lời cho sự hiệu quả trên ta có thể lý giải như sau: Thông tin sau khi bị gom cụm lại còn mất đi spatial information khiến cho việc khôi phục ở đầu ra rất khó. Vì vậy chúng ta cần nhập thẳng những output ở contracting path vào phần tương ứng ở expanding path làm input tương ứng của từng decoder layer, cung cấp thông tin vị trí đã bị mất trong quá trình encoder.

3.2.4 Transposed Convolution

Như đã biết, mạng CNN cổ điển chỉ có nhiệm vụ phân lớp dữ liệu. Trong mạng CNN cổ điển, dữ liệu sau khi được lọc các đặc tính sẽ được đưa vào lớp Fully Connected rồi qua hàm activation (thường là hàm sigmoid hoặc softmax) cho ra dữ liệu thuộc lớp gì (chó, mèo, người, xe cộ, v.v).

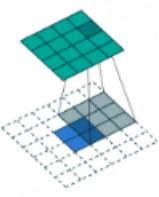
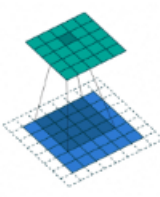
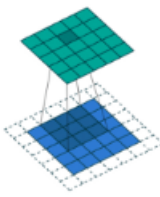
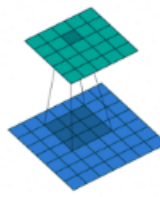
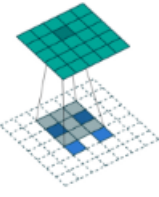
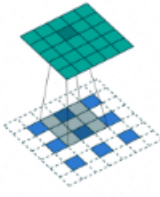
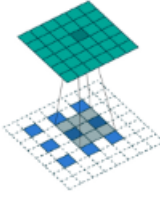


Hình 3: Mạng CNN truyền thống

Trong khi đó, bài toán Image Segmentation yêu cầu ta phải sinh ra mask có kích cỡ như ảnh ban đầu từ các ma trận đã bị downscale sau khi đi qua encoder. **Transposed convolution** trong decoder thực hiện điều này.

Giả sử **X** là ma trận kết quả sau khi thực hiện convolution trên ma trận **Y**. Transposed convolution giúp ta xây dựng ma trận có kích thước bằng ma trận **Y** từ ma trận **X**.

Về bản chất, **transposed convolution** cũng chỉ thực hiện tích chập như lớp convolution thông thường. Tuy nhiên, các thông số padding và stride được chọn sao cho sau khi thực hiện phép tích chập sẽ được ma trận mới có kích thước tương đương ma trận cũ. Ở đây, ngoài ý nghĩa là khoảng cách giữa hai lần tích chập liên tiếp, stride còn có ý nghĩa là khoảng cách giữa các pixels của ma trận đầu vào. Việc thực hiện transposed convolution không chỉ khôi phục kích thước ma trận mà còn khôi phục spatial relationship giữa các pixels.

			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			
No padding, strides, transposed	Padding, strides, transposed	Padding, strides, transposed (odd)	

Hình 4: Ví dụ về transposed convolution

Với mỗi convolution ta đều xây dựng được một transposed convolution tương ứng để trả lại ma trận có kích cỡ ban đầu.

Dưới đây xin giới thiệu một số cách xây dựng transposed convolution thông dụng dựa trên kích cỡ của lớp convolution và ma trận đầu vào. Ta quy ước i, o là kích thước của ma trận đầu vào của convolution. i', o' là kích thước ma trận đầu vào của transposed convolution. Gọi k, s, p lần lượt là kích thước, stride, padding của lớp convolution, và k', s', p' kích thước, stride, padding của lớp transposed convolution.

Vì mục đích giới thiệu, dưới đây chỉ xét các transposed convolution không giãn cách giữa các pixel trong ma trận đầu vào. Tổng hợp đầy đủ các loại transposed convolution được trình bày chi tiết trong tài liệu [3].

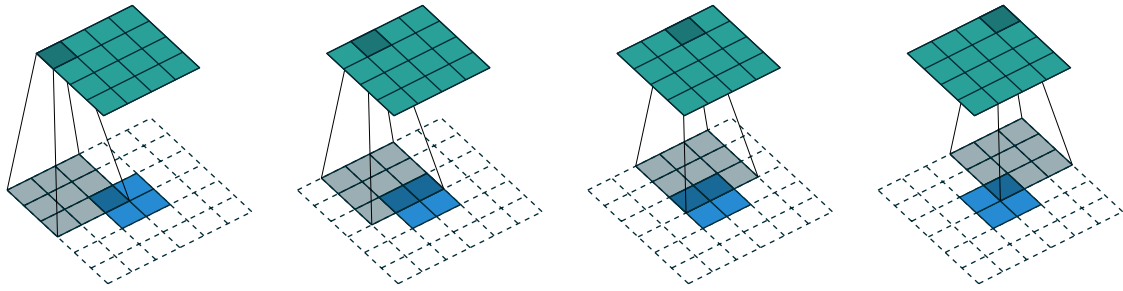
Lưu ý: $o = i', o' = i$

- **Zero padding, unit stride transposed:** Convolution có $s = 1$ có transposed convolution tương ứng là $k' = k, s' = s, p' = k - p - 1$, output size là $o' = i' + (k - 1) - 2p$.
- **Half (same) padding transposed:** Convolution có $k = 2n + 1, n \in \mathbb{N}, s = 1, p = n$ có transposed convolution tương ứng là $k' = k, s' = s, p' = p$, output size là:

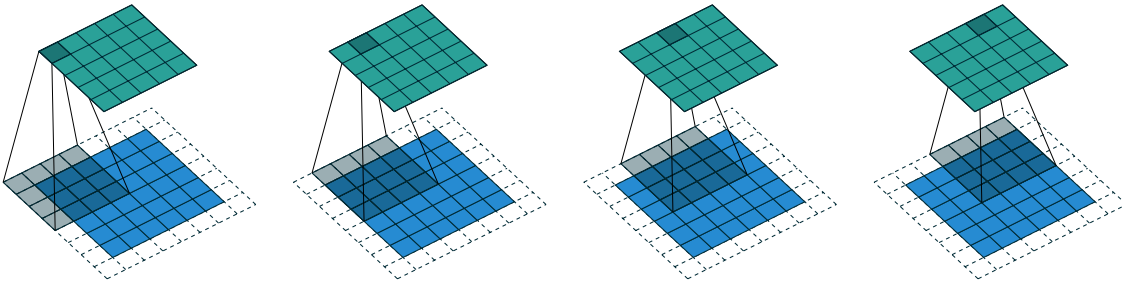
$$o' = i' + (k - 1) - 2p$$

$$o' = i' + 2n - 2n$$

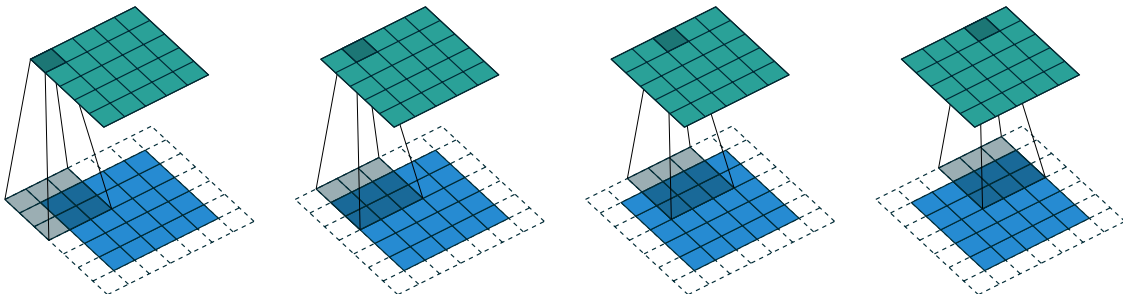
$$o' = i'$$
- **Full padding transposed:** Convolution có $s = 1, k$ tùy ý, $p = k - 1$ có transposed convolution là $k' = k, s' = s, p' = 0$ và output size là $o' = i' + (k - 1)$.



Hình 5: Transposed convolution tương ứng của phép convolution 3×3 trên ma trận đầu vào 4×4 với unit stride ($i = 4, k = 3, s = 1, p = 0, o = 2$). Transposed convolution này tương đương với phép convolution 3×3 trên ma trận 2×2 với padding 2, unit stride ($i' = 2, k' = k, s' = 1, p' = 2, o' = 4$).



Hình 6: Transposed convolution tương ứng của phép convolution 4×4 trên ma trận đầu vào 5×5 với padding 2, unit stride ($i = 5, k = 4, s = 1$ and $p = 2, o = 6$). Transposed convolution này tương đương với phép convolution 4×4 trên ma trận 6×6 với padding 1, unit stride ($i' = 6, k' = k, s' = 1, p' = 1, o' = 5$).



Hình 7: Transposed convolution tương ứng của phép convolution 3×3 trên ma trận đầu vào 5×5 với half padding, unit stride ($i = 5, k = 3, s = 1, p = 1, o = 5$). Transposed convolution này tương đương với phép convolution 3×3 trên ma trận 5×5 với half padding và unit stride ($i' = 5, k' = k, s' = 1, p' = 1, o' = 5$).

4 Áp dụng mô hình

Áp dụng mô hình mạng U-Net để giải bài toán Image Segmentation với tập dữ liệu Carnava Image Masking Challenge ta nhận được kết quả khá tốt.

Dữ liệu đầu vào gồm 5088 tấm ảnh xe (kích thước 1918×1280) nằm trên nền trắng, cùng với ảnh mask là đánh dấu vị trí của xe. Có tổng cộng 318 xe, mỗi xe có 16 tấm ảnh tương ứng với 16 góc quay khác nhau. Ta chia tập dữ liệu ra làm 3 phần (train, test, validation) tương ứng theo tỉ lệ (70%, 15%, 15%).

Ta tiến hành dựng mô hình, dưới đây là một số phần chính trong đoạn code:

```
def conv_block(input_tensor, num_filters):
    encoder = layers.Conv2D(num_filters, (3, 3),
        padding='same')(input_tensor)
    encoder = layers.BatchNormalization()(encoder)
    encoder = layers.Activation('relu')(encoder)
    encoder = layers.Conv2D(num_filters, (3, 3), padding='same')(encoder)
    encoder = layers.BatchNormalization()(encoder)
    encoder = layers.Activation('relu')(encoder)
    return encoder

def encoder_block(input_tensor, num_filters):
    encoder = conv_block(input_tensor, num_filters)
    encoder_pool = layers.MaxPooling2D((2, 2), strides=(2, 2))(encoder)

    return encoder_pool, encoder

def decoder_block(input_tensor, concat_tensor, num_filters):
    decoder = layers.Conv2DTranspose(num_filters, (2, 2), strides=(2,
        2), padding='same')(input_tensor)
    decoder = layers.concatenate([concat_tensor, decoder], axis=-1)
    decoder = layers.BatchNormalization()(decoder)
    decoder = layers.Activation('relu')(decoder)
    decoder = conv_block(decoder, num_filters)
    return decoder

def createModel():
    inputs = layers.Input(shape=(256, 256, 3))
    # beginning of encoder blocks
    encoder0_pool, encoder0 = encoder_block(inputs, 32)
    encoder1_pool, encoder1 = encoder_block(encoder0_pool, 64)
    encoder2_pool, encoder2 = encoder_block(encoder1_pool, 128)
    encoder3_pool, encoder3 = encoder_block(encoder2_pool, 256)
    encoder4_pool, encoder4 = encoder_block(encoder3_pool, 512)
    # end

    center = conv_block(encoder4_pool, 1024)
    # bottleneck
```

```

# beginning of decoder blocks
decoder4 = decoder_block(center, encoder4, 512)
decoder3 = decoder_block(decoder4, encoder3, 256)
decoder2 = decoder_block(decoder3, encoder2, 128)
decoder1 = decoder_block(decoder2, encoder1, 64)
decoder0 = decoder_block(decoder1, encoder0, 32)
#end

outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(decoder0)

model = models.Model(inputs=[inputs], outputs=[outputs])

model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['acc', f1_m, precision_m, recall_m])

model.summary()

return model

```

Model gồm có 5 khối trong encoder, 5 khối trong decoder và 2 lớp convolutions ở giữa. Mỗi khối trong encoder gồm 2 lớp convolution kết hợp 1 lớp max pooling. Mỗi khối trong decoder gồm 1 lớp transposed convolution và 2 lớp convolution. Sau mỗi lớp convolutions đều có một lớp batch-normalization.

4.1 Data Preprocessing

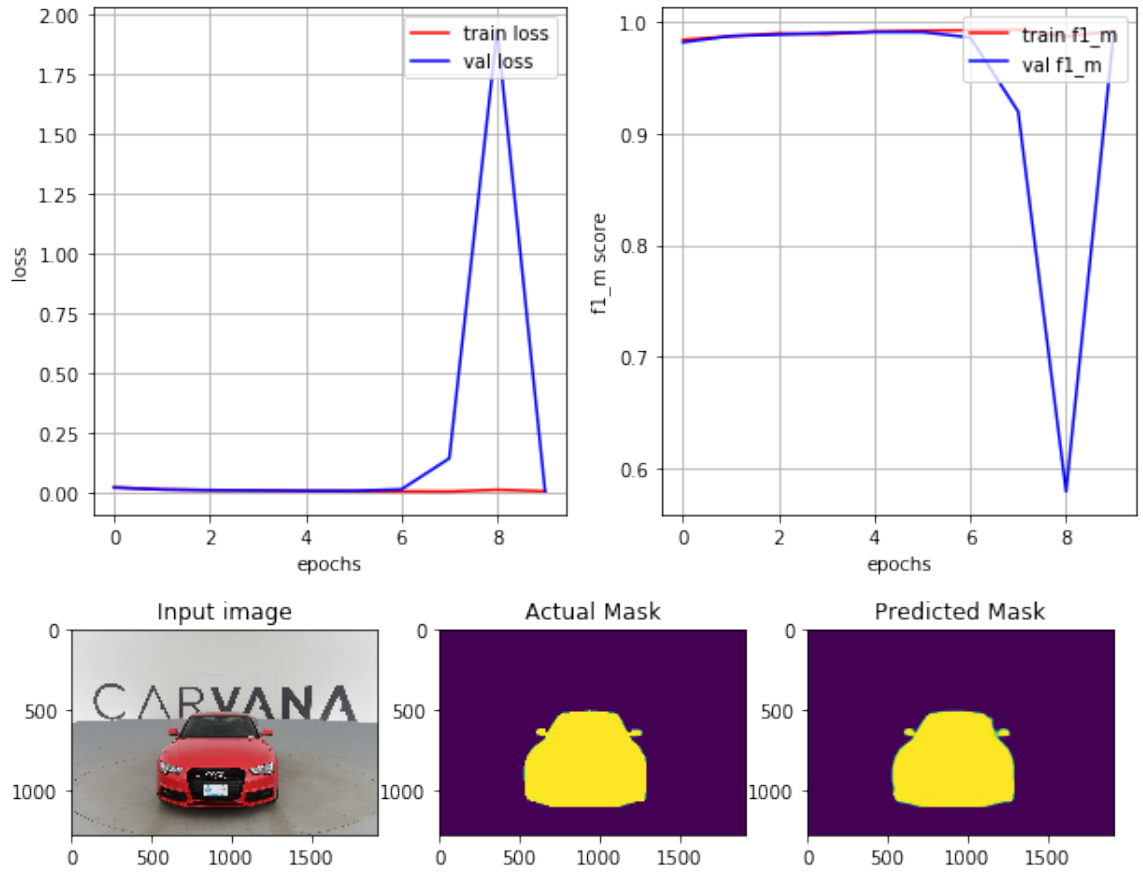
Trước hết, do ảnh của bộ dữ liệu có kích thước khá lớn, nên tất cả những ảnh đầu vào và đầu ra đều được thay đổi kích thước thành 256×256 .

Do bộ dữ liệu Carnava không phức tạp (chỉ có xe trên nền là phòng triển lãm), vì thế gần như không có noise, do đó khi thử những cách preprocessing như chuyển sang ảnh đen trắng, chỉ lấy 1 kênh màu thì kết quả không thay đổi 1 cách rõ rệt. Nên có thể nói rằng với bộ dữ liệu này preprocessing không giúp ích gì nhiều. Ngoài ra, bên cạnh việc sử dụng binary cross-entropy làm hàm loss, ta còn dùng **F1-score** để tính độ chính xác (chỉ với mục đích theo dõi, không tối ưu).

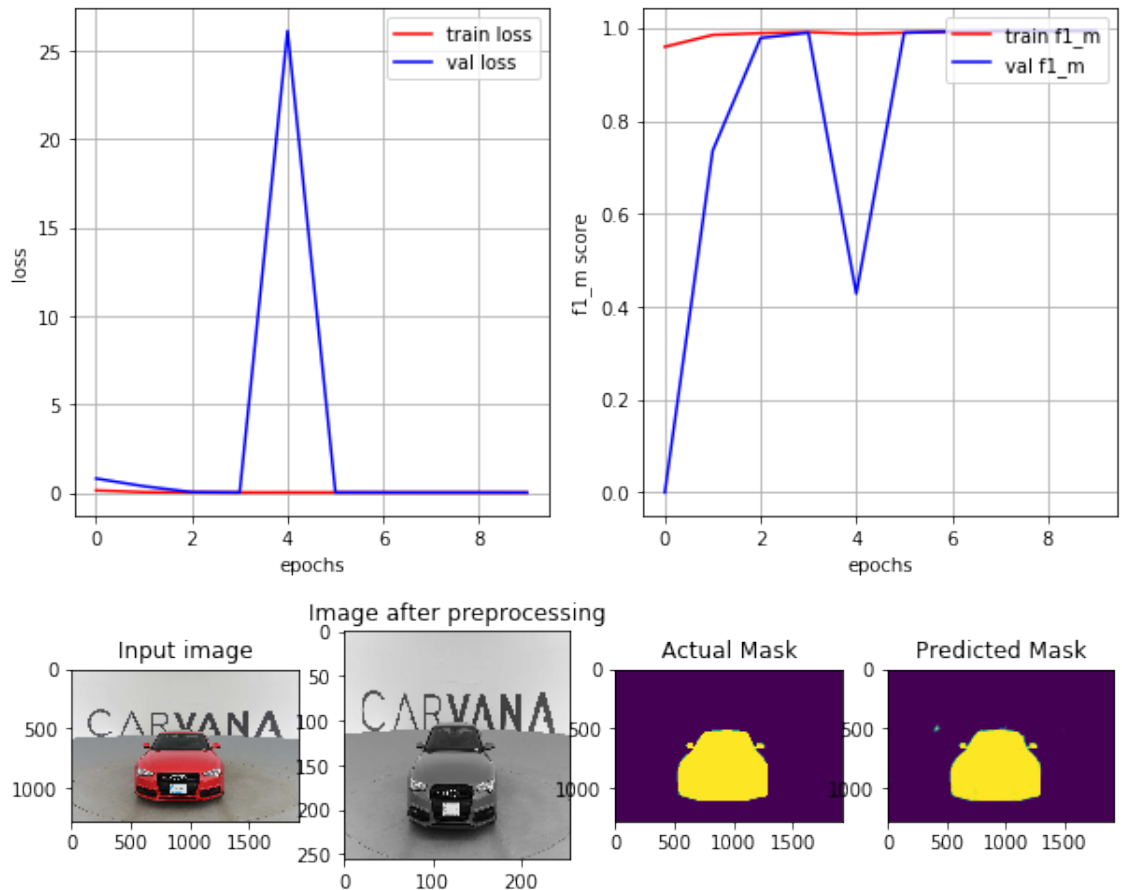
	loss	F1-score	precision	recall
Chuyển sang ảnh đen trắng	0.009431599	0.9915069	0.9913708	0.9916525
Chỉ lấy kênh màu xanh	0.010412044	0.99094784	0.99542123	0.9865221
Không tiền xử lý	0.010159233	0.9912001	0.99542534	0.98701715

Dưới đây là một số đồ thị, output của hàm:

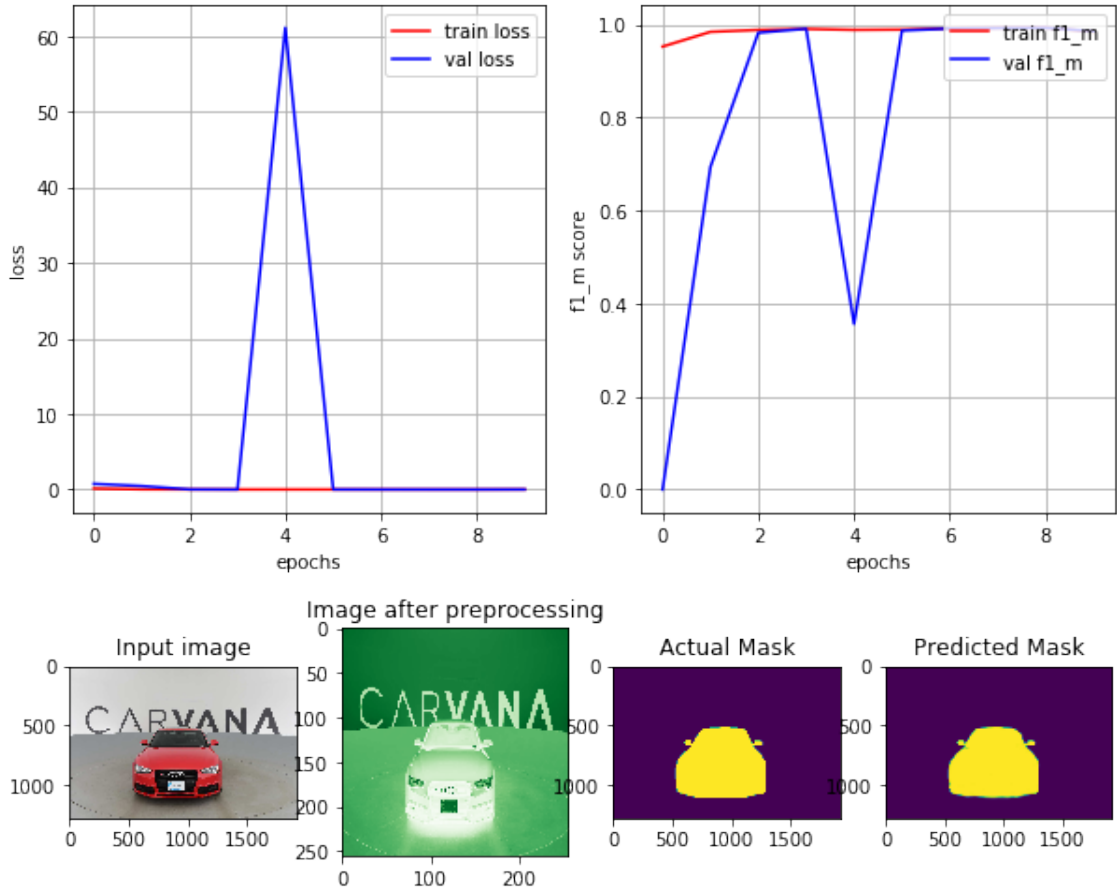
- Khi không tiền xử lý:



- Khi có chuyển ảnh sang dạng đen trắng:



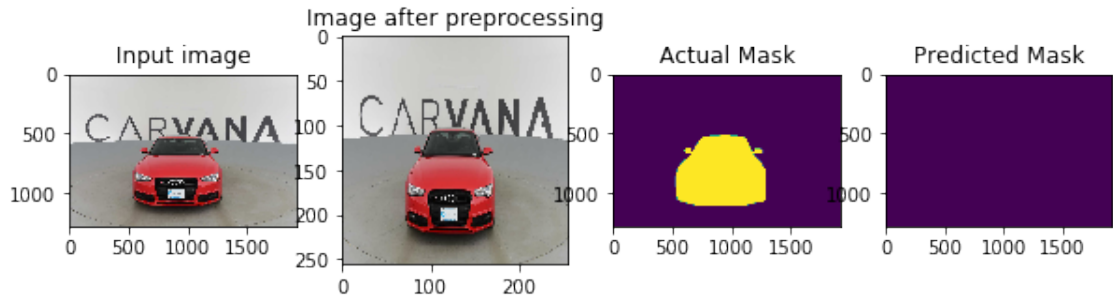
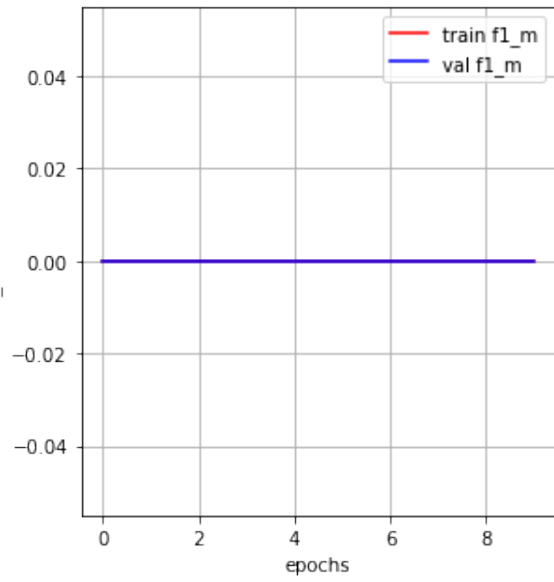
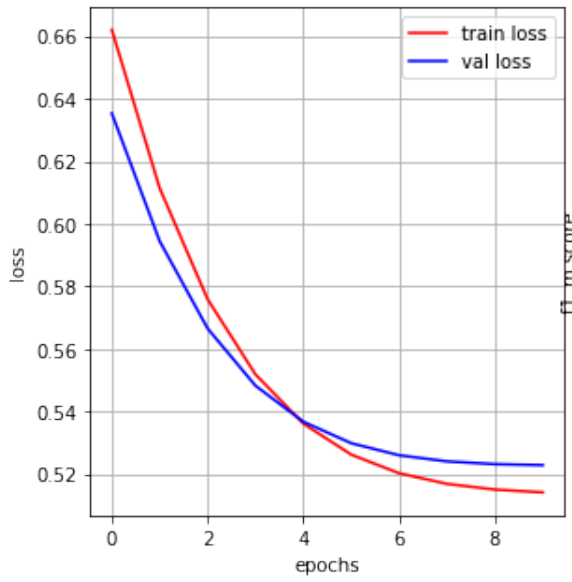
- Khi chỉ lấy một kênh màu green:



4.2 Initialization

Bên cạnh việc tiền xử lý, một điều quan trọng không kém, ảnh hưởng lớn đến quá trình train đó là việc khởi tạo các tham số của mô hình trước khi bắt đầu train. Mặc định, tất cả các tham số đều đã được khởi tạo trước bằng khởi tạo Glorot (còn gọi là khởi tạo Xavier).

Khi khởi tạo tất cả các tham số bằng 0, dù đã train cùng với số epochs như các mô hình trước (10 epochs), nhưng sau khi train thì mô hình đưa ra kết quả hoàn toàn không có giá trị gì, thậm chí F1-score của mô hình sau train chỉ là 0.0.



5 Nhận xét và Đánh giá

Trong bài toán Image Segmentation, mạng CNN tỏ ra rất vượt trội nhờ vào tính chất có thể tự trích xuất ra các đặc tính của ảnh trong quá trình train. Thêm nữa, mạng CNN có thể tìm được các đặc tính của ảnh mà không cần qua các bước xử lý ảnh thông thường, vậy nên nó có thể làm việc rất tốt với raw data.

Dù vậy, mô hình CNN cũng như mạng U-Net vẫn cần một lượng data cực kì lớn để train (đây là vấn đề mà đa số các mô hình đều gặp phải), và tốc độ train của mô hình CNN khá lâu nếu phải train từ đầu. Chúng ta có thể giải quyết phần nào vấn đề này nhờ pretrain.

Với bộ dữ liệu từ Carvana, do dữ liệu khá đơn giản nên model đạt kết quả khá tốt. Tuy nhiên, khi thử model với những tấm ảnh phức tạp hơn (có nhiều hơn 1 xe hoặc xe được chụp trên nền phức tạp hơn) thì kết quả đưa ra không được tốt lắm. Để model xử lý được những ảnh phức tạp hơn, chúng ta cần tăng độ phức tạp của data set (thêm ảnh với các nền khó hơn, nhiều xe trong một ảnh hơn, v.v)

Tài liệu

- [1] Nhóm mentor PiMA. Tài liệu trại hè PiMA 2019.
- [2] Vũ Hữu Tiệp. Blog Machine Learning cơ bản.
- [3] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016.
- [4] Michal Drozdal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The Importance of Skip Connections in Biomedical Image Segmentation, 2016.
- [5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015.
- [6] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015.
- [7] James Le. How to do Semantic Segmentation using Deep learning.
- [8] Saurabh Pal. Semantic Segmentation: Introduction to the Deep Learning Technique Behind Google Pixel's Camera!
- [9] Wikipedia. Convolutional neural network.
- [10] Nhiều tác giả. Website Towards Data Science.