PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

1. Thông tin về sinh viên

Họ và tên sinh viên: Đào Ngọc An

Điện thoại liên lạc 0349030007 Email: 20140003@student.hust.edu.vn

Lớp: CNTT 2.01 Hệ đào tạo: Đại học chính quy

Đồ án tốt nghiệp được thực hiện tại: Đại học Bách Khoa Hà Nội

Thời gian làm ĐATN: Từ ngày / / đến / /

2. Mục đích nội dung của ĐATN

Nghiên cứu và ứng dụng mô hình Deeplearning đối với bài toán Phân vùng ảnh, áp dụng vào bài toán phát hiện đường đi từ ảnh vệ tinh.

3. Các nhiệm vụ cụ thể của ĐATN

- Tìm hiểu tổng quan về Deeplearning : Các phép toán được sử dụng trong Deeplearning và trong bài toán Phân vùng ảnh.
- Các kiến trúc để giải quyết bài toán phân vùng ảnh.
- Cài đặt kiến trúc và chạy thử nghiệm.

4. Lời cam đoan của sinh viên:

Tôi – *Đào Ngọc An* - cam kết ĐATN là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *TS. Ban Hà Bằng*

Các kết quả nêu trong ĐATN là trung thực, không phải là sao chép toàn văn của bất kỳ công trình nào khác.

Hà Nội, ngày tháng năm

Tác giả ĐATN

Đào Ngọc An

5. Xác nhận của giáo viên hướng dẫn về mức độ hoàn thành của ĐATN và cho phép bảo vệ:

Hà Nội, ngày tháng năm

Giáo viên hướng dẫn

TS. Ban Hà Bằng

Lời cảm ơn

Lời đầu tiên, em xin chân thành cảm ơn các thầy, cô ở trường Đại học Bách Khoa Hà Nội nói chung và các thầy, cô ở Viện Công nghệ Thông tin và Truyền thông nói riêng, đã giảng dạy và truyền cảm hứng cho em trong suốt thời gian học tập và nghiên cứu ở trường, tạo một nền tảng vững chắc để có thể phát triển tốt hơn sau này.

Đặc biệt, em xin cảm ơn sâu sắc đến thầy Hà Ban Bằng – Giảng viên bộ môn Khoa học Máy tính, Viện Công nghệ Thông tin và Truyền Thông, đã tận tình hướng dẫn em trong suốt quá trình thực hiện đồ án này.

Dù đã cố gắng nhưng do thời gian và trình độ của bản thân có giới hạn nên chắc chắn nội dung đồ án này còn nhiều thiếu sót và tồn đọng một số hạn chế nhất định. Kính mong nhận được sự đóng góp của thầy cô và các bạn.

Sinh Viên

Đào Ngọc An

TÓM TẮT NỘI DUNG ĐỒ ÁN TỐT NGHIỆP

Trong những năm trở lại đây, học máy và khai phá dữ liệu đã trở thành trung tâm của sự chú ý và là chủ đề nghiên cứu phổ biến, rộng rãi trong cộng đồng các người nghiên cứu. Cùng với sự phát triển của các nước phát triển, một kho dữ liệu khổng lồ về ảnh vệ tinh đang được chờ khai thác. Mục tiêu của đồ án là ứng dụng Deeplearning để tự động phát hiện đường đi. Để thực hiện điều đó, đồ án của em sẽ đi theo các bước sau: (i) tổng quan về Deeplearning và các phép toán được sử dụng trong đó, (ii) một số nghiên cứu từ trước để giải quyết bài toán phân vùng ảnh, (iii) cài đặt mô hình chạy trên bộ dữ liệu có sẵn, (iv) kết quả và hướng phát triển. Đồ án sẽ tập trung vào phần thiết kế cho mô hình.

Mục lục

Danh mục hình vẽ

Danh mục bảng

Danh mục các từ viết tắt

CNN	Convolutional Neural Network
NN	Neural Network
DCNN	Deep Convolutional Neural Network
FCN	Fully-Convolutional Network
Conv	Convolution
DConv	Delation Convolution

Danh sách thuật ngữ

Loss function	Hàm mất mát
Mask	Mặt nạ nhị phân cho bài toán
Semantic Segmentation	

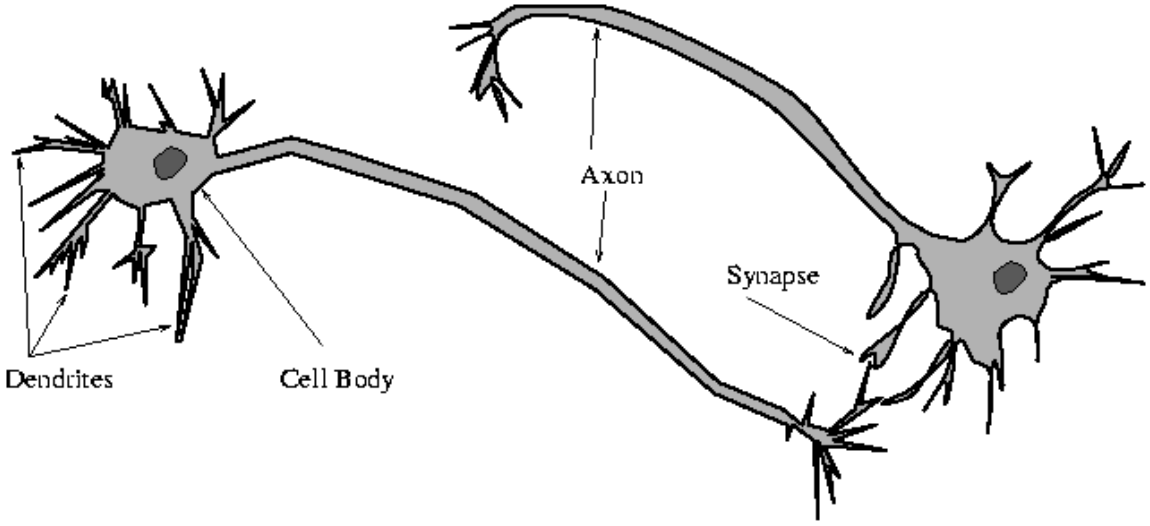
PHẦN 1: Tổng quan kiến trúc mạng NN và các vấn đề của một bài toán học máy

1.1 Tổng quan mạng Nơ-ron

1.1.1 Giới thiệu về mạng nơ-ron sinh học

Các mạng nơ-ron hiện nay được lấy cảm hứng chủ yếu từ mục tiêu mô hình hóa hệ thống thần kinh sinh học, trong phần này sẽ giới thiệu qua về cấu trúc nơ-ron sinh học.

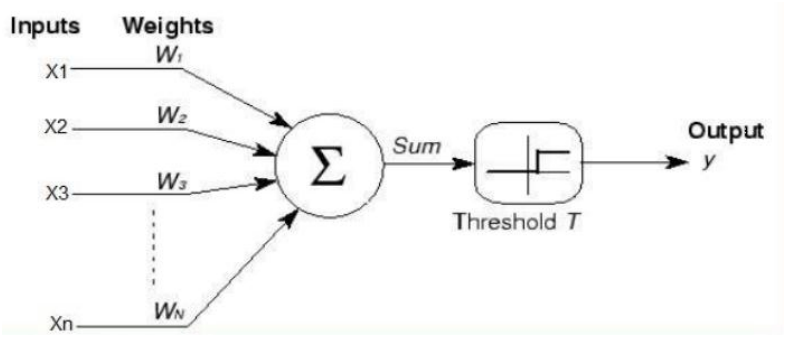
Theo các nhà nghiên cứu sinh học đã nghiên cứu, bộ não của con người chứa khoảng 10^{11} tế bào thần kinh (hay còn được gọi là nơ-ron). Mỗi nơ-ron kết nối tới khoảng 10^4 nơ-ron khác. Thành phần chính trong cấu trúc của một nơ-ron gồm: Cell Body, Dendrites, Synapses và Axon. Trong đó, Cell Body là thân của nơ-ron, các dendrites là các dây mảnh, dài, hình rễ cây, gắn liền với Cell Body, tiếp nhận dữ liệu từ các nơ-ron xung quanh liên kết với nó dưới dạng xung điện, cho Cell Body xử lý, Một loại dây dẫn tín hiệu khác cũng gắn với Cell Body là axon. Khác với Dendrites, Axon có khả năng phát các xung điện thế, chúng là dây dẫn tín hiệu từ nơron đi các nơi khác. Axon nối với các dendrites của các nơron khác thông qua những mối nối đặc biệt gọi là synapse.



Hình 1. Các thành phần chính của nơ-ron sinh học

Mỗi nơron nhận các tín hiệu hóa điện từ nơron khác ở các dendrites, khi một nơron nhận một tín hiệu, nơron đó có thể bị kích thích. Cell Body của các nơron nhận được tín hiệu sẽ tính tổng các tín hiệu đến. Một nơron sẽ bị kích thích nếu tổng số tín hiệu nhận được ở Cell Body vượt quá một ngưỡng nhất định. Nếu tổng các tín hiệu điện đầu vào đủ mạnh để kích thích nơron, nó sẽ truyền một tín hiệu hóa điện dọc axon và đưa tín hiệu này tới các nơron khác. Bộ não của con người bao gồm các nơron truyền tín hiệu hóa điện này liên kết với nhau. Từ một số lượng lớn các đơn vị xử lý cơ bản (mỗi đơn vị thực hiện tính tổng trọng số các giá trị đầu vào của nó, sau đó kích thích một tín hiệu nhị phân nếu tổng của giá trị đầu vào vượt qua một ngưỡng nhất định) bộ não người có thể thực hiện các nhiệm vụ phức tạp.

1.1.2 Perceptron



Hình 2. Cấu trúc mạng Perceptron

Perceptron là một mô hình Neural Network đơn giản nhất. Trong mô hình này, một nơ-ron sẽ nhận các tín hiệu vào X_i với các trọng số là W_i , và cho ra một tín hiệu là Y . Tổng trọng số của các tín hiệu vào là:

$$sum = \sum_{i=1}^{n} W_i x_i$$

Kết quả này được so sánh với ngưỡng T của nơ-ron, Nếu *sum* lớn hơn hoặc bằng ngưỡng T thì Y cho kết quả là 1, và ngược lại Y cho kết quả là 0. Đặt $b = -T$,Y biểu diễn toán học như sau:

$$Y = f(\sum_{i=1}^{n} W_i x_i - T)$$

Để thuận tiện trong tính toán, người ta thường đặt $x_0 = 1$ và $W_0 = b$, khi đó, đặt:

$$z = \sum_{i=0}^{n} W_i x_i$$

Hàm f gọi là hàm kích hoạt, có tác dụng biến đổi tín hiệu vào thành tín hiệu ra:

$$f = \begin{cases} 0, z < 0 \\ 1, z \geq 0 \end{cases}$$

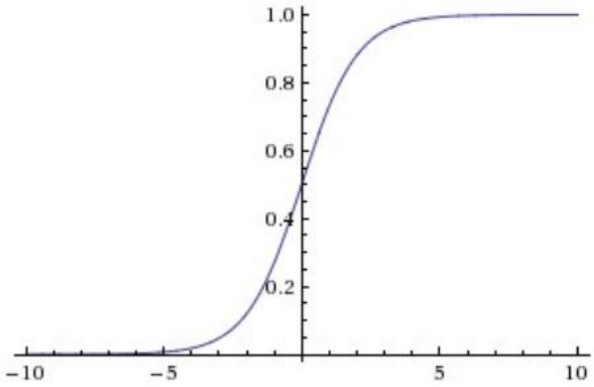
Trong mô hình trên, các trọng số của tín hiệu vào đóng vai trò là Synapse, các tín hiệu vào tương ứng với các Dendrites, tín hiệu ra truyền qua Axon tương ứng với giá trị đầu ra Y

1.1.3 Hàm kích hoạt .

Hàm kích hoạt f có vai trò biến đổi tín hiệu vào thành tín hiệu ra. Để thuật tiện cho các thuật toán tối ưu tham số sau này, hàm kích hoạt thường có đạo hàm khác 0 trên hầu hết mọi điểm (thay vì sử dụng hàm phân loại nhị phân như trên). Một số hàm thường được sử dụng là:

- Sigmoid: là một hàm phi tuyến, có công thức:

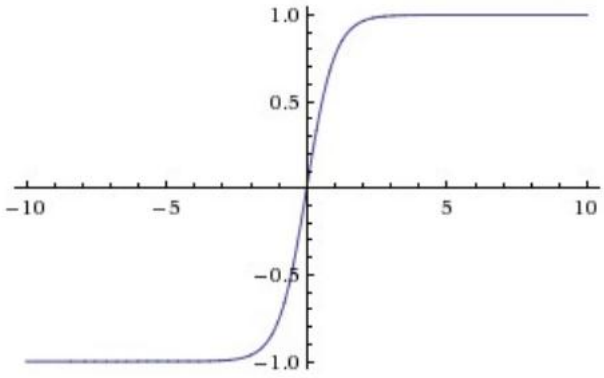
$$\sigma(x) = \frac{1}{1 + e^{(-x)}}$$



Hình 3. Hàm sigmoid, nhận đâu vào là một số thực và cho ra một số thuộc (0, 1)

- Tanh: là một hàm phi tuyến, có công thức:

$$\tanh(x) = \frac{e^{2x}+1}{e^{2x}-1}$$



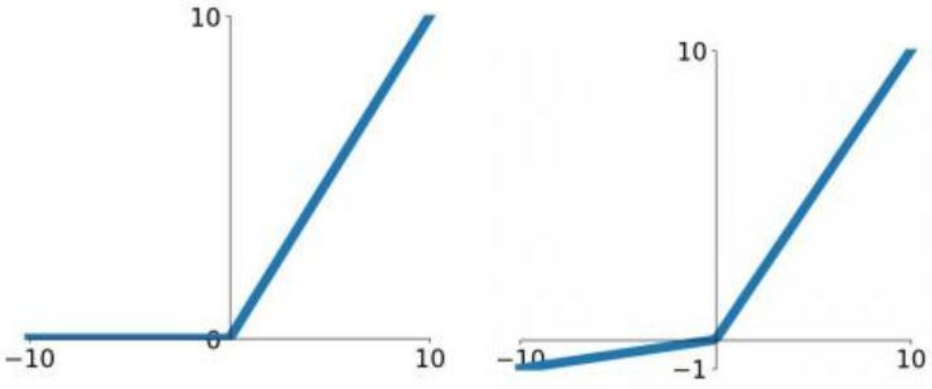
Hình 4. Hàm tanh, nhận đầu vào là một số thực và cho ra một số thuộc (-0.5, 0.5)

- ReLU: là một hàm tuyến tính, được sử dụng phổ biến hiện này, có công thức

$$f(x) = \max(0, x)$$

- Leaky ReLU: khắc phục một số hạn chế của hàm ReLU, có công thức:

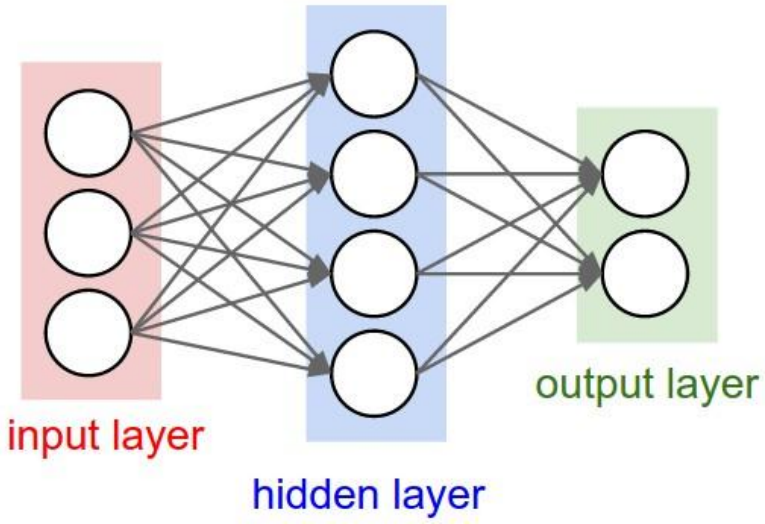
$$f(x) = \begin{cases} -\alpha x, x < 0 \\ x, x \geq 0 \end{cases}$$



Hình 5. Hàm ReLU(bên trái) và hàm Leaky ReLU(bên phải)

1.1.4 Kiến trúc mạng NN

Mạng nơ-ron là sự kết hợp của các tầng perceptron hay còn được gọi là perceptron đa tầng(Multilayer perceptron, MLP), như hình vẽ dưới:

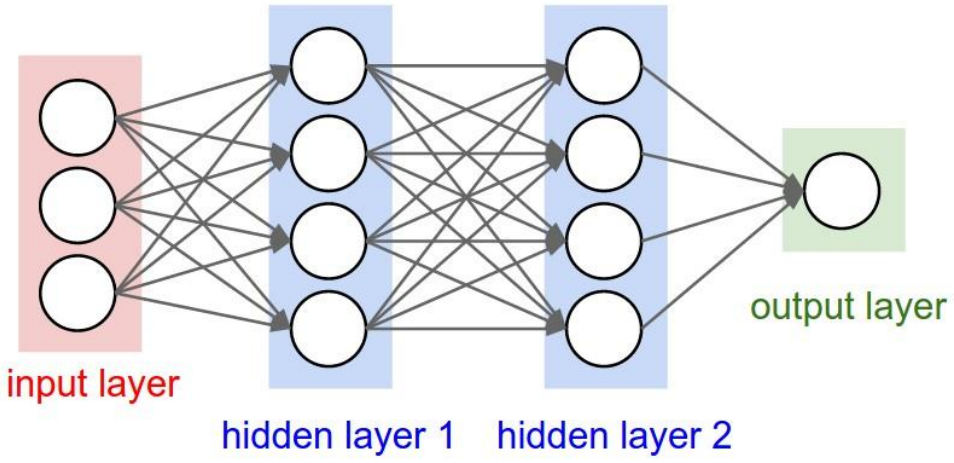


Hình 6. Mạng nơ-ron với một lớp ẩn

Một mạng nơ-ron sẽ có 3 kiểu tầng:

- Input layer: là tầng bên trái cùng của mạng, thể hiện cho các đầu vào của mạng.
- Output layer: là tầng bên phái cùng của mạng, thể hiện cho các đầu ra của mạng.
- Hidden layer: là tầng giữa Input layer và Hidden layer, thể hiện cho suy luận của mạng

Hình trạng của mạng được định nghĩa bởi: số lớp trong mạng, số đơn vị trong mỗi lớp và sự liên kết giữa các lớp trong mạng. Trong mạng nơ-ron mỗi nút có thể sử dụng các hàm kích hoạt tùy ý, nhưng trong lập trình thường dùng chung một loại cho thuận tiện. Ở mỗi tầng, số lượng mỗi đơn vị có thể tùy ý, nhưng khi làm việc người ta thường để số đơn vị ở các lớp ẩn là như nhau. Ngoài ra, các nơ-ron ở các tầng thường được liên kết đầy đủ với tầng phía trước, tạo thành mạng liên kết đầy đủ(full-connected network).



Hình 7. Mạng nơ-ron với hai lớp ẩn

1.1.5 Lan Truyền tiến.

Ta thấy, tất cả các nốt mạng được kết nối đôi một với nhau theo một chiều duy nhất từ tầng vào tới tầng ra, tức là mỗi nốt ở một tầng nào đó sẽ nhận đầu vào là tất cả các nốt ở tầng trước đó mà không có chiều ngược lại – lan truyền tiến.

Đặt $W^{(l+1)}$ là ma trận trọng số liên kết giữa lớp (l) và $(l+1)$, $a^{(l)}$ là output của tầng (l), $z^{(l)}$ được định nghĩa như công thức dưới đây, được lưu lại trong quá trình lan truyền tiến để sử dụng cho phần lan truyền ngược(được giới thiệu sau):

$$z^{(l+1)} = W^{(l+1)}.a^{(l)}$$

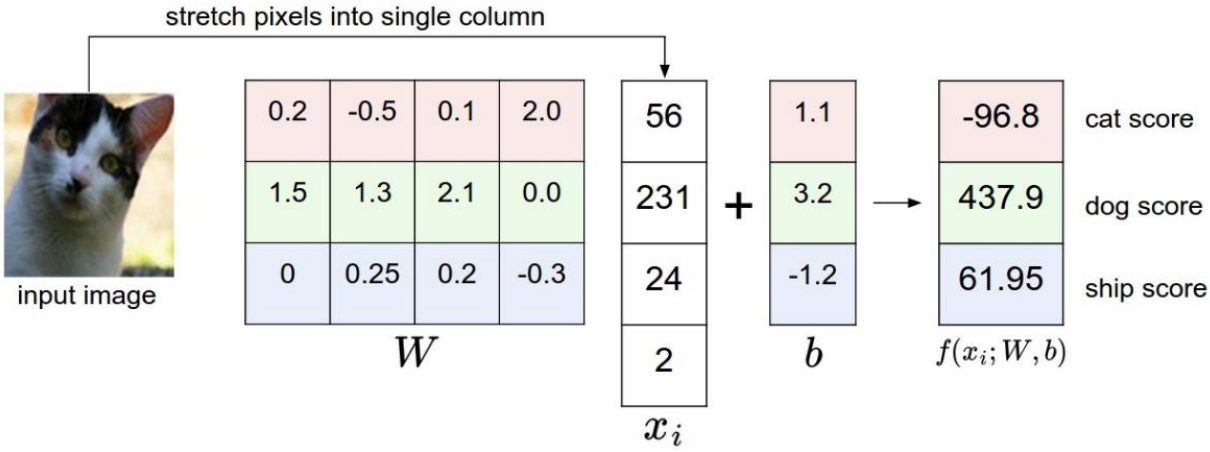
$$a^{(l+1)} = f(z^{(l+1)})$$

Với $a^{(0)}$ là input layer, $a^{(L)}$ là output layer, l từ 0 đến $L-1$ (L = số lớp ẩn +1).

1.1.6 Huấn luyện mạng.

1.1.6.1 Loss function

Giả sử có một training dataset gồm nhiều ảnh $x_i \in \mathbb{R}^D$, mỗi ảnh được gắn với nhãn y_i, với $i = 1...N$ và $y_i \in 1...K$, N là số ví dụ để học, K là số nhãn riêng biệt trong tập học. Ví dụ , trong bộ dữ liệu CIFAR-10 , chúng ta có tập training dataset của N = 50,000 ảnh, với mỗi ảnh gồm D = 32 x 32 x 3 = 3072 điểm ảnh, và K = 10, do có 10 nhãn riêng biệt (chó, mèo, ô-tô, …). Sử dụng mạng nơ-ron như phần trước chúng ta ánh xạ mỗi bức ảnh với một vector ứng với điểm của từng lớp $f : \mathbb{R}^D \mapsto \mathbb{R}^K$.



Hình 8. Mạng nơ-ron như một ánh xạ đầu vào với vector điểm của từng lớp

Như hình trên, ví dụ ánh xạ một bức ảnh tới điểm của từng lớp. Để dễ hình dung, chúng ta giả sử ảnh chỉ có 4 điểm ảnh như trên, và chúng ta có 3 lớp (màu đỏ (con mèo), màu xanh lá (con chó), màu xanh lá trời (con thuyền). Mạng nơ-ron của chúng ta có L = 1 (không sử dụng lớp ẩn, còn gọi là linear classifer). Như hình trên, ma trận trọng số W không được coi là tốt: bộ phân loại gán ảnh con mèo của chúng ta với một điểm con mèo rất thấp, thực tế, tập trọng số của chúng ta tin ảnh con mèo là một con chó (điểm của con chó rất cao).

Để đánh giá tập trọng số của chúng ta tốt hay xấu, chúng ta định nghĩa một hàm gọi là loss function (còn được gọi là cost hay object). Để dễ hình dung, loss có giá trị cao khi bộ phân loại của chúng ta hoạt động kém (như điểm của con chó là rất cao so với điểm của con mèo như ảnh ở trên).

Ta có $z_j^{(L)}$ là điểm của lớp j, sau khi sử dụng thuật toán lan truyền tiến. Sử dụng hàm kích hoạt như sau:

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Ta thu được đầu ra của mạng là xác xuất của từng nhãn, ví dụ $z^{(L)} = (3.2, 5.1, -1.7)$ thì $a^{(L)} = f(z^{(L)}) = (0.13, 0.87, 0.00)$ nghĩa là xác xuất đầu vào được phân vào nhãn 1 là 13%, vào nhãn 2 là 87%,… Theo Lý thuyết thông tin, cross-entropy giữa một phân bố "đúng" p và một phân bố ước lượng q được định nghĩa như sau:

$$H(p,q) = -\sum_x p(x) \log q(x)$$

Với các bài toán phân loại nhãn, p_i là one-hot vector – vector có các phần tử bằng 0 ngoại trừ vị trí y_i có giá trị bằng 1 và $q_i = a_i^{(L)}$.

Loss function được định nghĩa như sau:

$$L(\mathrm{W}) = \frac{1}{N}\sum_{i=1}^{N} L_i = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} -y_{i,j} \log(a_j^{(L)})$$

Qúa trình học là quá trình tìm W sao cho Loss đặt giá trị nhỏ nhất.

1.1.6.2 Back Propagation – Giải thuật lan truyền ngược.

Để tính đạo hàm của hàm lỗi $\nabla L(W)$ trong mạng nơ-ron, ta sử dụng một giải thuật đặc biệt là giải thuật lan truyền ngược (back propagation).Nhờ có giải thuật này mà mạng NN thực thi hiệu quả được và ứng dụng ngày một nhiều đến tới tận ngày nay.

Về cơ bản phương pháp này được dựa theo quy tắc đạo hàm của hàm hợp và phép tính ngược đạo hàm để thu được đạo hàm theo tất cả các tham số cùng lúc chỉ với 2 lần duyệt mạng .

Ví dụ với mạng NN như các phần trình bày bên trên(giải thuật này áp dụng cho các mạng khác được trình bày sau này:

1. Lan truyền tiến:

   Tính lần lượt $a^{(l)}$ với $l = 1...L$ theo công thức:

   $$z^{(l)} = \mathrm{W}^{(l)}.a^{(l-1)}$$

   $$a^{(l)} = f(z^{(l)})$$

   Trong đó a^0 là đầu vào của mạng NN

2. Tính đạo hàm theo z ở tầng ra

   $$\frac{\partial L}{\partial \mathrm{z}^{(L)}} = \frac{\partial L}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial \mathrm{z}^{(L)}}$$

   Với $a^{(L)}, z^{(L)}$ được tính ở bước 1.

3. Lan truyền ngược:

   Tính đạo hàm theo z ngược từ $l = (L-1) \to 1$ theo công thức:

   $$\frac{\partial L}{\partial \mathrm{z}^{(l)}} = \frac{\partial L}{\partial \mathrm{z}^{(l+1)}} \frac{\partial \mathrm{z}^{(l+1)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial \mathrm{z}^{(l)}}$$

   $$\frac{\partial L}{\partial \mathrm{z}^{(l)}} = \left((\mathrm{W}^{(l+1)})^T \frac{\partial L}{\partial \mathrm{z}^{(l+1)}} \right) \frac{\partial a^{(l)}}{\partial \mathrm{z}^{(l)}}$$

   Với $z^{(l)}$ được tính ở bước 1 và $\frac{\partial L}{\partial \mathrm{z}^{(l+1)}}$ được tính ở bước trước.

4. Tính đạo hàm

   Đạo hàm theo tham số W được tính bằng công thức:

   $$\frac{\partial L}{\partial \mathrm{W}^{(l)}} = \frac{\partial L}{\partial \mathrm{z}^{(l)}} \frac{\partial \mathrm{z}^{(l)}}{\partial \mathrm{W}^{(l)}}$$

   $$\frac{\partial L}{\partial \mathrm{W}^{(l)}} = \frac{\partial L}{\partial \mathrm{z}^{(l)}} (a^{(l-1)})^T$$

   Với $a^{(l-1)}$ được tính ở bước 1 và $\frac{\partial L}{\partial \mathrm{z}^{(l)}}$ được tính ở bước 3

Với một mạng nơ-ron tổng quát, thuật toán lan truyền ngược sẽ chia thành hai giai đoạn: lan truyền, cập nhật trọng số.

Giai đoạn 1: Lan truyền

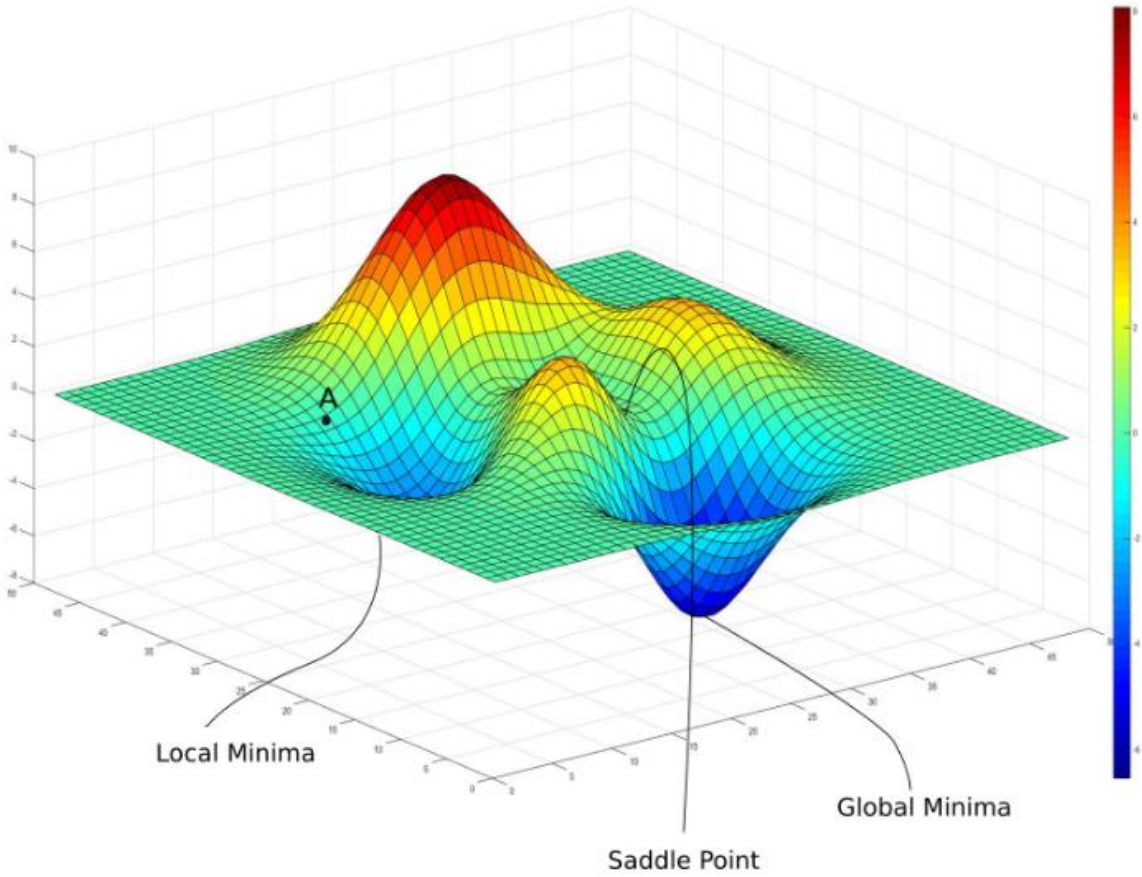
- Lan truyền tiến của một đầu vào của mô hình huấn luyện thông qua mạng nơ-ron, thu được đầu ra của mạng này, tính giá trị của loss function.
- Lan truyền ngược: tính đạo hàm các tham số mô hình của từng layer từ dưới lên trên, dùng quy tắc đạo hàm chuỗi.

Giai đoạn 2: Cập nhật trọng số

- Đi ngược đạo hàm: các tham số mô hình của từng layer được trừ một tỷ lệ nhỏ của đạo hàm của chính mình
- Một số chiến lược sẽ được trình bày trong phần sau.

Lặp lại hai giai đoạn đến khi loss function hội tụ (local minima) hoặc thoải mãn điều kiện nào đó như: đến số một bước lặp nhất định, hoặc quá lâu mà loss function không được cải thiện, …

1. 1.6.3 Thuật toán tối ưu cho mạng NN



Trong học máy, để mô hình chúng ta "fit" với tập dữ liệu, chúng ta phải tìm giá trị nhỏ nhất của loss function. Với môi hình thực tế có hàng triệu tham số, việc tìm được dường như là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm cực trị địa phương (local optimum), và ở một mức nào đó, giá trị đó được coi như là nghiệm cần tìm của bài toán.

Các điểm local optimum được định nghĩa là nghiệm của phương trình đạo hàm bậc nhất. Hiển nhiên, nếu ta tìm được tất cả các điểm local optimum thì ta có điểm global optimum. Tuy nhiên, trong hầu hết các trường hợp thì việc giải phương trình đạo hàm bằng 0 là bất khả thi (do tính phức tạp của hàm hợp, số lượng các tham số, dữ liệu có số chiều lớn hoặc từ việc có quá nhiều dữ liệu).

Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta khởi tạo ngẫu nhiên, sau đó dùng thuật toán lặp để mỗi bước chúng ta cần càng đến gần điểm local optimum. Gradient Descent (GD) và các biến thể là một trong những thuật toán phổ biến nhất để thực hiện việc tối ưu hóa và vì thế nó là cách phổ biến để tối ưu hóa cho các NN. Tại thời điểm này, hầu như tất cả các framework Deep-learning đều implement các thuật toán GD với công thức cơ bản như sau:

```
while True:
    dW = compute_grad(loss, weights, data)
    W  = W - learning_rate * dW
```

Trong đó dW là đạo hàm của từng tham số của loss function và *learning_rate* (thường được kí hiệu là α) là tốc độ học (cũng là một tham số đặc biệt của mô hình)

Thuật toán này sẽ tính toán gradient cho toàn bộ tập dữ liệu và thực hiện cập nhật một lần. Việc này sẽ mất rất nhiều bộ nhớ (đặc biệt trong deep-learning khi thuật toán phải lưu lại rất nhiều feature maps) với tập dữ liệu lớn. Thuật toán **Stochastic Gradient descent** (SGD) giải quyết vấn đề này bằng cách chia nhỏ tập dữ liệu thành nhiều batch, khi duyệt qua toàn bộ dữ liệu thì tham số của mô hình sẽ được cập nhật nhiều lần (bằng số lượng batch):

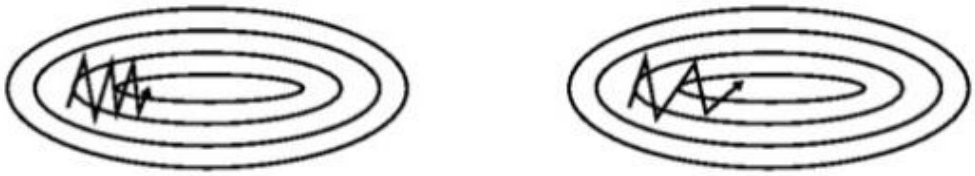
```
while True:
    while data_batch = next_batch(data, batch_size):
        dW = compute_grad(loss, weights, data_batch )
        W  = W - learning_rate * dW
```

SGD cùng với Momentum

Dưới góc nhìn vật lí, SGD với momentum (đà) cho việc hội tự có gia tốc, làm nhanh quá trình hội tự trên các đường cong có độ dốc lớn, nhưng cũng đồng thời làm giảm sự dao động khi gần hội tụ khi gần hội tụ:

```
v = 0
while True:
    while data_batch = next_batch(data, batch_size):
        dW = compute_grad(loss, weights, data_batch )
        v = mu * v - learning_rate * dW
        W = W + v
```

Công thức trên chúng ta thêm phần "quán tính" vào trước và hệ số *mu* thường bằng 0.9



Hình 9. SGD (bên trái) và SGD với momentum (bên phải)

Adagrad

Không giống như các cách thức trước, learning_rate được giữ nguyên trong quá trình học, thì adagrad coi learning_rate như là một tham số:

```
cache = 0
while True:
    while data_batch = next_batch(data, batch_size):
        dW = compute_grad(loss, weights, data_batch )
        cache += dW ** 2
        W = W - learning_rate / (math.sqrt(cache) + eps) * dW
```

Công thức trên biến *cache* có kích thước bằng kích thước của tham số của mô hình, nó giữ tổng bình phương của đạo hàm, nó cập nhật tạo các update lớn với các tham số có sự thay đổi nhiều và các update nhỏ với các tham số có sự thay đổi ít. Lợi ích của Adagrad là tránh việc điều chỉnh learning_rate bằng tay, thường khởi tạo là 0.01 và thuật toán sau đó sẽ tự động điều chỉnh. Điểm yêu của Adagra là tổng bình phương biến thiên sẽ lớn theo thời gian dẫn đến làm learning_rate cực kì nhỏ, làm việc tranning trở nên đóng băng.

RMSprop

Là thuật toán khắc phục nhược điểm của Adagrad (learning_rate giảm quá nhanh):

```
cache = 0
while True:
    while data_batch = next_batch(data, batch_size):
        dW = compute_grad(loss, weights, data_batch )
        cache += decay_rate * cache + (1 - decay_rate) * dW**2
        W = W - learning_rate / (math.sqrt(cache) + eps) * dW
```

Công thức trên, *decay_rate* là hyper parameter, thường được mặc định là [0.9, 0.99, 0.999]

Adam

Là thuật toán gần giống RMSprop kết hợp với momentum:

```
m = 0
v = 0
while True:
    while data_batch = next_batch(data, batch_size):
        dW = compute_grad(loss, weights, data_batch )
        m = beta1 * m + (1-beta1) * dW
        v = beta2 * v + (1-beta2) * (dW**2)
        W = W - learning_rate / (math.sqrt(v) + eps) * m
```

Công thức trên, bước cập nhật (08.) gần như giống hệt bước cập nhật của RMSprop, ngoại trừ việc sử dụng *m* thay cho *d*W (theo tác giả thì có tác dụng làm mượt hơn). Các hyper parameter *beta1, beta2, eps* được tác giả đề xuất lần lượt là [0.9, 0.999, 1e-8].

1.2 Mạng Convolutional Neural Networks

1.2.1 Giới thiệu

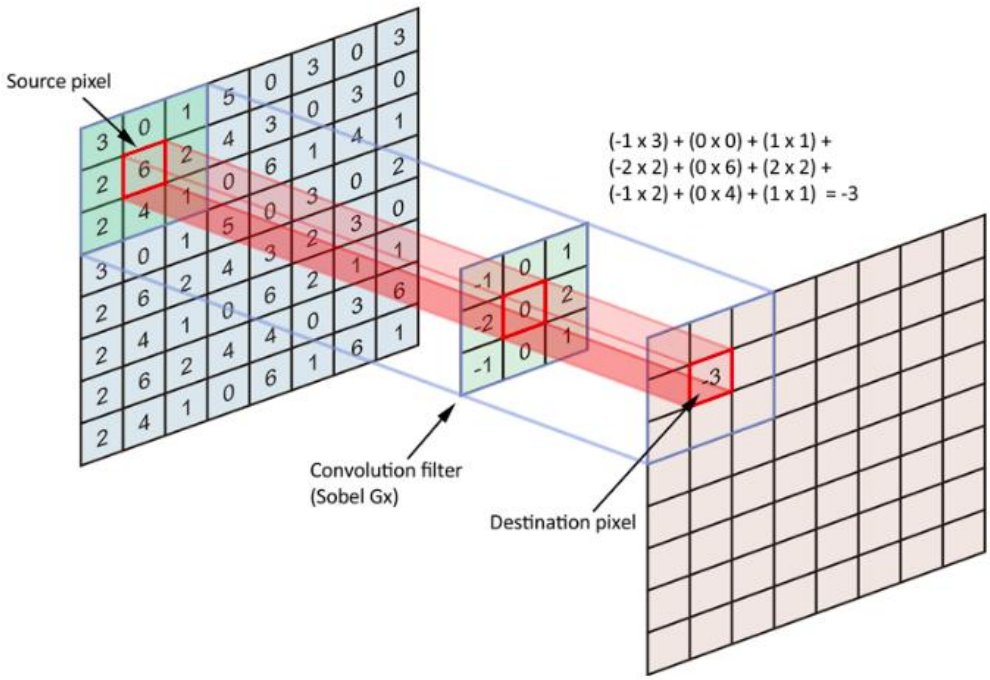
Những năm gần đây, ta đã chứng kiến được nhiều thành tựu vượt bậc trong ngành Thị giác máy tính (Computer vision). Các hệ thống xử lý ảnh lớn như Facebook, Google hay Amazon đã đưa vào sản phẩm của mình những chức năng thông minh như nhận diện khuôn mặt người dùng, phát triển xe hơi tự lái hay tìm kiếm theo hình ảnh.

Convolutional Neural Networks (CNNs- Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến giúp chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay. Deep ở đây là nhiều lớp được sử dụng, được phát triển theo thời gian, những năm 2012 thì trên được lớp được sử dụng được gọi là deep, hiện nay đã

có nhiều mô hình có trên một trăm lớp được sử dụng (ResNet). Trong đồ án này em sẽ giới thiệu các lớp được sử dụng để giải quyết bài toán Semantic Segmentation.

1.2.2 Lớp tích chập

Convolution layer (Conv) là tầng quan trọng nhất trong cấu trúc của CNNs. Conv dựa trên lý thuyết xử lý tín hiệu số, việc lấy tích chập sẽ giúp trích xuất được thông tin quan trọng từ dữ liệu. Lớp tích chập sử dụng một bộ lọc (filter, còn được gọi là kernel) có kích thước nhỏ (thường là 3x3 hoặc 5x5) chiếu vào một vùng dữ liệu đầu vào và tiến hành tính tích chập (tổng của tích từng phần tử tương ứng) trên bộ lọc (đã được quay 180^o) và giá trị trên vùng dữ liệu đầu vào được chiếu. Hình dưới minh họa cho phép tích chập:



Hình 10. Tích chập giữa vùng dữ liệu vào và bộ lọc (đã được quay 180^o)

Đặt Vùng dữ liệu vào là $F:\mathbb{Z}^2 \to \mathbb{R}$, $\Omega_r = [-r,r]^2 \cap \mathbb{Z}^2$ và $k:\Omega_r \to \mathbb{R}$ là hàm rời rạc cho kernel kích thước $(2r+1)^2$. Phép tích chập * giữa F và k được tính theo công thức sau:

$$(F*k)(p) = \sum_{s+t=p} F(s)k(t).$$

Với mọi điểm p trên vùng dữ liệu, ví dụ $p = (100,100)$, r = 1 thì:

$(F*k)(100,100) = F(99,99).k(1,1) + ... + F(101,101).k(-1,-1)$ là tổng của 9 phép nhân.

Áp dụng trong CNNs, giả sử Vùng dữ liệu vào có kích thước (C_{in}, H, W) và có C_{out} kernel thì output có kích thước $(C_{out}, H_{out}, \mathrm{W}_{out})$ được tính như sau:

$$\mathrm{out}(\mathrm{C}_{\mathrm{out}_j}) = \mathrm{bias}(\mathrm{C}_{\mathrm{out}_j}) + \sum_{k=0}^{C_{in}-1} \mathrm{weight}(\mathrm{C}_{\mathrm{out}_j}) * \mathrm{input}(k)$$

Với C là số channel, H và W là độ dài và độ rộng của vùng dữ liệu.

Với padding_h, padding_w là số số 0 thêm vào cả 2 cạnh của input theo lần lượt chiều dài và rộng; stride_h, stride_w là bước dịch chuyển của kernel; kernel_size_h, kernel_size_w là kích thước của kernel thì (H_{out}, W_{out}) được tính như sau:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding\_h} - \text{kernel\_size\_h}}{\text{stride\_h}} + 1 \right\rfloor$$

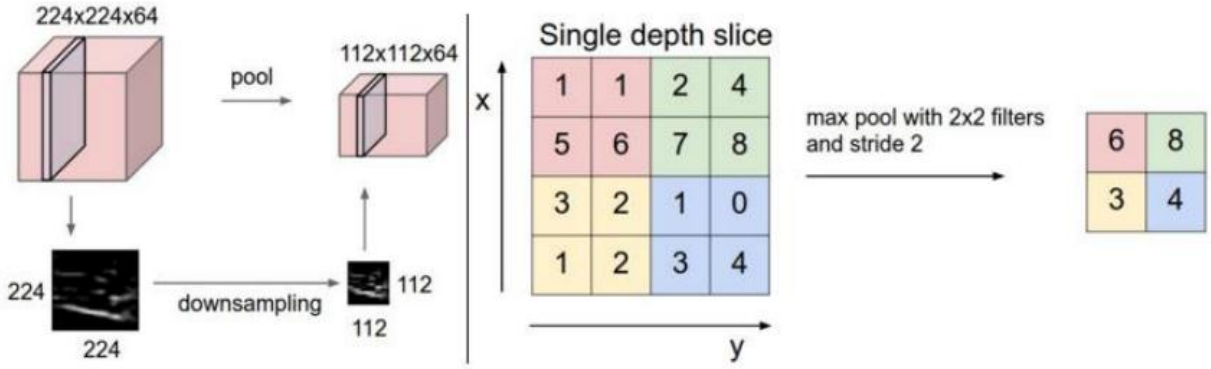
$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding\_w} - \text{kernel\_size\_w}}{\text{stride\_w}} + 1 \right\rfloor$$

Back propagation: tinh

Receptive field: vùng tiếp nhận, là vùng dữ liệu đầu vào (thường là bức ảnh ban đầu) mà kernel hiện tại chiếu vào. Là một thuộc tính quan trọng trong quá trình thiết kế mạng NN. Cách tính kích thước receptive field sẽ được trình bày trong phần tiếp.

1.2.3 Lớp pooling

Pooling layer là một trong những thành phần tính toán chính trong cấu trúc CNNs. Xét về mặt toán học pooling thực chất là quá trình tính toán trên ma trận, trong đó mục tiêu khi tính toán là giảm kích thước ma trận nhưng vẫn làm nổi bật lên được đặc trưng có trong ma trận vào. Có nhiều toán tử pooling như Mean-Pooling, Max-Pooling, L_2-Pooling, … Công thức tính kích thước đầu ra tương tự như công thức tính kích thước đầu ra của Conv. Thông thường kernel_size được chọn bằng 3, stride bằng 2 thì kích thước ra mỗi chiều bị giảm đi 2. Ví dụ dưới đây cho phép Max-Pooling, các phép khác được thực hiện tương tự:



Hình 11. Max Pooling với stride = 2, kernel_size = 2.

Backpropagation:

Receptive field: Người ta coi ảnh đầu vào có receptive size (r) bằng 1, dùng biến jump (j) để lưu bước nhảy của kernel hiện tại so với ảnh ban đầu. Qua một layer có kernel kích thước k, và stride s, thì receptive size được tính như sau:

$$j_{out} = j_{in} \times s$$

$$r_{out} = r_{in} + (k-1) \times j_{in}$$

Ví dụ, dùng 3 lớp Conv có kernel size là 3x3 sau ảnh đầu vào thì feature maps ra sẽ có kích thước của receptive là 7x7. Ghi chú, để feature maps có kích thước receptive là 7x7 thì có thể dùng trực tiếp một lớp Conv có kernel size là 7x7, nhưng khi dùng 3 lớp Conv3x3 thì

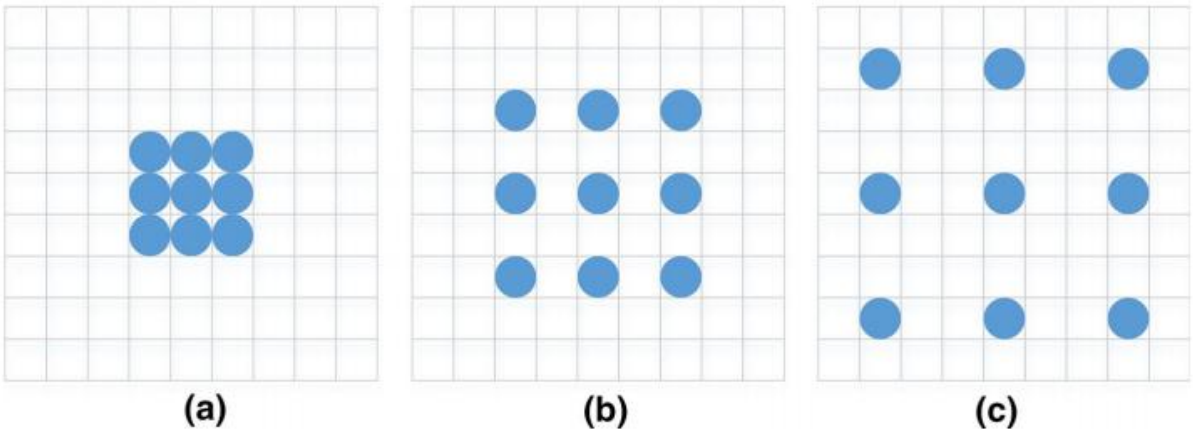
số tham số là $3\times(9\times C^2)$, nhỏ hơn số tham số khi dùng Conv7x7 là $49\times C^2$, rất hữu ích khi tạo mô hình NN có hàng trăm layer (giả sử số channel của feature maps của input layer vào output layer đều là C).

1.2.4 Tích chập mở rộng (Dilated Convolution layer)

Là bản mở rộng của tích chập bên trên. Công thức là :

$$(F *_l k)(p) = \sum_{s+lt=p} F(s)k(t).$$

Với l là chỉ số mở rộng của tích chập, nếu l là 1 thì công thức trên trở thành phép convolution thông thường.

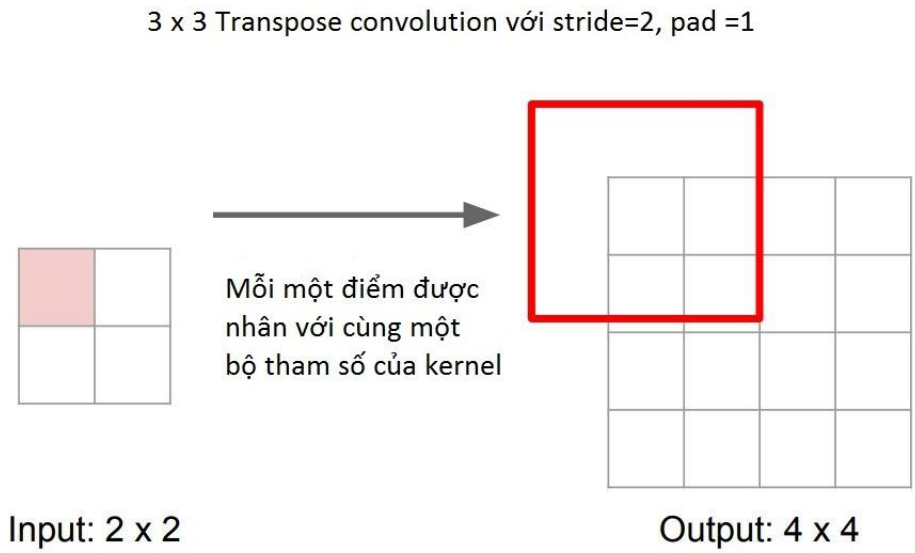


Hình xx. Một ví dụ cho dilated convolution. Kernel có kích thước 3x3, (a) l=1 là phép convolution thông thường, (b) phép dilated convolution với tham số dilation l=2, (c) phép dilated convolution với tham số dilation l=3.

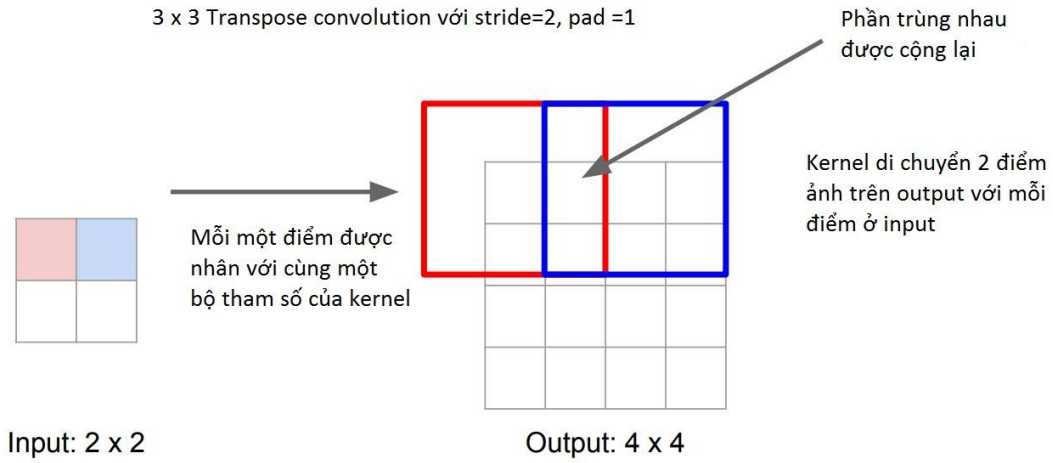
1.2.5 Transposed Convolution

Là một layer dùng để khôi phục kích thước của feature maps và có khả năng "học", được dùng trong các bài toán l

Một lớp Convolution (hoặc Pooling) có kernel 3x3, stride 2 và padding 1 (với Pooling thì padding 0) thì với input có kích thước là $(C_{in}, H_{in}, \mathrm{W}_{in})$ thì output có kích thước là $(C_{out}, H_{in}/2, \mathrm{W}_{in}/2)$. Thì một lớp Transpose Convolution có kernel 3x3, stride 2 và padding 1 sẽ khôi phục kích thước của feature maps $(C_{in}, H_{in}, \mathrm{W}_{in}) \rightarrow (C_{out}, H_{in}\times 2, \mathrm{W}_{in}\times 2)$ .Hình dưới đây sẽ miêu tả cụ thể hơn quá trình này:



Hình xx. Transpose Convolution (a)



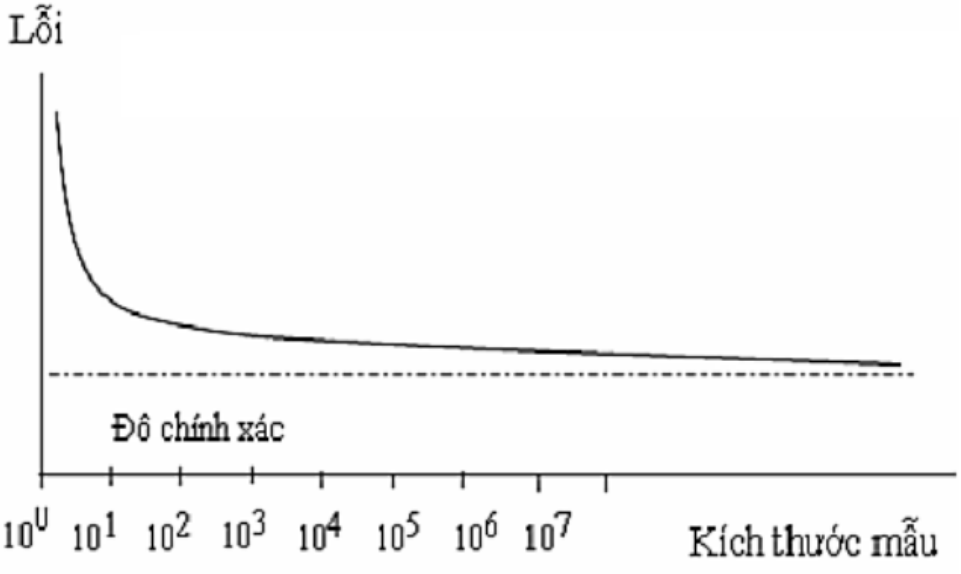
Hình xx. Transpose Convolution (b)

1.3 Các vấn đề của bài toán học máy và hướng giải quyết.

Xác định số lượng mẫu của tập học

Không có nguyên tắc nào hướng dẫn kích thước mẫu phải là bao nhiêu đối với một bài toán cho trước. Hai yếu tố quan trọng ảnh hưởng đến kích thước mẫu:

- Dạng hàm đích: khi mô hình càng phức tạp thì số lượng tham số càng cao, mô hình càng "đói" dữ liệu.
- Sự đa dạng của dữ liệu, nếu lượng dữ liệu bao phủ toàn bộ không gian của bài toán, thì mô hình của chúng ta đạt được độ chính xác cần tốt, điều này dường như bất khả thi. Một số phương pháp sinh thêm dữ liệu được sử dụng để mở rộng tập học.



Hình xx. Ảnh hưởng của kích thước mẫu.

Vấn đề quá khớp

Vấn đề quá khớp xảy ra khi mạng được luyện quá khớp (quá sát) với dữ liệu huấn luyện, nên nó sẽ trả lời chính xác những gì đã được học, còn những gì không được học thì nó không quan tâm. Như vậy, trong quá trình suy luân trong tương lai, mạng sẽ không có được khả năng tổng quát hóa. Vấn đề quá khớp xảy ra vì mạng có năng lực quá lớn (số lượng tham số quá lớn chẳng hạn). Một số cách để hạn chế bớt năng lực của mạng:

- Hạn chế số lượng layer.
- Giới hạn không gian tìm kiếm của tham số mô hình. Dùng Regularization: L2 regularization, L1 regularization, Elastic net regularization, …
- Giới hạn số bước huấn luyện. Dừng quá trình huấn luyện khi "cảm thấy" mô hình đã đủ tốt. Người ta thường lấy một phần nhỏ của tập học (validation dataset) để đánh giá độ tốt của mô hình trong quá trình học: mỗi lần duyệt qua toàn bộ tập học còn lại, sử dụng validation dataset để xem độ chính xác của mô hình hiện tại, từ đó ra quyết định học tiếp hoặc dừng. Người ta cùng dùng validation dataset để chọn siêu tham số của mô hình (kiến trúc của mô hình).

Sự triệt tiêu Gradient (Vanishing Gradient)

Vanishing Gradient là một khó khăn trong việc huấn luyện mạng thần kinh nhân tạo với các phương pháp dựa trên gradient. Vấn đề này làm cho việc học và điều chỉnh tham số của các lớp phía trước trong mạng trở nên khó khăn. Điều này càng trở nên tồi hơn khi mà số lượng các lớp trong kiến trúc tăng dần. Vần đề này sảy ra bởi một số hàm activation. Khi thực hiện lan truyền ngược để tính gradient của tham số, gradient có xu hướng nhỏ dần. Điều này có nghĩa các lớp dứng trước sẽ học chậm hơn so với các lớp phía sau đó. Khi mô hình càng nhiều lớp thì các lớp phía trước còn có thể không được cập nhật. Vấn đề này thường xuất hiện khi mô hình sử dụng các hàm Activation như là Sigmoid và Tanh nên đó cũng là lý do vì sao hai hàm này ít được sử dụng.

Tập dữ liệu mất cân bằng.

Vấn đề này xảy ra khi tập huấn luyện của chúng ta có chứa một lớp có số lượng mẫu quá ít so với một số lớp còn lại. Khi đó, trong quá trình học, mỗi batch của chúng ta có thể không

chứa, hoặc chứa một số ít mẫu là lớp đó, dẫn đến mô hình của chúng ta dường như quên sự tồn tại của lớp đó, quá trình suy luận của mô hình sẽ phần lớn bỏ qua lớp đó. Tùy từng tập dữ liệu mà chúng ta phải giải quyết vấn đề này theo cách khác nhau.

PHẦN 2 Tổng quan bài toán trích xuất đường đi từ ảnh vệ tinh và bài toán phân vùng của ảnh.

2.1 Giới thiệu bài toán trích xuất đường đi từ ảnh vệ tinh.

Công nghệ viễn thám (remote sensing), một trong những thành tựu khoa học vũ trụ đã đạt đến trình độ cao và đã trở thành kỹ thuật phổ biến được ứng dụng rộng rãi trong nhiều lĩnh vực kinh tế xa hội ở nhiều nước trên thế giới. Nhu cầu ứng dụng công nghệ viễn thám trong lĩnh vực điều tra nghiên cứu, khai thác sử dụng quản lý tài nguyên thiên nhiên và môi trường ngày càng gia tăng nhanh chóng không những trong phạm vi Quốc gia, mà cả phạm vi Quốc tế. Những kết quả thu được từ công nghệ viễn thám giúp các nhà khoa học và các nhà hoạch định chính sách các phương án lựa chọn có tính chiến lược về sử dụng và quản lý tài nguyên thiên nhiên và môi trường. Vì vậy viễn thám được sử dụng như là một công nghệ đi đầu rất có ưu thế hiện nay. Với mục đích nghiên cứu và ứng dung deep-learning, em đã chọn một ứng dụng của viên thám để áp dụng, đó là trích xuất đường từ ảnh vệ tinh.

Trích đường đi từ ảnh vệ tính đóng vai trò quan trọng trong quản lý giao thông, điều hướng xe cộ, cập nhật bản đồ, quy hoạch đô thị và cả trong lĩnh vực quân sự. Hình ảnh từ vệ tinh cung cấp thông tin quy mô lớn (độ phân giải cao), rất phù hợp để phân tích mạng lưới đường bộ một cách hiệu quả. Trong một vài thập kỷ qua, đã có những nghiên cứu, cố gắng lớn đã được thực hiện để trích xuất và cập nhật mạng lưới đường bộ.

Mạng lưới đường thường có một số hình thái hình học tiêu chuẩn (do quy hoạch đô thị), nhưng nói chung, nó không dễ dàng để trích xuất một cách chính xác từ hình ảnh vệ tinh. Lý do là các mạng lưới đường trong thực tế thường được bao phủ bởi nhiều loại vật thể trên mặt đất như xe cộ, cây cối và bóng râm. Do đó, hình dạng và màu sắc của đường thường đa dạng ở các khu khác nhau. Nhiều phương pháp trích xuất đường đã được đề xuất trong những năm gần đây. Phần lớn các phương pháp đều được chia thành 2 loại: Trích xuất bề mặt đường với từng điểm ảnh và trích xuất "khung xương" của đường. Do khung xương của đường có thể được sinh ra từ bề mặt đường bằng sử dụng thuật toán như morphological thinning nên trong báo cáo này tập trung vào trích xuất bề mặt đường từ ảnh vệ tinh có độ phân giải cao.

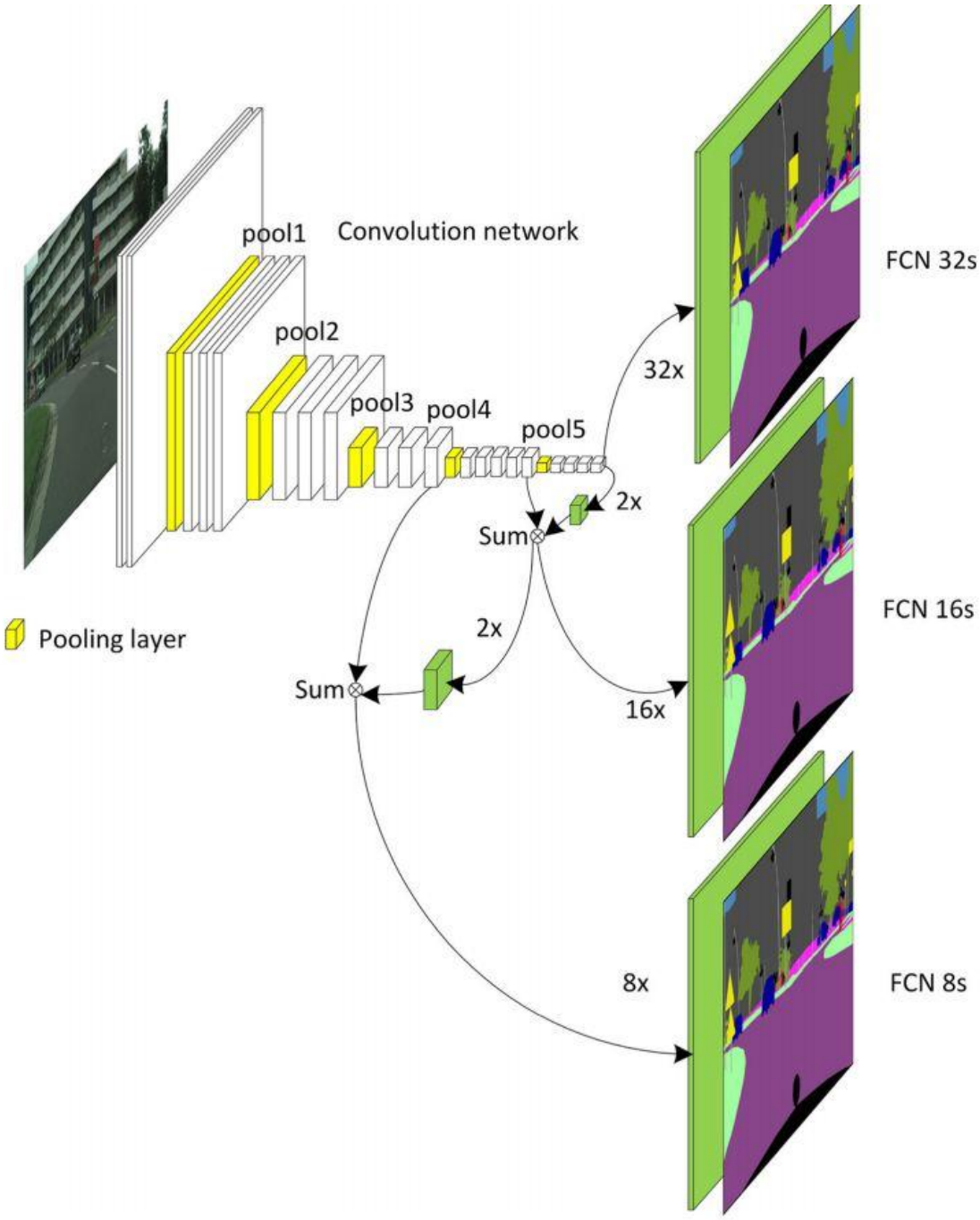
Trích xuất bề mặt đường có thể được xem như bài toán semantic segmentation hoặc phân loại nhị phân: gán nhãn cho mỗi điểm ảnh là đường đi hoặc là background (nền). Em sẽ giới thiệu tổng quan các mạng DCNN dùng cho bài toán semantic segmentation hiện nay.

2.2 Tổng quan một số mạng DCNN hiện đại cho bài Semantic Segmentation.

2.2.1 Fully convolutional network (FCN)

Long và cộng sự của ông, năm 2014, đã ra một bài báo lần đầu tiên giới thiệu Artificial Neural Network Fully Convolutional Network (ANNFCN) cho lĩnh vực phân chia hình ảnh. Thay đổi lớn nhất là thay thế các lớp kết nối đầy đủ (fully connected layer) bằng các lớp tích chập đầy đủ (fully convolutional layer). Với việc sử dụng upsample layer, kích thước đầu ra của mạng giống như kích thước đầu vào của mạng, đây là điều cần thiết trong bài toán Semantic Segmentation. Để tăng cường khả năng phân chia ảnh, các kết nối tắt được sử dụng (giữ thông tin môi trường tốt hơn). Quan trọng hơn, mạng có thể được học từ đầu tới cuối, lấy đầu vào kích thước bất kỳ và tạo ra đầu ra có kích thước tương ứng với suy luận và học tập hiệu quả. FCN được triển khai trên mô hình VGG-Net (được sử dụng từ trước để phân loại ảnh) và đạt được kết quả tốt nhất vào thời điểm đó trên tập dữ liệu

PASCAL VOC (giá trị trung bình IU tăng 20% so với kết quả tốt nhất năm 2012, đạt 62.2 %). Kiến trúc chính được thể hiện trong hình sau:



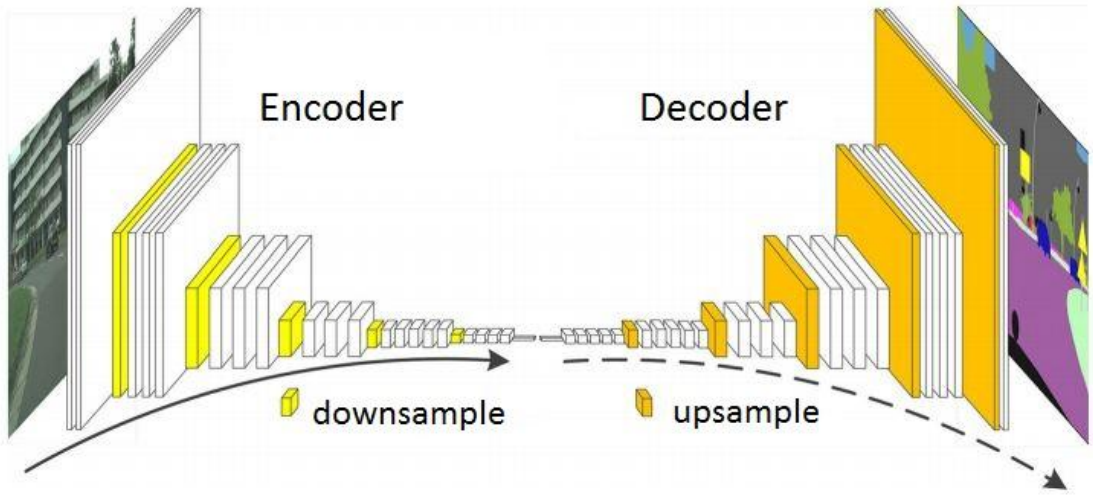
Hình xx. Kiến trúc mạng Fully convolutional network (FCN)

FCN sử dụng 5 lớp Downsample (Pooling) nên kích thước mỗi cạnh của feature maps giảm đi 32 lần so với ảnh input. FCN 32s: output của VGG-Net được nối với 1 lớp Upsample 32x, làm cho feature maps có kích thước bằng đúng kích thước ban đầu. FCN 16s: output của VGG-Net được nối với 1 lớp Upsample 2x, kích thước của feature maps bằng kích thước của feature maps trước khi thực hiện Pool5, sau đó được kết nối tắt với feature maps đó (thông qua phép cộng), tất cả được kết nối với 1 lớp Upsample 16x để feature maps có kích thước được giữ nguyên. FCN 8s: tương tự khi Output được thông qua

2 lớp Upsample 2x rồi kết nối tắt với features map trước khi thực thiện Pool4, tất cả được kết nối với 1 lớp Upsample 8x.

2.2.2 Kiến trúc encoder-decoder

Phát triển từ kiến trúc FCN bên trên, encoder là quá trình giảm kích thước của kích thước của feature maps (sử dụng các lớp downsample) , decoder là quá trình khôi phục kích thước feature maps về kích thước của ảnh vào ban đầu (sử dụng các lớp upsample).

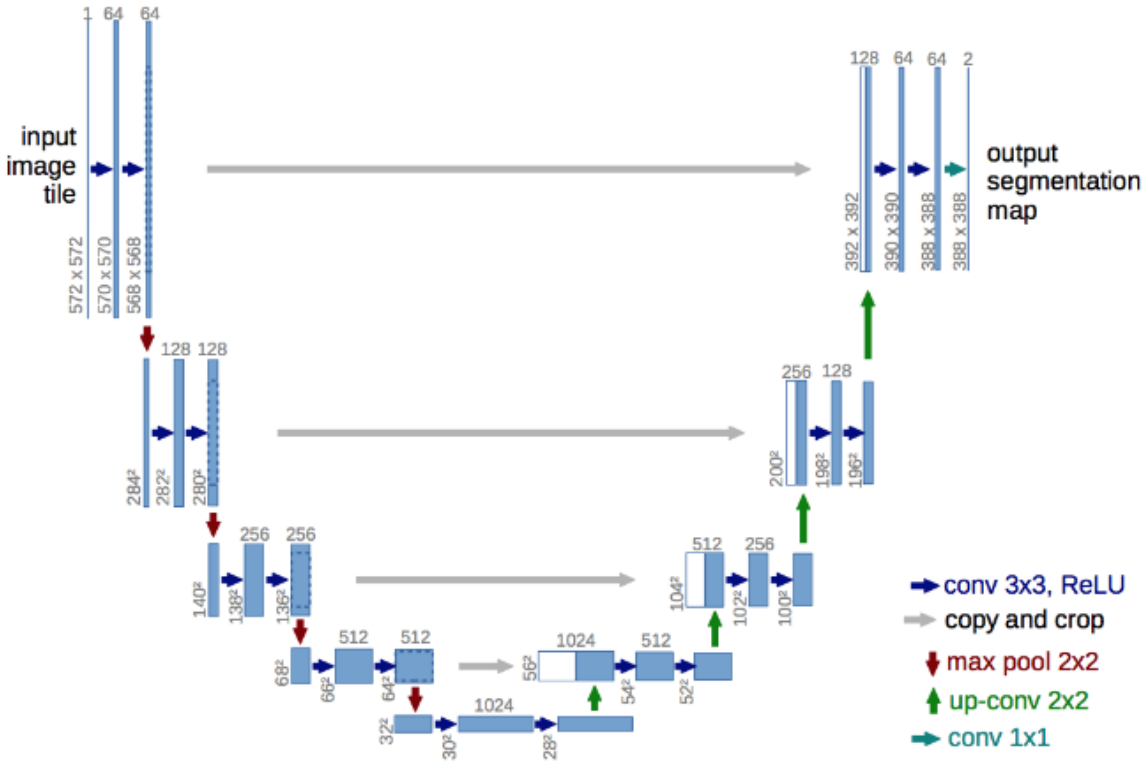


Hình xx. Kiến trúc Encoder-Decoder cho bài toán semantic segmentation

Hình trên Encoder sử dụng kiến trúc VGG-Net như FCN ở trên, Decoder sử dụng ngược lại VGG-Net với thay thế các lớp Downsample (Pooling) bằng các lớp Upsample (Transpose Convolutional). Một vài kiến trúc sau đây được phát triển dựa trên kiến trúc Encoder-Decoder.

2.2.2.1 Unet

Là một mạng FCN, đạt được kết quả tốt nhất cho bài toán phân chia ảnh y sinh.

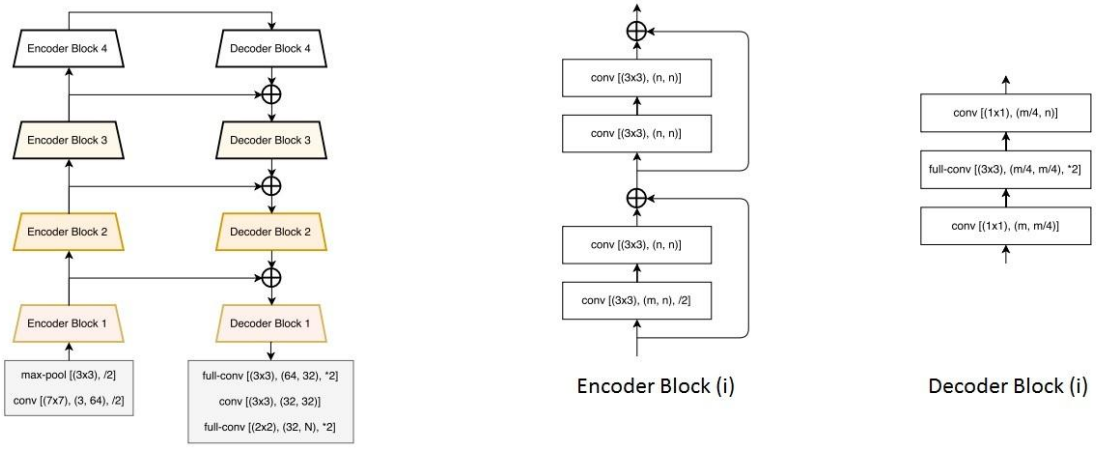


Hình xx. Kiến trúc mạng Unet

Hình trên là kiến trúc mạng Unet "gốc". Trong phần decoder, ngoài việc upsample ta còn thực hiện kết nối đối xứng với các layer phần encoder cho đến tận layer cuối cùng. Rõ ràng nếu như ta Upsample ngay từ layer cuối cùng ở phần encoder thì thông tin của bức ảnh ban đầu bị mất nhiều. Do đó việc ta kết nối đối xứng với phần encoder sẽ giúp ta phục hồi lại thông tin đã mất tại các lớp pooling. Thường khi cài đặt, người ta sẽ dùng ảnh kích thước là 2^i với i lớn hơn 7, các lớp Conv sẽ sử dụng padding để feature maps ra có cùng kích thước. Như vậy, các phép kết nối của chúng ta không cần phải crop để tránh mất mát thông tin.

2.2.2.2 LinkNet

Sử dụng phần encoder có chi phí tính toán và độ hiệu quả tốt hơn. Kiến trúc được mô tả theo hình dưới đây.



Hình xx. Kiến trúc của Linknet.

Trong kiến trúc trên, *conv[(kxk),(m, n), *2 hoặc /2]* nghĩa kernel size bằng k, số lượng input channel là m, số lượng output channel là n, */2* nghĩa là thực hiện phép downsample với hệ số 2 bằng cách dùng convolution có stride = 2 và **2* nghĩa là thực hiện phép upsample với hệ số 2 bằng cách sử dụng transpose convolution có stride = 2. Giữa 2 lớp convolution , tác giả sử dụng lớp batch normalization và theo sau đó là hàm ReLU. Phần encoder bắt đầu với convolution có kernel size bằng 7 và stride bằng 2, kế tiếp đó là lớp max pooling có kernel size bằng 3 và stride bằng 2, kế tiếp là các khối Encoder và Decoder như hình trên. Các feature maps giữa các khối Decoder cũng được kết nối với phần Encoder như mạng Unet để giữ lại thông tin môi trường, nhưng phép sử dụng là phép "cộng" thay cho phép "concatenate" ở Unet, điều này giúp cho mạng lan truyền tiến nhanh hơn. Tác giả cũng sử dụng các convolution có kernel size bằng 1 ở phần decoder để giảm chi phí tính toán.

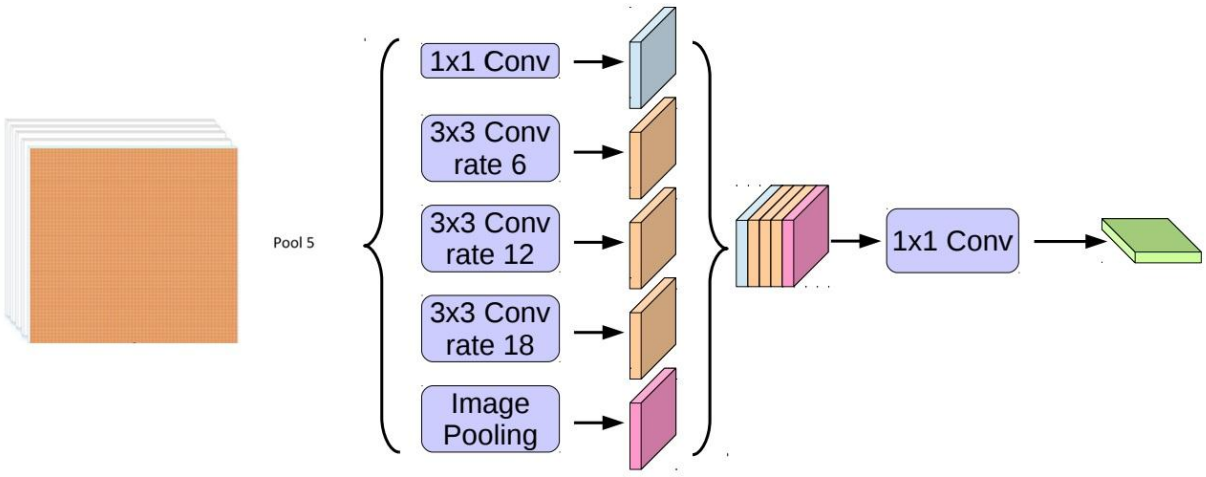
Sự phát triển của bài toán phân loại ảnh.

Hầu hết các mạng Semantic Segmentation đều sử dụng các mô hình của bài toán phân loại ảnh (image classification task) như phần chính của mô hình (encoder trong kiến trúc encoder-decoder) do các mạng classification có khả năng nhận diện tốt các góc cạnh của ảnh vào. Như mạng FCN đã sử dụng mạng VGG-16-Net làm xương sống, mạng Linknet được tác giả sử dụng ResNet-16 hoặc ResNet-34 làm phần encoder, DeepLab cũng đã sử dụng mạng ResNet (Deeplab v2, Deeplab v3) và đã đạt nhiều kết quả tốt nhất tại từng thời điểm. Vì thế các bài toán Semantic Segmentation được coi là phát triển song song với các bài toán Classification.

Một số mạng NN cho bài toán Classification tốt nhất hiện nay như: ResNeXt(thế hệ kế tiếp của ResNet, Inception-v2, Inception-v3, Inception-v4, Inception-ResNet, Xception, … được nhiều người sử dụng làm xương sống cho bài toán Semantic Segmentation và đều được kết quả tốt hơn.

2.2.3 Deeplab v3+

Để thu được thông tin môi trường ở nhiều góc độ nhìn khác nhau (multi-scale), Deeplab áp dụng các lớp tích chập mở rộng song song với tỉ lệ mở rộng khác nhau, gọi là khối ASPP (Atrous Spatial Pyramid Pooling):



Hình xx. Khối ASPP. Rate là hệ số mở rộng trong tích chập mở rộng. Trước khi thực hiện Convolution, các feature maps đều được padding hợp lý để giữ nguyên kích thước.

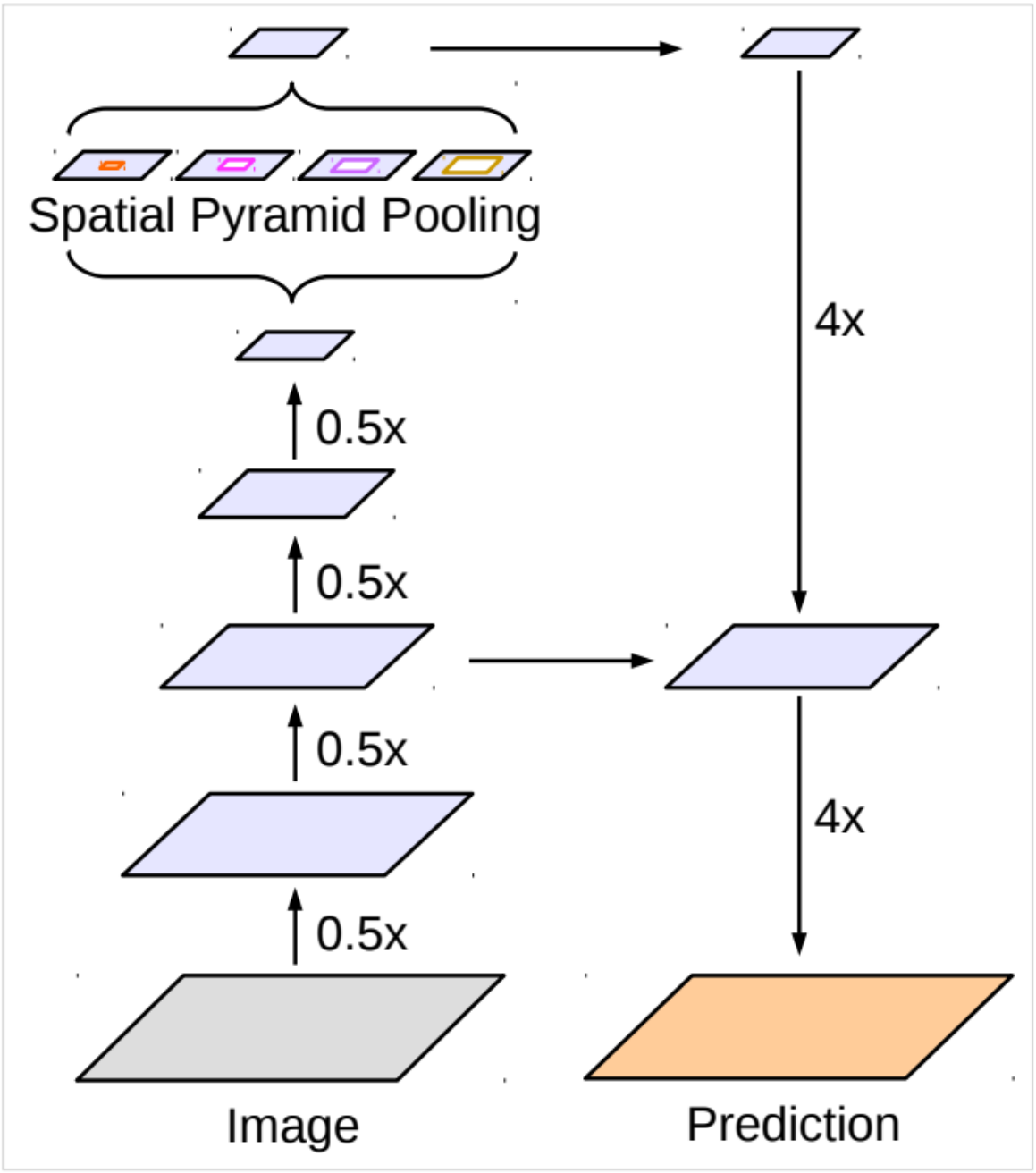
DeepLab cũng sử dụng Upsample là bi-linear interpolation để thay thế lớp Transpose Convolution như giới thiệu ở phần trước, mục đích là giảm chi phí tính toán và cũng làm tránh tình trạng triệt tiêu gradient ở các layer encoder. Phần hậu sử lý (post-processing) được sử dụng Fully connected Conditional Random Field (CRF) để giảm nhiễu.

Upsample sử dụng bi-linear interpolation được biểu diễn như sau:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 2 & 2 \\ 3 & 3 & 2 & 2 \end{bmatrix}$$

Hình xx. Sử dụng bilinear interpolation để upsample mỗi cạnh lên 2 lần (2x)

Kiến trục mạng Deeplab v3+ như sau:



Hình xx. Kiến trúc mạng DeepLabV3+.

Phần Encoder của mạng sử dụng mạng Xception (có chỉnh sửa) và theo sau là khối ASPP được giới thiệu ở trên. Phần Decoder sử dụng 2 lần Upsample 4x để khôi phục lại kích thước ảnh ban đầu. DeepLabV3 đã đạt 89.0% trên tập dữ liệu PASCAL VOC 2012.

PHẦN 3. Thực nghiệm và đánh giá.

3.1 Mô tả bộ dữ liệu.

Bộ dữ liệu em chọn để sử dụng là Massachusetts roads dataset của Mnih Volodymyr, là bộ dữ liệu đầu tiên được "public" cho bài toán trích xuất đường đi từ ảnh vệ tinh, dùng để so sánh các phương pháp. Mnih và Hinton cũng là người đầu tiên sử dụng mạng NN cho bài toán trích xuất đường đi này.

Hình xx. Ảnh vệ tinh (bên trái) và mặt nạ nhị phân (bên phải) để xác định các chỗ là đường

Đầu vào các ảnh vệ tinh RGB và đầu ra là các mặt nạ nhị phân có cùng kích thước, mỗi điểm thuộc mặt nạ nhị phân có giá trị là 1 nghĩa là đường và 0 nghĩa là nền (background). Dữ liệu có kích thước 1500 x 1500 điểm ảnh, độ phân giải là 1.2 mét trên một điểm ảnh và được tác giả chia làm 3 bộ: 1108 ảnh dùng để học, 14 ảnh dùng để xác định các siêu tham số (validation), 49 ảnh để kiểm tra. Bộ dữ liệu này bao phủ khoảng 500 ki-lô-mét vuông không gian, từ thành thị, tiểu đô thị tới các vùng nông thôn và các loại đối tượng trên mặt đất như đường đi, sông, biển, các công trình khác nhau, thảm thực vật, trường học, cầu, cảng, xe cộ, v.v … Để thực nghiệm, em sẽ đào tạo mạng lưới của mình trên tập học và kết quả được lấy từ tập thử nghiệm.

3.2 Cài đặt thực nghiệm.

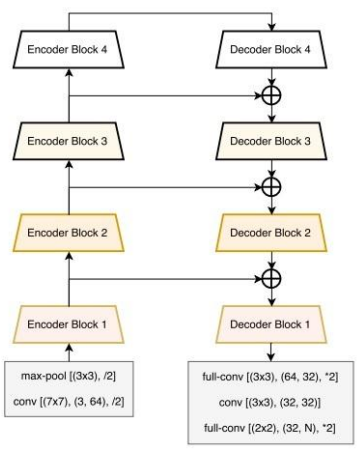
3.2.1 Các mô hình

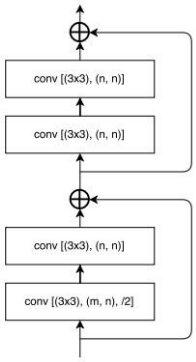
Mô hình cho bài toán Semantic Segmentation tốt nhất hiện nay là DeepLab v3 plus đã được giới thiệu trong phần 2. DeepLab v3 plus sử dụng rất nhiều layer (trên 100 layer) nên bộ nhớ tốn rất nhiều, và chi phí tính toán cho lan truyền rất lớn. Thêm vào đó, tính chất của con đường đi thường hẹp (mảnh, nhỏ), các con đường đi có thể bị tiêu biến sau các lớp pooling, khi upsample bằng bi-linear interpolation sẽ không thể phát hiện ra .Do vậy, em xin đề xuất một số mô hình "nhẹ" hơn, ứng dụng các module của các mạng tốt.

Các lớp Conv sử dụng trong các bảng dưới đây đều là 1 nhóm tuyến tính của 3 lớp [Convolution layer -> BatchNormalize layer -> ReLU].

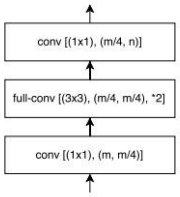
3.2.1.1 LinkNet + ASPP.

LinkNet có bộ decoder rất hiệu quả, đã đạt kết quả tốt hơn DeepLab v2 trên một số bộ dữ liệu. ASPP là một module của mạng DeepLab v3 plus, giúp tổng hợp thông tin về không gian dưới nhiều tỉ lệ (multi-scale) rất tốt. Kết hợp 2 mạng này ta có kiến trúc của mô hình như sau:





Encoder Block (i)



Decoder Block (i)

Chi tiết cài đặt của mạng LinkNet, phần Encoder sử dụng ResNet34, kiến trúc của ResNet34 như sau với đầu vào là ảnh 1024x1024, 2 lớp Conv trong cùng 1 ngoặc vuông (“[ ]”) nghĩa là sử dụng shortcut giữa input của lớp 1 vào output của lớp 2 :

Layer name	Output size	
Conv1	$64\times512\times512$	$\text{Conv}([7\times7],(3,64),/2)$
Conv2_x	$64\times256\times256$	$\text{Max-pool}([3\times3],/2)$
		$\begin{bmatrix}\text{Conv}([3\times3],(64,64))\\ \text{Conv}([3\times3],(64,64))\end{bmatrix}\times3$
Conv3_x	$128\times128\times128$	$\begin{bmatrix}\text{Conv}([3\times3],(64,128),\backslash2)\\ \text{Conv}([3\times3],(128,128))\end{bmatrix}$
		$\begin{bmatrix}\text{Conv}([3\times3],(128,128))\\ \text{Conv}([3\times3],(128,128))\end{bmatrix}\times3$
Conv4_x	$256\times64\times64$	$\begin{bmatrix}\text{Conv}([3\times3],(128,256),\backslash2)\\ \text{Conv}([3\times3],(256,256))\end{bmatrix}$
		$\begin{bmatrix}\text{Conv}([3\times3],(256,256))\\ \text{Conv}([3\times3],(256,256))\end{bmatrix}\times5$
Conv5_x	$512\times32\times32$	$\begin{bmatrix}\text{Conv}([3\times3],(256,512),\backslash2)\\ \text{Conv}([3\times3],(512,512))\end{bmatrix}$
		$\begin{bmatrix}\text{Conv}([3\times3],(512,512))\\ \text{Conv}([3\times3],(512,512))\end{bmatrix}\times2$

Bảng xx. Kiến trúc phần encoder của LinkNet

Kiến trúc phần Decoder của LinkNet như sau:

Layer name	Output Size	
DeConv5_x	$256\times64\times64$	$Conv([1\times1],(512,128))$
		$TransposeConv([3\times3],(128,128),*2)$
		$Conv([1\times1],(128,256))$
DeConv4_x	$128\times128\times128$	$Conv([1\times1],(256,64))$
		$TransposeConv([3\times3],(64,64),*2)$
		$Conv([1\times1],(64,128))$
DeConv3_x	$64\times256\times256$	$Conv([1\times1],(128,32))$
		$TransposeConv([3\times3],(32,32),*2)$
		$Conv([1\times1],(32,64))$
DeConv2_1	$64\times512\times512$	$Conv([1\times1],(64,16))$
		$TransposeConv([3\times3],(16,16),*2)$
		$Conv([1\times1],(16,64))$
DeConv1_1	$1\times1024\times1024$	$TransposeConv([3\times3],(64,32),*2)$
		$Conv([3\times3],(32,32))$
		$Conv([3\times3],(32,1))$
		Sigmoid()

Bảng xx. Kiến trúc phần Decoder của LinkNet.

Kiến trúc phần ASPP như sau:

<table>
<tr><td colspan="5">Input ($512\times32\times32$)</td></tr>
<tr><td>Global-Avg-pool ()
($512\times1\times1$)</td><td rowspan="3">Dconv([1x1],
(512,256),
rate=1)
($256\times32\times32$)</td><td rowspan="3">Dconv([3x3],
(512,256),
rate=6)
($256\times32\times32$)</td><td rowspan="3">Dconv([3x3],
(512,256),
rate=12)
($256\times32\times32$)</td><td rowspan="3">Dconv([3x3],
(512,256),
rate=18)
($256\times32\times32$)</td></tr>
<tr><td>Conv ($[1\times1],(512,256)$)
($256\times1\times1$)</td></tr>
<tr><td>Upsample(size
$=(32\times32)$)
($256\times32\times32$)</td></tr>
<tr><td colspan="5">Concatenate()
($1280\times32\times32$)</td></tr>
<tr><td colspan="5">Conv([1x1],(1280,512))</td></tr>
</table>

(512×32×32)

Như vậy, mạng đề xuất 1 của em, output của encoder được nối với module ASPP, output của ASPP được nối với Decoder, giữa các khối Decoder sử dụng shortcut với các phần tương ứng bên Encoder như LinkNet.

3.2.1.2 LinkNet + multi-scale context aggregation (MCA) module

multi-scale context aggregation: được tác giả giới thiệu như là một module để làm kích thước của receptive filed theo cấp số mũ mà không cần làm giảm kích thước của feature map, bằng cách xếp trồng các lớp tích chập mở rộng với rate khác nhau:

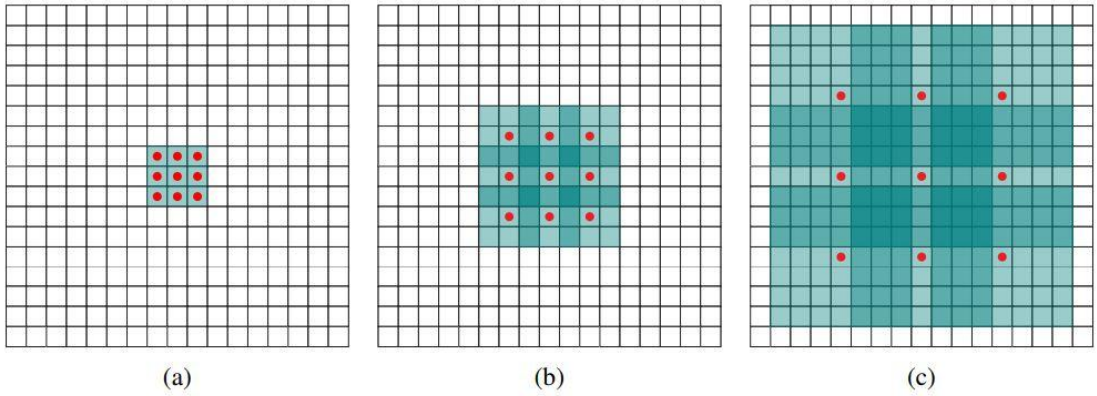


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

Như hình trên, nếu tỉ lệ mở rộng của các lớp tích chập mở rộng được xếp chồng lần lượt là 1, 2, 4, 8, 16, thì trường tiếp nhận của mỗi lớp sẽ là 3, 7, 15, 31, 63. Phần Encoder của LinkNet (sử dụng Resnet34) có 5 lớp làm giảm kích thước feature map (downsampling layers), nếu đầu vào là một ảnh kích thước 1024 x 1024 thì khi lan truyền qua phần encoder, đầu ra của feature map sẽ có kích thước 32 x 32. Trong trường hợp này, mạng đề xuất 2 của em sẽ sử dụng lớp các lớp tích chập mở rộng là 1, 2, 4, 8 ở MCA module, do vậy mỗi điểm của feature maps sau khi đi qua MCA module sẽ quan sát 31x31 điểm của feature maps trước đi qua MCA module,

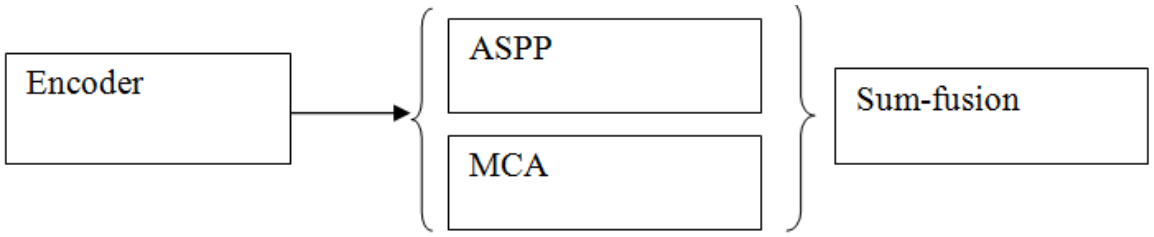
Kiến trúc của MCA module như sau:

Layer	Output size	
Dconv_1	(512×32×32)	Dconv([3,3],(512,512), rate=1)
Dconv_2	(512×32×32)	Dconv([3,3],(512,512), rate=2)
Dconv_3	(512×32×32)	Dconv([3,3],(512,512), rate=4)
Dconv_4	(512×32×32)	Dconv([3,3],(512,512), rate=8)

Như vậy, mạng đề xuất 2 của em, output của encoder được nối với module MCA, output của MCA được nối với Decoder, giữa các khối Decoder sử dụng shortcut với các phần tương ứng bên Encoder như LinkNet.

3.2.1.3 LinkNet + ASPP + MCA

Là mạng thứ 3 em đề xuất, feature maps sau khi qua Encoder sẽ được đi qua 2 module ASPP và MCA và được kết hợp lại bằng phép cộng:



3.3 Loss Function.

Loss function được sử dụng là Binary cross entropy (BCE) cho bài toán phân loại nhị phân. Do trong mỗi ảnh, số điểm ảnh là đường rất ít so với số điểm ảnh là background, nên em cũng sử dụng thêm Dice coeffient để khắc phục bài toán mất cân bằng trên tập dữ liệu. Loss function được tính như sau:

$$L = \sum_{i=1}^{N} \text{BCE-Loss}(\text{P}_i, \text{GT}_i) + 1 - \frac{\sum_{i=1}^{N} 2 \times |\text{P}_i \cap \text{GT}_i|}{\sum_{i=1}^{N} (|\text{P}_i| + |\text{GT}_i|)}$$

Binary cross entropy được tính bằng:

$$BCELoss(P, GT) = \sum_{i=1}^{W} \sum_{j=1}^{H} [gt_{ij} \times log(p_{ij}) + (1 - gt_{ij}) \times log(1 - p_{ij})$$

Với P là ảnh được dự đoán; GT là Ground truth; p là một điểm ảnh trong P; gt là một điểm dảnh trong GT.

3.4 Data Augmentation.

Dùng để tăng dữ liệu học tập, tránh bị overfit khi khả năng học của mô hình quá lớn còn dữ liệu lại ít. Ảnh ban đầu có kích thước 1500x1500, mỗi ảnh được Crop thành kích thước 1024x1024, ngẫu nhiên và 10 lần, ta thu được 11080 ảnh kích thước 1024x1024 cho tập học. Sau đó mỗi ảnh sẽ được biến đổi tiếp bằng các phép biến đổi sau.

Biến đổi màu sắc(HSV).

Không gian màu HSV là một không gian màu dựa trên 3 số liệu:

- H: (Hue) vùng màu
- S: (Saturation) Độ bão hòa màu
- V: (Value) Độ sáng

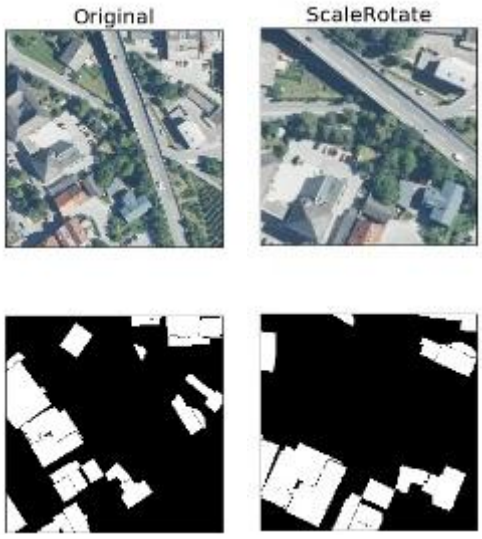
Phép biến đổi theo các bước: biến đổi không gian màu RGB thành HSV, tách mỗi giá trị H, S, V ra rồi cộng trừ với một lượng nhỏ (Hue, sau đó lại biến đổi từ không gian HSV thành RGB:

```
def randomHueSaturationValue(image, hue_shift_limit=(-30, 30),
                             sat_shift_limit=(-15, 15),
                             val_shift_limit=(-15, 15), u=0.5):
    if np.random.random() < u:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        h, s, v = cv2.split(image)
        hue_shift = np.random.randint(hue_shift_limit[0], hue_shift_limit[1]+1)
        hue_shift = np.uint8(hue_shift)
        h += hue_shift
        sat_shift = np.random.uniform(sat_shift_limit[0], sat_shift_limit[1])
        s = cv2.add(s, sat_shift)
        val_shift = np.random.uniform(val_shift_limit[0], val_shift_limit[1])
        v = cv2.add(v, val_shift)
        image = cv2.merge((h, s, v))
        image = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)

    return image
```

Biến đổi không gian (Scale-Rotate)

Scale là phép phóng to hoặc thu nhỏ ảnh, Rotate là phép xoay ảnh từ điểm trung tâm. Cả 2 phép đều áp dụng cho cả ảnh vệ tinh và mask.

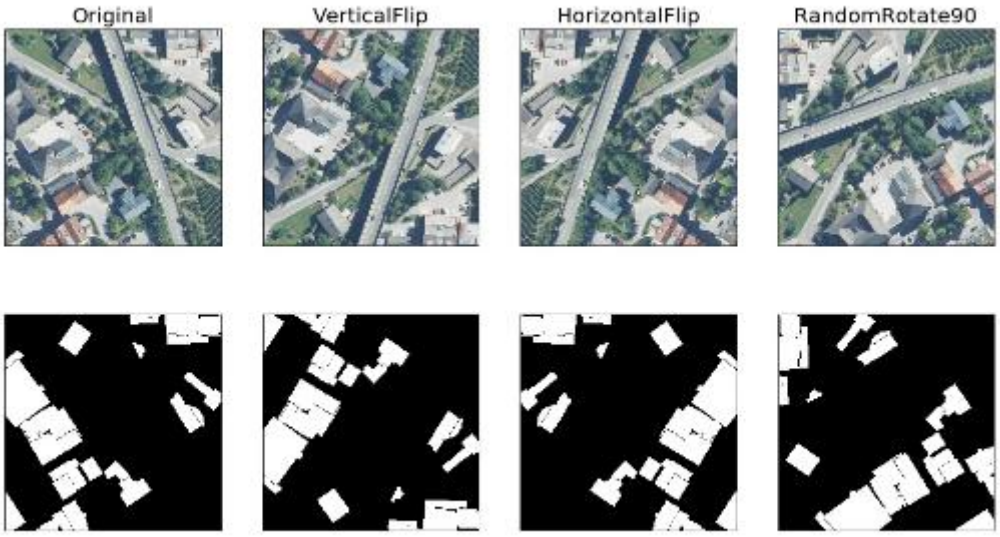
```
def rotate(img, angle, interpolation=cv2.INTER_LINEAR, border_mode=cv2.BORDER_REFLECT_101):
    height, width = img.shape[:2]
    matrix = cv2.getRotationMatrix2D((width / 2, height / 2), angle, 1.0)
    img = cv2.warpAffine(img, matrix, (width, height), flags=interpolation, borderMode=border_mode)
    return img

def scale(img, scale, interpolation=cv2.INTER_LINEAR):
    height, width = img.shape[:2]
    new_height, new_width = int(height * scale), int(width * scale)
    img = cv2.resize(img, (new_width, new_height), interpolation=interpolation)
    return img
```



Hình xx. Phép Scale và Rotate cho bài toán Semantic Segmentation

Các phép lật ảnh.

Sử dụng các phép lật ảnh như lật đối xứng trục ngang, trục dọc, trục chéo. Như vậy một ảnh có thể thu được 8 ảnh khác nhau theo phép lật.



Hình xx. Các phép lật cho bài toán Semantic Segmentation

3.5 Chạy thực nghiệm

Chương trình được cài đặt bằng ngôn ngữ Python kết hợp với các thư viện PyTorch, OpenCV, pydensecrf… OpenCV là một thư viện mã nguồn mở cho thị giác máy tính (Computer Vision), xử lý ảnh và học máy, và có tính năng hỗ trợ GPU giúp tăng tốc độ xử lý. Pytorch là một framework, cung cấp nền tảng tính toán khoa học phục vụ lĩnh vực Deep Learning, được Facebook nghiên cứu và sử dụng, hiện nay đã ra phiên bản 1.0 hỗ trợ cho cả nghiên cứu và triển khai sản phẩm. Pydensecrf là một thư viện cài đặt CRF, dùng để cho bước hậu xử lý (post-processing).

Chương trình được chạy trên Google Colab với cấu hình sau:

STT	Phần cứng	Loại
1	CPU	Intel(R) Xeon(R) CPU @ 2.20GHZ
2	Ram	12 GB
2	GPU	1 Tesla K80

Độ đo sự chính xác, intersection over union:

$$\text{IoU}=\frac{1}{n}\sum_{i=1}^{n}\frac{\text{P}_i\cap\text{GT}_i}{\text{P}_i\cup\text{GT}_i}$$

Phần Encoder của các mạng đề xuất đều sử dụng pre-trained model ResNet34 để tăng khả năng phân loại. Em sử dụng Adam optimizer để tối ưu hàm mất mát. Learning rate ban đầu được thiết lập là 2e-4 và được giảm 5 lần khi quan sát hàm mất mát không giảm qua 10 epochs. Batch size trong giai đoạn học được cố định là 4. Thường mất khoảng 150 epochs để các mạng hội tụ.

Giai đoạn đánh giá, một bức ảnh kích thước 1500x1500 được chia thành 4 ảnh kích thước 1024x1024, các phần trùng nhau giữa 2 ảnh khi kết hợp sẽ lấy giá trị lớn nhất; mỗi ảnh trong 4 ảnh đấy, được sử dụng các phép lật, sẽ được dự đoán 8 lần, sau đó được lấy trung bình. Sử dụng ngưỡng dự đoàn bằng 0.5 để phân loại nhị phân.

3.6 Kết quả.

Em so sánh với 2 mạng: Unet với 7 lớp pooling (để có thể bao phủ ảnh kích thước 1024x1024 như 3 mạng đề xuất của em) và mạng LinkNet với Encoder là pre-trained model ResNet34 . Kết quả của các mô hình khác nhau được thể hiện trong bảng dưới đây

Model	IoU trên tập thử nghiệm
Unet (7 lớp pooling)	
LinkNet	
LinkNet +ASPP	
LinkNet + MCA	
LinkNet + ASPP + MCA	
LinkNet + MCA + CRF	

Bảng xx. Kết quả một số mô hình khác nhau trên tập thử nghiệm

3.7 Đánh giá

Em thấy rằng LinkNet với encoder là ResNet34 chỉ tốt hơn một chút so với Unet được huấn luyện từ ban đầu. Em nghĩ rằng hai mô hình này có thể đạt được độ chính xác gần như nhau trong các cách khác nhau. Unet cơ bản có trường tiếp nhận lớn nhưng lại không sử dụng transfer learning và feature map ở đầu ra encoder chỉ có kích thước 8 x8, quá nhỏ để giữ thông tin chi tiết về không gian. LinkNet với Resnet34 là encoder có kết quả tốt hơn, nhưng nó chỉ có 5 lớp downsampling, hầu như không bao phủ hết được hình kích thước 1024 x 1024. Trong khi xem xét các kết quả từ hai mô hình này, chúng tôi nhận thấy rằng mặc dù LinkNet tốt hơn Unet trong khi đánh giá một điểm là đường đi hay không, nhưng nó có vấn đề về tính chất kết nối ở đường đi. Vì vậy, bằng cách thêm các lớp tích chập mở rộng như ASPP hoặc MCA sau phần Encoder, mạng đề xuất có thể thu được trường tiếp nhận lớn hơn LinkNet cũng như bảo toàn thông tin về không gian chi tiết hơn, và do đó làm giảm bớt vấn đề kết nối đường xảy ra trong LinkNet.

3.8 Một số kết quả khác với ảnh thực tế ở Việt Nam và hướng phát triển.

Với mục đích ứng dụng vào bài toán thực tế, em sử dụng trang website https://www.planet.com/ để lấy một số ảnh biên giới của nước ra để chạy thử nghiệm(do ở các vùng thành thị thì google maps đã trích xuất chính xác hầu hết các đường). Một số kết quả sau:

Kết luận

Trong quá trình học hỏi và tìm hiểu về Deeplearning nói chung và các mạng Semantic Segmentation nói riêng em đã tích lũy, trau dồi thêm được nhiều kiến thức bổ ích. Qua đây, em cũng thấy được ý nghĩa to lớn và những ứng dụng quan trọng trong lĩnh vực thị giác máy

Về mặt lí thuyết, em đã tìm hiểu và trình bày được trong đồ án:

- Các khái niệm cơ bản về học máy cũng như nền tảng mô hình mạng nơ-ron và các vấn đề khi huấn luyện một mạng nơ-ron sâu.
- Kiến trúc mạng Semantic Segmentation, các khái niệm liên quan và các nguyên lí hoạt động. Ngoài ra đồ án còn trình bày sơ lược về các kiến trúc hiện đại của mạng Semantic Segmentation cũng như các kĩ thuật tăng độ chính xác của mô hình trong bài toán phân lại ảnh.

Về mặt thực nghiệm, em đã cài đặt một số mạng đề xuất và đạt được kết quả khả quan trên bộ dữ liệu "*Massachusetts roads dataset*". Tuy nhiên, do hạn chế về thời gian và tài nguyên nên em chưa thể thử các mô hình tốt nhất hiện tại để có thế so sánh tốt hơn.

Hướng phát triển:

- Kết hợp nhiều bộ dữ liệu về trích xuất đường để mô hình tổng quát hơn
- Nghiên cứu và phát triển các mạng Semantic Segmentation để đạt độ chính xác tốt hơn.
- Xử dụng các thông tin thêm về đường, ảnh vệ tinh như độ rộng của con đường, độ phân giải của ảnh vệ tinh (bao nhiêu mét trên một điểm ảnh) để lọc nhiễu.
- Phát triển thành hệ thống tự động cập nhật và phát hiện thay đổi đường.

Tài liệu tham khảo

….

https://dominhhai.github.io/vi/2018/04/nn-intro/

https://cs231n.github.io

https://www.teco.edu/~albrecht/neuro/html/node7.html

https://viblo.asia/p/thuat-toan-toi-uu-adam-aWj53k8Q56m

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807