Bài 8 - Convolutional Neural Network

22 Aug 2019 - phamdinhkhanh

Menu

1. Lý thuyết về mạng tích chập

1.1. Giới thiệu tích chập

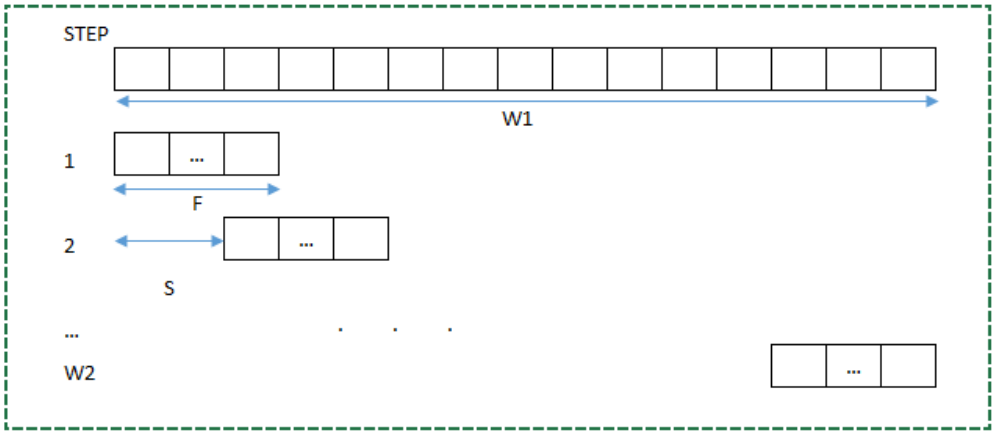
Tích chập là một khái niệm trong xử lý tín hiệu số nhằm biến đổi thông tin đầu vào thông qua một phép tích chập với bộ lọc để trả về đầu ra là một tín hiệu mới. Tín hiệu này sẽ làm giảm những đặc trưng mà bộ lọc không quan tâm và chỉ giữ những đặc trưng chính.

Tích chập thông dụng nhất là tích chập 2 chiều được áp dụng trên ma trận đầu vào và ma trận bộ lọc 2 chiều. Phép tích chập của một ma trận $\mathbf{X} \in \mathbb{R}^{W_1 \times H_1}$ với một *bộ lọc* (receptive field) $\mathbf{F} \in \mathbb{R}^{F \times F}$ là một ma trận $\mathbf{Y} \in \mathbb{R}^{W_2 \times H_2}$ sẽ trả qua những bước sau:

- Tính tích chập tại 1 điểm: Tại vị trí đầu tiên trên cùng của ma trận đầu vào ta sẽ lọc ra một ma trận con $\mathbf{X}_{sub} \in \mathbb{R}^{F \times F}$ có kích thước bằng với kích thước của bộ lọc. Giá trị y_{11} tương ứng trên $\mathbf{Y}$ là tích chập của $\mathbf{X}_{sub}$ với $\mathbf{F}$ được tính như sau: $y_{11} = \sum_{i=1}^{F}\sum_{j=1}^{F} x_{ij}f_{ij}$
- Tiến hành trượt dọc ma trận theo chiều từ trái qua phải, từ trên xuống dưới với *bước nhảy* (stride) S ta sẽ tính được các giá trị y_{ij} tiếp theo. Sau khi quá trình này kết thúc ta thu được trọn vẹn ma trận đầu ra $\mathbf{Y}$.

Trong một mạng nơ ron tích chập, các tầng (*layer*) liền sau lấy đầu vào từ tầng liền trước nó. Do đó để hạn chế lỗi trong thiết kế mạng nơ ron chúng ta cần xác định kích thước đầu ra ở mỗi tầng. Điều đó có nghĩa là dựa vào kích thước ma trận đầu vào (W_1, H_1), kích thước bộ lọc (F, F) và bước nhảy S để xác định kích thước ma trận đầu ra (W_2, H_2).

Xét quá trình trượt trên chiều W_1 của ma trận đầu vào.



Hình 1: Quá trình trượt theo chiều rộng W_1. Mỗi dòng tương ứng với một bước. Mỗi bước chúng ta dịch sang phải một khoảng S đơn vị cho tới khi đi hết W_1 ô. Nếu bước cuối cùng bị dư thì chúng ta sẽ lát (*padding*) thêm để mở rộng ma trận sao cho quá trình tích chập không bị dư ô.

Giả sử quá trình này sẽ dừng sau W_2 bước. Tại bước đầu tiên ta đi được đến vị trí thứ F. Sau mỗi bước liền sau sẽ tăng so với vị trí liền trước là S. Như vậy đến bước thứ i quá trình trượt sẽ đi đến vị trí $F + (i-1)S$. Suy ra tại bước cuối cùng W_2 ma trận sẽ đi đến vị trí $F + (W_2 - 1)S$. Đây là vị trí lớn nhất gần với vị trí cuối cùng là W_1. Trong trường hợp lý tưởng thì $F + (W_2 - 1)S = W_1$. Từ đó ta suy ra:

$$W_2 = \frac{W_1 - F}{S} + 1 \quad (1)$$

Top

Khi vị trí cuối cùng không trùng với W_1 thì số bước W_2 sẽ được lấy phần nguyên:

$$W_2 = \backslash\text{floor}\frac{W_1 - F}{S} + 1$$

Chúng ta luôn có thể tạo ra đẳng thức (1) nhờ thêm phần *đường viền* (padding) tại các cạnh của ảnh với độ rộng viền là P sao cho phép chia cho S là chia hết. Khi đó:

$$W_2 = \frac{W_1 + 2P - F}{S} + 1$$

Hình 2: Thêm padding kích thước P vào 2 lề chiều rộng (W_1)

Hoàn toàn tương tự ta cũng có công thức ứng với chiều cao:

$$H_2 = \frac{H_1 + 2P - F}{S} + 1$$

1.2. Thực hành mạng tích chập hai chiều

Trong ví dụ bên dưới ta sẽ thực hành sử dụng mạng tích chập hai chiều để trích xuất các đặc trưng chính của một bức ảnh. Do tích chập hai chiều là tích chập phổ biến nhất nên để ngắn gọn, kể từ đây ta sẽ coi thuật ngữ tích chập là tích chập hai chiều. Chúng ta sẽ tìm ra output của tích chập thông qua hai bộ lọc thông dụng nhất là bộ lọc ngang

$$-1 \quad -1 \quad -1 \; 0 \quad 0 \quad 0 \; 1 \quad 1 \quad 1$$

được sử dụng để trích xuất các đường nằm ngang và bộ lọc dọc

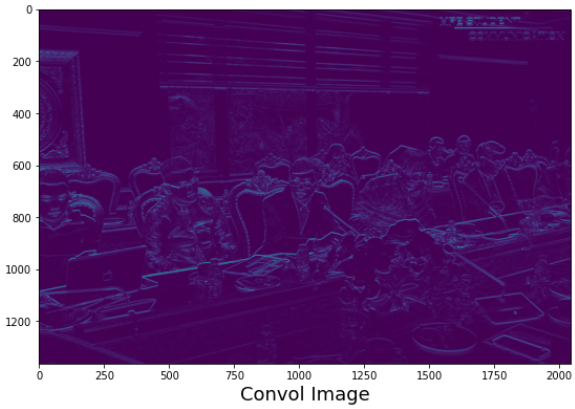
$$-1 \quad 0 \quad 1 \; -1 \quad 0 \quad 1 \; -1 \quad 0 \quad 1$$

dùng để trích xuất các đường nét nằm dọc từ 1 bức ảnh.

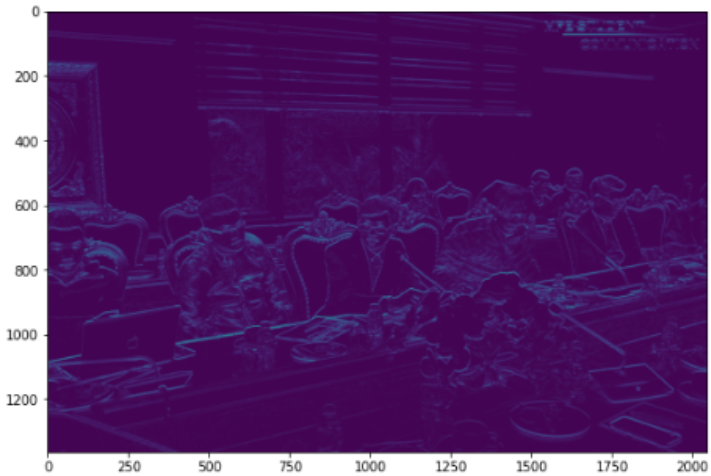
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from PIL import Image
import urllib.request
from io import BytesIO

%matplotlib inline

url = str('https://imgur.com/nmGz7Lv.png')
with urllib.request.urlopen(url) as url:
    f = BytesIO(url.read())

X = np.array(Image.open(f))
print('Image shape: %s'%str(X.shape))
# Convert to grey
X = X.dot([0.299, 0.5870, 0.114])
print('Image shape: %s'%str(X.shape))
plt.imshow(X)
```

Top

Convolution with Horizontal Filter



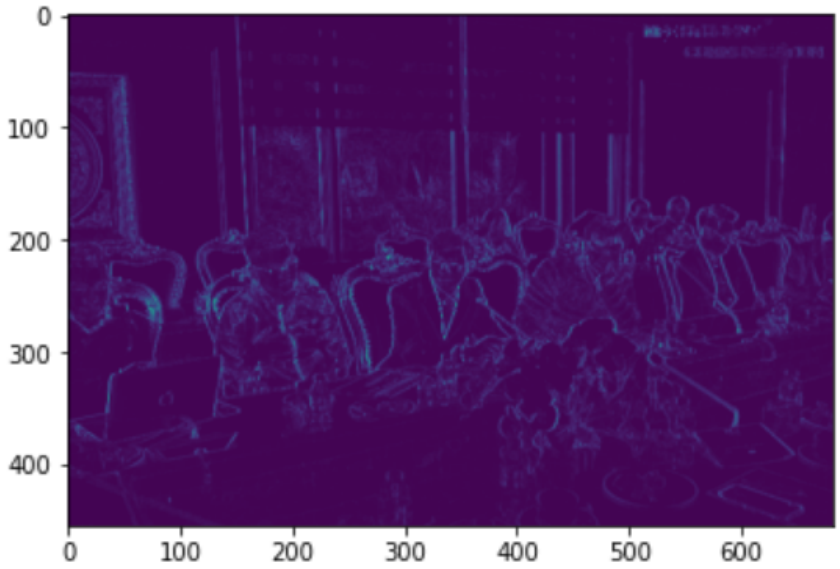

```
#Tạo bộ lọc ngang F1
F1 = np.array([[-1, -1, -1],
               [0, 0, 0],
               [1, 1, 1]])
#Tính tích chập 2 chiều.
def conv2d(X, F, s = 1, p = 0):
    """
    X: Ma trận đầu vào
    F: Ma trận bộ lọc
    s: Bước trượt
    p: Độ rộng lề thêm vào
    """
    (w1, h1) = X.shape
    f = F.shape[0]
    w2 = int((w1 + 2*p - f)/s) + 1
    h2 = int((h1 + 2*p - f)/s) + 1
    Y = np.zeros((w2, h2))
    X_pad = np.pad(X, pad_width = p, mode = 'constant', constant_values = 0)
    for i in range(w2):
        for j in range(h2):
            idw = i*s
            idh = j*s
            Y[i, j] = np.abs(np.sum(X_pad[idw:(idw+f), idh:(idh+f)]*F))
    return Y

Y1 = conv2d(X, F1, s = 1, p = 0)
plt.imshow(Y1)
```



Ta nhận thấy bộ lọc trên có tác dụng nhận diện những đường nét theo chiều ngang của bức ảnh như các đường viền của cửa sổ, cạnh bàn, vai áo,…. Sở dĩ bộ lọc này làm nổi bật các đường nét nằm ngang là bởi vì tích chập của chúng bằng hiệu của tổng giá trị các điểm phía dưới trừ các điểm phía trên. Đối với các đường nét nằm ngang thì cường độ sáng ít thay đổi theo phương ngang nhưng xét theo chiều dọc thì chúng sẽ thay đổi. Do đó hiệu giữa 2 tổng phía trên và dưới càng lớn dẫn tới giá trị của tích chập càng lớn khi trượt theo các đường nét nằm ngang này. Khi hoàn thành thiện ma trận tích chập các đường nét nằm ngang sẽ có cường độ sáng lớn hơn nên nổi bật hơn. Chúng ta sẽ thử nghiệm một bộ lọc khác để nhận diện chiều dọc của bức ảnh.

Top

```
#Tạo bộ lọc dọc F2
F2 = np.array([[1, 0, -1],
               [1, 0, -1],
               [1, 0, -1]])
Y2 = conv2d(X, F2, s = 3, p = 0)
plt.imshow(Y2)
```



Bộ lọc cho thấy các đường nét dọc theo bức ảnh như dáng người ngồi thẳng đã được nhận diện rõ ràng, các đướng nét ngang như cửa sổ, vai áo, cạnh bàn,… đã biến mất. Như vậy chúng ta có thể thấy mỗi bộ lọc sẽ có 1 tác dụng trích xuất một đặc trưng khác nhau từ cùng 1 bức ảnh.

1.3. Mạng nơ ron tích chập (mạng CNN)

1.3.1. Các Thuật ngữ

Do bài này khá nhiều thuật ngữ chuyên biệt trong mạng CNN nên để dễ hiểu hơn cho bạn đọc tôi sẽ diễn giải trước khái niệm.

- **Đơn vị (Unit)**: Là giá trị của một điểm nằm trên ma trận khối ở mỗi tầng của mạng CNN.
- **Vùng nhận thức (Receptive Field)**: Là một vùng ảnh trên khối ma trận đầu vào mà bộ lọc sẽ nhân tích chập để ánh xạ tới một đơn vị trên layer tiếp theo.
- **Vùng địa phương (Local region)**: Theo một nghĩa nào đó sẽ bao hàm cả vùng nhận thức. Là một vùng ảnh cụ thể nằm trên khối ma trận ở các tầng (*layer*) của mạng CNN.
- **Bản đồ đặc trưng (Feature Map)**: Là ma trận đầu ra khi áp dụng phép tích chập giữa bộ lọc với các vùng nhận thức theo phương di chuyển từ trái qua phải và từ trên xuống dưới.
- **Bản đồ kích hoạt (Activation Map)**: Là output của *bản đồ đặc trưng* CNN khi áp dụng thêm hàm activation để tạo tính phi tuyến.

1.3.2. Kiến trúc chung của mạng neural tích chập

Tích chập được ứng dụng phổ biến trong lĩnh vực thị giác máy tính. Thông qua các phép tích chập, các đặc trưng chính từ ảnh được trích xuất và truyền vào các tầng *tích chập* (layer convolution). Mỗi một tầng tích chập sẽ bao gồm nhiều đơn vị mà kết quả ở mỗi đơn vị là một phép biến đổi tích chập từ layer trước đó thông qua phép nhân tích chập với bộ lọc.

Về cơ bản thiết kế của một mạng nơ ron tích chập 2 chiều có dạng như sau:

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

Trong đó:

- INPUT: Tầng đầu vào
- CONV: Tầng tích chập
- RELU: Tầng kích hoạt. Thông qua hàm kích hoạt (*activation function*), thường là ReLU hoặc LeakyReLU để kích hoạt phi tuyến

Top

- POOL: Tầng tổng hợp, thông thường là Max pooling hoặc có thể là Average pooling dùng để giảm chiều của ma trận đầu vào.
- FC: Tầng kết nối hoàn toàn. Thông thường tầng này nằm ở sau cùng và kết nối với các đơn vị đại diện cho nhóm phân loại.

Các kí hiệu []*N, []*M hoặc []*K ám chỉ các khối bên trong [] có thể lặp lại nhiều lần liên tiếp nhau. M, K là số lần lặp lại. Kí hiệu -> đại diện cho các tầng liền kề nhau mà tầng đứng trước sẽ làm đầu vào cho tầng đứng sau. Dấu ? sau POOL để thể hiện tầng POOL có thể có hoặc không sau các khối tích chập.

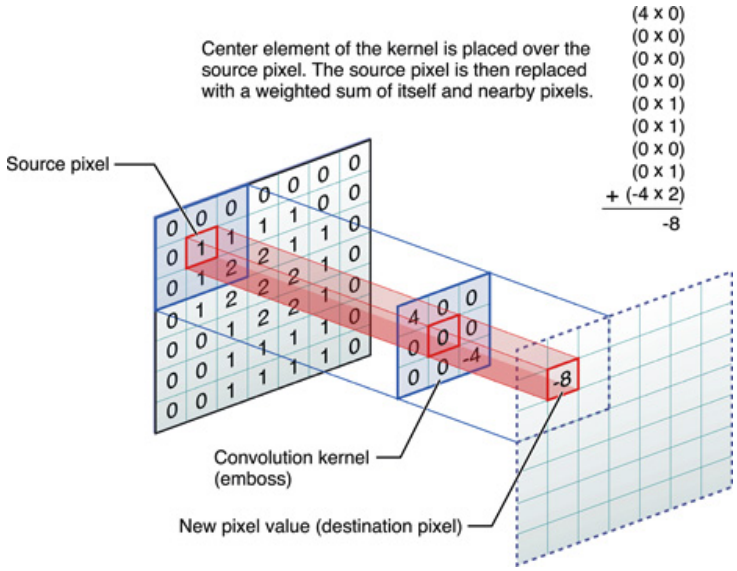
Như vậy ta có thể thấy một mạng nơ ron tích chập về cơ bản có 3 quá trình khác nhau:

- Quá trình tích chập (convolution): Thông qua các tích chập giữa ma trận đầu vào với bộ lọc để tạo thành các đơn vị trong một tầng mới. Quá trình này có thể diễn ra liên tục ở phần đầu của mạng và thường sử dụng kèm với hàm kích hoạt ReLU. Mục tiêu của tầng này là trích suất đặc trưng hai chiều.
- Quá trình tổng hợp (max pooling): Các tầng càng về sau khi trích xuất đặc trưng sẽ cần số lượng tham số lớn do chiều sâu được qui định bởi số lượng các kênh ở các tầng sau thường tăng tiến theo cấp số nhân. Điều đó làm tăng số lượng tham số và khối lượng tính toán trong mạng nơ ron. Do đó để giảm tải tính toán chúng ta sẽ cần giảm kích thước các chiều của khối ma trận đầu vào hoặc giảm số đơn vị của tầng. Vì mỗi một đơn vị sẽ là kết quả đại diện của việc áp dụng 1 bộ lọc để tìm ra một đặc trưng cụ thể nên việc giảm số đơn vị sẽ không khả thi. Giảm kích thước khối ma trận đầu vào thông qua việc tìm ra 1 giá trị đại diện cho mỗi một vùng không gian mà bộ lọc đi qua sẽ không làm thay đổi các đường nét chính của bức ảnh nhưng lại giảm được kích thước của ảnh. Do đó quá trình giảm chiều ma trận được áp dụng. Quá trình này gọi là tổng hợp nhằm mục đích giảm kích thước dài, rộng.
- Quá trình kết nối hoàn toàn (fully connected): Sau khi đã giảm kích thước đến một mức độ hợp lý, ma trận cần được trải phẳng (flatten) thành một vector và sử dụng các kết nối hoàn toàn giữa các tầng. Quá trình này sẽ diễn ra cuối mạng CNN và sử dụng hàm kích hoạt là ReLU. Tầng kết nối hoàn toàn cuối cùng (fully connected layer) sẽ có số lượng đơn vị bằng với số classes và áp dụng hàm kích hoạt là softmax nhằm mục đích tính phân phối xác xuất.

Hình 3: Cấu trúc đại diện của một mạng nơ ron tích chập, source: Mathworks.com (Source: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html)

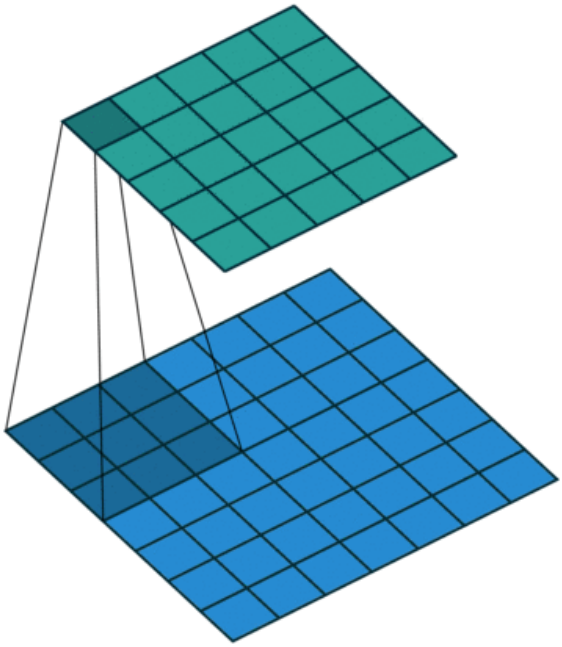
1.4. Tính chất của mạng nơ ron tích chập

Tính kết nối trượt: Khác với các mạng nơ ron thông thường, mạng nơ ron tích chập không kết nối tới toàn bộ hình ảnh mà chỉ kết nối tới từng *vùng địa phương* (local region) hoặc *vùng nhận thức* (receptive field) có kích thước bằng kích thước bộ lọc của hình ảnh đó. Các bộ lọc sẽ trượt theo chiều của ảnh từ trái qua phải và từ trên xuống dưới đồng thời tính toán các giá trị tích chập và điền vào *bản đồ kích hoạt* (activation map) hoặc *bản đồ đặc trưng* (feature map).



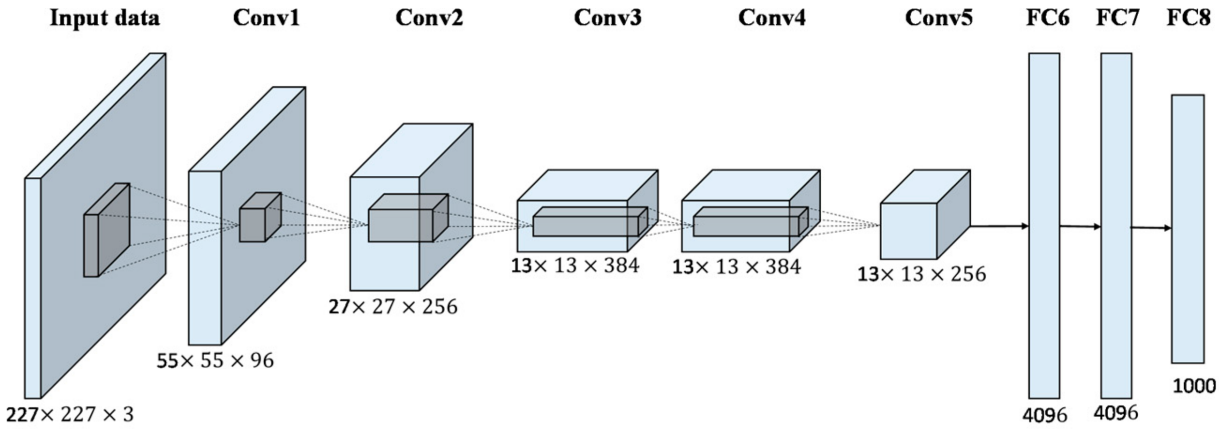
Hình 4: Tính tích chập trên bản đồ kích hoạt, Source: developer.apple.com (https://developer.apple.com/library/archive/documentation/Performance/Conceptual/vImage/Art/kernel_convolution.jpg)

Top



Hình 5: Quá trình trượt và tính tích chập của một bộ lọc kích thước 3x3 trên ảnh và kết nối tới bản đồ kích hoạt, Source: github - iamaaditya (https://raw.githubusercontent.com/iamaaditya/iamaaditya.github.io/master/images/conv_arithmetic/full_padding_no_strides_transposed.g

Các khối nơ ron 3D: Không giống như những mạng nơ ron thông thường khi cấu trúc ở mỗi tầng là một ma trận 2D (batch size x số đơn vị ở mỗi tầng). Các kết quả ở mỗi tầng của một mạng nơ ron là một khối 3D được sắp xếp một cách hợp lý theo 3 chiều `rộng (width), cao (height), sâu (depth)`. Trong đó các chiều rộng và cao được tính toán theo công thức tích chập mục 1.1. Giá trị chiều rộng và cao của một tầng phụ thuộc vào kích thước của bộ lọc, kích thước của tầng trước, kích thước mở rộng (*padding*) và bước trượt bộ lọc (stride). Tuy nhiên chiều sâu lại hoàn toàn không phụ thuộc vào những tham số này mà nó bằng với số bộ lọc trong tầng đó. Quá trình tính bản đồ kích hoạt dựa trên một bộ lọc sẽ tạo ra một ma trận 2D. Như vậy khi áp dụng cho d bộ lọc khác nhau, mỗi bộ lọc có tác dụng trích suất một dạng đặc trưng trên mạng nơ ron, ta sẽ thu được d ma trận 2D có cùng kích thước mà mỗi ma trận là một bản đồ đặc trưng. Khi sắp xếp chồng chất các ma trận này theo chiều sâu kết quả đầu ra là một khối nơ ron 3D. Thông thường đối với xử lý ảnh thì tầng đầu vào có depth = 3 (số kênh) nếu các bức ảnh đang để ở dạng màu gồm 3 kênh RBG. Bên dưới là một cấu trúc mạng nơ ron điển hình có dạng khối.

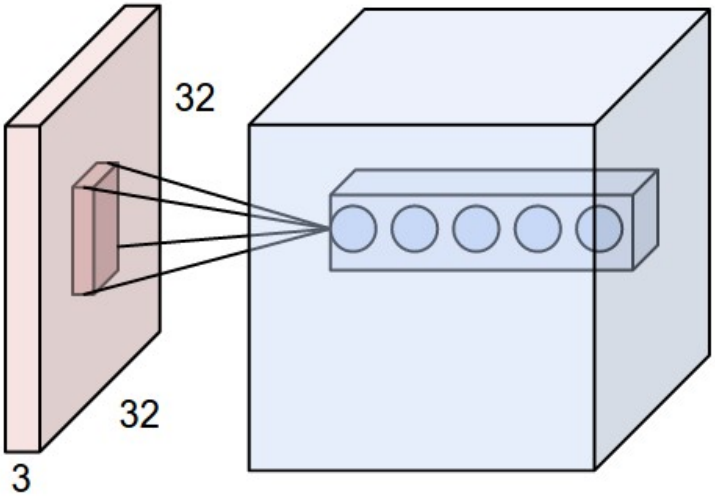


Hình 6: Cấu trúc các khối nơ ron 3D mạng Alexnet, Source: mdpi.com (https://www.mdpi.com/remotesensing/remotesensing-09-00848/article_deploy/html/images/remotesensing-09-00848-g001.png)

Tính chia sẻ kết nối và kết nối cục bộ: Chúng ta đã biết quá trình biến đổi trong mạng tích chập sẽ kết nối các khối nơ ron 3D. Tuy nhiên các đơn vị sẽ không kết nối tới toàn bộ khối 3D trước đó theo chiều rộng và cao mà chúng sẽ chọn ra các *vùng địa phương* (hoặc vùng nhận thức) có kích thước bằng với bộ lọc. Các vùng địa phương sẽ được chia sẻ chung một bộ siêu tham số có tác dụng nhận thức đặc trưng của bộ lọc. Các kết nối cục bộ không chỉ diễn ra theo chiều rộng và cao mà kết nối sẽ mở rộng hoàn toàn theo chiều sâu. Như vậy số tham số trong một tầng sẽ là $F \times F \times D$ (F, D lần lượt là kích thước bộ lọc và chiều depth).

Mỗi bộ lọc sẽ có khả năng trích xuất một đặc trưng nào đó như đã giải thích ở mục 1. Do đó khi đi qua toàn bộ các vùng địa phương của khối nơ ron 3D, các đặc trưng được trích xuất sẽ hiển thị trên tầng mới.

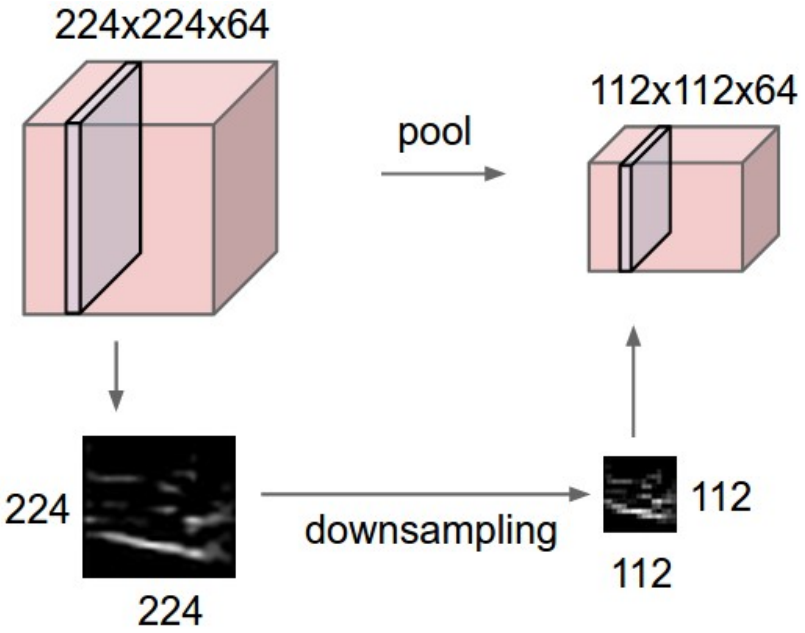
Top



Hình 7: Kết nối cục bộ, Source: cs231n - stanford (http://cs231n.github.io/assets/cnn/depthcol.jpeg)

> Giả sử ta có đầu vào là một bức ảnh 3 chiều kích tước 32x32x3. Khi đó mỗi đơn vị sẽ chỉ kết nối tới một vùng địa phương theo chiều rộng và cao nhưng sẽ mở rộng hoàn toàn kết nối theo chiều sâu. Chúng ta có tổng cộng 5 đơn vị (nơ ron) trong tầng cùng nhìn vào một vùng địa phương này và sẽ tạo ra cùng 1 vùng địa phương kích thước 1x1x5 trên khối nơ ron 3D mới.

Tính tổng hợp: Ở các tầng tích chập gần cuối số tham số sẽ cực kì lớn do sự gia tăng của chiều sâu và thông thường sẽ theo cấp số nhân. Như vậy nếu không có một cơ chế kiểm soát sự gia tăng tham số, chi phí tính toán sẽ cực kì lớn và vượt quá khả năng của một số máy tính cấu hình yếu. Một cách tự nhiên là chúng ta sẽ giảm kích thước các chiều rộng và cao bằng kỹ thuật giảm mẫu (*down sampling*) mà vẫn giữ nguyên được các đặc trưng của khối. Theo đó những bộ lọc được di chuyển trên bản đồ đặc trưng và tính trung bình (average pooling) hoặc giá trị lớn nhất (max pooling) của các phần tử trong vùng nhận thức. Trước đây các tính trung bình được áp dụng nhiều nhưng các mô hình hiện đại đã thay thế bằng giá trị lơn nhất do tốc độ tính max nhanh hơn so với trung bình.

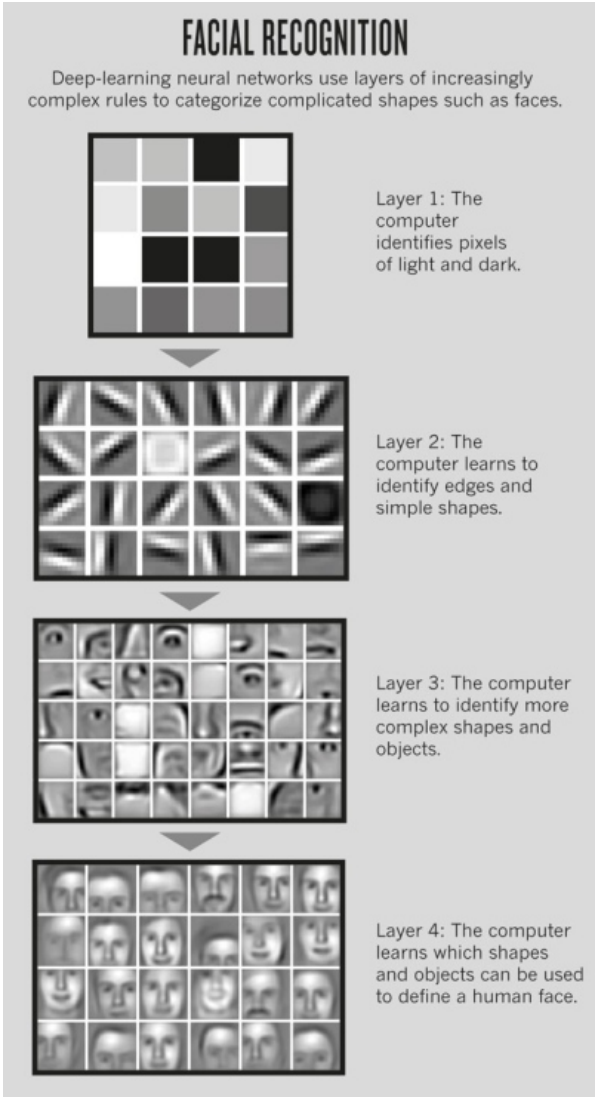


Hình 8: Quá trình tổng hợp, Source: cs231n - stanford (http://cs231n.github.io/assets/cnn/depthcol.jpeg)

> Chẳng hạn chúng ta có một khối nơ ron 3D kích thước 224x224x64. Sẽ cần 224x224x64 = 3211264 tham số để kết nối tới khối này. Chúng ta sẽ giảm kích thước kết nối đến khối 4 lần thông qua giảm chiều rộng và cao 2 lần mỗi chiều. Quá trình giảm chiều dữ liệu sẽ thực hiện lần lượt trên các lát cắt theo chiều sâu và không làm thay đổi độ lớn của chiều sâu. Khối mới vẫn giữ đặc trưng của khối cũ. Để đơn giản, bạn hình dung quá trình này cũng giống như zoom nhỏ bức ảnh lại.

Top

Độ phức tạp phát hiện hình ảnh tăng dần: Ở tầng đầu tiên, hình ảnh mà chúng ta có chỉ là những giá trị pixels. Sau khi đi qua tầng thứ 2 máy tính sẽ nhận diện được các hình dạng cạnh, rìa và các đường nét đơn giản được gọi là đặc trưng bậc thấp (low level). Càng ở những tầng tích chập về sau càng có khả năng phát hiện các đường nét phức tạp, đã rõ ràng hình thù và thậm chí là cấu thành vật thể, đây được gọi là những đặc trưng bậc cao (high level). Máy tính sẽ học từ tầng cuối cùng để nhận diện nhãn của hình ảnh.



Hình 9: Hình ảnh mô phỏng các phát hiện sau mỗi tầng

2. Xây dựng mạng nơ ron tích chập

Bên dưới ta sẽ tiến hanh xây dựng một mạng nơ ron tích chập phân biệt chữ số viết tay trong bộ số liệu mnist thông qua sử dụng API estimator của tensorflow. Phần source code này được lấy từ trang chủ của tensorflow và được hiệu chỉnh để phù hợp với mục đích của bài viết.

Top

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

tf.logging.set_verbosity(tf.logging.INFO)

def cnn_model_fn(features, labels, mode):
    """Model function for CNN"""
    #Input layer
    input_layer = tf.reshape(features['x'], shape = [-1, 28, 28, 1])

    #Convolution layer 1
    conv1 = tf.layers.conv2d(
        inputs = input_layer,
        filters = 32,
        kernel_size = [5, 5],
        padding = 'same',
        activation = tf.nn.relu)
    #Apply formula:N1 = (N+2P-f)/S + 1
    #in which: N is input image size, P is padding size, f is filter size and S is step
    #Output tensor shape: N1 = (28-5)/1+1 = 24 => shape = [-1, 24, 24, 1]
    #But we at parameter we set padding = 'same' in order to keep output shape unchange to
    #Thus output shape is [-1, 28, 28, 1]

    #Max pooling layer 1
    pool1 = tf.layers.max_pooling2d(
        inputs = conv1,
        pool_size = [2, 2],
        strides = 2)
    #Output tensor shape: N2 = (28-2)/2+1 = 14 => shape = [-1, 14, 14, 1]

    #Convolution layer 2
    conv2 = tf.layers.conv2d(
        inputs = pool1,
        filters = 64,
        kernel_size = [5, 5],
        padding = 'same',
        activation = tf.nn.relu)
    #Output tensor shape: N3 = (14-5)/1+1 = 10 => shape = [-1, 10, 10, 1]
    #But padding = 'same' so output shape is [-1, 14, 14, 1]

    #Max pooling layer 2
    pool2 = tf.layers.max_pooling2d(
        inputs = conv2,
        pool_size = [2, 2],
        strides = 2)
    #Output tensor shape: N4 = (14-2)/2+1 = 7 => shape = [-1, 7, 7, 1]

    #Dense layer
    flat = tf.reshape(pool2, [-1, 7*7*64])
    dense = tf.layers.dense(
        inputs = flat,
        units = 1024,
        activation = tf.nn.relu)

    dropout = tf.layers.dropout(
        inputs = dense,
        rate = 0.4,
        training = mode == tf.estimator.ModeKeys.TRAIN)

    #Logits layer
    logits = tf.layers.dense(inputs = dropout, units = 10)

    predictions = {
        'classes': tf.argmax(input = logits, axis = 1, name = 'class_tensor'),
        'probabilities': tf.nn.softmax(logits, name = 'softmax_tensor')}

    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(mode = mode, predictions = predictions)

    loss = tf.losses.sparse_softmax_cross_entropy(labels = labels, logits = logits)
```

Top

```
        if mode == tf.estimator.ModeKeys.TRAIN:
            optimizer = tf.train.AdamOptimizer(learning_rate = 0.001)
            train_op = optimizer.minimize(
                loss = loss,
                global_step = tf.train.get_global_step())
            return tf.estimator.EstimatorSpec(mode = mode, loss = loss, train_op = train_op)

        if mode == tf.estimator.ModeKeys.EVAL:
            eval_metric_ops = {
                'accuracy': tf.metrics.accuracy(
                labels = labels, predictions = predictions['classes'])}
            return tf.estimator.EstimatorSpec(
                mode = mode, loss = loss, eval_metric_ops = eval_metric_ops)
```

Như vậy mạng nơ ron của chúng ta sẽ có cấu trúc:

- tầng input: Có kích thước [-1, 28, 28, 1], số -1 biểu thị bất kì số lượng bức ảnh nào có thể truyền vào mô hình. 3 thành phần còn lại là chiều rộng, chiều cao và kênh của bức ảnh.
- tầng tích chập số 1: Gồm 32 bộ lọc có kích thước [5, 5]. Chúng ta có thể khai báo đơn giản là `kernel_size = 5` trong trường hợp bộ lọc là vuông. Tham số `padding = same` để cố định kích thước của đầu ra so với đầu vào. Khi đó tầng sẽ tự động thêm viền ngoài để kích thước không đổi theo công thức $P = \frac{W_1(S-1)-1+F}{2}$. Như vậy sau bước này kích thước đầu ra vẫn sẽ là [-1, 28, 28, 1].
- tầng chồng chất số 1: Có kích thước của bộ lọc là [2, 2] và bước nhảy là 2. Áp dụng công thức tính kích thước đầu ra ta sẽ suy ra w2 = h2 = (28-2)/2+1 = 14. Kích thước đầu ra sau bước này là [-1, 14, 14, 1].
- tầng tích chập số 2: Gồm 64 bộ lọc có kích thước [5, 5] và thám số `padding = same` sẽ không thay đổi kích thước đầu ra so với tầng trước là [-1, 14, 14, 1].
- tầng chồng chất số 2: Giống với tầng chồng chất số 1 với bộ lọc kích thước [2, 2] và bước nhảy 2. Do đó chiều dài và rộng của ma trận đầu ra sẽ là w2 = h2 = (14-2)/2+1 = 7. Kích thước đầu ra: [-1, 7, 7, 1].
- tầng vector dàn phẳng: Ma trận ở tầng trước sẽ được dành phẳng nên có kích thước là 7x7. Kết hợp với chiều sâu = 64 là số lượng đơn vị ở layer trước ta suy ra kích thước của tầng này là 7x7x64 = 3136.
- tầng dropout: tầng này không làm thay đổi kích thước của tầng trước mà chỉ tác động vào quá trình training khi sẽ tắt ngẫu nhiên các đơn vị của tầng trước với xác xuất là `rate = 0.4` bằng cách gán cho trọng số ứng với đơn vị bị tắt bằng 0. Đây là một kĩ thuật trong *kiểm soát* (regularization) mô hình nhằm giảm thiểu overfiting và tăng mức độ chính xác của dự báo và tốc độ huấn luyện.
- tầng output: Là một kết nối hoàn toàn tới 10 đơn vị đại diện cho 10 nhóm chữ số cần phân loại.

Bên dưới chúng ta sẽ load dữ liệu đầu vào dưới dạng numpy.

```
    import sys
    !{sys.executable} -m pip install python-mnist

    from mnist import MNIST
    mndata = MNIST('../input')

    mndata.load_training()
    train_data = np.asarray(mndata.train_images)/255.0
    train_labels = np.array(mndata.train_labels.tolist())

    mndata.load_testing()
    test_data = np.asarray(mndata.test_images)/255.0
    test_labels = np.array(mndata.test_labels.tolist())

    print('Train images shape      : %s'%str(train_data.shape))
    print('Train labels shape shape: %s'%str(train_labels.shape))
    print('Test  images shape      : %s'%str(test_data.shape))
    print('Test  labels shape shape: %s'%str(test_labels.shape))
```

Khởi tạo Estimator

```
    #Create the Estimator
    mnist_classifier = tf.estimator.Estimator(
        model_fn = cnn_model_fn,
        model_dir = './tmp/conv2_checkpoints' #temporary file to save model
    )

```

Top

Khởi tạo hàm truyền dữ liệu

```
#Training model
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x = {'x': train_data},
    y = train_labels,
    batch_size = 100,
    num_epochs = 50,
    shuffle = True
)
```

Hàm truyền dữ liệu sẽ bao gồm 2 biến chính là biến dự báo $\mathbf{x}$ và nhãn $\mathbf{y}$ với kích thước bach_size = 100 và mỗi batch sẽ được cập nhật dữ liệu 1 lần. Khi chuyển qua batch mới sẽ thay đổi vị trí các quan sát.

Huẩn luyện mô hình

```
mnist_classifier.train(
    input_fn = train_input_fn,
    steps = 10000
#     hooks = [logging_hook]
)
```

Đánh giá mô hình trên tập test

```
#Validation on test
eval_input_fn = tf.estimator.inputs.numpy_input_fn(
      x = {"x": test_data},
      y = test_labels,
      num_epochs = 1,
      shuffle = False)

eval_results = mnist_classifier.evaluate(input_fn=eval_input_fn)
print(eval_results)
```

3. Tài liệu

1. Tài liệu CS231n - Mạng nơ ron tích chập ứng dụng trong nhận diện hình ảnh - Standford (http://cs231n.github.io/convolutional-networks/)
2. Tích chấp 2 chiều - Blog machine learning cơ bản - Vũ Hữu Tiệp (https://machinelearningcoban.com/2018/10/03/conv2d)
3. Xây dựng mạng nơ ron tích chập sử dụng estimator - Tensoflow (https://www.tensorflow.org/tutorials/estimators/cnn)
4. Image kenel - Victor Powell (http://setosa.io/ev/image-kernels/)
5. Image Filtering - Blog Machine Learning Guru (http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html)

Top