TRƯỜNG ĐẠI HỌC TRẦN ĐẠI NGHĨA

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

BÀI TẬP LỚN MÔN HỌC
ĐỒ ÁN CHUYÊN NGÀNH

Đề tài: "Nguyên cứu, xây dựng nhận diện khuôn mặt sử dụng Deeplearning"

TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2019

TRƯỜNG ĐẠI HỌC TRẦN ĐẠI NGHĨA

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

BÀI TẬP LỚN MÔN HỌC
ĐỒ ÁN CHUYÊN NGÀNH

Đề tài: "Nguyên cứu, xây dựng nhận diện khuôn mặt sử dụng Deeplearning"

GVHD: Ts. Phùng Thế Bảo

Sinh viên thực hiện:

Nguyễn Lê Xuân Phước

Tôn Nữ Nguyên Hậu

Lớp: 16DDS06031

TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2019

MỤC LỤC

DANH MỤC HÌNH ẢNH

LỜI MỞ ĐẦU

Công nghệ thông tin đang được ứng dụng trong mọi lĩnh vực của cuộc sống. Với một hệ thống máy tính, chúng ta có thể làm được rất nhiều việc, tiết kiệm thời gian và công sức. Điển hình như công việc nhận dạng mặt người. Ngày xưa, muốn tìm kiếm một kẻ tình nghi trong siêu thị hay sân bay, các nhân viên an ninh phải tìm kiếm trên từng màn hình camera theo dõi. Ngày nay, công việc đấy đã được làm tự động nhờ các hệ thống nhận dạng mặt người. Phát hiện mặt người trong ảnh là một phần quan trọng của hệ thống nhận dạng mặt người đó, giải quyết tốt việc phát hiên mặt người sẽ giúp tiết kiệm thời gian và nâng cao độ chính xác của việc nhận dạng khuôn mặt.

Phát hiện mặt người cũng là một bài toán nhận dạng đơn giản, hệ thống chỉ cần phân loại đối tượng đưa vào có phải mặt người hay không phải mặt người. Ở mức độ cao hơn, sau khi đã phát hiện được khuôn mặt, các khuôn mặt đó sẽ được so sánh với các khuôn mặt có trong dữ liệu để nhận dạng xem khuôn mặt đấy là của ai (thường áp dụng trong nhận dạng khuôn mặt của người nổi tiếng hoặc của tội phạm đang bị truy nã).

Với mục tiêu chính là tìm hiểu thuật toán Facenet và MTCNN, các đặc trưng phân lớp SVM, mô hình Pre-train of Classifiers, đồng thời áp dụng vào bài toán phát hiện mặt người trong ảnh, video, webcam.

Em xin chân thành cảm ơn thầy ***Phùng Thế Bảo*** đã tận tình hướng dẫn và giúp đỡ em hoàn thành cuốn đồ án này.

Do kinh nghiệm và kiến thức chưa được sâu sắc nên trong báo cáo về đề tài của nhóm mong thầy góp ý thêm để nhóm có thể hoàn thiện tốt hơn các đề tài nghiên cứu về sau.

Chúng em xin chân thành cảm ơn !

TP. Hồ Chí Minh, ngày 05 tháng 12 năm 2019

CHƯƠNG 1. TÌM HIỂU NGÔN NGỮ LẬP TRÌNH PYTHON

1.1. Giới thiệu ngôn ngữ Python

Python là một ngôn ngữ lập trình thông dịch do Guido van Rossum tạo ra năm 1990. Python hoàn toàn tạo kiểu động và dùng cơ chế cấp phát bộ nhớ tự động; do vậy nó tương tự như Perl, Ruby, Scheme, Smalltalk, và Tcl. Python được phát triển trong một dự án mã mở, do tổ chức phi lợi nhuận Python Software Foundation quản lý.

Theo đánh giá của Eric S. Raymond, Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình. Cấu trúc của Python còn cho phép người sử dụng viết mã lệnh với số lần gõ phím tối thiểu, như nhận định của chính Guido van Rossum trong một bài phỏng vấn ông.

Ban đầu, Python được phát triển để chạy trên nền Unix. Nhưng rồi theo thời gian, nó đã "bành trướng" sang mọi hệ điều hành từ MS-DOS đến Mac OS, OS/2, Windows, Linux và các hệ điều hành khác thuộc họ Unix. Mặc dù sự phát triển của Python có sự đóng góp của rất nhiều cá nhân, nhưng Guido van Rossum hiện nay vẫn là tác giả chủ yếu của Python. Ông giữ vai trò chủ chốt trong việc quyết định hướng phát triển của Python.

1.1.1. Lịch sử phát triển

Python được hình thành vào cuối những năm 1980, và việc thực hiện nó vào tháng 12 năm 1989 bởi Guido van Russom tại Centrum Wiskunde & Informatica (CWI) ở Hà Lan như là một kế thừa cho ngôn ngữ ABC (tự lấy cảm hứng từ SETL) có khả năng xử lý ngoại lệ và giao tiếp với Hệ điều hành Amoeba. Van Rossum là tác giả chính của Python, và vai trò trung tâm của ông trong việc quyết định hướng phát triển của Python.

Hình 1. 1: Guido van Russom

Sự phát triển Python đến nay có thể chia làm các giai đoạn:

Python 1: bao gồm các bản phát hành 1.x. Giai đoạn này, kéo dài từ đầu đến cuối thập niên 1990. Từ năm 1990 đến 1995, Guido làm việc tại CWI (Centrum voor Wiskunde en Informatica - Trung tâm Toán-Tin học tại Amsterdam, Hà Lan). Vì vậy, các phiên bản Python đầu tiên đều do CWI phát hành. Phiên bản cuối cùng phát hành tại CWI là 1.2.

Vào năm 1995, Guido chuyển sang CNRI (Corporation for National Research Initiatives) ở Reston, Virginia. Tại đây, ông phát hành một số phiên bản khác. Python 1.6 là phiên bản cuối cùng phát hành tại CNRI.

Sau bản phát hành 1.6, Guido rời bỏ CNRI để làm việc với các lập trình viên chuyên viết phần mềm thương mại. Tại đây, ông có ý tưởng sử dụng Python với các phần mềm tuân theo chuẩn GPL. Sau đó, CNRI và FSF (Free Software Foundation - Tổ chức phần mềm tự do) đã cùng nhau hợp tác để làm bản quyền Python phù hợp với GPL. Cùng năm đó, Guido được nhận Giải thưởng FSF vì Sự phát triển Phần mềm tự do (Award for the Advancement of Free Software).

Phiên bản 1.6.1 ra đời sau đó là phiên bản đâu tiền tuân theo bản quyền GPL. Tuy nhiên, bản này hoàn toàn giống bản 1.6, trừ một số sửa lỗi cần thiết.

Python 2: vào năm 2000, Guido và nhóm phát triển Python dời đến BeOpen.com và thành lập BeOpen PythonLabs team. Phiên bản Python 2.0 được phát hành tại đây. Sau khi phát hành Python 2.0, Guido và các thành viên PythonLabs gia nhập Digital Creations.

Python 2.1 ra đời kế thừa từ Python 1.6.1 và Python 2.0. Bản quyền của phiên bản này được đổi thành Python Software Foundation License. Từ thời điểm này trở đi, Python thuộc sở hữu của Python Software Foundation (PSF), một tổ chức phi lợi nhuận được thành lập theo mẫu Apache Software Foundation.

Python 3, còn gọi là Python 3000 hoặc Py3K: Dòng 3.x sẽ không hoàn toàn tương thích với dòng 2.x, tuy vậy có công cụ hỗ trợ chuyển đổi từ các phiên bản 2.x sang 3.x. Nguyên tắc chủ đạo để phát triển Python 3.x là "bỏ cách làm việc cũ nhằm hạn chế trùng lặp về mặt chức năng của Python". Trong PEP (Python Enhancement Proposal) có mô tả chi tiết các thay đổi trong Python. Các đặc điểm mới của Python 3.0 sẽ được trình bày phần cuối bài này.

1.1.2. Phiên bản của Python

- *CPython*

Đây là phiên bản đầu tiên và được duy trì lâu nhất của Python, được viết trong C. Những đặc điểm của ngôn ngữ mới xuất hiện đầu tiên từ đây

- *Jython*

Là phiên bản Python trong môi trường Java. Bản này có thể được sử dụng như là một ngôn ngữ script cho những ứng dụng Java. Nó cũng thường được sử dụng để tạo ra những tests thử nghiệm cho thư viện Java

- *Python for .NET*

Phiên bản này thật ra sử dụng phiên bản Cpython, nhưng nó là một ứng dụng .NET được quản lý, và tạo ra thư viện .NET sẵn dùng. Bản này được xây dựng bởi Brian Lloyd.

- *IronPython*

Là một bản Python tiếp theo của .NET, không giống như Python.NET đây là một bản Python hoàn chỉnh, tạo ra IL, và biên dịch mã Python trực tiếp vào một tập hợp .NET.

- *PyPy*

PyPy được viết trên Python, thậm chí cả việc thông dịch từng byte mã cũng được viết trên Python. Nó sử dụng Cpython như một trình thông dịch cơ sở. Một trong những mục đích của dự án cho ra đời phiên bản này là để khuyến khích sự thử nghiệm bản thân ngôn ngữ này làm cho nó dễ dàng hơn để sửa đổi thông dịch (bởi vì nó được viết trên Python).

1.1.3. Một điểm khác nhau giữa phiên bản 3.x và 2.x

- *Division operator*

Thay đổi này đặc biệt nguy hiểm nếu bạn đang thực thi code Python 3 trong Python 2, vì thay đổi trong hành vi phân chia số nguyên thường có thể không được chú ý (nó không làm tăng cú pháp SyntaxError).

```
print 7 / 5
print -7 / 5
Output in Python 2.x:
1
-2
Output in Python 3.x:
1.4
-1.4
```

- *Print function*

Đây là một trong những sự thay đổi được biết đến nhiều nhất từ bản Python 2.x lên Python 3.x. Python 2.x không có vấn đề với các lệnh bổ sung, nhưng ngược lại, Python 3.x sẽ tăng cú pháp SyntaxError nếu chúng ta gọi hàm in mà không có dấu ngoặc đơn.

```
print 'Hello, Geeks'    # Python 3.x doesn't support
print('Hope You like these facts')
Output in Python 2.x:
Hello, Geeks
Hope You like these facts

Output in Python 3.x:
File "a.py", line 1
    print 'Hello, Geeks'
                        ^
SyntaxError: invalid syntax
```

- *Unicode*

Trong Python 2.x, kiểu mặc định của String là ASCII, nhưng ở Python 3.x kiểu mặc định của String là Unicode và 2 lớp byte: byte và bytearrays.

```
print(type('default string '))
print(type(u'string with b '))
Output in Python 2.x (Unicode and str are different):
<type 'str'>
<type 'unicode'>

Output in Python 3.x (Unicode and str are same):
<class 'str'>
<class 'str'>
```

- *Xrange*

Việc sử dụng xrange () rất phổ biến trong Python 2.x để tạo một đối tượng có thể lặp lại, ví dụ: trong vòng lặp for hoặc list / set-dictionary-comprehension. xrange-iterable không phải hoàn toàn có nghĩa là bạn có thể lặp lại nó vô hạn.

Trong Python 3, range () đã được thực hiện giống như hàm xrange () sao cho hàm xrange () chuyên dụng không còn tồn tại nữa (xrange () tăng một NameError trong Python 3).

```
for x in xrange(1, 5):
    print(x),

for x in range(1, 5):
    print(x),
Output in Python 2.x:
1 2 3 4 1 2 3 4
Output in Python 3.x:
```

```
NameError: name 'xrange' is not defined
```

- *Error Handling*

Đây là một thay đổi nhỏ trên phiên bản 3.x, từ khoá as đã trở thành bắt buộc Kiểm tra không có từ khoá as trong 2 phiên bản Python

```
try:
    trying_to_check_error
except NameError, err:
    print err, 'Error Caused'   # Would not work in Python 3.x
Output in Python 2.x:
name 'trying_to_check_error' is not defined Error Caused


Output in Python 3.x :
File "a.py", line 3
    except NameError, err:
                    ^
SyntaxError: invalid syntax
```

Kiểm tra khi có từ khoá as trong 2 phiên bản Python

```
try:
     trying_to_check_error
except NameError as err: # 'as' is needed in Python 3.x
     print (err, 'Error Caused')
Output in Python 2.x:
(NameError("name 'trying_to_check_error' is not defined",), 'Error Caused')

Output in Python 3.x:
name 'trying_to_check_error' is not defined Error Caused
```

- *The next() function and next() method*

Next() hay (.next ()) là một hàm thường được sử dụng (method), đây là một thay đổi cú pháp khác (hay đúng hơn là thay đổi trong thực thi) sử dụng tốt trong python 2 nhưng bị loại bỏ trong python3

Vd: python 2.x:

```
my_generator = (letter for letter in 'abcdefg')

next(my_generator)
my_generator.next()
--------------------------------
result:
'a'
'b'
```

python3.x:

```
my_generator = (letter for letter in 'abcdefg')
next(my_generator)

--------------------------------
'a'

my_generator.next()

---------------------------------------------------------------------------
AttributeError              Traceback (most recent call last)

<ipython-input-14-125f388bb61b> in <module>()
----> 1 my_generator.next()

AttributeError: 'generator' object has no attribute 'next'
```

- *Các biến vòng lặp và rò rỉ namespace chung*

Trong Python 3.x cho các biến vòng lặp không bị rò rỉ vào không gian tên chung nữa List comprehensions không còn hỗ trợ mẫu cú pháp [... for var in item1, item2, ...]. Sử dụng [... for var in (item1, item2, ...)] thay thế. Cũng lưu ý rằng việc hiểu danh sách có ngữ nghĩa khác nhau: chúng gần với cú pháp cho biểu thức máy hiểu bên trong một hàm tạo danh sách (và đặc biệt là các biến điều khiển vòng lặp không còn bị rò rỉ ra ngoài phạm vi. ”

Vd: Python 2.x

```
i = 1
print 'before: i =', i
print 'comprehension: ', [i for i in range(5)]

print 'after: i =', i
--------------------------------
before: i = 1
comprehension:  [0, 1, 2, 3, 4]
after: i = 4
```

Python 3.x:

```
i = 1
print('before: i =', i)

print('comprehension:', [i for i in range(5)])
print('after: i =', i)
--------------------------------
before: i = 1
comprehension: [0, 1, 2, 3, 4]
after: i = 1
```

- *So sánh khác loại*

  Ở python 2 ,có thể so sánh list với string,... nhưng ở python 3 thì không

  Vd: python 2.x

```
print "[1, 2] > 'foo' = ", [1, 2] > 'foo'
print "(1, 2) > 'foo' = ", (1, 2) > 'foo'
print "[1, 2] > (1, 2) = ", [1, 2] > (1, 2)
--------------------------------
[1, 2] > 'foo' =  False
(1, 2) > 'foo' =  True
[1, 2] > (1, 2) =  False
```

Python 3.x

```
print("[1, 2] > 'foo' = ", [1, 2] > 'foo')
print("(1, 2) > 'foo' = ", (1, 2) > 'foo')
print("[1, 2] > (1, 2) = ", [1, 2] > (1, 2))
--------------------------------
TypeError                    Traceback (most recent call last)

<ipython-input-16-a9031729f4a0> in <module>()
      1 print('Python', python_version())
----> 2 print("[1, 2] > 'foo' = ", [1, 2] > 'foo')
      3 print("(1, 2) > 'foo' = ", (1, 2) > 'foo')
      4 print("[1, 2] > (1, 2) = ", [1, 2] > (1, 2))
      TypeError: unorderable types: list() > str()
```

- *Phân tích cú pháp đầu vào của người dùng thông qua input()*

Hàm input () được cố định trong Python 3 để nó luôn lưu trữ đầu vào của người dùng làm các đối tượng str. Để tránh hành vi nguy hiểm trong Python 2 để đọc trong các loại khác so với chuỗi, chúng ta phải sử dụng raw_input () để thay thế.

Vd: Python 2.x

```
Python 2.7.6
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> my_input = input('enter a number: ')

enter a number: 123

>>> type(my_input)
<type 'int'>

>>> my_input = raw_input('enter a number: ')
enter a number: 123

>>> type(my_input)
```

```
<type 'str'>
```

Python 3.x

```
Python 3.4.1
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> my_input = input('enter a number: ')

enter a number: 123

>>> type(my_input)
<class 'str'>
```

Trả về các đối tượng có thể lặp lại thay vì danh sách

- *Một số hàm và phương thức trả về các đối tượng có thể lặp lại trong Python 3 ngay-thay vì các danh sách trong Python 2*

  Vd: Python 2.x

```
print range(3)
print type(range(3))
--------------------------------
[0, 1, 2]
<type 'list'>
```

Python 3

```
print(range(3))
print(type(range(3)))
print(list(range(3)))
--------------------------------
range(0, 3)
<class 'range'>
[0, 1, 2]
```

Một số hàm và phương thức thường được sử dụng không trả về danh sách nữa trong Python 3:

```
zip()
map()
filter()
dictionary’s .keys() method
dictionary’s .values() method
```

```
dictionary’s .items() method
```

1.1.4. Đặc điểm của Python

- Dễ học, dễ đọc

Python được thiết kế để trở thành một ngôn ngữ dễ học, mã nguồn dễ đọc, bố cục trực quan, dễ hiểu, thể hiện qua các điểm sau:

- Từ khóa
  - Python tăng cường sử dụng từ khóa tiếng Anh, hạn chế các kí hiệu và cấu trúc cú pháp so với các ngôn ngữ khác.
  - Python là một ngôn ngữ phân biệt kiểu chữ HOA, chữ thường.
  - Như C/C++, các từ khóa của Python đều ở dạng chữ thường.

- Khối lệnh

Trong các ngôn ngữ khác, khối lệnh thường được đánh dấu bằng cặp kí hiệu hoặc từ khóa. Ví dụ, trong C/C++, cặp ngoặc nhọn { } được dùng để bao bọc một khối lệnh. Python, trái lại, có một cách rất đặc biệt để tạo khối lệnh, đó là thụt các câu lệnh trong khối vào sâu hơn (về bên phải) so với các câu lệnh của khối lệnh cha chứa nó. Ví dụ, giả sử có đoạn mã sau trong C/C++:

```
#include <math.h>
//...
delta = b * b - 4 * a * c;
if (delta > 0) {
// Khoi lenh moi bat dau tu kí tu { den }
x1 = (- b + sqrt(delta)) / (2 * a);
x2 = (- b - sqrt(delta))/ (2 * a);
printf("Phuong trinh co hai nghiem phan biet:\n");
printf("x1 = %f; x2 = %f", x1, x2);
}
```

Đoạn mã trên có thể được viết lại bằng Python như sau:

```
# -*- coding: utf-8 -*-
"""
Spyder Editor
This is a temporary script file.
"""
import math
  #...
```

```
    delta = b * b – 4 * a * c
    if delta > 0:
    # Khối lệnh mới, thụt vào đầu dòng
    x1 = (- b + math.sqrt(delta)) / (2 * a)
    x2 = (- b – math.sqrt(delta)) / (2 * a)
    print "Phuong trinh co hai nghiem phan biet:"
print "x1 = ", x1, "; ", "x2 = ", x2
```

Ta có thể sử dụng dấu tab hoặc khoảng trống để thụt các câu lệnh vào.

- Các bản hiện thực

Python được viết từ những ngôn ngữ khác, tạo ra những bản hiện thực khác nhau. Bản hiện thực Python chính, còn gọi là CPython, được viết bằng C, và được phân phối kèm một thư viện chuẩn lớn được viết hỗn hợp bằng C và Python. CPython có thể chạy trên nhiều nền và khả chuyển trên nhiều nền khác. Dưới đây là các nền trên đó, CPython có thể chạy.

- Các hệ điều hành họ Unix: AIX, Darwin, FreeBSD, Mac OS X, NetBSD, Linux, OpenBSD, Solaris,…
- Các hệ điều hành dành cho máy desktop: Amiga, AROS, BeOS, Mac OS 9, Microsoft Windows, OS/2, RISC OS.
- Các hệ thống nhúng và các hệ đặc biệt: GP2X, Máy ảo Java, Nokia 770 Internet Tablet, Palm OS, PlayStation 2, PlayStation Portable, Psion, QNX, Sharp Zaurus, Symbian OS, Windows CE/Pocket PC, Xbox/XBMC, VxWorks.
- Các hệ máy tính lớn và các hệ khác: AS/400, OS/390, Plan 9 from Bell Labs, VMS, z/OS.

Ngoài CPython, còn có hai hiện thực Python khác: Jython cho môi trường Java và IronPython cho môi trường .NET và Mono.

- Khả năng mở rộng

Python có thể được mở rộng: nếu ta biết sử dụng C, ta có thể dễ dàng viết và tích hợp vào Python nhiều hàm tùy theo nhu cầu. Các hàm này sẽ trở thành hàm xây dựng sẵn (built-in) của Python. Ta cũng có thể mở rộng chức năng của trình thông dịch, hoặc liên kết các chương trình Python với các thư viện chỉ ở dạng nhị phân (như các thư viện đồ họa do nhà sản xuất thiết bị cung cấp). Hơn thế nữa, ta cũng có thể liên kết trình thông dịch của Python với các ứng dụng viết từ C và sử dụng nó như là một mở rộng hoặc một ngôn ngữ dòng lệnh phụ trợ cho ứng dụng đó.

- Trình thông dịch

Python là một ngôn ngữ lập trình dạng thông dịch, do đó có ưu điểm tiết kiệm thời gian phát triển ứng dụng vì không cần phải thực hiện biên dịch

và liên kết. Trình thông dịch có thể được sử dụng để chạy file script, hoặc cũng có thể được sử dụng theo cách tương tác. Ở chế độ tương tác, trình thông dịch Python tương tự shell của các hệ điều hành họ Unix, tại đó, ta có thể nhập vào từng biểu thức rồi gõ Enter, và kết quả thực thi sẽ được hiển thị ngay lập tức. Đặc điểm này rất hữu ích cho người mới học, giúp họ nghiên cứu tính năng của ngôn ngữ; hoặc để các lập trình viên chạy thử mã lệnh trong suốt quá trình phát triển phần mềm. Ngoài ra, cũng có thể tận dụng đặc điểm này để thực hiện các phép tính như với máy tính bỏ túi.

- Lệnh và cấu trúc điều khiển

Mỗi câu lệnh trong Python nằm trên một dòng mã nguồn. Ta không cần phải kết thúc câu lệnh bằng bất kì kí tự gì. Cũng như các ngôn ngữ khác, Python cũng có các cấu trúc điều khiển. Chúng bao gồm:

Cấu trúc rẽ nhánh: cấu trúc if (có thể sử dụng thêm elif hoặc else), dùng để thực thi có điều kiện một khối mã cụ thể.

Cấu trúc lặp, bao gồm:

- Lệnh while: chạy một khối mã cụ thể cho đến khi điều kiện lặp có giá trị false.
- Vòng lặp for: lặp qua từng phần tử của một dãy, mỗi phần tử sẽ được đưa vào biến cục bộ để sử dụng với khối mã trong vòng lặp.

Python cũng có từ khóa class dùng để khai báo lớp (sử dụng trong lập trình hướng đối tượng) và lệnh def dùng để định nghĩa hàm.

- Hệ thống kiểu dữ liệu

Python sử dụng hệ thống kiểu duck typing, còn gọi là latent typing (tự động xác định kiểu). Có nghĩa là, Python không kiểm tra các ràng buộc về kiểu dữ liệu tại thời điểm dịch, mà là tại thời điểm thực thi. Khi thực thi, nếu một thao tác trên một đối tượng bị thất bại, thì có nghĩa là đối tượng đó không sử dụng một kiểu thích hợp.

Python cũng là một ngôn ngữ định kiểu mạnh. Nó cấm mọi thao tác không hợp lệ, ví dụ cộng một con số vào chuỗi kí tự.

Sử dụng Python, ta không cần phải khai báo biến. Biến được xem là đã khai báo nếu nó được gán một giá trị lần đầu tiên. Căn cứ vào mỗi lần gán, Python sẽ tự động xác định kiểu dữ liệu của biến. Python có một số kiểu dữ liệu thông dụng sau:

- int, long: số nguyên (trong phiên bản 3.x long được nhập vào trong kiểu int). Độ dài của kiểu số nguyên là tùy ý, chỉ bị giới hạn bởi bộ nhớ máy tính.
- float: số thực
- complex: số phức, chẳng hạn 5+4j

- list: dãy trong đó các phần tử của nó có thể được thay đổi, chẳng hạn [8, 2, 'b', -1.5] . Kiểu dãy khác với kiểu mảng (array) thường gặp trong các ngôn ngữ lập trình ở chỗ các phần tử của dãy không nhất thiết có kiểu giống nhau. Ngoài ra phần tử của dãy còn có thể là một dãy khác.
- tuple: dãy trong đó các phần tử của nó không thể thay đổi.
- str: chuỗi kí tự. Từng kí tự trong chuỗi không thể thay đổi. Chuỗi kí tự được đặt trong dấu nháy đơn, hoặc nháy kép.
- dict: từ điển, còn gọi là "hashtable": là một cặp các dữ liệu được gắn theo kiểu {từ khóa: giá trị}, trong đó các từ khóa trong một từ điển nhất thiết phải khác nhau. Chẳng hạn {1: "Python", 2: "Pascal"}
- set: một tập không xếp theo thứ tự, ở đó, mỗi phần tử chỉ xuất hiện một lần.

Ngoài ra, Python còn có nhiều kiểu dữ liệu khác. Xem thêm trong phần "Các kiểu dữ liệu" bên dưới.

- Module

Python cho phép chia chương trình thành các module để có thể sử dụng lại trong các chương trình khác. Nó cũng cung cấp sẵn một tập hợp các modules chuẩn mà lập trình viên có thể sử dụng lại trong chương trình của họ. Các module này cung cấp nhiều chức năng hữu ích, như các hàm truy xuất tập tin, các lời gọi hệ thống, trợ giúp lập trình mạng (socket),…

- Đa năng

Python là một ngôn ngữ lập trình đơn giản nhưng rất hiệu quả.

- So với Unix shell, Python hỗ trợ các chương trình lớn hơn và cung cấp nhiều cấu trúc hơn.
- So với C, Python cung cấp nhiều cơ chế kiểm tra lỗi hơn. Nó cũng có sẵn nhiều kiểu dữ liệu cấp cao, ví dụ như các mảng (array) linh hoạt và từ điển (dictionary) mà ta sẽ phải mất nhiều thời gian nếu viết bằng C.

Python là một ngôn ngữ lập trình cấp cao có thể đáp ứng phần lớn yêu cầu của lập trình viên:

- Python thích hợp với các chương trình lớn hơn cả AWK và Perl.
- Python được sử dụng để lập trình Web. Nó có thể được sử dụng như một ngôn ngữ kịch bản.
- Python được thiết kế để có thể nhúng và phục vụ như một ngôn ngữ kịch bản để tuỳ biến và mở rộng các ứng dụng lớn hơn.
- Python được tích hợp sẵn nhiều công cụ và có một thư viện chuẩn phong phú, Python cho phép người dùng dễ dàng tạo ra các dịch vụ Web, sử dụng các thành phần COM hay CORBA, hỗ trợ các loại định dạng dữ liệu Internet như email, HTML, XML và các ngôn ngữ

đánh dấu khác. Python cũng được cung cấp các thư viện xử lý các giao thức Internet thông dụng như HTTP, FTP,…

- Python có khả năng giao tiếp đến hầu hết các loại cơ sở dữ liệu, có khả năng xử lí văn bản, tài liệu hiệu quả, và có thể làm việc tốt với các công nghệ Web khác.
- Python đặc biệt hiệu quả trong lập trình tính toán khoa học nhờ các công cụ Python Imaging Library, pyVTK, MayaVi 3D Visualization Toolkits, Numeric Python, ScientificPython,…
- Python có thể được sử dụng để phát triển các ứng dụng desktop. Lập trình viên có thể dùng wxPython, PyQt, PyGtk để phát triển các ứng dụng giao diện đồ họa (GUI) chất lượng cao. Python còn hỗ trợ các nền tảng phát triển phần mềm khác như MFC, Carbon, Delphi, X11, Motif, Tk, Fox, FLTK, …
- Python cũng có sẵn một unit testing framework để tạo ra các các bộ test (test suites).

- Multiple paradigms (đa biến hóa)

Python là một ngôn ngữ đa biến hóa (multiple paradigms). Có nghĩa là, thay vì ép buộc mọi người phải sử dụng duy nhất một phương pháp lập trình, Python lại cho phép sử dụng nhiều phương pháp lập trình khác nhau: hướng đối tượng, có cấu trúc, chức năng, hoặc chỉ hướng đến một khía cạnh. Python kiểu kiểu động và sử dụng bộ thu gom rác để quản lí bộ nhớ. Một đặc điểm quan trọng nữa của Python là giải pháp tên động, kết nối tên biến và tên phương thức lại với nhau trong suốt thực thi của chương trình.

- Sự tương đương giữa true và một giá trị khác 0

Cũng như C/C++, bất kì một giá trị khác 0 nào cũng tương đương với true và ngược lại, một giá trị 0 tương đương với false. Như vậy:

```
if a != 0:
```

tương đương với:

```
if a:
```

- Cú pháp

Sau đây là cú pháp cơ bản nhất của ngôn ngữ Python:

+ *Toán tử*

```
+ - * / // (chia làm tròn) % (phần dư) ** (lũy thừa)
    ~ (not) & (and) | (or) ^ (xor)
    << (left shift) >> (right shift)
    == (bằng) <= >= != (khác)
```

Python sử dụng kí pháp trung tố thường gặp trong các ngôn ngữ lập trình khác.

+ *Các kiểu dữ liệu*

✍ Kiểu số

```
1234585396326 (số nguyên dài vô hạn) -86.12 7.84E-04
    2j 3 + 8j (số phức)
```

✍ Kiểu chuỗi (string)

```
"Hello" "It's me" '"OK"-he replied'
```

✍ Kiểu bộ (tuple)

```
(1, 2.0, 3) (1,) ("Hello",1,())
```

✍ Kiểu danh sách (list)

```
[4.8, -6] ['a','b']
```

✍ Kiểu từ điển (dictionary)

```
  {"Vietnam":"Hanoi", "Netherlands":"Amsterdam", "France":"Paris"}
```

+ *Chú thích*

```
  # dòng chú thích
```

+ *Lệnh gán*

```
tên biến = biểu thức
    x = 23.8
    y = -x ** 2
    z1 = z2 = x + y
    loiChao = "Hello!"

i += 1 # tăng biến i thêm 1 đơn vị
```

+ *In giá trị*

```
print biểu thức
    print (7 + 8) / 2.0
    print (2 + 3j) * (4 - 6j)
```

+ *Nội suy chuỗi (string interpolation)*

```
print "Hello %s" %("world!")

    print "i = %d" %i
```

```
    print "a = %.2f and b = %.3f" %(a,b)
```

+ *Cấu trúc rẽ nhánh*

✍ *Dạng 1:*

```
if biểu_thức_điều_kiện:
    # lệnh ...
```

✍ *Dạng 2:*

```
if biểu_thức_điều_kiện:
    # lệnh ...
    else:
    # lệnh ...
```

✍ *Dạng 3:*

```
if biểu_thức_điều_kiện_1:
    # lệnh ... (được thực hiện nếu biểu_thức_điều_kiện_1 là đúng/true)
    elif biểu_thức_điều_kiện_2:
    # lệnh ... (được thực hiện nếu biểu_thức_điều_kiện_1 là sai/false,
nhưng biểu_thức_điều_kiện_2 là đúng/true)
    else:
    # lệnh ... (được thực hiện nếu tất cả các biểu thức điều kiện đi kèm if
và elif đều sai)
```

+ *Cấu trúc lặp*

```
while biểu_thức_đúng:
    # lệnh ...
    for phần_tử in dãy:
    # lệnh ...
    L = ["Ha Noi", "Hai Phong", "TP Ho Chi Minh"]
    for thanhPho in L:
    print thanhPho

for i in range(10):
    print i
```

+ *Hàm*

```
def tên_hàm (tham_biến_1, tham_biến_2, tham_biến_n):
    # lệnh ...
    return giá_trị_hàm
    def binhPhuong(x):
    return x*x
```

+ *Hàm với tham số mặc định:*

```
def luyThua(x, n=2):
    """Lũy thừa với số mũ mặc định là 2"""
    return x**n
```

```
print luyThua(3) # 9
    print luyThua(2,3) # 8
```

+ *Lớp*

```
class Tên_Lớp_1:
    # ...

class Tên_Lớp_2(Tên_Lớp_1):
    """Lớp 2 kế thừa lớp 1"""
    x = 3 # biến thành viên của lớp
    #
    def phương_thức(self, tham_biến):
    # ...

# khởi tạo
    a = Tên_Lớp_2()
    print a.x
    print a.phương_thức(m) # m là giá trị gán cho tham biến
    List Comprehension
```

+ *List Comprehension*

Là dạng cú pháp đặc biệt (syntactic sugar) (mới có từ Python 2.x) cho phép thao tác trên toàn bộ dãy (list) mà không cần viết rõ vòng lặp. Chẳng hạn y là một dãy mà mỗi phần tử của nó bằng bình phương của từng phần tử trong dãy x:

```
y = [xi**2 for xi in x]
```

+ *Xử lý lỗi*

```
try:
    câu_lệnh
    except Loại_Lỗi:
    thông báo lỗi
```

+ *Tốc độ thực hiện*

Là một ngôn ngữ thông dịch, Python có tốc độ thực hiện chậm hơn nhiều lần so với các ngôn ngữ biên dịch như Fortran, C, v.v... Trong số các ngôn ngữ thông dịch, Python được đánh giá nhanh hơn Ruby và Tcl, nhưng chậm hơn Lua.

+ *Các đặc điểm mới trong Python 3.x*

Nội dung phần này được trích từ tài liệu của Guido van Rossum. Phần này không liệt kê đầy đủ tất cả các đặc điểm; chi tiết xin xem tài liệu nói trên.

+ *Một số thay đổi cần lưu ý nhất*

Lệnh print trở thành hàm print. Theo đó sau print() ta cần nhớ gõ vào cặp ngoặc ():

```
print("Hello")
    print(2+3)
```

Trả lại kết quả không còn là list trong một số trường hợp.

- dict.keys(), dict.items(), dict.values() kết quả cho ra các "view" thay vì list.
- map và filter trả lại các iterator.
- range bây giờ có tác dụng như xrange, và không trả lại list.

✍ So sánh

Không còn hàm cmp, và cmp(a, b) có thể được thay bằng (a > b) - (a < b)

✍ Số nguyên

- Kiểu long được đổi tên thành int.
- 1/2 cho ta kết quả là số thực chứ không phải số nguyên.
- Không còn hằng số sys.maxint
- Kiểu bát phân được kí hiệu bằng 0o thay vì 0, chẳng hạn 0o26.

Phân biệt văn bản - dữ liệu nhị phân thay vì Unicode - chuỗi 8-bit

- Tất cả chuỗi văn bản đều dưới dạng Unicode, nhưng chuỗi Unicode mã hóa lại là dạng dữ liệu nhị phân. Dạng mặc định là UTF-8.
- Không thể viết u"a string" để biểu diễn chuỗi như trong các phiên bản 2.x

+ *Các thay đổi về cú pháp*

✍ Cú pháp mới

- Các tham biến chỉ chấp nhận keyword: Các tham biến phía sau *args phải được gọi theo dạng keyword.
- Từ khóa mới nonlocal. Muốn khai báo một biến x với có phạm vi ảnh hưởng rộng hơn, nhưng chưa đến mức toàn cục, ta dùng nonlocal x.
- Gán giá trị vào các phần tử tuple một cách thông minh, chẳng hạn có thể viết (a, *rest, b) = range(5) để có được a = 0; b = [1,2,3]; c = 4.
- Dictionary comprehension, chẳng hạn {k: v for k, v in stuff} thay vì dict(stuff).
- Kiểu nhị phân, chẳng hạn b110001.

✍ Cú pháp được thay đổi

- raise [biểu_thức [from biểu_thức]]
- except lệnh as biến
- Sử dụng metaclass trong đối tượng:

```
class C(metaclass=M):
    pass
```

Cách dùng biến __metaclass__ không còn được hỗ trợ.

\+ *Cú pháp bị loại bỏ*

- Không còn dấu ``, thay vì đó, dùng repr.
- Không còn so sánh <> (dùng !=).

Không còn các lớp kiểu classic.

1.2. Hướng dẫn cài đặt Ananconda

1.2.1. Giới thiệu Ananconda

Anaconda là nền tảng mã nguồn mở về Khoa học dữ liệu trên Python thông dụng nhất hiện nay. Anaconda hướng đến việc quản lí các package một cách đơn giản, phù hợp với mọi người. Hệ thống quản lí package của Anaconda là Conda. Anaconda Với hơn 11 triệu người dùng, Anaconda là cách nhanh nhất và dễ nhất để học Khoa học dữ liệu với Python hoặc R trên Windows, Linux và Mac OS X. Lợi ích của Anaconda:

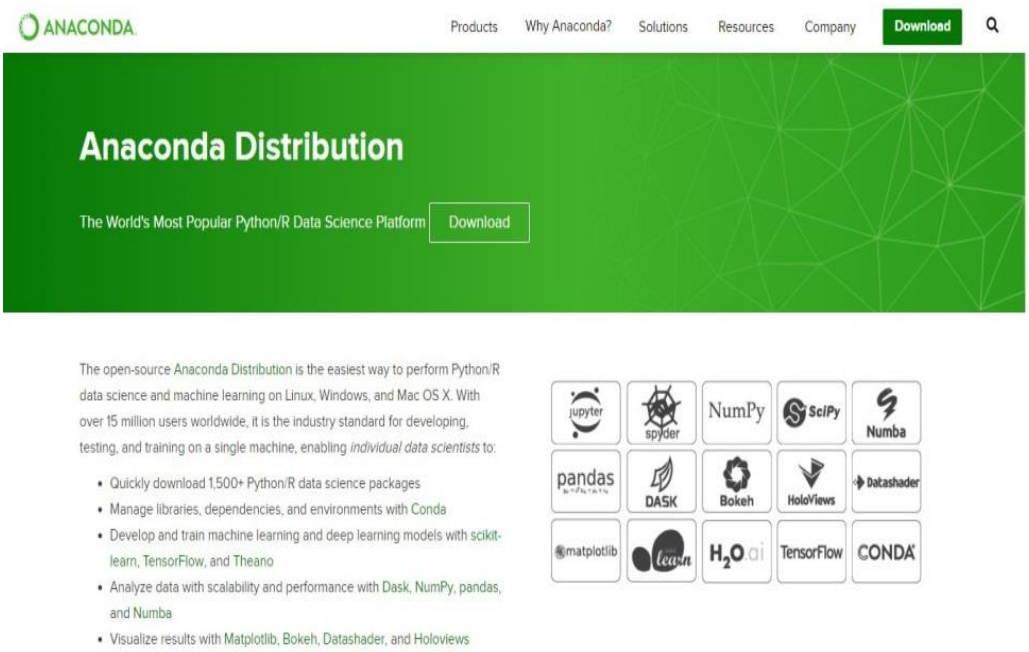
- Dễ dàng tải 1500+ packages về Python/R cho data science
- Quản lý thư viện, môi trường và dependency giữa các thư viện dễ dàng
- Dễ dàng phát triển mô hình machine learning và deep learning với scikit-learn, tensorflow, keras
- Xử lý dữ liệu tốc độ cao với numpy, pandas
- Hiện thị kết quả với Matplotlib, Bokeh

Trong khi đó Spyder là 1 trong những IDE (môi trường tích hợp dùng để phát triển phần mềm) tốt nhất cho data science và quang trọng hơn là nó được cài đặt khi bạn cài đặt Anaconda.

Yêu cầu phần cứng và phần mềm:

- Hệ điều hành: Win 7, Win 8/8.1, Win 10, Red Hat Enterprise Linux/CentOS 6.7, 7.3, 7.4, and 7.5, and Ubuntu 12.04+.
- Ram tối thiểu 4GB

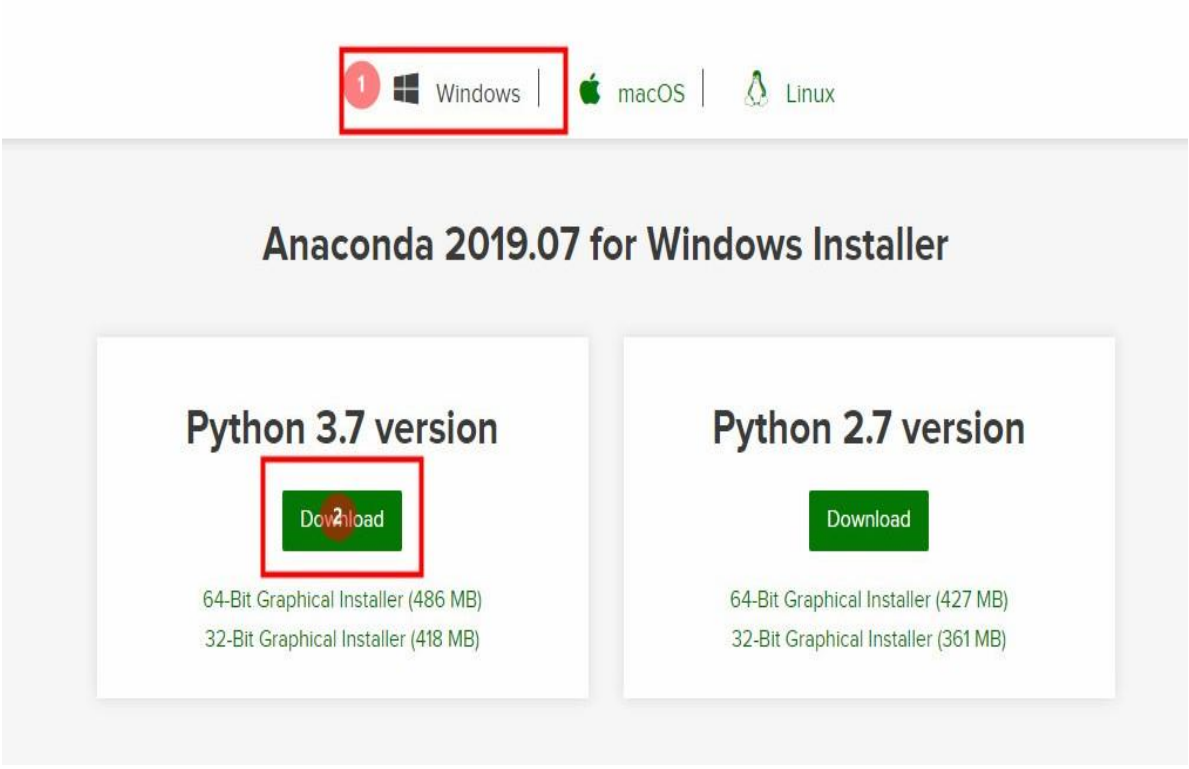
Ổ cứng trống tối thiểu 3GB để tải và cài đặt



Hình 1.2:Giao diện trang chủ Anaconda

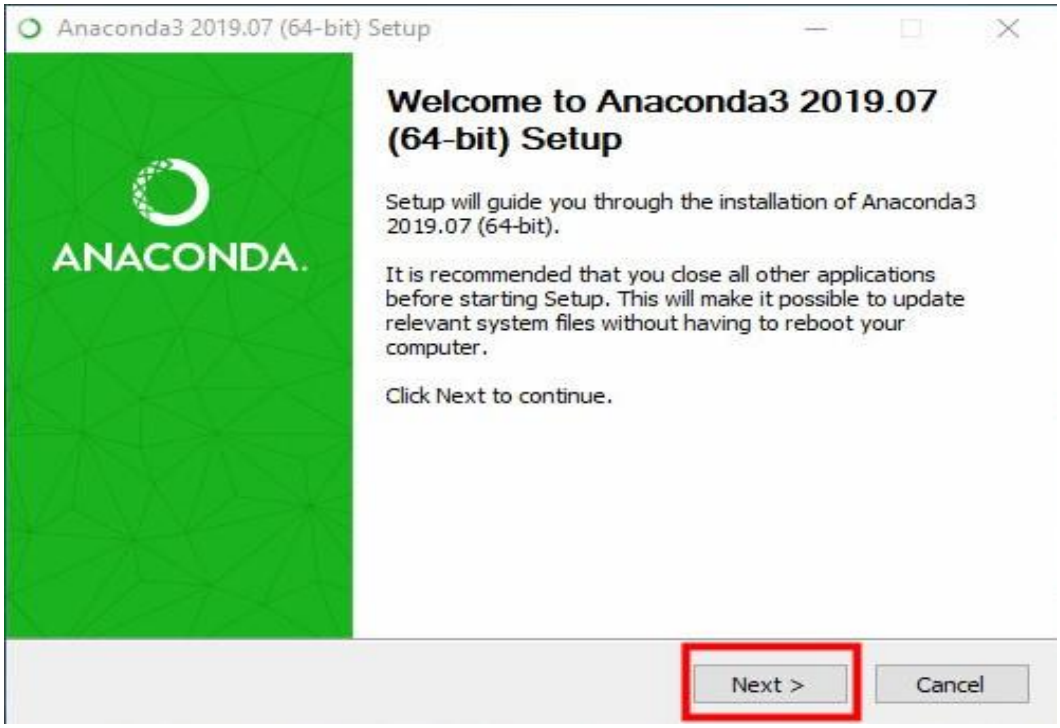
1.2.2. Download Anaconda và hướng dẫn cài đặt trên HĐH Windown

- Download Anaconda (python3) cho HĐH Windown về tại https://www.anaconda.com/distribution/
- Hiện tại phiên bản Python mà ta sử dụng là python3.7 bản phân phối Anaconda 2019.07

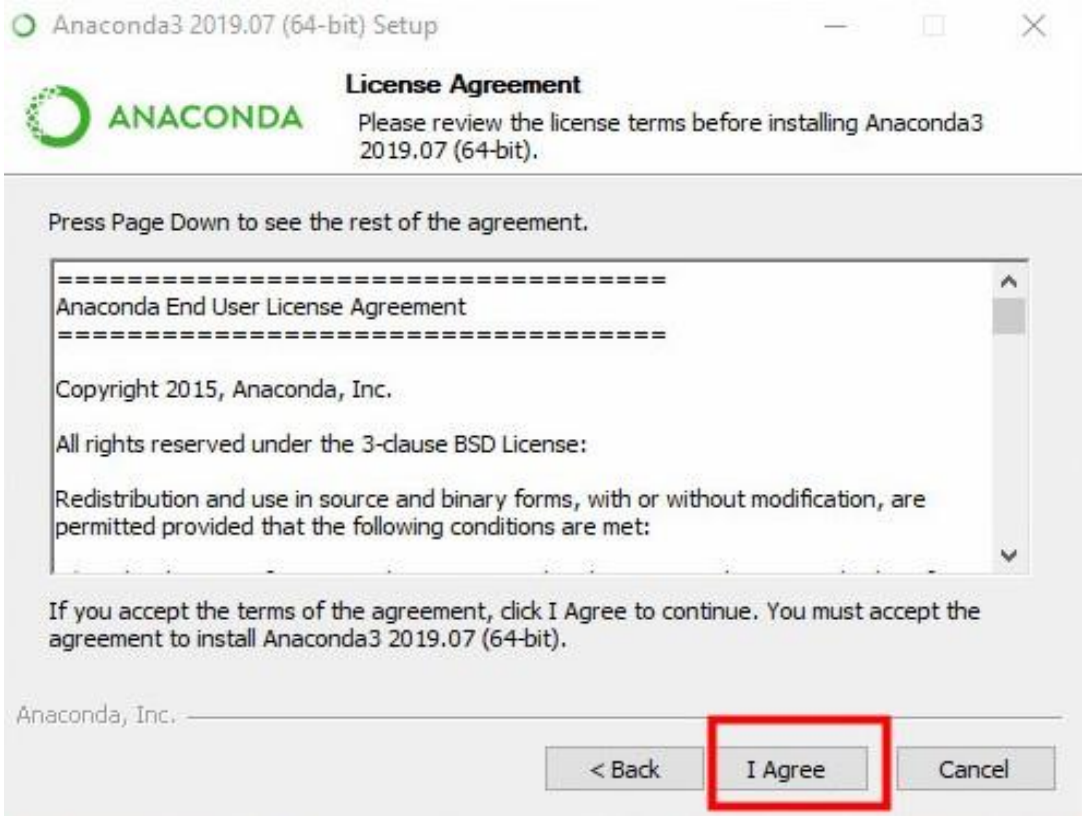


Hình 1.3: Trang tải Anaconda

Sau khi tải về xong bạn chạy file "Anaconda3-2019.07-Windows-x86_64.exe"

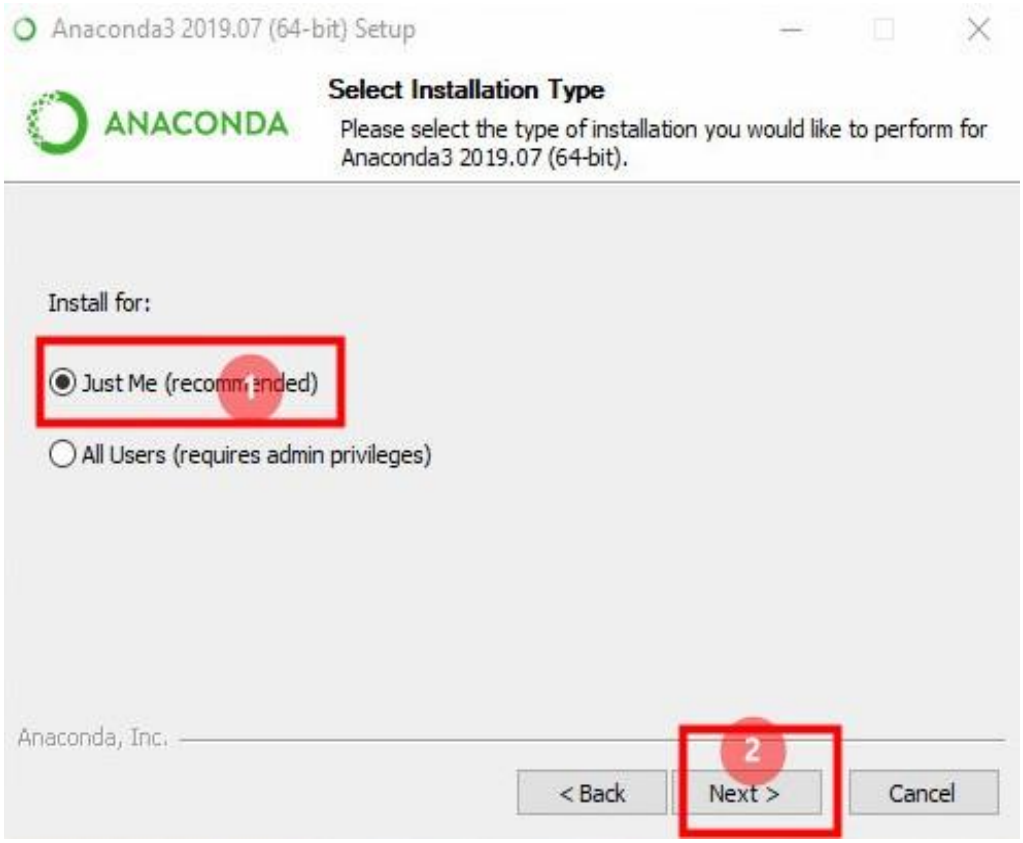


Hình 1. 4: Chạy chương trình vừa tải về



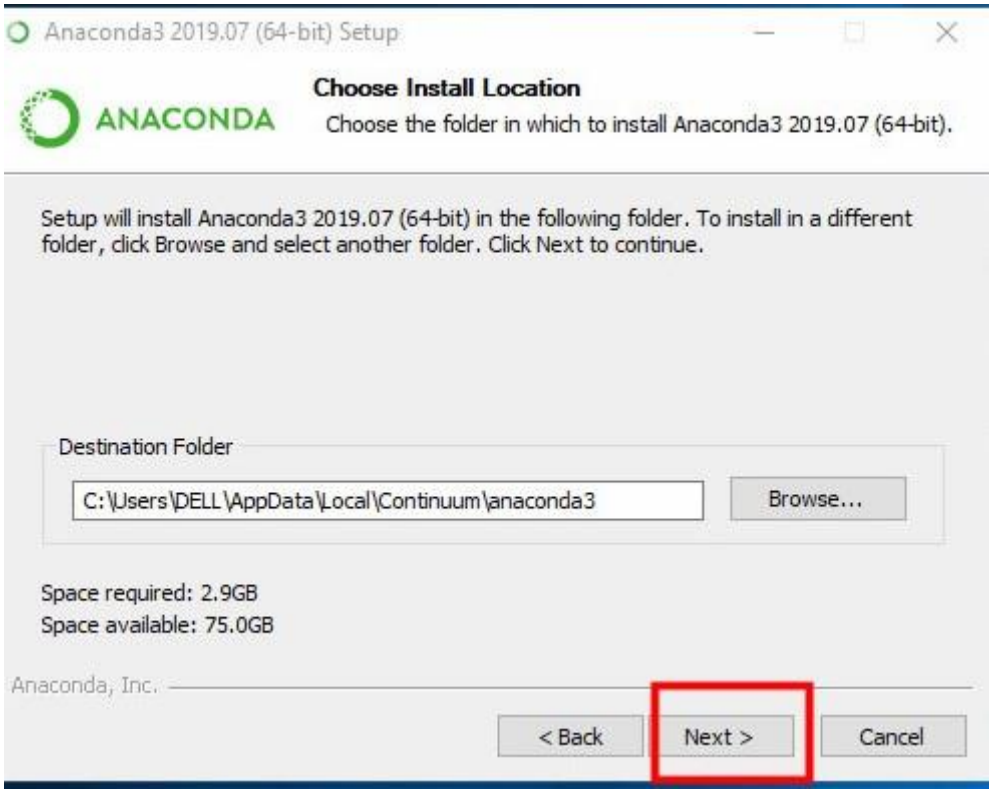
Hình 1. 5: License Agreement

- *Click* **I Agree**



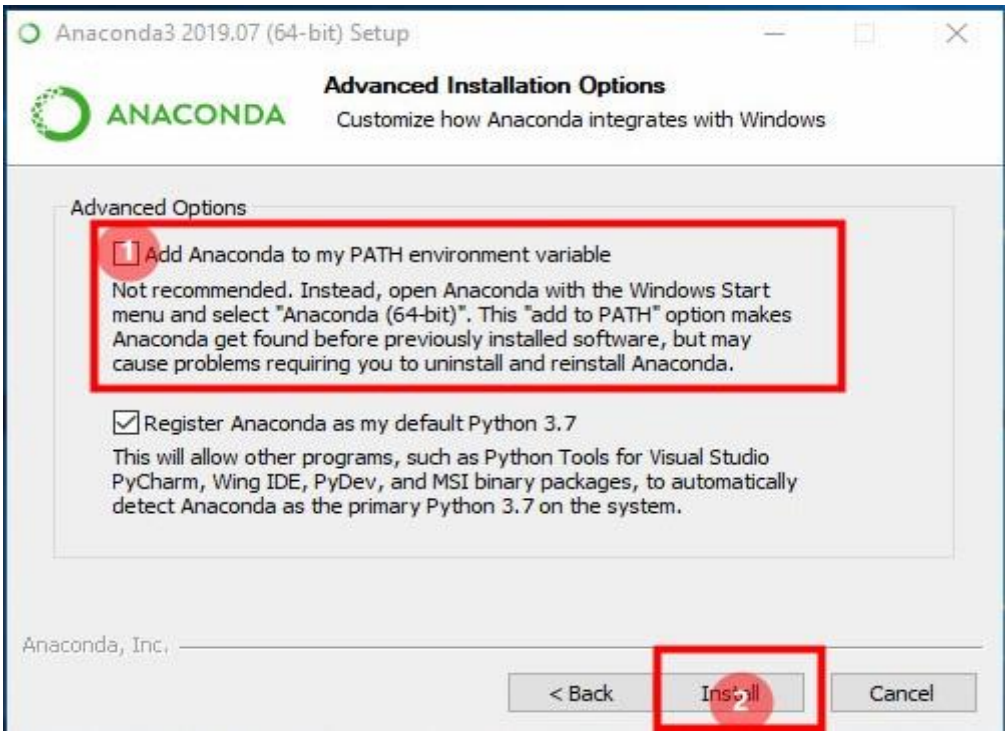
Hình 1.6:Chọn Loại cài đặt

- Bạn chọn "Just Me" sau đó *Click* **Next**



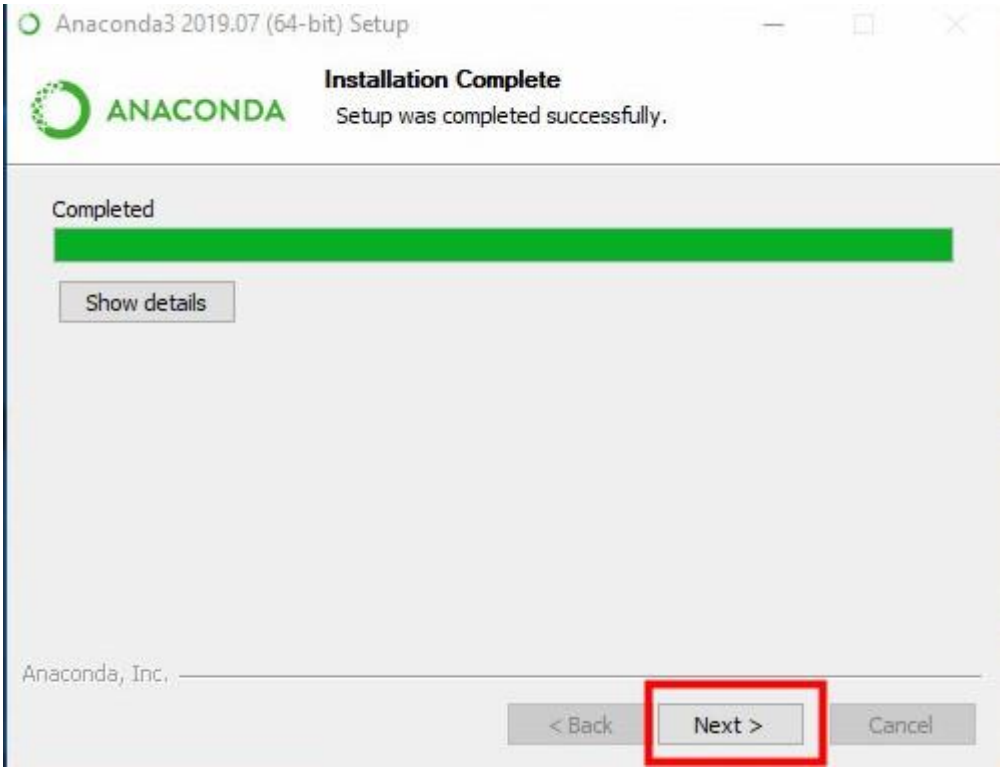
Hình 1. 7: Chọn Vị trí cài đặt

- *Click* **Next**



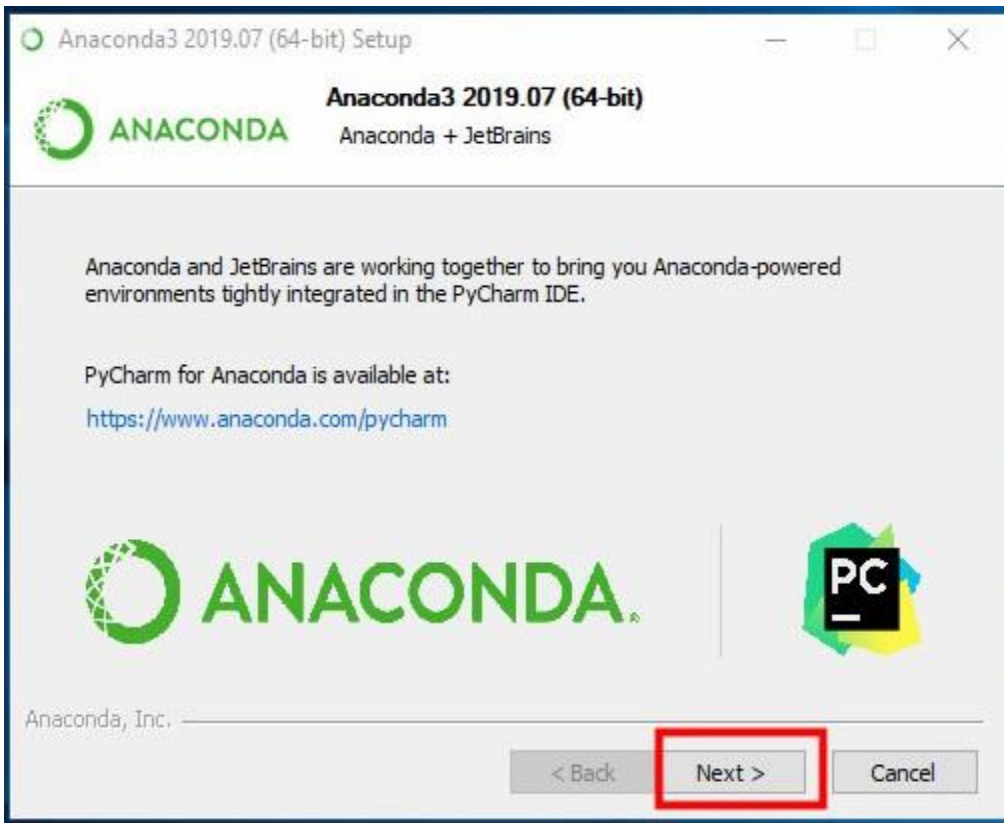
Hình 1. 8: Advanced Options

- Lưu ý: hãy chắc là bạn chọn tuỳ chọn được khoanh đỏ bên dưới để thêm các câu lệnh Anaconda vào **System Environment Variable (PATH)** của Windows
- *Sau đó Click* **Next**



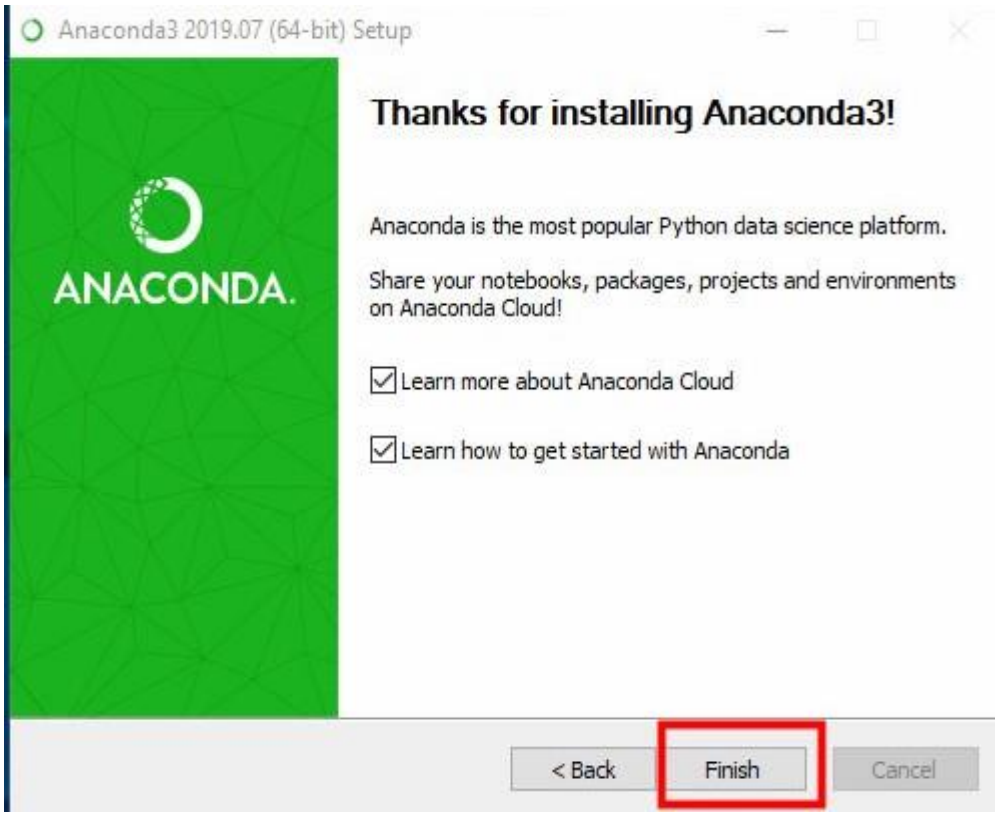
Hình 1. 9:Hoàn thành Cài đặt

- *Tiếp tục Click* **Next**



Hình 1. 10: Anaconda và JetBrains

- *Click* **Next**



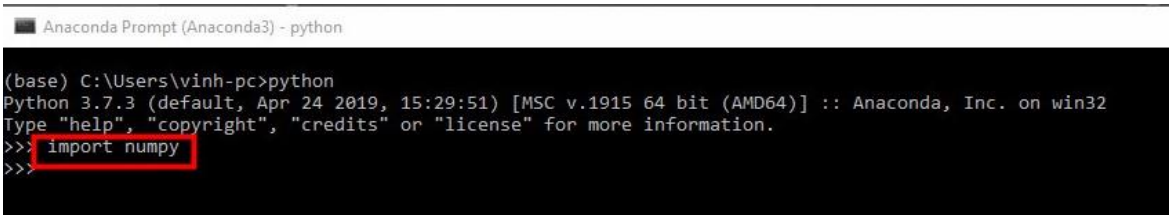
Hình 1.11: Cài đặt hoàn tất và khởi động chương trình

- Tiếp theo bạn click **Finish** để hoàn tất việc cài đặt

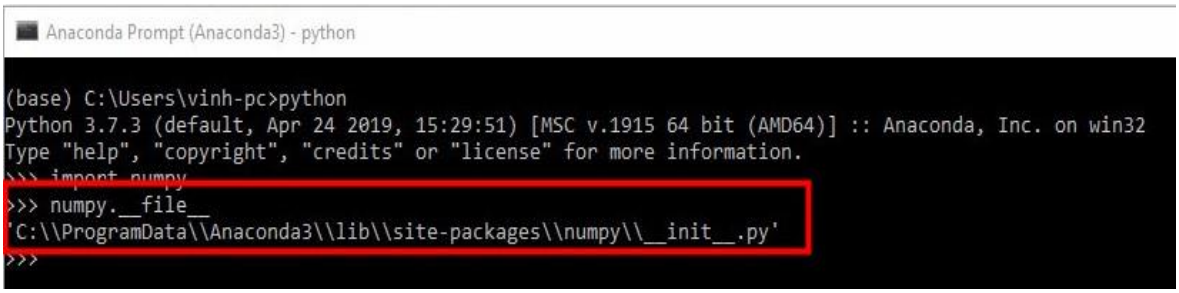
1.2.3. Hướng dẫn cài đặt thêm thư viện

Anaconda đã có sẵn khá là nhiều thư viện python như : Numpy, Scipy, Matplotlib, sklearn,…

Để kiểm tra python của Anaconda đã có thư viện nào đó, chúng ta sẽ thử import nó trong Console

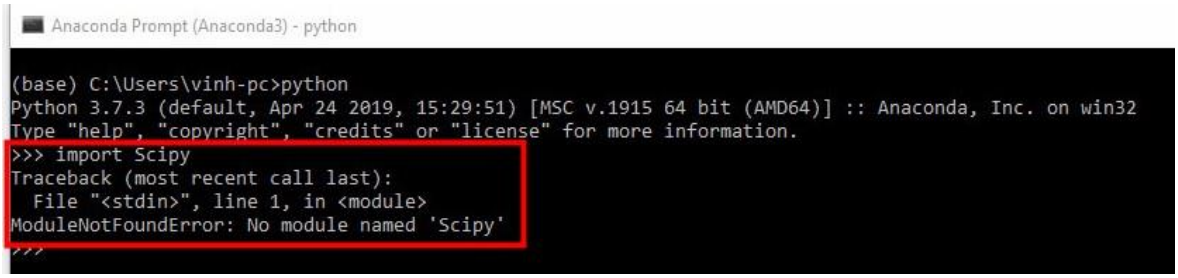


Không có lỗi được thông báo nghĩa là python đã biết được thư viện này. Để kiểm tra thư viện này ở đâu, sau khi *import*, ta truy xuất đường dẫn của thư viện như sau:



Thư viện Numpy của tôi nằm ở đường dẫn *'C:\ProgramData\Anaconda3\lib\site-packages\'*.

Anaconda đã có sẵn thư viện Numpy

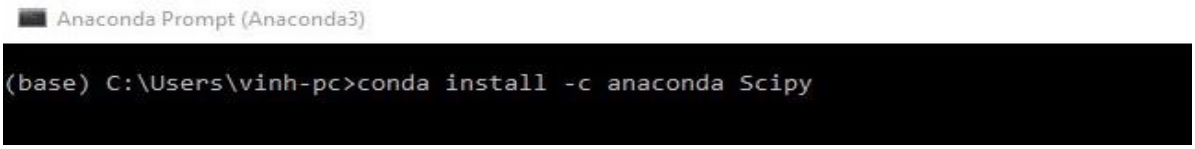


Nếu như Python trả về lỗi Import như trên thì có nghĩa trong Anaconda chúng ta chưa có thư viện đó.

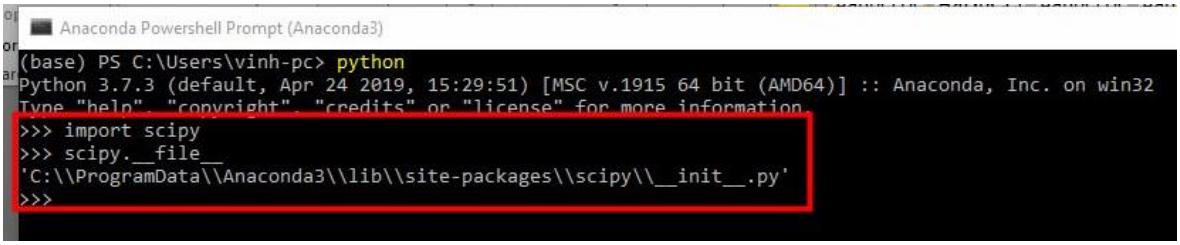
Ở phần trên python của tôi chưa có thư viện *Scipy*, nên tôi phải đi cài đặt nó. Vì tôi sử dụng Anaconda cho lập trình python nên tôi cần phải *(1) cài đặt thư viện mới vào đường dẫn libs python của Anaconda* hoặc *(2) chỉ cho python của Anaconda biết về đường dẫn tới thư viện mới này*.

Với Anaconda, việc cài đặt 1 thư viện đang được hỗ trợ cực kỳ đơn giản, tôi chỉ cần dùng tools *pip* hoặc *conda* mà Anaconda đã cài sẵn. Cụ thể, ở đây tôi muốn cài thư viện *Scipy* tôi truy cập vào trang chủ của Scipy. Trang này ghi rằng chúng ta có thể cài bằng *pip* hoặc *conda*.

Chúng ta sẽ bật Anaconda Prompt (Anaconda3) lên và gõ conda install -c anaconda Scipy. Conda sẽ tự động tìm thư viện *Scipy* và cài vào đường dẫn Anaconda giúp chúng ta.



Chờ cho thư viện và các thư viện liên quan hoàn tất cài đặt, chúng ta vào spyder kiểm tra lại đã có *Scipy* chưa. Và python trả về đã có Scipy trong Anaconda. Và chúng ta đã có thể sử dụng Scipy



Với 1 thư viện chưa có trên Anaconda, cách cài đặt sẽ phức tạp hơn chút nhưng hầu hết các thư viện lớn thường dùng đều có thể cài đặt thông qua Anaconda, nên chúng ta không phải lo lắng lắm.

Ngoài ra chúng ta có thể cài đặt thêm các thư viện bằng Anaconda Navigator

1.3.Hướng dẫn cài đặt Pycharm

1.3.1. Giới thiệu Pycharm

JetBrains PyCharm cung cấp một bộ công cụ hoàn chỉnh cho các nhà phát triển Python chuyên nghiệp. PyCharm được xây dựng xung quanh một trình soạn thảo hiểu mã sâu sắc, và một trình sửa lỗi cho cái nhìn rõ ràng về hoạt động của mã. PyCharm cung cấp khả năng tích hợp với các công cụ cộng tác như hệ thống kiểm soát phiên bản và các tracker. Trình biên tập Chuyên nghiệp mở rộng các yếu tố cần thiết bằng cách tích hợp liền mạch với các khuôn khổ web, các công cụ JavaScript, ảo hóa và hỗ trợ containerization.

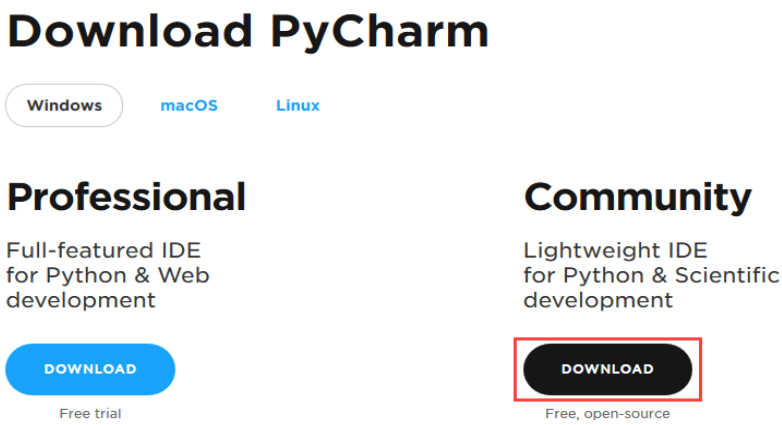
Một khía cạnh quan trọng của chương trình là hiểu được nền tảng mã mà bạn đang đưa vào. PyCharm đảm bảo bạn có thể khám phá dự án của bạn chỉ với một vài thao tác trên phím, nó cung cấp cho bạn một cái nhìn tổng quan về cấu trúc dự án và cho phép bạn truy cập vào các tài liệu có liên quan ngay từ trình soạn thảo. Hiểu được một nền tảng code nhanh hơn có nghĩa là thúc đẩy nhanh hơn quá trình phát triển của bạn.

1.3.2. Hướng dẫn cài đặt Pycharm

Có 2 phiên bản PyCharm:

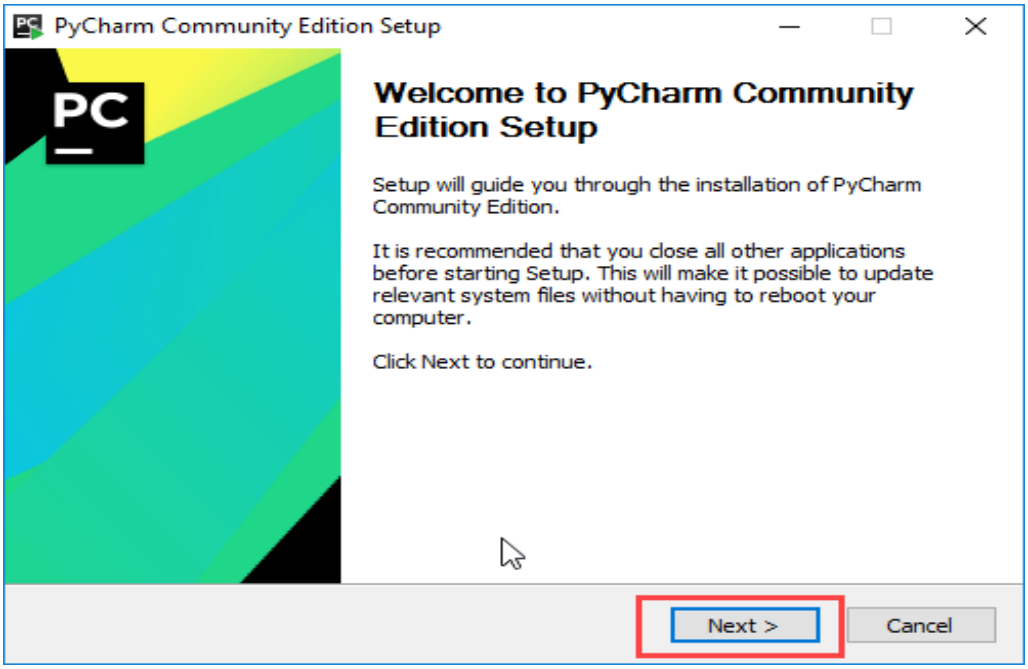
+ Bản **Professional**: Có đầy đủ tất cả các tính năng từ cơ bản đến nâng cao để phát triển Python, nhưng ta phải mua bản quyền. Ta có thể download bản dùng thử.
+ Bản **Community**: Là bản chứa các tính năng cơ bản, để có thể phát triển Python. Bản này được tải miễn phí.

Đối với các dự án cá nhân thì bản Community đã hoàn toàn đầy đủ tính năng. Hãy Download bản đó về.



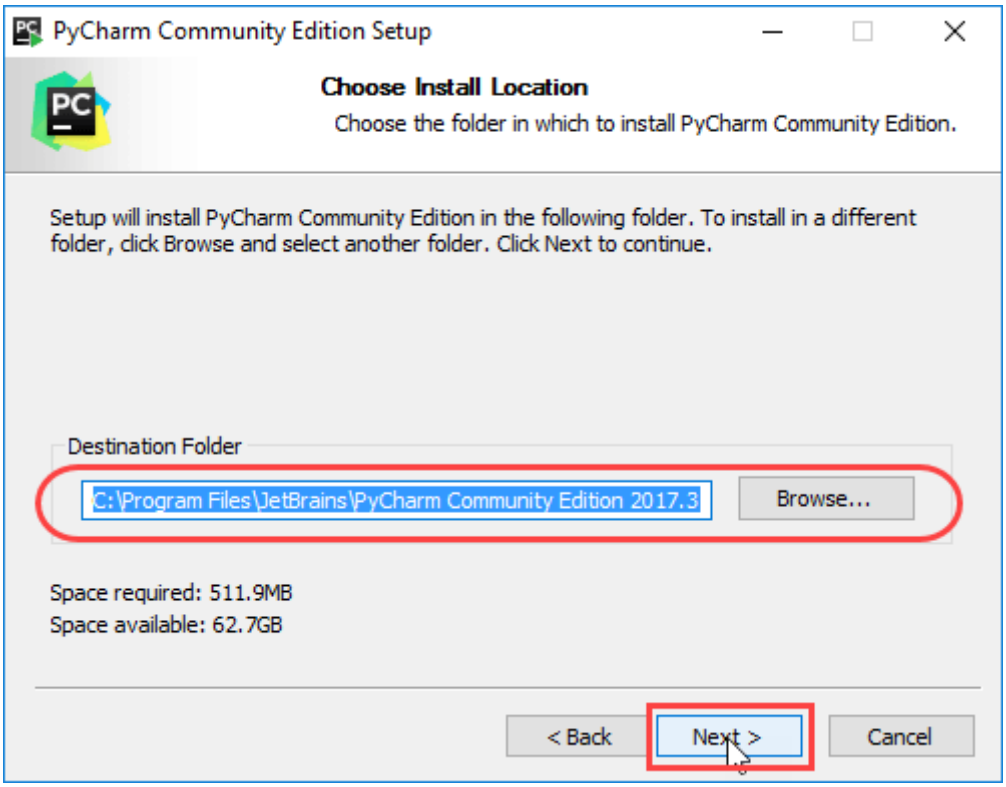
Hình 1.12. Trang dowload pycharm

Sau khi download thành công, PyCharm sẽ được lưu tại một thư mục Download của máy tính. Ta click đúp lên file bộ cài, để tiến hành cài đặt PyCharm. Khi quá trình cài đặt đã hoàn tất, chạy tập tin exe để cài đặt PyCharm. Chương trình cài đặt sẽ được khởi động. Click vào "Next":



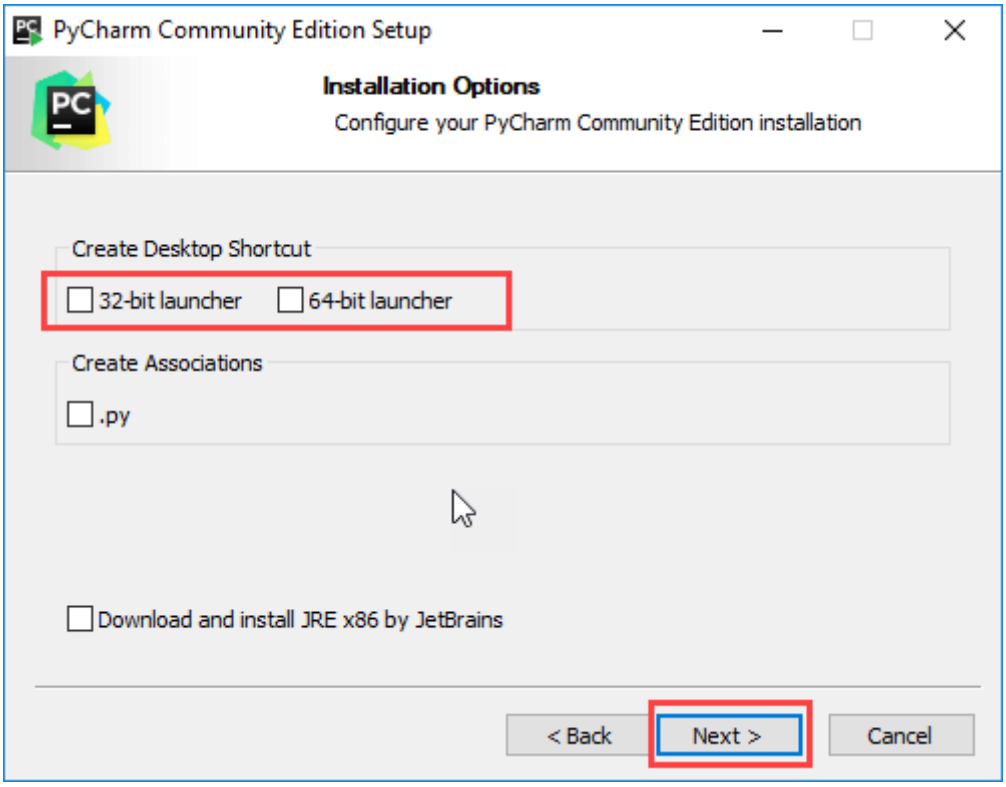
Hình 1. 13. Setup pycharm

Trên màn hình tiếp theo, thay đổi đường dẫn cài đặt nếu cần thiết. Sau đó click vào "Next":



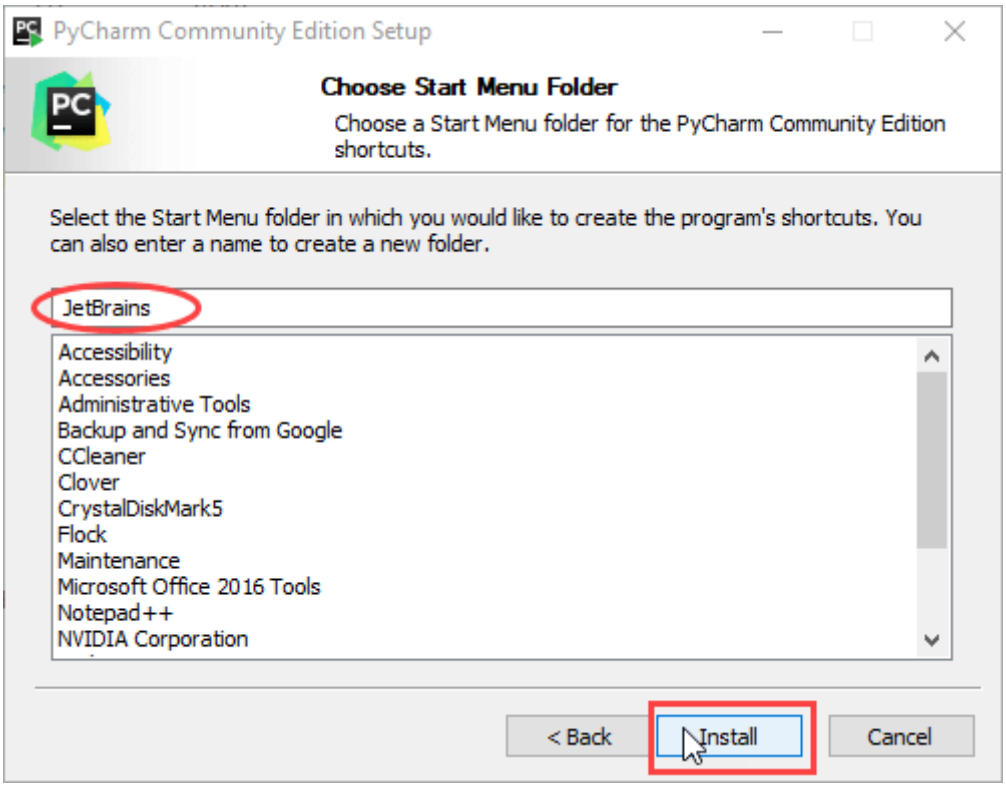
Hình 1. 14. Vị trí cài đặt

Trên màn hình tiếp theo, bạn có thể lựa chọn tạo một biểu tượng trên màn hình desktop nếu bạn muốn và sau đó click "Next":



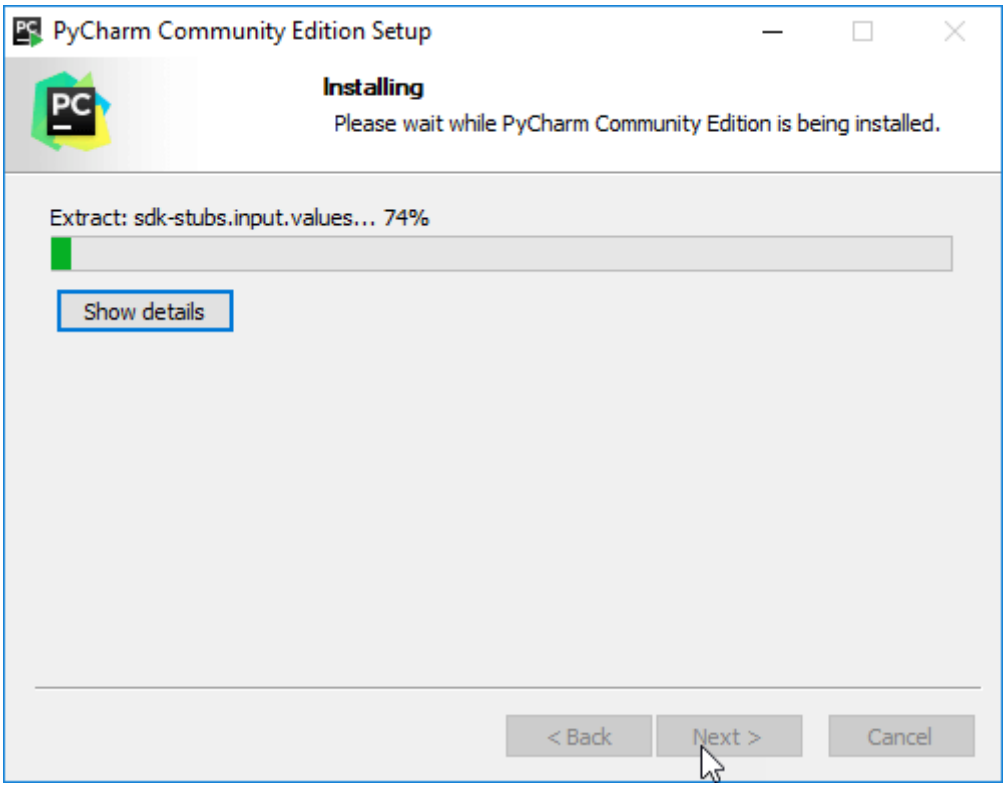
Hình 1. 15. Cài đặt Options

Lựa chọn thư mục Start Menu. Tiếp tục lựa chọn JetBrains sau đó click vào “Install”:



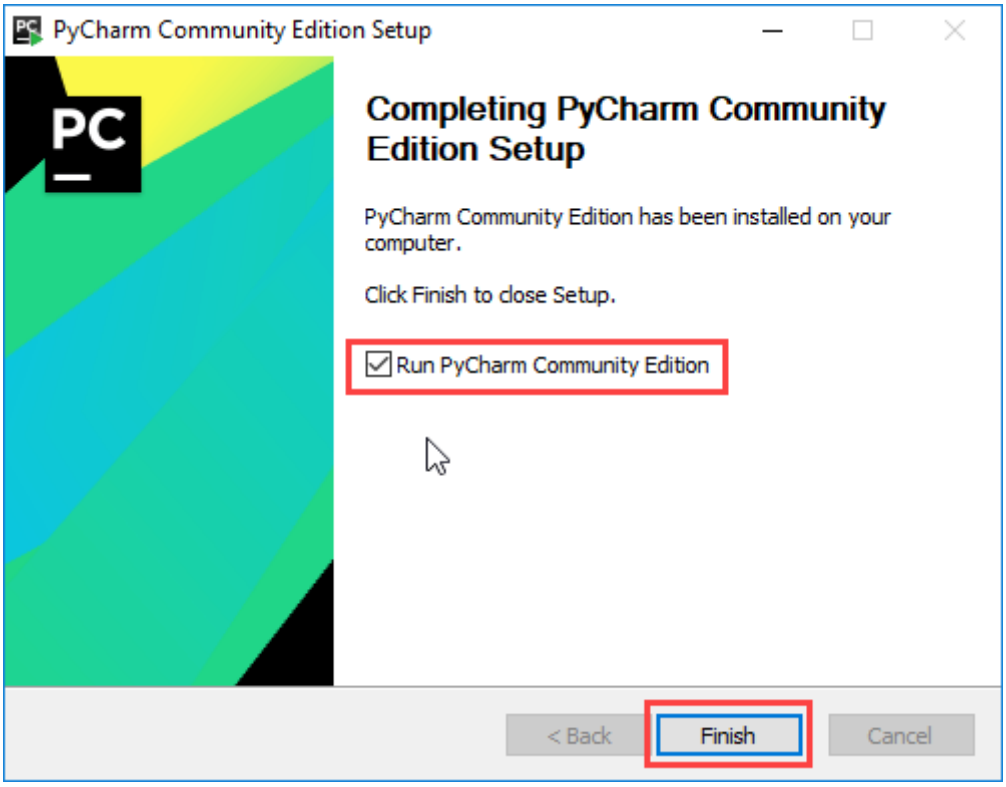
Hình 1.16. chọn thư mục Start menu

Chờ đợi cho tới khi quá trình cài đặt kết thúc.



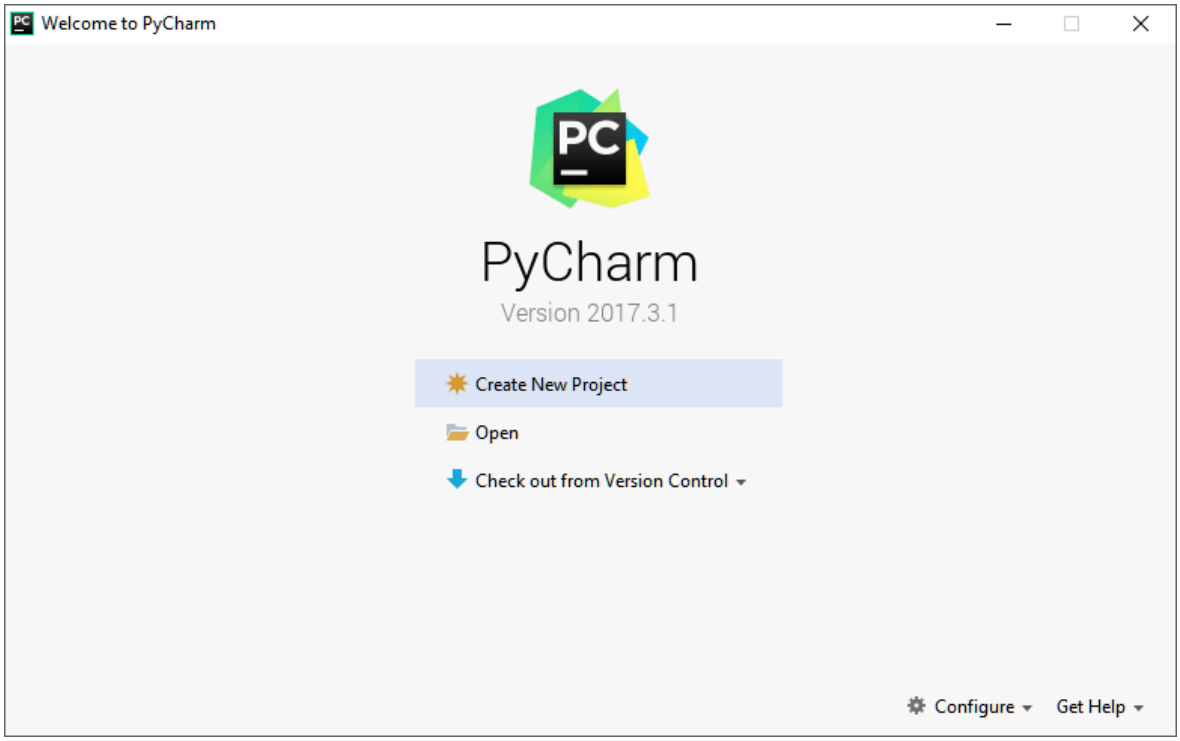
Hình 1. 17. Quá trình cài đặt pycharm

Khi quá trình cài đặt kết thúc, bạn sẽ nhận được thông báo trên màn hình rang PyCharm đã được cài đặt. Nếu bạn muốn tiếp tục và chạy thử nó, click vào ô "Run PyCharm Community Edition", sau đó click "Finish".



Hình 1. 18. Kết thúc cài đặt

Sau khi bạn click vào "Finish", màn hình sau sẽ hiện ra:



Hình 1. 19. Giao diện pycharm

CHƯƠNG 2. GIỚI THIỆU VỀ MẠNG NƠRON

2.1. Giới thiệu về Maching Learning

Những năm gần đây, AI - Artificial Intelligence (Trí Tuệ Nhân Tạo), và cụ thể hơn là Machine Learning (Học Máy hoặc Máy Học) nổi lên như một bằng chứng của cuộc cách mạng công nghiệp lần thứ tư (1 - động cơ hơi nước, 2 - năng lượng điện, 3 - công nghệ thông tin). Trí Tuệ Nhân Tạo đang len lỏi vào mọi lĩnh vực trong đời sống mà có thể chúng ta không nhận ra. Xe tự hành của Google và Tesla, hệ thống tự tag khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm của Amazon, hệ thống gợi ý phim của Netflix, máy chơi cờ vây AlphaGo của Google DeepMind, …, chỉ là một vài trong vô vàn những ứng dụng của AI/Machine Learning.

Machine Learning là một tập con của AI. Theo định nghĩa của Wikipedia, *Machine learning is the subfield of computer science that "gives computers the ability to learn without being explicitly programmed"*. Nói đơn giản, Machine Learning là một lĩnh vực nhỏ của Khoa Học Máy Tính, nó có khả năng tự học hỏi dựa trên dữ liệu đưa vào mà không cần phải được lập trình cụ thể. Vậy Machine Learning là gì?

Machine learning gây nên cơn sốt công nghệ trên toàn thế giới trong vài năm nay. Trong giới học thuật, mỗi năm có hàng ngàn bài báo khoa học về đề tài này. Trong giới công nghiệp, từ các công ty lớn như Google, Facebook, Microsoft đến các công ty khởi nghiệp đều đầu tư vào machine learning. Hàng loạt các ứng dụng sử dụng machine learning ra đời trên mọi linh vực của cuộc sống, từ khoa học máy tính đến những ngành ít liên quan hơn như vật lý, hóa học, y học, chính trị. *AlphaGo*, cỗ máy đánh cờ vây với khả năng tính toán trong một không gian có số lượng phần tử còn nhiều hơn số lượng hạt trong vũ trụ, tối ưu hơn bất kì đại kì thủ nào, là một trong rất nhiều ví dụ hùng hồn cho sự vượt trội của machine learning so với các phương pháp cổ điển

Để giới thiệu về machine learning, mình xin dựa vào mối quan hệ của nó với ba khái niệm sau:

- Machine learning và trí tuệ nhân tạo (Artificial Intelligence hay AI)
- Machine learning và Big Data.
- Machine learning và dự đoán tương lai.

Trí tuệ nhân tạo, AI, một cụm từ vừa gần gũi vừa xa lạ đối với chúng ta. Gần gũi bởi vì thế giới đang phát sốt với những công nghệ được dán nhãn AI. Xa lạ bởi vì một AI thực thụ vẫn còn nằm ngoài tầm với của chúng ta. Nói đến AI, hẳn mỗi người sẽ liên tưởng đến một hình ảnh khác nhau. Vài thập niên gần đây có một sự thay đổi về diện mạo của AI trong các bộ phim quốc

tế. Trước đây, các nhà sản xuất phim thường xuyên đưa hình ảnh robot vào phim (như *Terminator*), nhằm gieo vào đầu người xem suy nghĩ rằng trí tuệ nhân tạo là một phương thức nhân bản con người bằng máy móc. Tuy nhiên, trong những bộ phim gần hơn về đề tài này, ví dụ như *Transcendence* do Johny Depp vào vai chính, ta không thấy hình ảnh của một con robot nào cả. Thay vào đó là một bộ não điện toán khổng lồ chỉ huy hàng vạn con Nanobot, được gọi là Singularity. Tất nhiên cả hai hình ảnh đều là hư cấu và giả tưởng, nhưng sự thay đổi như vậy cũng một phần nào phản ánh sự thay đổi ý niệm của con người về AI. AI bây giờ được xem như vô hình vô dạng, hay nói cách khác có thể mang bất cứ hình dạng nào. Vì nói về AI là nói về một *bộ não*, chứ không phải nói về một cơ thể, là software chứ không phải là hardware.

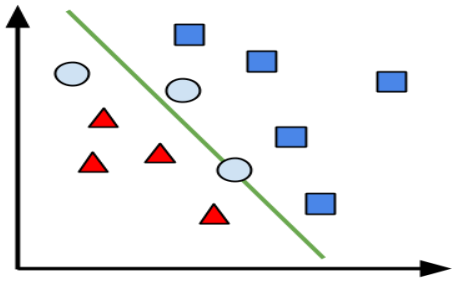
AI thể hiện một *mục tiêu* của con người. Machine learning là một *phương tiện* được kỳ vọng sẽ giúp con người đạt được mục tiêu đó. Và thực tế thì machine learning đã mang nhân loại đi rất xa trên quãng đường chinh phục AI. Nhưng vẫn còn một quãng đường xa hơn rất nhiều cần phải đi. Machine learning và AI có mối quan hệ chặt chẽ với nhau nhưng không hẳn là trùng khớp vì một bên là mục tiêu (AI), một bên là phương tiện (machine learning). Chinh phục AI mặc dù vẫn là mục đích tối thượng của machine learning, nhưng hiện tại machine learning tập trung vào những mục tiêu ngắn hạn hơn như: Làm cho máy tính có những khả năng nhận thức cơ bản của con người như nghe, nhìn, hiểu được ngôn ngữ, giải toán, lập trình, …Và Hỗ trợ con người trong việc xử lý một khối lượng thông tin khổng lồ mà chúng ta phải đối mặt hàng ngày, hay còn gọi là Big Data.

Big Data thực chất không phải là một ngành khoa học chính thống. Đó là một cụm từ dân gian và được giới truyền thông tung hô để ám chỉ thời kì bùng nổ của dữ liệu hiện nay. Nó cũng không khác gì với những cụm từ như "cách mạng công nghiệp", "kỉ nguyên phần mềm". Big Data là một hệ quả tất yếu của việc mạng Internet ngày càng có nhiều kết nối. Với sự ra đời của các mạng xã hội nhưng Facebook, Instagram, Twitter, nhu cầu chia sẻ thông của con người tăng trưởng một cách chóng mặt. Youtube cũng có thể được xem là một mạng xã hội, nơi mọi người chia sẻ video và comment về nội dung của video.

Bùng nổ thông tin không phải là lý do duy nhất dẫn đến sự ra đời của cụm từ Big Data. Nên nhớ rằng Big Data xuất hiện mới từ vài năm gần đây nhưng khối lượng dữ liệu tích tụ kể từ khi mạng Internet xuất hiện vào cuối thế kỉ trước cũng không phải là nhỏ. Thế nhưng, lúc ấy con người ngồi quanh một đống dữ liệu và không biết làm gì với chúng ngoài lưu trữ và sao chép. Cho đến một ngày, các nhà khoa học nhận ra rằng trong đống dữ liệu ấy thực ra chứa một khối lượng tri thức khổng lồ. Những tri thức ấy có thể giúp cho ta hiểu thêm về con người và xã hội. Từ danh sách bộ phim yêu thích của một cá nhân chúng ta có thể rút ra được sở thích của người đó và giới thiệu những bộ

phim người ấy chưa từng xem, nhưng phù hợp với sở thích. Từ danh sách tìm kiếm của cộng đồng mạng chúng ta sẽ biết được vấn đề nóng hổi nhất đang được quan tâm và sẽ tập trung đăng tải nhiều tin tức hơn về vấn đề đó. Big Data chỉ thực sự bắt đầu từ khi chúng ta hiểu được gía trị của thông tin ẩn chứa trong dữ liệu, và có đủ tài nguyên cũng như công nghệ để có thể khai thác chúng trên quy mô khổng lồ. Và không có gì ngạc nhiên khi machine learning chính là thành phần mấu chốt của công nghệ đó. Ở đây ta có một quan hệ hỗ tương giữa machine Learning và Big Data: machine learning phát triển hơn nhờ sự gia tăng của khối lượng dữ liệu của Big Data; ngược lại, gía trị của Big Data phụ thuộc vào khả năng khai thác tri thức từ dữ liệu của machine learning.

Ngược dòng lịch sử, machine learning đã xuất hiện từ rất lâu trước khi mạng Internet ra đời. Một trong những thuật toán machine learning đầu tiên là *thuật toán perceptron* được phát minh ra bởi Frank Rosenblatt vào năm 1957. Đây là một thuật toán kinh điển dùng để phân loại hai khái niệm. Một ví dụ đơn gỉan là phân loại thư rác (tam gíac) và thư bình thường (vuông). Chắc các bạn sẽ khó hình dung ra được làm thế nào để làm được điều đó. Đối với perceptron, điều này không khác gì với việc vẽ một đường thẳng trên mặt phẳng để phân chia hai tập điểm:

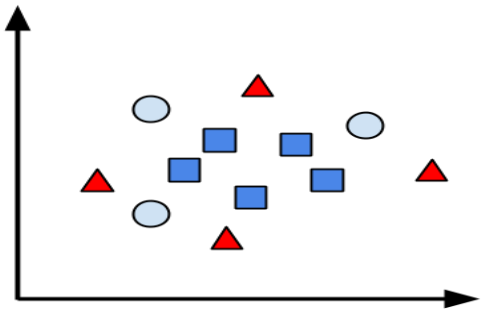


Hình2.1. Đường thẳng trên mặt phẳng để phân chia hai tập điểm

Những điểm tam giác và vuông đại diện cho những email chúng ta đã biết nhãn trước. Chúng được dùng để "huấn luyện" (train) perceptron. Sau khi vẽ đường thẳng chia hai tập điểm, ta nhận thêm các điểm chưa được dán nhãn, đại diện cho các email cần được phân loại (điểm tròn). Ta dán nhãn của một điểm theo nhãn của các điểm cùng nửa mặt phẳng với điểm đó.

Sơ lược quy trình phân loại thư được mô tả sau. Trước hết, ta cần một thuật toán để chuyển email thành những điểm dữ liệu. Công đoạn này rất rất quan trọng vì nếu chúng ta chọn được biểu diễn phù hợp, công việc của perceptron sẽ nhẹ nhàng hơn rất nhiều. Tiếp theo, perceptron sẽ đọc tọa độ của từng điểm và sử dụng thông tin này để cập nhật tham số của đường thẳng cần tìm. Các bạn có thể xem qua demo của perceptron (điểm xanh lá cây là điểm perceptron đang xử lý)

Vì là một thuật toán khá đơn giản, có rất nhiều vấn đề có thể nảy sinh với perceptron, ví dụ như điểm cần phân loại nằm ngay trên đường thẳng phân chia. Hoặc tệ hơn là với một tập dữ liệu phức tạp hơn, đường thẳng phân chia không tồn tại:



Hình 2.2 Đường thẳng phân chia không tồn tại

Lúc này, ta cần các loại đường phân chia "không thẳng". Nhưng đó lại là một câu chuyện khác.

Perceptron là một thuật toán supervised learning: ta đưa cho máy tính hàng loạt các ví dụ cùng câu trả lời mẫu với hy vọng máy tính sẽ tìm được những đặc điểm cần thiết để đưa ra dự đoán cho những ví dụ khác chưa có câu trả lời trong tương lai. Ngoài ra, cũng có những thuật toán machine learning không cần câu trả lời mẫu, được gọi là unsupervised learning. Trong trường hợp này, máy tính cố gắng khai thác ra cấu trúc ẩn của một tập dữ liệu mà không cần câu trả lời mẫu. Một loại machine learning khác được gọi là reinforcement learning. Trong dạng này, cũng không hề có câu trả lời mẫu, nhưng thay vì đó máy tính nhận được phản hồi cho mỗi hành động. Dựa vào phản hồi tích cực hay tiêu cực mà máy tính sẽ điều chỉnh hoạt động cho phù hợp. Sau đây là một ví dụ minh họa

Mục tiêu của chiếc xe là leo lên được đỉnh đồi và lấy được ngôi sao. Chiếc xe có hai chuyển động tới và lui. Bằng cách thử các chuyển động và nhận được phản hồi là độ cao đạt được và thời gian để lấy được ngôi sao, chiếc xe dần trở nên thuần thục hơn trong việc leo đồi lấy sao.

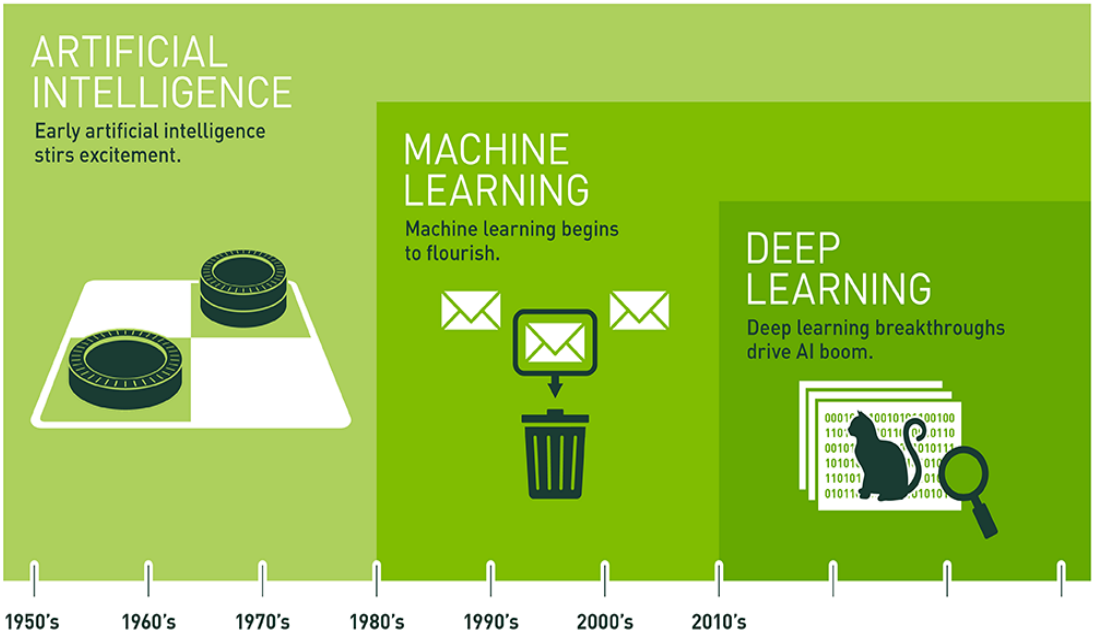
Machine learning có mối quan hệ rất mật thiết đối với statistics (thống kê). Machine learning sử dụng các mô hình thống kê để "ghi nhớ" lại sự phân bố của dữ liệu. Tuy nhiên, không đơn thuần là ghi nhớ, machine learning phải có khả năng tổng quát hóa những gì đã được nhìn thấy và đưa ra dự đoán cho những trường hợp chưa được nhìn thấy. Bạn có thể hình dung một mô hình machine learning không có khả năng tổng quát như một đứa trẻ học vẹt: chỉ trả lời được những câu trả lời mà nó đã học thuộc lòng đáp án. Khả năng tổng quát là một khả năng tự nhiên và kì diệu của con người: bạn không thể nhìn thấy tất cả các khuôn mặt người trên thế giới nhưng bạn có thể nhận biết được

một thứ có phải là khuôn mặt người hay không với xác suất đúng gần như tuyệt đối. Đỉnh cao của machine learning sẽ là mô phỏng được khả năng tổng quát hóa và suy luận này của con người.

Như ta đã thấy, nói đến machine learning là nói đến "dự đoán": từ việc dự đoán nhãn phân loại đến dự đoán hành động cần thực hiện trong bước tiếp theo. Vậy machine learning có thể dự đoán tương lai hay không? Có thể có hoặc có thể không: machine learning có thể dự đoán được tương lai, nhưng chỉ khi tương lai có mối liên hệ mật thiết với hiện tại.

Để kết thúc, mình muốn cùng các bạn xem xét một ví dụ đơn giản sau. Giả sử bạn được đưa cho một đồng xu, rồi được yêu cầu tung đồng xu một số lần. Vấn đề đặt ra là: dựa vào những lần tung đồng xu đó, bạn hãy tiên đoán ra kết quả lần tung tiếp theo. Chỉ cần dựa vào tỉ lệ sấp/ngửa của những lần tung trước đó, bạn có thể đưa ra một dự đoán khá tốt. Nhưng nếu mỗi lần tung, người ta đưa cho bạn một đồng xu khác nhau thì mọi chuyện sẽ hoàn toàn khác. Các đồng xu khác nhau có xác suất sấp ngửa khác nhau. Lúc này việc dự đoán gần như không thể vì xác suất sấp ngửa của lần tung sau không hề liên quan gì đến lần tung trước. Điều tương tự cũng xảy ra với việc dự đoán tương lai bằng machine learning, nếu ta xem như mỗi ngày có một "đồng xu" được tung ra để xem một sự kiện có diễn ra hay không. Nếu "đồng xu" của ngày mai được chọn một cách tùy ý không theo phân bố nào cả thì machine learning sẽ thất bại. Rất may là trong nhiều trường hợp điều đó không hoàn toàn đúng, thế giới hoạt động theo những quy luật nhất định và machine learning có thể nhận ra những quy luật đó. Nhưng nói cho cùng, machine learning hoàn toàn không phải là một bà phù thủy với quả cầu tiên tri mà cũng giống như chúng ta: phán đoán bằng cách tổng quát hóa những kinh nghiệm, những gì đã được học từ dữ liệu.

Những năm gần đây, khi mà khả năng tính toán của các máy tính được nâng lên một tầm cao mới và lượng dữ liệu khổng lồ được thu thập bởi các hãng công nghệ lớn, Machine Learning đã tiến thêm một bước dài và một lĩnh vực mới được ra đời gọi là Deep Learning (Học Sâu - *thực sự tôi không muốn dịch từ này ra tiếng Việt*). Deep Learning đã giúp máy tính thực thi những việc tưởng chừng như không thể vào 10 năm trước: phân loại cả ngàn vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn hay âm nhạc



Hình 2.3. Lịch sử phát triển của Maching Learning

2.2. *Lịch sử phát triển mạng nơron nhân tạo- ANN*

Các nghiên cứu về bộ não con người đã được tiến hành từ hàng nghìn năm nay. Cùng với sự phát triển của khoa học kĩ thuật đặc biệt là những tiến bộ trong ngành điện tử hiện đại, việc con người bắt đầu nghiên cứu các nơron nhân tạo là hoàn toàn tự nhiên. Có thể tính từ nghiên cứu của William (1890) về tâm lý học với sự liên kết các noron thần kinh. Sự kiện đầu tiên đánh dấu sự ra đời của mạng nơron nhân tạo diễn ra vào năm 1943 khi nhà thần kinh học Warren McCulloch và nhà toán học Walter Pitts viết bài báo mô tả cách thức các nơron hoạt động. Họ cũng đã tiến hành xây dựng một mạng nơron đơn giản bằng các mạch điện. Các nơron của họ được xem như là các thiết bị nhị phân với ngưỡng cố định. Kết quả của các mô hình này là các hàm logic đơn giản chẳng hạn như " a OR b" hay "a AND b" . Những tiến bộ của máy tính đầu những năm 1950 giúp cho việc mô hình hóa các nguyên lý của những lý thuyết liên quan tới cách thức con người suy nghĩ đã trở thành hiện thực. Nathanial Rochester sau nhiều năm làm việc tại các phòng thí nghiệm nghiên cứu của IBM đã có những nỗ lực đầu tiên để mô phỏng một mạng nơron.

Năm 1956 dự án Dartmouth nghiên cứu về trí tuệ nhân tạo (Artificial Intelligence) đã mở ra thời kỳ phát triển mới cả trong lĩnh vực trí tuệ nhân tạo lẫn mạng nơron. Tác động tích cực của nó là thúc đẩy hơn nữa sự quan tâm của các nhà khoa học về trí tuệ nhân tạo và quá trình xử lý ở mức đơn giản của mạng nơron trong bộ não con người.

Những năm tiếp theo của dự án Dartmouth, John von Neumann đã đề xuất việc mô phỏng các nơron đơn giản bằng cách sử dụng rơle điện áp hoặc đèn chân không. Nhà sinh học chuyên nghiên cứu về nơron Frank Rosenblatt cũng bắt đầu nghiên cứu về Perceptron năm 1958. Sau thời gian nghiên cứu này Perceptron đã được cài đặt trong phần cứng máy tính và được xem như là mạng nơron lâu đời nhất còn được sử dụng đến ngày nay. Perceptron một tầng rất hữu ích trong việc phân loại một tập các đầu vào có giá trị liên tục vào một trong hai lớp. Perceptron tính tổng có trọng số các đầu vào, rồi trừ tổng này cho một ngưỡng và cho ra một trong hai giá trị mong muốn có thể. Tuy nhiên Perceptron còn rất nhiều hạn chế, những hạn chế này đã được chỉ ra trong cuốn sách về Perceptron của Marvin Minsky và Seymour Papert của MIT (Massachurehs Insritute of Technology) viết năm 1969 đã chứng minh nó không dùng được cho các hàm logic phức.

Năm 1959, Bernard Widrow và Marcian Hoff thuộc trường đại học Stanford đã xây dựng mô hình ADALINE (ADAptive LINear Elements) và MADALINE. (Multiple ADAptive LINear Elements). Các mô hình này sử dụng quy tắc học Least-Mean-Squares (LMS : Tối thiểu bình phương trung bình). MADALINE là mạng nơron đầu tiên được áp dụng để giải quyết một bài toán thực tế. Nó là một bộ lọc thích ứng có khả năng loại bỏ tín hiệu dội lại trên đường dây điện thoại. Ngày nay mạng nơron này vẫn được sử dụng trong các ứng dụng thương mại. Năm 1973 Von der Marlsburg: đưa ra quá trình học cạnh tranh và self – organization.

Năm 1974 Paul Werbos đã phát triển và ứng dụng phương pháp học lan truyền ngược (back-propagation). Tuy nhiên phải mất một vài năm thì phương pháp này mới trở lên phổ biến. Các mạng lan truyền ngược được biết đến nhiều nhất và được áp dụng rộng dãi nhất nhất cho đến ngày nay.

Năm 1985, viện vật lý Hoa Kỳ bắt đầu tổ chức các cuộc họp hàng năm về mạng nơron ứng dụng trong tin học (Nơron Networks for Computing).

Năm 1987, hội thảo quốc tế đầu tiên về mạng neuron của Viện các kỹ sư điện và điện tử IEEE (Institute of Electrical and Electronic Engineer) đã thu hút hơn 1800 người tham gia. Tính từ năm 1987 đến nay, hàng năm thế giới đều mở hội nghị toàn cầu chuyên ngành nơron IJCNN (International Joit Conference on Nơron Networks).

Ngày nay, không chỉ dừng lại ở mức nghiên cứu lý thuyết, các nghiên cứu ứng dụng mạng nơron để giải quyết các bài toán thực tế được diễn ra ở khắp mọi nơi. Các ứng dụng mạng nơron ra đời ngày càng nhiều và ngày càng hoàn thiện hơn. Điển hình là các ứng dụng: xử lý ngôn ngữ (Language Processing), nhận dạng ký tự (Character Recognition), nhận dạng tiếng nói

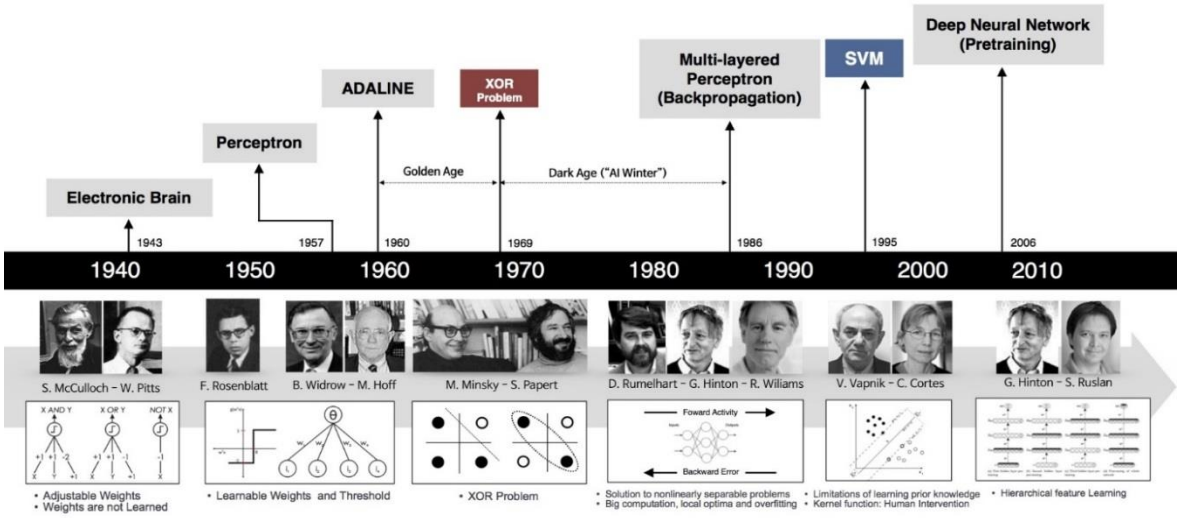
(Voice Recognition), nhận dạng mẫu (Pattern Recognition), xử lý tín hiệu (Signal Processing), Lọc dữ liệu (Data Filtering),…

2.3. Lịch sử phát triển Deep Learing

Trí tuệ nhân tạo đang len lỏi vào trong cuộc sống và ảnh hưởng sâu rộng tới mỗi chúng ta. Kể từ khi tôi viết bài đầu tiên, tần suất chúng ta nghe thấy các cụm từ ‘artificial intelligence’, ‘machine learning’, ‘deep learning’ cũng ngày một tăng lên. Nguyên nhân chính dẫn đến việc này (và việc ra đời blog này) là sự xuất hiện của deep learning trong 5-6 năm gần đây.

Deep learning được nhắc đến nhiều trong những năm gần đây, nhưng những nền tảng cơ bản đã xuất hiện từ rất lâu …

Chúng ta cùng quan sát hình dưới đây:



Hình 2.4. Lịch sử phát triển Deep Learning

- Perceptron (60s)

Một trong những nền móng đầu tiên của neural network và deep learning là perceptron learning algorithm(hoặc gọn là perceptron). Perceptron là một thuật toán supervised learning giúp giải quyết bài toán phân lớp nhị phân, được khởi nguồn bởi Frank Rosenblatt năm 1957 trong một nghiên cứu được tài trợ bởi Văn phòng nghiên cứu hải quân Hoa Kỳ (U.S Office of Naval Research – *từ một cơ quan liên quan đến quân sự*). Thuật toán perceptron được chứng minh là hội tụ nếu hai lớp dữ liệu là *linearly separable*. Với thành công này, năm 1958, trong một hội thảo, Rosenblatt đã có một phát biểu gây tranh cãi. Từ phát biểu này, tờ New York Times đã có một bài báo cho rằng perceptron được Hải quân Hoa Kỳ mong đợi “có thể đi, nói chuyện, nhìn, viết, tự sinh sản, và tự nhận thức được sự tồn tại của mình”. (*Chúng ta biết rằng cho tới giờ các hệ thống nâng cao hơn perceptron nhiều lần vẫn chưa thể*).

Mặc dù thuật toán này mang lại nhiều kỳ vọng, nó nhanh chóng được chứng minh không thể giải quyết những bài toán đơn giản. Năm 1969, Marvin Minsky và Seymour Papert trong cuốn sách nổi tiếng Perceptrons đã chứng minh rằng không thể 'học' được hàm số XOR khi sử dụng perceptron. Phát hiện này làm choáng váng giới khoa học thời gian đó . Perceptron được chứng minh rằng chỉ hoạt động nếu dữ liệu là *linearly separable*.

Phát hiện này khiến cho các nghiên cứu về perceptron bị gián đoạn gần 20 năm. Thời kỳ này còn được gọi là ***Mùa đông AI thứ nhất (The First AI winter)***. Cho tới khi…

- MLP và Backpropagation ra đời (80s)

Geoffrey Hinton tốt nghiệp PhD ngành neural networks năm 1978. Năm 1986, ông cùng với hai tác giả khác xuất bản một bài báo khoa học trên Nature với tựa đề "Learning representations by back-propagating errors". Trong bài báo này, nhóm của ông chứng minh rằng neural nets với nhiều hidden layer (được gọi là multi-layer perceptron hoặc MLP) có thể được huấn luyện một cách hiệu quả dựa trên một quy trình đơn giản được gọi là **backpropagation**. Việc này giúp neural nets *thoát*được những hạn chế của perceptron về việc chỉ biểu diễn được các quan hệ tuyến tính. Để biểu diễn các quan hệ phi tuyến, phía sau mỗi layer là một hàm kích hoạt phi tuyến, ví dụ hàm sigmoid hoặc tanh. (ReLU ra đời năm 2012). Với hidden layers, neural nets được chứng minh rằng có khả năng xấp xỉ hầu hết bất kỳ hàm số nào qua một định lý được gọi là universal approximation theorem.

Thuật toán này mang lại một vài thành công ban đầu, nổi trội là **convolutional neural nets** (convnets hay CNN) (còn được gọi là LeNet) cho bài toán nhận dạng chữ số viết tay được khởi nguồn bởi Yann LeCun tại AT&T Bell Labs (Yann LeCun là sinh viên sau cao học của Hinton tại đại học Toronto năm 1987-1988). Dưới đây là bản demo được lấy từ trang web của LeNet, network là một CNN với 5 layer, còn được gọi là LeNet-5 (1998).

- Mùa đông AI thứ hai (90s - đầu 2000s)

Các mô hình tương tự được kỳ vọng sẽ giải quyết nhiều bài toán image classification khác. Tuy nhiên, không như các chữ số, các loại ảnh khác lại rất hạn chế vì máy ảnh số chưa phổ biến tại thời điểm đó. Ảnh được gán nhãn lại càng hiếm. Trong khi để có thể huấn luyện được mô hình convnets, ta cần rất nhiều dữ liệu huấn luyện. Ngay cả khi dữ liệu có đủ, một vấn đề nan giải khác là khả năng tính toán của các máy tính thời đó còn rất hạn chế.

Một hạn chế khác của các kiến trúc MLP nói chung là hàm mất mát không phải là một hàm lồi. Việc này khiến cho việc tìm nghiệm tối ưu toàn cục cho bài toán tối ưu hàm mất mát trở nên rất khó khăn. Một vấn đề khác liên quan đến giới hạn tính toán của máy tính cũng khiến cho việc huấn luyện

MLP không hiệu quả khi số lượng hidden layers lớn lên. Vấn đề này có tên là **vanishing gradient**.

Nhắc lại rằng hàm kích hoạt được sử dụng thời gian đó là sigmoid hoặc tanh – là các hàm bị chặn trong khoảng (0, 1) hoặc (-1, 1) (Nhắc lại đạo hàm của hàm sigmoid $\sigma(z)\sigma(z)$ là $\sigma(z)(1-\sigma(z))\sigma(z)(1-\sigma(z))$ là tích của hai số nhỏ hơn 1). Khi sử dụng backpropagation để tính đạo hàm cho các ma trận hệ số ở các lớp đầu tiên, ta cần phải nhân rất nhiều các giá trị nhỏ hơn 1 với nhau. Việc này khiến cho nhiều đạo hàm thành phần bằng 0 do xấp xỉ tính toán. Khi đạo hàm của một thành phần bằng 0, nó sẽ không được cập nhật thông qua gradient descent!

Những hạn chế này khiến cho neural nets một lần nữa rơi vào thời kỳ *băng giá*. Vào thời điểm những năm 1990 và đầu những năm 2000, neural nets dần được thay thế bởi support vector machines –SVM. SVMs có ưu điểm là bài toán tối ưu để tìm các tham số của nó là một bài toán lồi – có nhiều các thuật toán tối ưu hiệu quả giúp tìm nghiệm của nó. Các kỹ thuật về kernel cũng phát triển giúp SVMs giải quyết được cả các vấn đề về việc dữ liệu không phân biệt tuyến tính.

Nhiều nhà khoa học làm machine learning chuyển sang nghiên cứu SVM trong thời gian đó, trừ một vài nhà khoa học cứng đầu…

- Cái tên được làm mới – Deep Learning (2006)

Năm 2006, Hinton một lần nữa cho rằng ông biết bộ não hoạt động như thế nào, và giới thiệu ý tưởng của *tiền huấn luyện không giám sát* (*unsupervised pretraining*) thông qua deep belief nets (DBN). DBN có thể được xem như sự xếp chồng các unsupervised networks đơn giản như restricted Boltzman machinehay autoencoders.

Lấy ví dụ với autoencoder. Mỗi autoencoder là một neural net với một hidden layer. Số hidden unit ít hơn số input unit, và số output unit bằng với số input unit. Network này đơn giản được huấn luyện để kết quả ở output layer giống với kết quả ở input layer (và vì vậy được gọi là autoencoder). Quá trình dữ liệu đi từ input layer tới hidden layer có thể coi là *mã hoá*, quá trình dữ liệu đi từ hidden layer ra output layer có thể được coi là *giải mã*. Khi output giống với input, ta có thể thấy rằng hidden layer với ít unit hơn có để mã hoá input khá thành công, và có thể được coi mang những tính chất của input. Nếu ta bỏ output layer, *cố định* (*freeze*) kết nối giữa input và hidden layer, coi đầu ra của hidden layer là một input mới, sau đó huấn luyện một autoencoder khác, ta được thêm một hidden layer nữa. Quá trình này tiếp tục kéo dài ta sẽ được một network đủ *sâu* mà output của network lớn này (chính là hidden layer của autoencoder cuối cùng) mang thông tin của input ban đầu. Sau đó ta có thể thêm các layer khác tuỳ thuộc vào bài toán (chẳng hạn thêm softmax layer ở

cuối cho bài toán classification). Cả network được huấn luyện thêm một vài epoch nữa. Quá trình này được gọi là *tinh chỉnh* (*fine tuining*).

Tại sao quá trình huấn luyện như trên mang lại nhiều lợi ích?

Một trong những hạn chế đã đề cập của MLP là vấn đề *vanishing gradient*. Những ma trận trọng số ứng với các layer đầu của network rất khó được huấn luyện vì đạo hàm của hàm mất mát theo các ma trận này nhỏ. Với ý tưởng của DBN, các ma trận trọng số ở những hidden layer đầu tiên được *tiền huấn luyện*(*pretrained*). Các trọng số được tiền huấn luyện này có thể coi là giá trị khởi tạo tốt cho các hidden layer phía đầu. Việc này giúp phần nào tránh được sự phiền hà của *vanishing gradient*.

Kể từ đây, neural networks với nhiều hidden layer được đổi tên thành **deep learning**.

Vấn đề *vanishing gradient* được giải quyết phần nào (vẫn chưa thực sự triệt để), nhưng vẫn còn những vấn đề khác của deep learning: dữ liệu huấn luyện quá ít, và khả năng tính toán của CPU còn rất hạn chế trong việc huấn luyện các deep networks.

Năm 2010, giáo sư Fei-Fei Li, một giáo sư ngành computer vision đầu ngành tại Stanford, cùng với nhóm của bà tạo ra một cơ sở dữ liệu có tên ImageNet với hàng triệu bức ảnh thuộc 1000 lớp dữ liệu khác nhau đã được gán nhãn. Dự án này được thực hiện nhờ vào sự bùng nổ của internet những năm 2000 và lượng ảnh khổng lồ được upload lên internet thời gian đó.

Bộ cơ sở dữ liệu này được cập nhật hàng năm, và kể từ năm 2010, nó được dùng trong một cuộc thi thường niên có tên ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Trong cuộc thi này, dữ liệu huấn luyện được giao cho các đội tham gia. Mỗi đội cần sử dụng dữ liệu này để huấn luyện các mô hình phân lớp, các mô hình này sẽ được áp dụng để dự đoán nhãn của dữ liệu mới (được giữ bởi ban tổ chức). Trong hai năm 2010 và 2011, có rất nhiều đội tham gia. Các mô hình trong hai năm này chủ yếu là sự kết hợp của SVM với các feature được xây dựng bởi các bộ *hand-crafted descriptors* (SIFT, HoG, v.v.). Mô hình giành chiến thắng có top-5 error rate là 28% (càng nhỏ càng tốt). Mô hình giành chiến thắng năm 2011 có top-5 error rate là 26%. Cải thiện không nhiều!

- Đột phá (2012)

Năm 2012, cũng tại ILSVRC, Alex Krizhevsky, Ilya Sutskever, và Geoffrey Hinton (lại là ông) tham gia và đạt kết quả top-5 error rate 16%. Kết quả này làm sững sờ giới nghiên cứu thời gian đó. Mô hình là một Deep Convolutional Neural Network, sau này được gọi là AlexNet.

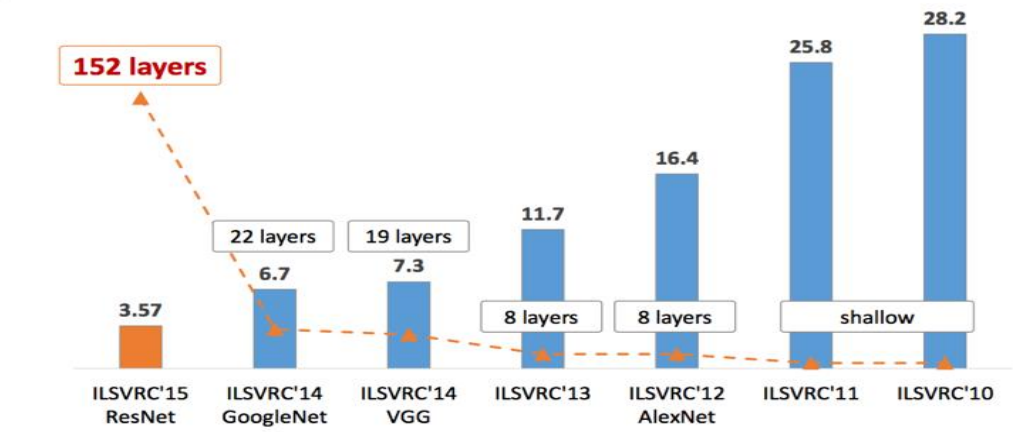
Trong bài báo này, rất nhiều các kỹ thuật mới được giới thiệu. Trong đó hai đóng góp nổi bật nhất là hàm ReLU và dropout. Hàm ReLU

(ReLU(x)=max(x,0)ReLU(x)=max(x,0)) với cách tính và đạo hàm đơn giản (bằng 1 khi đầu vào không âm, bằng 0 khi ngược lại) giúp tốc độ huấn luyện tăng lên đáng kể. Ngoài ra, việc ReLU không bị chặn trên bởi 1 (như softmax hay tanh) khiến cho vấn đề vanishing gradient cũng được giải quyết phần nào. Dropout cũng là một kỹ thuật đơn giản và cực kỳ hiệu quả. Trong quá trình training, nhiều hidden unit bị *tắt* ngẫu nhiên và mô hình được huấn luyện trên các bộ tham số còn lại. Trong quá trình test, toàn bộ các unit sẽ được sử dụng. Cách làm này khá là có lý khi đối chiếu với con người. Nếu chỉ dùng một phần năng lực đã đem lại hiệu quả thì dùng toàn bộ năng lực sẽ mang lại hiệu quả cao hơn. Việc này cũng giúp cho mô hình tránh được overfitting và cũng được coi giống với kỹ thuật *ensemble*trong các hệ thống machine learning khác. Với mỗi cách *tắt* các unit, ta có một mô hình khác nhau. Với nhiều tổ hợp unit bị tắt khác nhau, ta thu được nhiều mô hình. Việc kết hợp ở cuối cùng được coi như sự kết hợp của nhiều mô hình (và vì vậy, nó giống với *ensemble learning*).

Một trong những yếu tố quan trọng nhất giúp AlexNet thành công là việc sử dụng GPU (card đồ hoạ) để huấn luyện mô hình. GPU được tạo ra cho game thủ, với khả năng chạy song song nhiều lõi, đã trở thành một công cụ cực kỳ phù hợp với các thuật toán deep learning, giúp tăng tốc thuật toán lên nhiều lần so với CPU.

Sau AlexNet, tất cả các mô hình giành giải cao trong các năm tiếp theo đều là các deep networks (ZFNet 2013, GoogLeNet 2014, VGG 2014, ResNet 2015). Tôi sẽ giành một bài của blog để viết về các kiến trúc quan trọng này. Xu thế chung có thể thấy là các mô hình càng ngày càng *deep*. Xem hình dưới đây.

Những công ty công nghệ lớn cũng để ý tới việc phát triển các phòng nghiên cứu deep learning trong thời gian này. Rất nhiều các ứng dụng công nghệ đột phá đã được áp dụng vào cuộc sống hàng ngày. Cũng kể từ năm 2012, số lượng các bài báo khoa học về deep learning tăng lên theo hàm số mũ. Các blog về deep learning cũng tăng lên từng ngày.



Hình2.5.Kết quả ILSVRC qua các năm. (Nguồn: CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more ...)

2.4. Một số thư viện Deep Learning nổi tiếng Hiện nay

Cùng với sự phát triển của các thuật toán Deep Learning thì các thư viện cũng như framework hỗ trợ các thuật toán này cũng ngày càng tăng về số lượng. Hầu hết các thư viện và framework này đều cung cấp dưới dạng mã nguồn mở do đó rất linh hoạt trong việc sử dụng và mở rộng, đây cũng là một trong những lý do DL được áp dụng trong nhiều bài toán với nhiều lĩnh vực khác nhau. Trong phần tiếp theo nội dung post này sẽ giới thiệu một số thư viện phổ biến đang được cộng đồng nghiên cứu sử dụng.

Hình 2.6. Một số thư viện Deep learning nổi tiếng

- TensorFlow. (Commits: 16785, Contributors: 795)

Do các developer của Google phát triển, TensorFlow là thư viện nguồn mở của graphs computations thuộc luồng dữ liệu, thích hợp với Machine Learning. TensorFlow đáp ứng các requirement cao cấp trong môi trường Google để train Neural Networks và thư viện kế nhiệm của DistBelief – 1 hệ thống Machine Learning dựa trên Neural Networks. Tuy nhiên, TensorFlow không chỉ sử dụng cho mục đích khoa học trong Google mà có thể áp dụng trong các dự án thực tế.

Tính năng quan trọng của TensorFlow is hệ thống nút đa layer, cho phép huấn luyện các neural networks trên datasets lớn 1 cách nhanh chóng, hỗ trợ khả năng nhận diện giọng nói và định vị vật thể trong ảnh của Google.

Một số tính năng nổi bật của Torch framework:

+ TensorFlow hỗ trợ cả hai ngôn ngữ c và python.
+ TensorFlow có thể chạy trên nhiều CPU cũng như GPU giúp đẩy nhanh quá trình huấn luyện cũng như xử lý dữ liệu thực từ mô mình đã được học. Ngoài ra với việc có thể sử dụng thư viện này trên các hệ thống cloud sẽ làm đẩy nhanh hiệu năng của các hệ thống sử dụng TensorFlow.

+ Với khả năng chạy trên nhiều hệ điều hành như bao gồm cả iOS, Android, hứa hẹn sẽ phát triển được các ứng dụng thông minh nhờ áp dụng các tính năng nổi bật của DL.

- Theano. (Commits: 25870, Contributors: 300)

Theano là package Python định dạng các arrays đa chiều tương tự như NumPy, đi kèm với các operation về toán và expressions. Thư viện này được compiled, chạy hiệu quả trên tất cả các architectures. Do đội ngũ Machine Learning của Université de Montréal, Theano được sử dụng chính cho các hoạt động liên quan đến Machine Learning.

Lưu ý là Theano tích hợp với NumPy ở mức độ operation cấp thấp. Thư viện này cũng tối ưu hóa khả năng sử dụng GPU & CPU, giúp cho hiệu năng của computation thiên về data nhanh chóng hơn.

Hiệu quả và sự ổn định cũng mang đến những kết quả chính xác hơn, dù đó là những giá trị rất nhỏ như computation của log(1+x) sẽ cho ra kết quả chính xác đối với các giá trị nhỏ nhất của x.

Theano sử dụng 2 gói thư viện để hỗ trợ cho việc định nghĩa mô hình mạng neural:

+ **Lasagne**: Là thư viện định nghĩa các lớp (trừ lớp cuối cùng) của mô hình mạng neural. Lasagne giúp người dùng lưu trữ dữ liệu trong mạng, tính toán giá trị hàm lỗi, cập nhật trọng số.
+ **Keras**: Là lớp cuối cùng trong cấu trúc mạng. Hỗ trợ cài đặt các hàm kích hoạt và định nghĩa lớp softmax.

- Keras. (Commits: 3519, Contributors: 428)

Keras là thư viện nguồn mở được viết bằng Python dùng để build các Neural Networks ở cấp độ cao cấp của interface. Thư viện này đơn giản và có khả năng mở rộng cao. Keras sử dụng backend là Theano hoặc TensorFlow nhưng gần đây, Microsoft đã cố gắng tích hợp CNTK (Cognitive Toolkit của Microsoft) thành back-end mới.

Cách tiếp cận đơn giản về thiết kế nhắm đến quy trình experimentation dễ dàng, nhanh chóng từ việc build các compact systems.

Bắt đầu dùng Keras rất đơn giản, tiếp theo là prototyping nhanh. Keras được viết bằng Python, theo mô-đun và mở rộng được. Không chỉ đơn giản và có tính định hướng cao, Keras vẫn hỗ trợ modeling rất mạnh mẽ.

Ý tưởng chung về Keras là dựa trên các layers và mọi thứ khác cũng đều được xây dựng xung quanh các layer này. Data được chuẩn bị trong các tensors, layer đầu tiên chịu trách nhiệm về input của các tensors, layer cuối cùng chịu trách nhiệm output và model được build ở giữa.

- SciKit-Learn (Commits: 21793, Contributors: 842)

Scikits là các packages bổ sung của SciPy Stack được thiết kế cho các chức năng chuyên biêt như xử lý ảnh và hỗ trợ Machine Learning. Riêng với mảng Machine Learning, một trong những ưu điểm nổi bật của các packages này là scikit-learn. Package được xây dựng trên nền tảng của SciPy và tận dụng các operations về toán.

Scikit-learn có giao diện đơn giản, nhất quán, exposes a concise and consistent interface to the common machine learning algorithms, hỗ trợ việc mang Machine Learning vào các hệ thống production trở nên đơn giản hơn. Thư viện này bao gồm các code chất lượng và documentation hay, dễ sử dụng, hiệu suất cao, là chuẩn mực thực tế cho xây dựng Machine Learning bằng Python.

- Matplotlib (Commits: 21754, Contributors: 588)

Một core package của SciPy Stack và 1 thư viện Python khác được xây dựng riêng cho việc generation các visualizations mạnh mẽ, đơn giản là Matplotlib. Matplotlib là 1 phần của phần mềm giúp cho Python (cùng với sự hỗ trợ của NumPy, SciPy và Pandas) trở thành đối thủ nổi bật với các công cụ khoa học như MatLab hoặc Mathematica.

Tuy nhiên, thư viện này ở cấp độ thấp, đồng nghĩa là bạn sẽ cần phải viết nhiều code hơn để tiếp cận các cấp độ visualization cao cấp và bạn sẽ phải nỗ lực hơn so với khi sử dụng các công cụ cấp cao, tuy nhiên nỗ lực này là hoàn toàn xứng đáng.

Chỉ cần nỗ lực 1 chút, bạn có thể tạo được các visualization bất kì:

+ Line plots;
+ Scatter plots;
+ Bar charts và Histograms;
+ Pie charts;
+ Stem plots;
+ Contour plots;
+ Quiver plots;
+ Spectrograms.

Có rất nhiều công cụ để tạo nhãn, lưới, các biểu tượng/ kí hiệu/ chú giải và rất nhiều yếu tố format khác với Matplotlib. Về cơ bản, mọi thứ đều có thể custom được.

Thư viện này còn được rất nhiều platform hỗ trợ và tận dụng các GUI kít khác nhau để mô tả các visualizations kết quả. Thay đổi các IDEs (như IPython) sẽ hỗ trợ chức năng của Matplotlib.

- SciPy (Commits: 17213, Contributors: 489)

SciPy là 1 thư viện phần mềm cho engineering và khoa học. Một lần nữa bạn cần phải hiểu sự khác biệt giữa SciPy Stack và thư viện SciPy. SciPy gồm các modules cho đại số tuyến tính, optimization, tích hợp và thống kế. Chức năng chính của thư viện SciPy được xây dựng trên NumPy, và arrays của nó sẽ tận dụng tối đa NumPy. Nó mang đến rất nhiều hoạt động hữu ích liên quan đến số như tích hợp số, optimization… qua các submodules chuyên biệt. Các hàm trong tất cả các submodules của SciPy đều được document tốt.

- NumPy (Commits: 15980, Contributors: 522)

Khi bắt đầu giải quyết task về khoa học bằng Python, tập hợp phần mềm được thiết kế riêng cho scientific computing trong Python sẽ không thể không hỗ trợ SciPy Stack của Python (đừng nhầm lẫn với thư viện SciPy – là 1 phần của stack này, và cộng đồng của stack này). Tuy nhiên, stack này khá rộng, có hơn cả tá thư viện trong nó và chúng ta thì lại muốn tập trung vào các core packages (đặc biệt là những packages quan trọng nhất).

Package cơ bản nhất, khi computation stack về khoa học được xây dựng là NumPy (viết tắt của Numerical Python), cung cấp rất nhiều tính năng hữu ích cho các phần operations trong n-arrays & matrics trong Python. Thư viện này cung cấp khả năng vector hóa các vận hành về toán trong type array NumPy, giúp cải thiện hiệu suất và theo đó là tốc độ execution.

- Pandas (Commits: 15089, Contributors: 762)

Pandas là 1 package Python được thiết kế để làm việc với dữ liệu đơn giản, trực quan, được "gắn nhãn" và có liên hệ với nhau. Pandas là công cụ hoàn hảo để tinh chỉnh và làm sạch dữ liệu. Pandas được thiết kế hỗ trợ cho các thao tác, tập hợp và visualize dữ liệu.

Có 2 data structure chính trong thư viện này:

"Series"—1 chiều

Series	
A	X0
B	X1
C	X2
D	X3

"Data Frames", 2 chiều

DataFrame				
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

- Dễ dàng xóa và thêm cột từ DataFrame
- Chuyển data structures đến các objects DataFrame
- Xử lý các data bị mất, như NaNs
- Khả năng bhóm lại theo chức năng

- Pytorch

PyTorch là một framework được xây dựng dựa trên python cung cấp nền tảng tính toán khoa học phục vụ lĩnh vực Deep learning. Pytorch tập trung vào 2 khả năng chính:

- Một sự thay thế cho bộ thư viện numpy để tận dụng sức mạnh tính toán của GPU.
- Một platform Deep learning phục vụ trong nghiên cứu, mang lại sự linh hoạt và tốc độ.

Ngoài ra còn có các thư viện khác cho DeepLearning như Neural Networks Zoo, Caffe, MXNet, Chainer (mới), Caffe2, ….

2.5. Các khái niệm trong mạng Nơron

Mạng Neuron nhân tạo (Artificial Neural Network- ANN) là mô hình xử lý thông tin được mô phỏng dựa trên hoạt động của hệ thống thần kinh của sinh vật, bao gồm số lượng lớn các Neuron được gắn kết để xử lý thông tin. ANN giống như bộ não con người, được học bởi kinh nghiệm (thông qua huấn luyện), có khả năng lưu giữ những kinh nghiệm hiểu biết (tri thức) và sử dụng những tri thức đó trong việc dự đoán các dữ liệu chưa biết (unseen data).

Các ứng dụng của mạng Neuron được sử dụng trong rất nhiều lĩnh vực như điện, điện tử, kinh tế, quân sự,… để giải quyết các bài toán có độ phức tạp và đòi hỏi có độ chính xác cao như điều khiển tự động, khai phá dữ liệu, nhận dạng,…

2.5.1. Epoch

Một Epoch là khi tất cả dữ liệu được đưa vào mạng neural network 1 lần. Khi dữ liệu quá lớn, chúng ta không thể đưa hết mỗi lần 1 epoch để huấn luyện. Buộc lòng chúng ta phải chia nhỏ 1 epoch ra thành các batchs nhỏ hơn.

2.5.2. Batch Size

Batch size là số lượng mẫu dữ liệu trong một batch. Ở đây, khái niệm batch size và số lượng batch(number of batch) là hoàn toàn khác nhau.

Chúng ta không thể đưa hết toàn bộ dữ liệu vào huấn luyện trong 1 epoch, vì vậy chúng ta cần phải chia tập dữ liệu thành các phần (number of batch), mỗi phần có kích thước là batch size.

2.5.3. Iterations

Iterations là số lượng batchs cần để hoàn thành 1 epoch.Ví dụ chúng ta có tập dữ liệu có 20,000 mẫu, batch size là 500, vậy chúng ta cần 50 lần lặp (iteration) để hoàn thành 1 epoch.

2.5.4. Loss Fuction

Kí hiệu là LLL, là thành phần cốt lõi của evaluation function và objective function. Cụ thể, trong công thức thường gặp:

$$\mathcal{L}_D(f_w) = \frac{1}{|D|} \sum_{(x,y)\in D} L\left(f_w(x), y\right)$$

thì hàm LLL chính là loss function.

Loss function trả về một số thực không âm thể hiện sự chênh lệch giữa hai đại lượng: y^, label được dự đoán và y, label đúng. Loss function giống như một hình thức để bắt model đóng phạt mỗi lần nó dự đoán sai, và số mức phạt tỉ lệ thuận với độ trầm trọng của sai sót. Trong mọi bài toán supervised learning, mục tiêu của ta luôn bao gồm giảm thiểu tổng mức phạt phải đóng. Trong trường hợp lý tưởng y^=y.loss function sẽ trả về giá trị cực tiểu bằng 0.

2.5.5. Momentum

Là một kỹ thuật đơn giản thường cải thiện cả tốc độ đào tạo và độ chính xác. Đào tạo một mạng nơ-ron là quá trình tìm các giá trị cho các trọng số và độ lệch sao cho đối với một tập hợp các giá trị đầu vào nhất định, các giá trị đầu ra được tính toán khớp với các giá trị đích đã biết, đúng, đã biết. Ví dụ: giả sử bạn đang cố gắng dự đoán các loài hoa iris (setosa, Versolor hoặc viriginica) dựa trên chiều dài sepal của hoa, chiều rộng của cánh hoa, chiều dài cánh hoa và chiều rộng cánh hoa.

2.5.6. Dataset

Là Tập dữ liệu huấn luyện trong Machine Learning; là bộ dữ liệu thực tế được sử dụng để huấn luyện mô hình để thực hiện các hành động khác nhau. Đây là dữ liệu thực tế mà các mô hình quy trình phát triển đang diễn ra học với nhiều thuật toán và API khác nhau để đào tạo máy hoạt động tự động

2.5.7. Learning rate

Là một siêu tham số có thể định cấu hình được sử dụng trong việc đào tạo các mạng thần kinh có giá trị dương nhỏ, thường nằm trong khoảng giữa

0,0 và 1,0. Tốc độ học tập kiểm soát mức độ nhanh chóng của mô hình thích ứng với vấn đề.

Learning rate có thể là siêu tham số quan trọng nhất khi định cấu hình mạng thần kinh của bạn. Do đó, điều quan trọng là phải biết cách điều tra các tác động của tốc độ học tập đối với hiệu suất của mô hình và xây dựng một trực giác về động lực của tốc độ học tập đối với hành vi của mô hình

2.5.8. Traing set

Là một tập dữ liệu có kích thước lớn, được dùng để training trong quá trình huấn luyện máy học. Đây chính là tập dữ liệu máy dùng để học và rút trích được những đặc điểm quan trọng để ghi nhớ lại. Tập training set sẽ gồm 2 phần:

- Input: sẽ là những dữ liệu đầu vào.
- Output: sẽ là những kết quả tương ứng với tập input.

Training set là tập các cặp input và output dùng để huấn luyện trong quá trình máy học.

2.5.9. Testing set

Testing set là tập dữ liệu dùng để test sau khi máy đã học xong. Một mô hình máy học sau khi được huấn luyện, sẽ cần phải được kiểm chứng xem nó có đạt hiểu quả không. Testing set chỉ gồm các giá trị input mà không có các giá trị output. Máy tính sẽ nhận những giá trị input này, và xử lý các giá trị, sau đó đưa ra output tương ứng cho giá trị input.

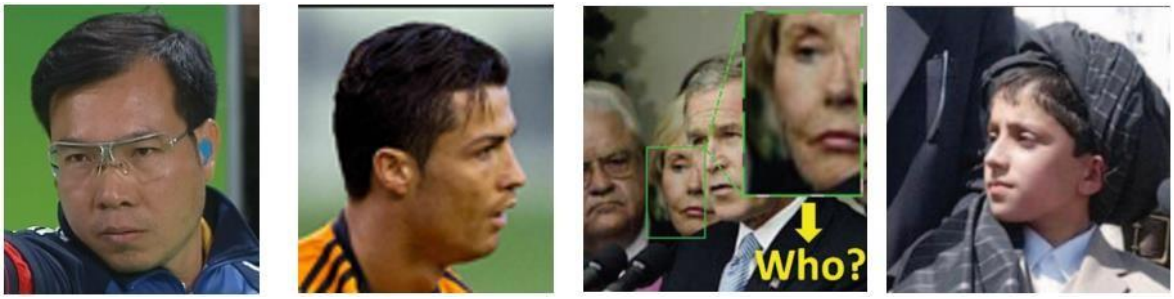
Testing set là tập các giá trị input và được dùng để kiểm thử độ chính xác của những mô hình máy học sau khi được huấn luyện.

CHƯƠNG 3. CÁC KỸ THUẬT CHO NHẬN DẠNG KHUÔN MẶT

3.1. Tổng quan về nhận dạng khuôn mặt

Nhận dạng khuôn mặt là một bài toán lâu đời và được nghiên cứu rộng rãi trong khoảng hơn 30 năm trở lại đây. Bài toán nhận dạng khuôn mặt có thể áp dụng rộng rãi trong nhiều lĩnh vực khác nhau. Các ứng dụng liên quan đến nhận dạng khuôn mặt có thể kể như: Hệ thống phát hiện tội phạm, hệ thống theo dõi nhân sự trong một đơn vị, hệ thống tìm kiếm thông tin trên ảnh, video dựa trên nội dung, … Hiện nay, bài toán nhận dạng khuôn mặt gặp nhiều thách thức, ví dụ như hệ thống camera công cộng, chụp hình vui chơi thì ảnh mặt nhận được có thể bị che khuất một phần, ảnh chụp không chính diện hay chất lượng ảnh không tốt, những yếu tố này ảnh hưởng không nhỏ đến các thuật toán nhận dạng khuôn mặt. Có nhiều thuật toán khắc phục điều này, họ sử dụng một số kỹ thuật như xác định nhiều điểm chính trên khuôn mặt, lấy những chi tiết nhỏ hay sử dụng các phương pháp Deep-learning.

Yêu cầu bài toán từ ảnh chụp một phần (hay một góc) của khuôn mặt, ta cần xác định xem mặt đó là của ai. Yêu cầu ảnh phải đảm bảo thấy được ít nhất 50% diện tích khuôn mặt và ít nhất một phần chi tiết ở mắt, mũi, miệng (xem Hình).

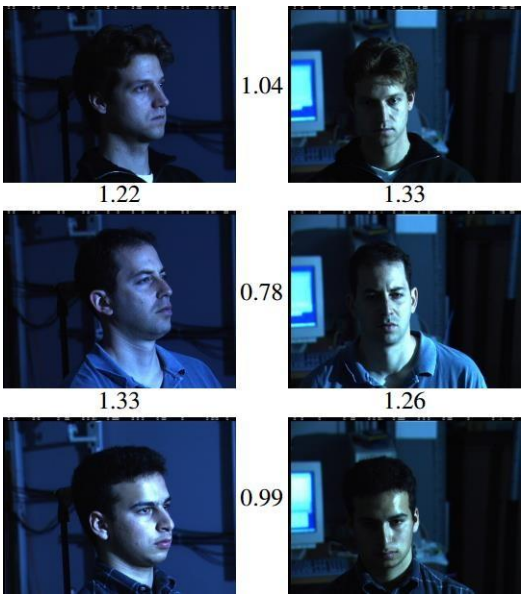


Hình 3.1. Ảnh chụp một phần khuôn mặt, ta cần xác định mặt đó là ai

3.2. Nhận dạng khuôn mặt sử dụng thuật toán Facenet

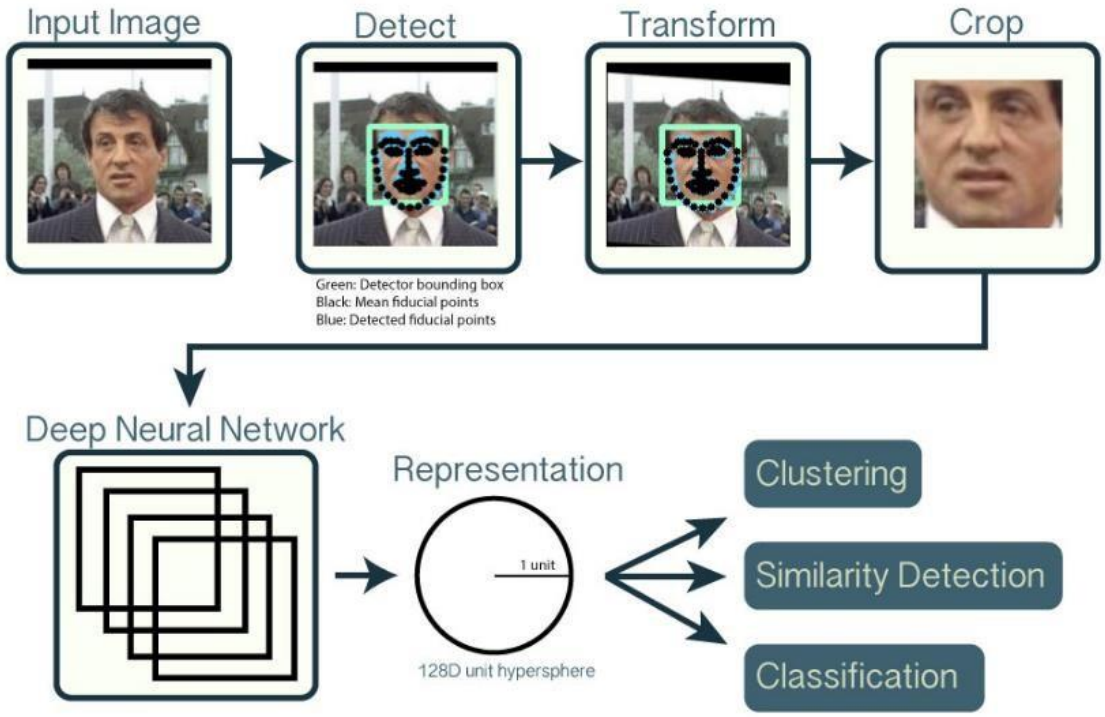
3.2.1. Tóm tắt

Nhóm tác giả từ Google đề xuất một thuật toán có tên là FaceNet sẽ học cách ánh xạ từ ảnh khuôn mặt vào không gian Euclide compact với khoảng cách đo được tương ứng với độ tương đồng của khuôn mặt. Thuật toán này có thể tạo ra vector đặc trưng và nhúng vào bài toán nhận dạng khuôn mặt, kiểm tra khuôn mặt và phân cụm khuôn mặt. Nhóm tác giả sử dụng Mạng Tích Chập Sâu (Deep Convolution Network - DNN) được huấn luyện để tự tối ưu hóa bài toán. Mạng được huấn luyện sao cho khoảng cách L2 bình phương trong không gian nhúng tương ứng với mức độ tương đồng của khuôn mặt: Mặt cùng người sẽ có khoảng cách nhỏ, mặt khác người sẽ có khoảng cách lớn (xem Hình 3.2).



Hình 3.2 Hình minh họa output khoảng cách khi sử dụng FaceNet giữa các cặp khuôn mặt. Nếu lấy ngưỡng là 1.1, ta thấy rằng 2 mặt có khoảng cách nhỏ hơn ngưỡng đều thuộc về một người (ví dụ như 2 ảnh ở dòng đầu tiên) và ngược lại.

Sau khi thực hiện phép nhúng, thu được vector đặc trưng thì ta có thể thực hiện được 3 bài toán: Kiểm tra khuôn mặt, ta chỉ cần phân ngưỡng khoảng cách giữa 2 vector đặc trưng của 2 khuôn mặt. Nhận dạng khuôn mặt là bài toán phân loại k-NN. Phân cụm khuôn mặt sử dụng k- mean (xem tóm tắt cách nhận dạng khuôn mặt sử dụng FaceNet ở Hình 3.3).

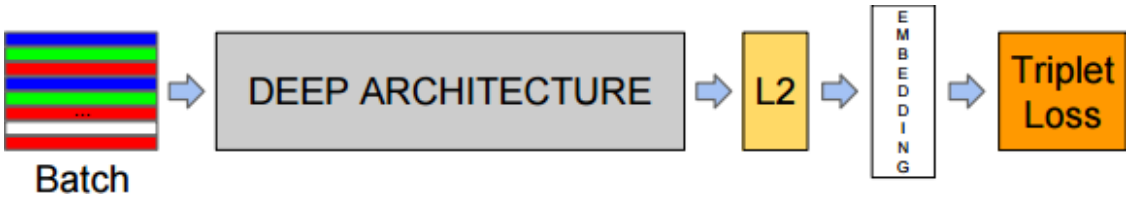


Hình 3.3. Tóm tắt quy trình nhận dạng khuôn mặt sử dụng FaceNet,từ ảnh vào (input image), sau đó xác định khuôn mặt, những

điểm chính trên mặt (Detect), canh chỉnh lại mặt (Transform), sau đó cắt khuôn mặt ra khỏi ảnh (Crop) và đưa vào Mạng Neuron Sâu (Deep Neural Network), thu được vector đặc trưng 128 chiều dùng để biểu diễn khuôn mặt (Representation). Từ vector đặc trưng này có thể dùng để phân cụm khuôn mặt (Clustering), xác định tính tương đồng (Similarity Detection) và phân loại (Classification).

Nhiều thuật toán nhận dạng khuôn mặt sử dụng DNN trước đây sử dụng lớp phân loại đã qua huấn luyện trên toàn bộ ảnh đã biết nhãn, sau đó lấy lớp thắt cổ chai trung bình để biểu diễn tổng quát cho tập huấn luyện. Tuy nhiên, mặt tiêu cực là lớp thắt cổ chai đôi khi không rõ 26as và không tiện lợi do lớp thắt cổ chai này thường rất lớn (khoảng 1000 chiều). Để khắc phục điều này, FaceNet huấn luyện output thành nhúng compact 128 chiều sử dụng hàm bộ ba sai số dựa trên LMNN, mẫu bộ ba này gồm 2 ảnh cùng loại và 1 ảnh khác loại và hàm sai số có nhiệm vụ tách ảnh đúng ra khỏi ảnh sai dựa vào biên khoảng cách. Nhóm tác giả sử dụng 2 kiến trúc Mạng Tích Chập Sâu, một mạng dựa theo mô hình của Zeiler và Fergus, mạng còn lại sử dụng mô hình Inception từ GoogLeNet.

3.2.2. Chi tiết thuật toán



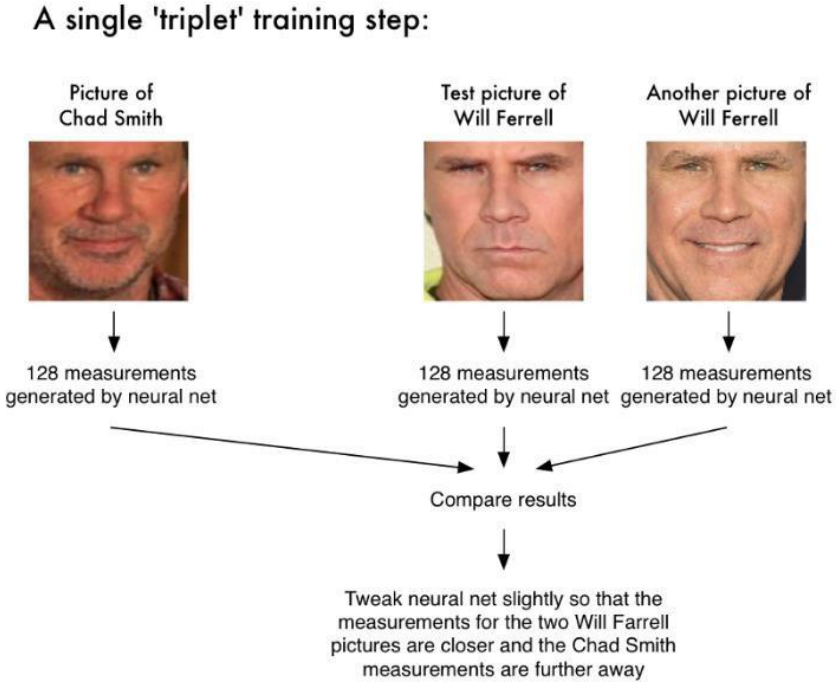
Hình 3.4. Cấu trúc mô hình, mạng bao gồm khối lớp vào (batch), đi qua cấu trúc Mạng Neuron Tích Chập Sâu (Convolution Neural Network – CNN), sau đó chuẩn hóa theo L_2 và đưa vào hệ thống nhúng. Trong quá trình huấn luyện có sử dụng bộ ba sai số.

FaceNet sử dụng DNN, giả sử cấu trúc mô hình là một khối lớp (xem Hình 3.4), sau khi sử dụng cấu trúc CNN, vấn đề quan trọng nằm ở kết quả sau khi huấn luyện. Do đó, nhóm tác giả sử dụng đến bộ ba sai số có thể giúp kiểm tra, nhận dạng và phân cụm khuôn mặt. Giả sử ta có ảnh x, đưa qua hàm nhúng (x) vào không gian đặc trưng $\mathbb{R}^d$ sao cho khoảng cách bình phương của tất cả khuôn mặt cùng loại phải nhỏ hơn khoảng cách bình phương với mặt khác loại.

3.2.2.1. Bộ ba sai số

Ta biểu diễn phép nhúng là hàm $(x) \in \mathbb{R}^d$ có chức năng nhúng ảnh x vào không gian Euclide d chiều. Hơn nữa, ta xét hàm nhúng này xác định trong siêu cầu d chiều, tức $\|(x)\|_2 = 1$. Giả sử x_i^a là ảnh kiểm tra người i, x_i^p là ảnh

của người i trong bộ dữ liệu và x_i^n là ảnh không phải của người i (xem Hình 3.5), ta muốn rằng khoảng cách từ x_i^a đến x_i^p phải ngắn hơn khoảng cách từ x_i^a đến x_i^n, tức $d(x_i^a, x_i^p) < d(x_i^a, x_i^n)$.



Hình 3.5. Ví dụ về bộ ba sai số, ảnh bên trái (Chad Smith) là x_i^n,, ảnh giữa (Will Ferrell) là x_i^a,, ảnh phải (Will Ferrell) là x_i^p.

Do đó, ta muốn

$$\left\|f(x_i^a) - f\left(x_i^p\right)\right\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2, \qquad (30)$$

$$\forall\left(f(x_i^a), f\left(x_i^p\right), f(x_i^n)\right) \in T \qquad (31)$$

với α là giá trị biên sao cho đảm bảo bất đẳng thức (30) xảy ra, T là tập các mẫu bộ 27as a thể xảy ra trong tập huấn luyện.

Ta cực tiểu hóa hàm sai số như sau

$$L = \sum_i^N \left[\left\|f(x_i^a) - f\left(x_i^p\right)\right\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]_+ \qquad (32)$$

Có rất nhiều bộ ba thỏa (30), những bộ ba này không đóng góp nhiều vào quá trình huấn luyện và khiến cho tốc độ hội tụ chậm. Do đó, ta cần chọn bộ ba thích hợp cho quá trình huấn luyện, giữ vai trò quan trọng trong mô hình.

3.2.2.2. Chọn Bộ ba

Để đảm bảo quá trình huấn luyện hội tụ nhanh, ta sẽ chọn bộ ba không thỏa bất đẳng thức (30), tức cho ảnh x_i^n của người i, ta chọn x_i^p $_p$ sao cho

$$\arg\max_{x_i^p} \left\| f(x_i^a) - f(x_i^p) \right\|_2^2 \qquad (33)$$

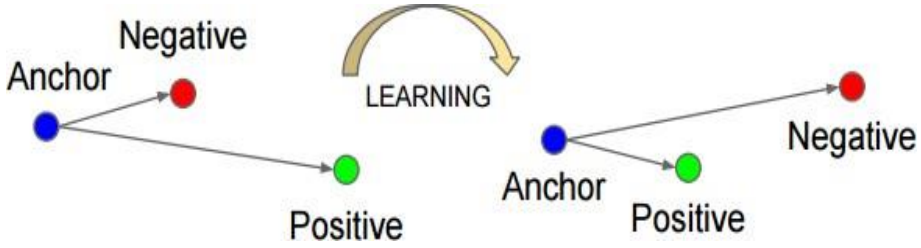
và chọn x_i^n sao cho

$$\arg\min_{x_i^n} \| f(x_i^a) - f(x_i^n) \|_2^2 \qquad (34)$$

Điều này có nghĩa trong tập ảnh cùng đối tượng với xia, ta chọn ảnh x_i^p (hard positive) sao cho khoảng cách giữa chúng là lớn nhất và trong tập ảnh khác đối tượng với x_i^a, chọn ảnh x_i^n (hard negative) sao cho khoảng cách giữa chúng là nhỏ nhất, khi đó có khả năng xảy ra trường hợp

$$\left\| f(x_i^a) - f(x_i^p) \right\|_2^2 > \| f(x_i^a) - f(x_i^n) \|_2^2 \qquad (35)$$

Ta sẽ huấn luyện sao cho bất đẳng thức (35) về lại kiểu (30). Hình 3.6 cho thấy quy trình huấn luyện sẽ rút ngắn khoảng cách giữa x_i^a, và x_i^p xuống thấp nhất và kéo dài khoảng cách giữa x_i^a và x_i^n ra xa nhất.



Hình 3.6 Bộ ba sai số tối thiểu hóa khoảng cách giữa ảnh vào (Anchor) và ảnh cùng loại với ảnh vào (Positive) và tối đa hóa khoảng cách giữa ảnh vào và ảnh khác loại với ảnh vào (Negative).

Không thể tính arg min và arg max trên toàn bộ tập huấn luyện vì có thể đưa ra kết quả huấn luyện kém và quá trình huấn luyện có thể lấy ảnh có chất lượng kém làm hard positive và hard negative. Để khắc phục tình trạng này, nhóm tác giả sử dụng khối mini lớn với vài ngàn mẫu, dùng khối này vào quá trình huấn luyện, tạo bộ ba sau mỗi n bước huấn luyện, chọn điểm checkpoint mới nhất ở trong mạng và tính arg min và arg max trong tập dữ liệu con. Để biểu diễn khoảng cách giữa x_i^a và x_i^p có nghĩa, ta cần đảm bảo tối thiểu các mẫu từ tất cả đối tượng phải có trong khối mini. Khi thực nghiệm, nhóm tác giả lấu mẫu dữ liệu huấn luyện sao cho mỗi đối tượng có 40 ảnh ở mỗi khối mini. Hơn nữa, chọn ngẫu nhiên ảnh x_i^n và thêm vào mỗi khối.

Thay vì chọn ảnh cùng loại có khoảng cách xa nhất, nhóm tác giả sử dụng tất cả cặp (x^a, x^p) trong khối mini, đồng thời tìm kiếm ảnh hard negative. Thực nghiệm cho thấy cặp (x^a, x^p) có tính ổn định và hội tụ nhanh. Chọn mẫu khác loại có khoảng cách gần nhất có thể đưa đến lỗi cực tiểu địa phương khi

huấn luyện, dễ dẫn đến mô hình bị sập (tức $(x) = 0$). Để tránh trường hợp này, ta chọn x_i^n sao cho

$$\left\|f(x_i^a) - f\left(x_i^p\right)\right\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2 \qquad (36)$$

Ta gọi các mẫu x_i^n thỏa bất đẳng thức (36) là semi-hard do khoảng cách từ mẫu này đến x_i^a xa hơn khoảng cách đến x_i^p, nhưng ta vẫn gặp khó khăn do bình phương khoảng cách sát với khoảng cách x_i^a đến x_i^p. Các mẫu x_i^n này nằm trong biên α.

Chọn đúng bộ ba sẽ giúp quá trình huấn luyện hội tụ nhanh. Mặt khác, nhóm tác giả sử dụng khối mini nhỏ do khối này giúp cải thiện khả năng hội tụ khi sử dụng kỹ thuật Trượt Dốc Ngẫu Nhiên (Stochastic Gradient Descent – SGD) [28] bằng cách lấy phần biểu thức trong dấu Σ ở (32)

$$\left\|f(x_i^a) - f\left(x_i^p\right)\right\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \qquad (37)$$

sau đó lấy đạo hàm hàm L theo từng biến x_i^a, x_i^p, x_i^n

$$\frac{\partial L}{\partial x_i^a} = \sum_{i=1}^{N} \begin{cases} 2\left(f(x_i^n) - f\left(x_i^p\right)\right), & \text{nếu (37)} \geq 0 \\ 0, & \text{ngược lại} \end{cases} \qquad (38)$$

$$\frac{\partial L}{\partial x_i^p} = \sum_{i=1}^{N} \begin{cases} -2\left(f(x_i^a) - f\left(x_i^p\right)\right), & \text{nếu (37)} \geq 0 \\ 0, & \text{ngược lại} \end{cases} \qquad (39)$$

$$\frac{\partial L}{\partial x_i^n} = \sum_{i=1}^{N} \begin{cases} -2\left(f(x_i^a) - f(x_i^n)\right), & \text{nếu (37)} \geq 0 \\ 0, & \text{ngược lại} \end{cases} \qquad (40)$$

sau đó, ta cập nhật tham số hàm sai số

$$f(x_i^a) = f(x_i^a) - \delta \frac{\partial L}{\partial x_i^a} \qquad (41)$$

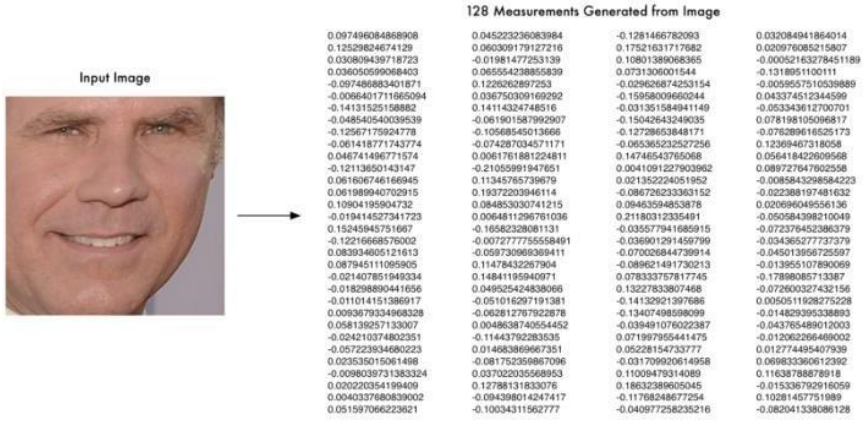
$$f\left(x_i^p\right) = f\left(x_i^p\right) - \beta \frac{\partial L}{\partial x_i^p} \qquad (42)$$

$$f(x_i^n) = f(x_i^n) - \gamma \frac{\partial L}{\partial x_i^n} \qquad (43)$$

với δ, β, γ là tốc độ học

3.2.2.3. Mạng tích chập sâu

Khi thực nghiệm, nhóm tác giả huấn luyện sử dụng CNN sử dụng SGD với kỹ thuật truyền ngược chuẩn [29] [30] và AdaGrad. Khi thực nghiệm, nhóm chọn tốc độ học $\delta = \beta = \gamma = 0.05$, huấn luyện qua một cụm CPU trong 1000 giờ đến 2000 giờ, hàm chi phí giảm dần (tức độ chính xác tăng dần) sau 500 giờ huấn luyện. Sau huấn luyện. mỗi ảnh trả về vector đặc trưng 128 chiều (xem Hình 3.7), tính chất với 2 ảnh cùng người thì khoảng cách hai vector gần hơn khoảng cách với ảnh của người khác.



Hình 3.7. Ảnh vào (trái) sau khi huấn luyện, thu được vector 128 chiều (phải)

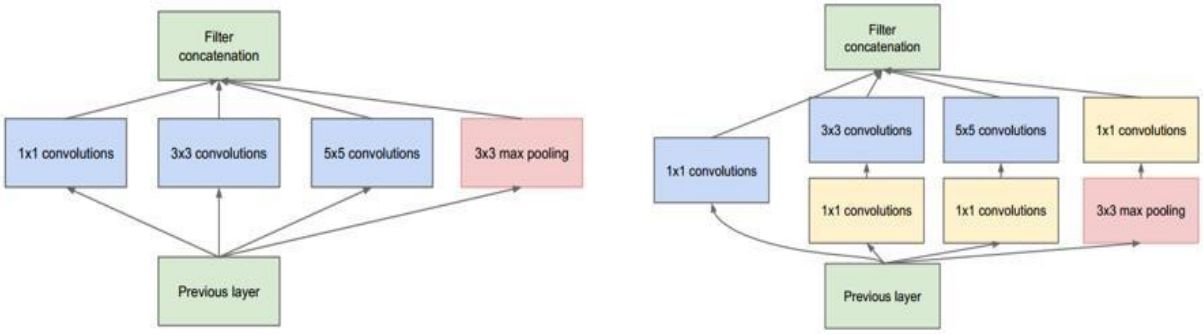
Nhóm tác giả sử dụng 2 kiến trúc để học, một kiến trúc của Zeiler và Fergus (xem Hình 3.8). Nhóm tác giả thêm lớp tích chập $1 \times 1 \times d$, thu được mô hình sâu 22 lớp với 140 triệu tham số và cần 1.6 tỷ toán tử/giây/ảnh.

layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

Hình 3.8. Cấu trúc mạng do Zeiler và Fergus đề xuất, với các lớp (layer), kích thước input (size-in) và output (sizeout) có dạng $rows \times cols \times \#filters$ (dòng ×

cột × số lượng lọc), riêng phần nhân (kernel) là $rows \times cols$, $stride$ (dòng × cột, bước sải) và kích thước maxout polling là $p = 2$.

kiến trúc còn lại từ mô hình Inception của GoogLeNet. Ý tưởng chính của kiến trúc Inception nhằm quan sát cách các bộ phận có sẵn của đối tượng bao phủ và xấp xỉ cấu trúc thưa địa phương tối ưu hóa của mạng thị giác tích chập. Module Inception được xây dựng như trong Hình 3.9. Mô hình này có khoảng 6.6 – 7.5 triệu tham số và khoảng 500 triệu – 1.6 tỷ toán tử. Chi tiết mô hình xem tại Hình 3.10.



Hình 3.9. Module Inception dạng nguyên thủy (ảnh trái) và dạng giảm chiều (ảnh phải)

type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

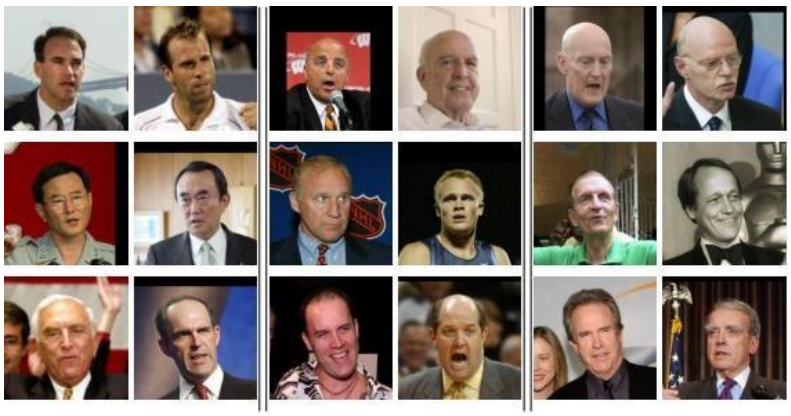
Hình 3.10. FaceNet sử dụng mô hình Inception tương tự với Hai điểm khác biệt chinh đó là FaceNet sử dụng L2 pooling thay vì max pooling. Kích thước polling luôn luôn 3 × 3 và tính song song với module tích chập trong mỗi module Inception.

3.2.3.4. Thực nghiệm

Nhóm tác giả đánh giá trên bộ dữ liệu LFW và Youtube Faces và thực nghiệm 3 vấn đề: Nhận dạng khuôn mặt, kiểm tra khuôn mặt và phân cụm khuôn mặt. Do mục tiêu của cuốn báo cáo nói về nhận dạng khuôn mặt nên tôi chỉ trình bày kết quả thực nghiệm nhận dạng khuôn mặt khi sử dụng

FaceNet. Khi huấn luyện, nhóm tác giả sử dụng 100 triệu đến 200 triệu ảnh khuôn mặt từ 8 triệu đối tượng, sau khi cắt phần khuôn mặt trong ảnh để tạo thành ảnh khuôn mặt, nhóm thay đổi kích thước ảnh khuôn mặt từ 96 × 96 điểm ảnh đến 224 × 224 điểm ảnh.

Với bài toán nhận dạng khuôn mặt, sau khi huấn luyện thu được vector đặc trưng 128 chiều thì ta sử dụng phân loại k-NN. Nhóm đánh giá trên bộ dữ liệu LFW với 13233 ảnh khuôn mặt từ 5749 người và thu được độ chính xác 98.87% ± 0.15 khi không canh chỉnh mặt và 99.63% ± 0.09 khi có canh chỉnh mặt. Hình 3.11 là một số cặp ảnh nhận dạng lỗi trong bộ LFW.



Hình 3.11. Một số cặp ảnh nhận dạng sai trong bộ dữ liệu LFW.

Trong bộ dữ liệu Youtube Face bao gồm 3425 video với 1595 người, trong 100 frame đầu tiên ở mỗi video, nhóm tác giả xác định khuôn mặt, tính trung bình tương đương cho tất cả cặp Trong bộ dữ liệu Youtube Face bao gồm 3425 video với 1595 người, trong 100 frame đầu tiên ở mỗi video, nhóm tác giả xác định khuôn mặt, tính trung bình tương đương cho tất cả cặp

- *Ưu điểm:*

Tính đến thời điểm FaceNet ra đời, thuật toán này đã lập nên kỷ lục mới trong nhận dạng khuôn mặt dưới nhiều điều kiện ảnh khác nhau

- *Nhược điểm*

FaceNet huấn luyện với một số lượng lớn hình ảnh (hơn 200 triệu ảnh của 8 triệu đối tượng),lớn gấp 3 lần so với các bộ dữ liệu hiện có. Để xây dựng bộ dữ liệu lớn như vậy rất khó thực hiện trong các phòng thiết bị, học thuật do đòi hỏi kiến trúc máy lớn

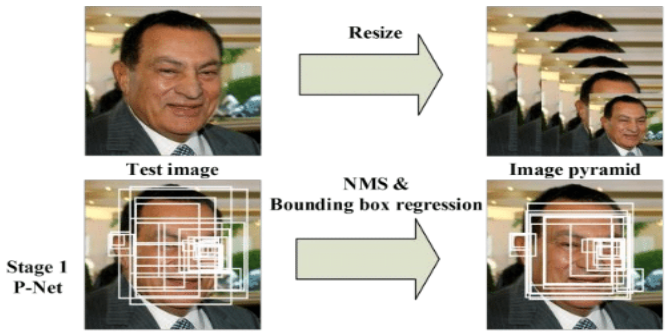
3.3. Phương pháp MTCNN (Multi-task Cascaded Convolutional Networks)

3.3.1. Tóm tắt

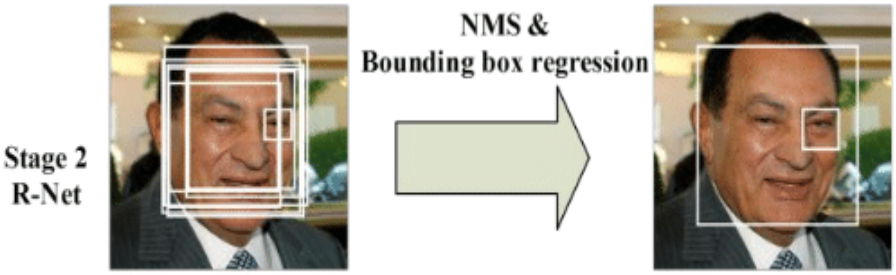
Phát hiện khuôn mặt và căn chỉnh trong môi trường không bị giới hạn là một thách thức do nhiều tư thế, ánh sáng và khớp khác nhau. Các nghiên cứu gần đây cho thấy phương pháp học sâu có thể đạt được hiệu suất ấn tượng trong hai nhiệm vụ này. Trong bài báo này, chúng tôi đề xuất một khung đa tác vụ xếp tầng sâu, khai thác mối tương quan vốn có giữa phát hiện và căn chỉnh để tăng hiệu suất của chúng. Đặc biệt, khung của chúng tôi tận dụng một kiến trúc xếp tầng với ba giai đoạn của mạng tích chập sâu được thiết kế cẩn thận để dự đoán vị trí mặt và mốc theo cách từ thô đến mịn. Ngoài ra, chúng tôi đề xuất một chiến lược khai thác mẫu cứng trực tuyến mới giúp cải thiện hơn nữa hiệu suất trong thực tế. Phương pháp của chúng tôi đạt được độ chính xác vượt trội so với các kỹ thuật tiên tiến trên các tiêu chuẩn FDDB và WIDER FACE đầy thách thức để phát hiện khuôn mặt và điểm chuẩn AFLW để căn chỉnh khuôn mặt, trong khi vẫn giữ hiệu suất thời gian thực.

Ở đây, nhóm tác giả đề xuất một khuôn khổ mới để tích hợp hai nhiệm vụ này sử dụng CNNs cascaded thống nhất bởi học đa nhiệm vụ. Các CNNs đề xuất bao gồm ba giai đoạn.Trong giai đoạn đầu tiên, nó tạo ra các cửa sổ ứng cử viên một cách nhanh chóng thông qua một nông CNN.Sau đó, nó trau chuốt các cửa sổ để từ chối một số lượng lớn các phi khuôn mặt cửa sổ thông qua một CNN phức tạp hơn.Cuối cùng, nó sử dụng một CNN mạnh mẽ hơn để tinh chỉnh kết quả và đầu ra vị trí địa điểm trên khuôn mặt.Nhờ khuôn khổ học tập đa nhiệm vụ này, việc thực hiện các thuật toán có thể được cải thiện đáng kể.Những đóng góp chính của nghiên cứu này được summa rized như sau: (1) Nhóm tác giả đề xuất một CNNs cascaded dựa khuôn khổ mới cho nhận diện khuôn mặt khớp và sự liên kết,và cẩn thận thiết kế kiến trúc CNN nhẹ cho hiệu suất theo thời gian thực. (2) Nhóm tác giả đề xuất một phương pháp hiệu quả để tiến hành khai thác mẫu trực tuyến khó có thể cải thiện hiệu suất.(3) các thí nghiệm được tiến hành rộng rãi trên thách thức tiêu chuẩn,để hiển thị sự cải thiện đáng kể hiệu suất của phương pháp đề xuất so với các kỹthuật hiện đại nhất trong cả nhận diện khuôn mặt và liên kết mặt nhiệm vụ.

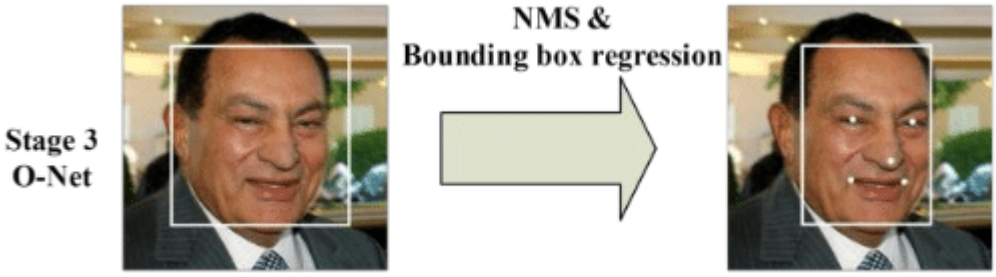
Các đường ống của khung xếp tầng bao gồm các mạng chập sâu đa nhiệm vụ ba giai đoạn.



Hình 3.12. Các cửa sổ ứng cử viên được sản xuất thông qua Mạng đề xuất nhanh (P-Net)



Hình 3.13. Tinh chỉnh các ứng cử viên này trong giai đoạn tiếp theo thông qua Mạng tinh chỉnh (R-Net).



Hình 3.14. Mạng đầu ra (O-Net) tạo hộp giới hạn cuối cùng và vị trí mốc mặt.

Phương pháp nhận diện khuôn mặt này có một lợi thế về các điều kiện ánh sáng khác nhau, khuôn mặt tạo ra các biến thể và biến thể thị giác của khuôn mặt.

3.3.2. Chi tiết

3.3.2.1. Tổng thể

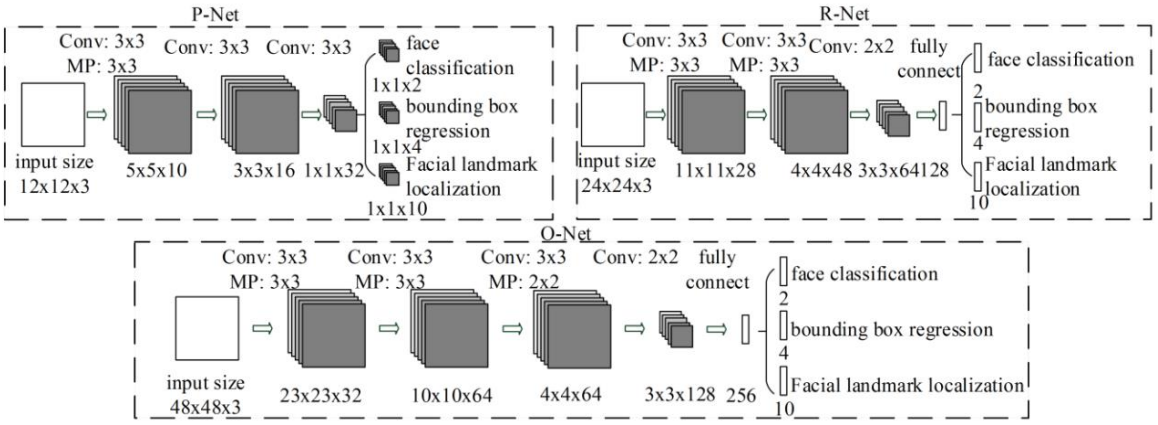
Đường ống dẫn dầu tổng thể của cách tiếp cận của chúng tôi được thể hiện trong hình. 3.12,3.13,3.14 .Với một hình ảnh, chúng tôi bước đầu thay đổi kích thước quy mô khác nhau để xây dựng một kim tự tháp hình ảnh,đó là đầu vào của các khuôn khổ cascaded ba giai đoạn sau:

Giai đoạn 1: Chúng tôi khai thác một mạng lưới hoàn toàn xoắn [?], Gọi là Proposal Network (P-Net), để có được các cửa sổ ứng cử viên và hộp bounding vectơ hồi quy của họ trong một cách tương tự Sau đó, chúng tôi sử dụng bounding vectơ hồi quy hộp ước tính để hiệu chỉnh các ứng cử viên. Sau đó, chúng tôi sử dụng ức chế phi tối đa (NMS) để hợp nhất các ứng cử viên chồng chéo cao.

Giai đoạn 2: tất cả các ứng cử viên được cho ăn khác CNN, gọi Lọc mạng (R-Net), trong đó bác bỏ một số lượng lớn các ứng cử viên sai hơn nữa,Thực hiện hiệu chuẩn với bounding hồi quy hộp, và ứng cử viên merge NMS.

Giai đoạn 3: Giai đoạn này cũng tương tự như giai đoạn thứ hai, nhưng trong giai đoạn này, chúng tôi nhằm mục đích để mô tả bộ mặt chi tiết hơn.Đặc biệt, các mạng di chúc đầu ra năm vị trí địa điểm trên khuôn mặt.

3.3.2.2. Quy trình làm việc của MTCNN



Hình 3.15. Quy trình làm việc MTCNN

Các kiến trúc của P-Net, R-Net, và O-Net, trong đó "MP" có nghĩa là tổng hợp tối đa và "Chuyển đổi" có nghĩa là chập. Kích thước bước chập và tổng hợp là 1 và 2, tương ứng.

Một bức ảnh đầu vào, nó sẽ tạo ra nhiều bản sao của hình ảnh đó với các kích thước khác nhau.

Tại P-Net, thuật toán sử dụng 1 kernel 12x12 chạy qua mỗi bức hình để tìm kiếm khuôn mặt Sau lớp convolution thứ 3, mạng chia thành 2 lớp. Convolution 4-1 đưa ra xác suất của một khuôn mặt nằm trong mỗi bounding boxes, và Convolution 4-2 cung cấp tọa độ của các bounding boxes.

R-Net có cấu trúc tương tự với P-Net. Tuy nhiên sử dụng nhiều layer hơn. Tại đây, network sẽ sử dụng các bounding boxes đc cung cấp từ P-Net và tinh chỉnh là tọa độ.

Tương tự R-Net chia ra làm 2 layers ở bước cuối,cung cấp 2 đầu ra đó là tọa độ mới của các bounding boxes, cùng độ tin tưởng của nó. O-Net lấy các bounding boxes từ R-Net làm đầu vào và đánh dấu các tọa độ của các mốc trên khuôn mặt. Ở bước này, thuật toán đưa ra 3 kết quả đầu ra khác nhau bao gồm: xác suất của khuôn mặt nằm trong bounding box, tọa độ của bounding box và tọa độ của các mốc trên khuôn mặt (vị trí mắt, mũi, miệng)

3.4. Thuật toán SVM (Support Vector Machine)

3.4.1. Giới thiệu

Support Vector Machine (SVM) là phương pháp mạnh và chính xác nhất trong số các thuật toán nổi bật ở lĩnh vực khai thác dữ liệu. SVM bao gồm hai nội dung chính là: support vector classifier (SVC), bộ phân lớp dựa theo hỗ trợ và support vector regressor (SVR), bộ hồi quy dựa theo vector hỗ trợ. Được phát triển đầu tiên bởi Vapnik vào những năm 1990, SVM có nền tảng lý thuyết được xây dựng trên nền móng lý thuyết xác suất thống kê. Nó yêu cầu số lượng mẫu huấn luyện không nhiều và thường không nhạy cảm với số chiều của dữ liệu. Trong những thập niên qua, SVM đã phát triển nhanh chóng cả về lý thuyết lẫn thực nghiệm.

3.4.2. Định nghĩa.

Là phương pháp dựa trên nền tảng của lý thuyết thống kê nên có một nền tảng toán học chặt chẽ để đảm bảo rằng kết quả tìm được là chính xác

Là thuật toán học giám sát (*supervied learning*) được sử dụng cho phân lớp dữ liệu.

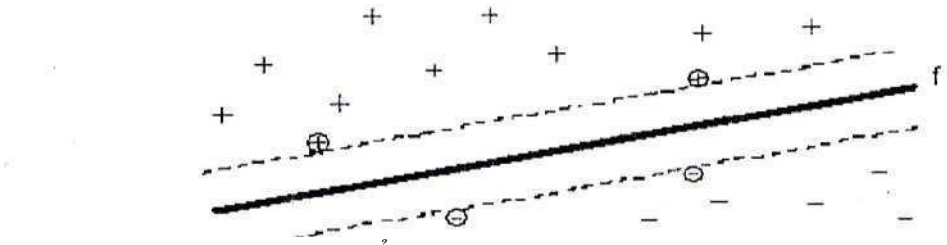
Là 1 phương pháp thử nghiệm, đưa ra 1 trong những phương pháp mạnh và chính xác nhất trong số các thuật toán nổi tiếng về phân lớp dữ liệu

SVM là một phương pháp có tính tổng quát cao nên có thể được áp dụng cho nhiều loại bài toán nhận dạng và phân loại

3.4.3. Ý tưởng

Cho trước một tập huấn luyện, được biểu diễn trong không gian vector, trong đó mỗi tài liệu là một điểm, phương pháp này tìm ra một siêu phẳng quyết định tốt nhất có thể chia các điểm trên không gian này thành hai lớp riêng biệt tương ứng là lớp + và lớp -. Chất lượng của siêu phẳng này được quyết định bởi khoảng cách (gọi là biên) của điểm dữ liệu gần nhất của mỗi lớp đến mặt phẳng này. Khi đó, khoảng cách biên càng lớn thì mặt phẳng quyết định càng tốt, đồng thời việc phân loại càng chính xác.

Mục đích của phương pháp SVM là tìm được khoảng cách biên lớn nhất, điều này được minh họa như sau:



Hình 3.16: Siêu phẳng phân chia dữ liệu học thành 2 lớp + và - với khoảng cách biên lớn nhất. Các điểm gần nhất (điểm được khoanh tròn) là các Support Vector.

3.4.4. Nội dung phương pháp

SVM thực chất là một bài toán tối ưu, mục tiêu của thuật toán này là tìm được một không gian F và siêu phẳng quyết định f trên F sao cho sai số phân loại là thấp nhất.

Bài toán SVM có thể giải bằng kỹ thuật sử dụng toán tử Lagrange để biến đổi về thành dạng đẳng thức. Một đặc điểm thú vị của SVM là mặt phẳng quyết định chỉ phụ thuộc các Support Vector và nó có khoảng cách đến mặt phẳng quyết

Cho dù các điểm khác bị xóa đi thì thuật toán vẫn cho kết quả giống như ban đầu. Đây chính là điểm nổi bật của phương pháp SVM so với các phương pháp khác vì tất cả các dữ liệu trong tập huấn luyện đều được đùng để tối ưu hóa kết quả.

TÓM LẠI: trong trường hợp nhị phân phân tách tuyến tính, việc phân lớp được thực hiện qua hàm quyết định f(x) = sign(<w.x> + b), hàm này thu được bằng việc thay đổi vectơ chuẩn w, đây là vectơ để cực đại hóa viền chức năng

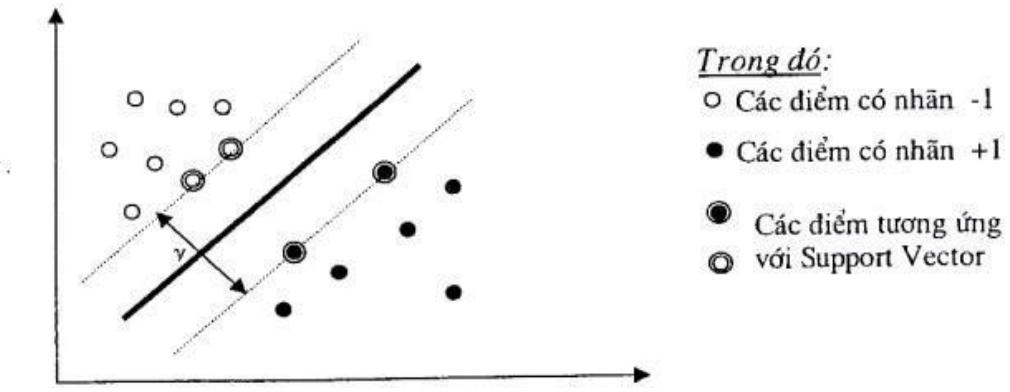
Việc mở rộng SVM để phân đa lớp hiện nay vẫn đang được đầu tư nghiên cứu. Có một phương pháp tiếp cận để giải quyết vấn để này là xây dựng và kết hợp nhiều bộ phân lớp nhị phân SVM (Chẳng hạn: trong quá trình luyện với SVM, bài toán phân m lớp có thể được biến đổi thành bài toán phân 2*m lớp, khi đó trong mỗi hai lớp, hàm quyết định sẽ được xác định cho khả năng tổng quát hóa tối đa). Trong phương pháp này có thể đề cập tới hai cách là một-đổi-một, một-đối-tất cả

3.4.5. Bài toán phân 2 lớp với SVM

Bài toán đặt ra là: Xác định hàm phân lớp để phân lớp các mẫu trong tọong lai, nghĩa là với một mẫu dữ liệu mới xi thì cần phải xác định xi được phân vào lớp +1 hay lớp -1

Để xác định hàm phân lớp dựa trên phương pháp SVM, ta sẽ tiến hành tìm hai siêu phẳng song song sao cho khoảng cách y giữa chúng là lớn nhất có

thể để phân tách hai lớp này ra làm hai phía. Hàm phân tách tọơng ứng với phương trình siêu phẳng nằm giữa hai siêu phẳng tìm được



Hình 3.17: Minh họa bài toán 2 phân lớp bằng phương pháp SVM

Các điểm mà nằm trên hai siêu phẳng phân tách được gọi là các Support Vector. Các điểm này sẽ quyết định đến hàm phân tách dữ liệu

3.4.6. Bài toán nhiều phân lớp với SVM

Để phân nhiều lớp thì kỹ thuật SVM nguyên thủy sẽ chia không gian dữ liệu thành 2 phần và quá trình này lặp lại nhiều lần. Khi đó hàm quyết định phân dữ liệu vào lớp thứ i của tập n , 2-Iớp sẽ là:

$$f_i(x) = w^i{}_i x + b_i$$

Những phần tử x là support vector sẽ thỏa điều kiện

$$f_i(x) = \begin{cases} +1 \text{ nếu thuộc lớp i} \\ -1 \text{ nếu thuộc phần còn lại} \end{cases}$$

Như vậy, bài toán phân nhiều lớp sử dụng phương pháp SVM hoàn toàn có thể thực hiện giống như bài toán hai lớp. Bằng cách sử dụng chiến lược "một- đối- một”(one - against - one).

Giả sử bài toán cần phân loại có k lớp (k > 2), chiến lược "một-đối-một”sẽ tiến hành k(k-l)/2 lần phân lớp nhị phân sử dụng phương pháp SVM. Mỗi lớp sẽ tiến hành phân tách với k-1 lớp còn lại để xác định k-1 hàm phân tách dựa vào bài toán phân hai lớp bằng phương pháp SVM.

3.4.7. Các Bước chính của phương pháp

Phương pháp SVM yêu cầu dữ liệu được diễn tả như các vector của các số thực. Như vậy nếu đầu vào chọa phải là số thì ta cần phải tìm cách chuyển chúng về dạng số của SVM

Tiền xử lý dữ liệu: Thực hiện biến đổi dữ liệu phù hợp cho quá trình tính toán, tránh các số quá lớn mô tả các thuộc tính. Thường nên co giãn (scaling) dữ liệu để chuyển về đoạn [-1, 1] hoặc [0, 1].

Chọn hàm hạt nhân: Lựa chọn hàm hạt nhân phù hợp tương ứng cho từng bài toán cụ thể để đạt được độ chính xác cao trong quá trình phân lớp.

Thực hiện việc kiểm tra chéo để xác định các thám số cho ứng đụng. Điều này cũng quyết định đến tính chính xác của quá trình phân lớp.

Sử dụng các tham số cho việc huấn luyện với tập mẫu. Trong quá trình huấn luyện sẽ sử dụng thuật toán tối ưu hóa khoảng cách giữa các siêu phẳng trong quá trình phân lớp, xác định hàm phân lớp trong không gian đặc trưng nhờ việc ánh xạ dữ liệu vào không gian đặc trưng bằng cách mô tả hạt nhân, giải quyết cho cả hai trường hợp dữ liệu là phân tách và không phân tách tuyến tính trong không gian đặc trưng.

CHƯƠNG 4. XÂY DỰNG ỨNG DỤNG

4.1. Hướng giải quyết bài toán nhận dạng khuôn mặt

Nhận dạng mặt khuôn mặt người là quá trình xác định danh tính tự động cho từng đôi tượng người trong ành/video dựa vào nội dung. Nhìn chung, quy trình giải quyết bài toán thường bao gôm các công đoạn cơ bàn như: (i) Thu nhận hình ánh; (ii) Tiên xử lý, tăng cường chất lượng hình ảnh; (iii) Phát hiện, căn chinh, crop ánh khuôn mặt; (iv) Nhận dạng (trích chọn đặc trưng và phân lớp) khuôn mặt.

Các hướng tiêp cận trước đây chủ yêu dựa trên đặc trưng (feature-based) và luôn cô găng đưa ra các định nghĩa tường minh để biểu diễn khuôn mặt dựa trôn các tỷ lệ khoảng cách, diện tích và góc Một biểu diễn khuôn mặt được định nghĩa tường minh hướng tới mục tiêu xây dựng một không gian đặc trưng trực quan. Tuy nhiên, trong thực tê các biểu diễn được định nghĩa tường minh thường không chính xác. Để khắc phục điều đó, các hướng tiếp cận sau này được đề xuất dựa trên ý tường sử dụng các mô hình học máy thông kê có khả năng học đê lựa chọn các đặc trưng khuôn mặt từ một tập mẫu cho trước điển hình như phương pháp PCA (Principal Component Analysis), trong đó mỗi khuôn mặt được biêu diễn dưới dạng tô hợp các eigenvectors, eigenfaces và fisherfaces phương pháp sử dụng các mô hình mạng neural tích chập CNN (Convolutional Neural Network).

Hiện tại, hiệu quả của các mô hình nhận dạng khuôn mặt đã được cài thiện đáng kê dựa trên việc kết hợp sử dụng các mô hình học sâu để tự động phát hiện các đặc trưng trên khuôn mặt và các kỹ thuật phân lớp thống kê. và dựa trên việc kết họp đấu ra của một mạng neural tích chập học sâu D-CNN (Deep Convolutional Neural Network) với PCA đê giảm chiều dữ liệu và bộ phân lớp SVM.

Zhenyao và cộng xây dựng một mạng neural học sâu đê căn chỉnh các khuôn mặt theo hướng nhìn trực diện sau đó huấn luyện một mạng CNN để phân lớp và xác định danh tính cho mỗi khuôn mặt. Y. Taigman và cộng sự đề xuất mô hình DeepFace dựa trên ý tưởng kết họp nhiều công đoạn (multi-stage): trước tiên sử dụng một mô hình khuôn mặt 3 chiều để chuẩn hóa các ảnh đầu vào (đã được thu thập với các tư thế, góc cạnh khác nhau) về tư thế nhìn thẳng (trực diện), sau đó xây dựng một kiến trúc mạng neural học sâu DNN (Deep Neural Net) với 120 triệu tham sô, có khả năng học từ một tập dữ liệu khổng lồ với trên 4.4 triệu khuôn mặt đã được gán nhãn. Trong kiên trúc mạng DNN DeepFace, lớp mạng cuối cùng dược loại bỏ và đâu ra của lớp mạng trước đó được sử dụng như một biếu diễn thấp chiều của khuôn mặt. Các kết quả thực nghiệm cho thây mô hình này đạt độ chính xác trên 97.35% đôi với tập dữ liệu LFW

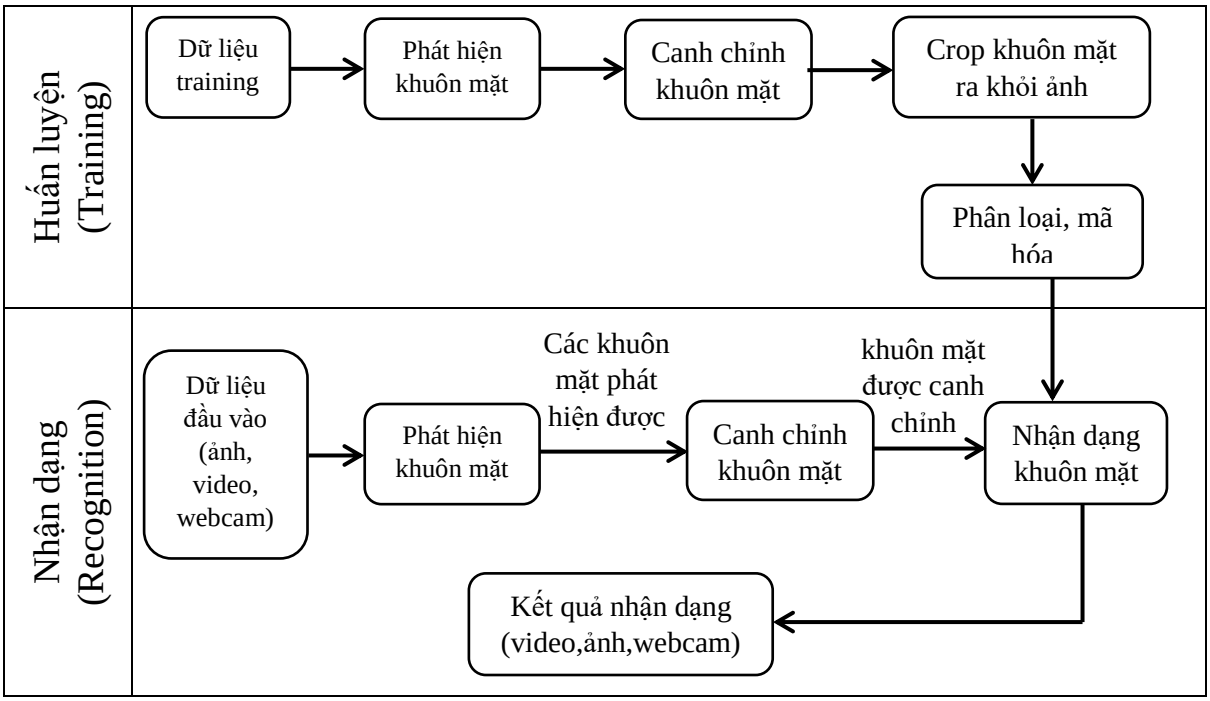
Mục tiêu của DeepFace cũng nhằm giải quyết bài toán đó, tuy nhiên đế có được sự biểu diễn này cần phải huấn luyện mạng trên một tập dữ liệu lớn. Đó cũng chính là điểm hạn chế của DeepFace.

Vì vậy, Florian Schroff và cộng sự đã đề xuất kiến trúc mạng học sâu FaceNet với hàm chi phí bộ ba (triplet loss function) được định nghĩa trực tiếp trên các biểu diễn. Hình 1 mô tả quá trình huấn luyện mạng FaceNet với hàm chi phí bộ ba để học cách phân cụm các biểu diễn khuôn mặt của cùng một người. Một siêu câu đơn vị (unit hypersphere) là một siêu cầu có sô chiêu lớn sao cho khoảng cách từ tất cà các điểm tới tâm của siêu cầu bằng 1.

Các cài tiến quan trọng của FaceNet bao gồm: (i) Đề xuất hàm chi phí bộ ba; (ii) thủ tục lựa chọn các bộ ba trong khi huấn luyện; (iii) cho phép học lù các tập dữ liệu khống lồ để tìm ra kiến trúc mạng thích hợp

4.2. Mô hình nhận dạng khuôn mặt

Các bước cơ bản nhận dạng khuôn mặt



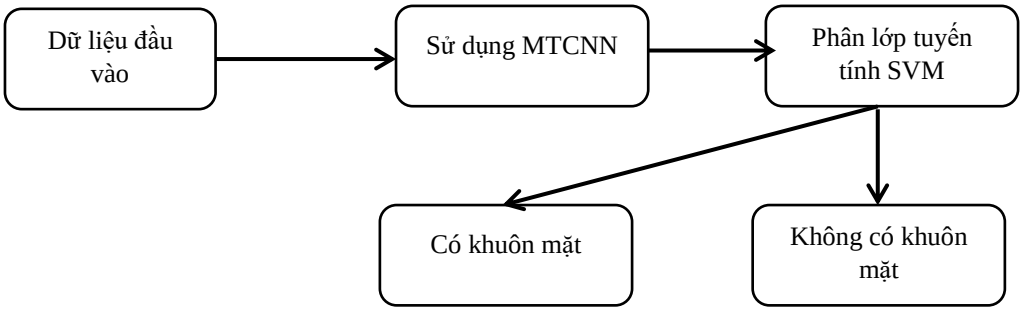
Hình 4.1. Mô hình nhận dạng khuôn mặt

Từ tín hiệu đầu vào, bước xử lý đầu tiên sẽ tiến hành phân đoạn thành các khung hình (frame) riêng biệt. Vì vậy, trong bước xử lý đầu tiên, thuật toán sẽ tiên hành phát hiện (face detection) và xác định vị trí của các khuôn mặt (nếu có) trên ảnh. Các khuôn mặt phát hiện được sau đó sẽ tiếp tục được tiền xừ lý nhăm tăng cường chất Iượng hình ảnh (loại nhiễu, khử bóng/mờ), chuẩn hóa kích thước và độ phân giải ảnh, căn chinh khuôn mặt về hướng trực diện (nhìn thẳng). Các khuôn mặt sau khi đã tiền xử lý sẽ được sử dụng làm đầu vào cho một mô hình mạng neral học sâu (DNN-Deep Neural Network). Mô hình này sẽ tự động học và trích chọn ra các đặc trưng đê nhận

dạng (phần lớp) khuôn mặt. Bước xử lý cuối cùa thuật toán sẽ tiến hành phần lớp (nhận diện) các khuôn mặt. Bản chất cùa việc phân lớp khuôn mặt là tìm kiêm đôi tượng người có mẫu khuôn mặt giống với khuôn mặt cần nhận dạng nhất. Để thực hiện được điều này, các mô hình phân lớp cần phải được huấn luyện với một tập mẫu cho trước. Trong đó, mỗi mẫu khuôn mặt được thể hiện bằng tập đặc trưng thu được từ các mô hình phát hiện đặc trưng DNN ở bước trên

4.2. Phát hiện khuôn mặt trên khung hình

Bản chất của việc phát hiện khuôn mặt là quá trình tìm kiếm và định vị khuôn mặt trên frame ảnh bất kỳ. Phương pháp phát hiện khuôn mặt ở đây được đề xuất sử dụng MTCNN và bộ phân lớp tuyến tính SVM (Support Vector Machines))



Hình 4.2. Phát hiện khuôn mặt

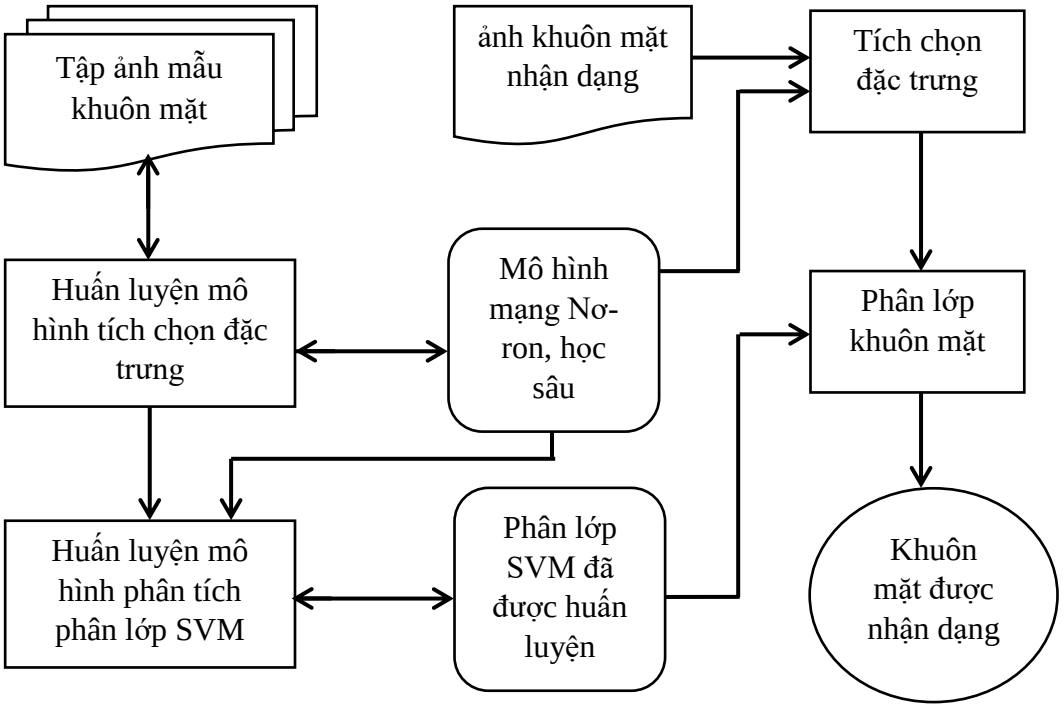
4.3. Nhận diện khuôn mặt

Nhận dạng thường gồm 2 bước xử lý chính là trích chọn đặc trưng và phân lớp khuôn mặt. Phương pháp trích chọn đặc trưng ờ đây được đề xuất sử dụng các lớp mạng neural học sâu FaceNet, được Florian Schroff và cộng sự đà đè xuất năm 2015 Đây là mô hình có khả năng học từ một tập mẫu cho trước nhằm tự động phát hiện các đặc trưng quan trọng nhất để nhận dạng đối tượng. Ý tưởng chính của hướng tiếp cận này dựa trên việc học một không gian Euclidean nhúng ứong mỗi ảnh sử dụng một cấu hình mạng neural tích chập học sâu (deep convolutional network). Mạng được huấn luyện sao cho khoảng cách L2 bình phương trong không gian nhúng là tương ứng trực tiêp với độ tương tự cùa khuôn mặt. Cụ thê là các khuôn mặt của cùng một người sẽ có khoảng cách nho và các khuôn mặt cùa các người khác nhau sẽ có khoáng cách lớn

Mạng được huấn một cách trực tiếp để đầu ra của nó trở thành một vector đặc trưng 128 chiều sử dụng hàm chi phí bộ ba (tripletbased loss function). Một bộ ba (triplet) được định nghĩa bao gôm hai khuôn mặt của cùng một người - positive và một khuôn mặt của người khác negative. Mục tiêu cùa hàm chi phí là phân tách cặp khuôn mặt positive ra khói khuôn mặt negative sử dụng một lê khoang cách - distance margin. Từ các độ đo thu

được, thuật toán sẽ ước lượng giá trị cùa hàm chi phi dựa trên việc so sánh khoáng cách giữa 2 tập dặc trung, được sinh ra từ 2 ảnh khuôn mặt khác nhau cua cùng một người (được gọi là người thứ nhât) và tạp đặc trưng thứ 3 được sinh ra từ ảnh khuôn mặt của một người khác (được gọi là người thứ hai). Các giá trị ước lượng của hàm chi phí sau khi tính sẽđược lan truyền ngược từ lớp cuối cùng đến lớp đau tiên cùa mạng đe tinh chỉnh trọng số (cập nhật lại trọng số) trên các lớp mạng. Quá trình tính toán, ước lượng và cập nhật trọng số của mạng được lặp di lặp lại liên tục cho đên khi giá trị của hàm chi phí thỏa mãn điều kiện đã cho. Lặp lại các bước trên đôi vối toàn bộ tập dữ liệu huấn luyện cho đến khi thuật toán huân luyện mạng hội tụ. Mô hình nhận dạng khuôn mặt được mô tả cụ thể:

Huấn luyện (xây dựng) mô hình Nhận dạng (sử dụng) mô hình



Hình 4.3. nhận dạng khuôn mặt

4.4. Cơ sở dữ liệu LFW

Bao gồm những khuôn mặt được gắn nhãn trong tự nhiên. Bộ dữ liệu gồm 13233 hình ảnh khuôn mặt của 5749 người được thu thập từ web. Mỗi khuôn mặt được gắn nhãn với tên của người đó trog đó, 1680 người có từ 2 hình ảnh khác biệt trở lên.

4.5. Demo chương trình

4.5.1. Tiền xử lý dữ liệu để cắt khuôn mặt từ ảnh gốc

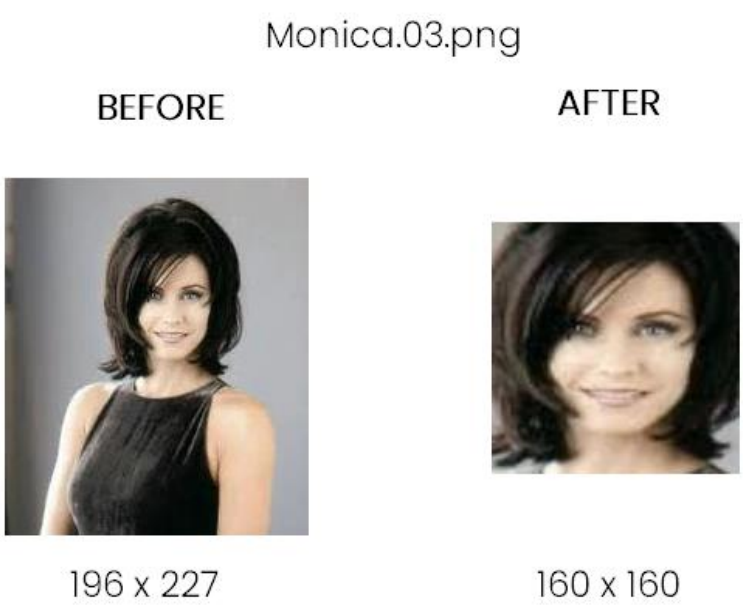
- Với chỗ ảnh mà đã chọn bên trên, có thể là ảnh cả người, bây giờ ta sẽ cắt riêng khuôn mặt ra để train. Chạy lệnh :

python src/align_dataset_mtcnn.py Dataset/FaceData/raw Dataset/FaceData/processed --image_size 160 --margin 32 --random_order --gpu_memory_fraction 0.25

Trong đó:

- align_dataset_mtcnn.py: Thực hiện căn chỉnh khuôn mặt bằng thuật toán mtcnn và lưu trữ hình thu nhỏ khuôn mặt trong thư mục đầu ra.

Chạy xong thấy nó hiển thị dạng "Total number of images: …" là thành công. ta sẽ thấy có thêm thư mục processed có cấu trúc tương tự thư mục raw nhưng chỉ chứa dữ liệu khuôn mặt dã được xử lý. Ví dụ như ảnh dưới:



Hình 4.4. hình ảnh sau khi căn chỉnh

4.5.2. Tiến hành train model để nhận diện khuôn mặt

python src/classifier.py TRAIN Dataset/FaceData/processed Models/20180402-114759.pb Models/facemodel.pkl --batch_size 1000

Trong đó:

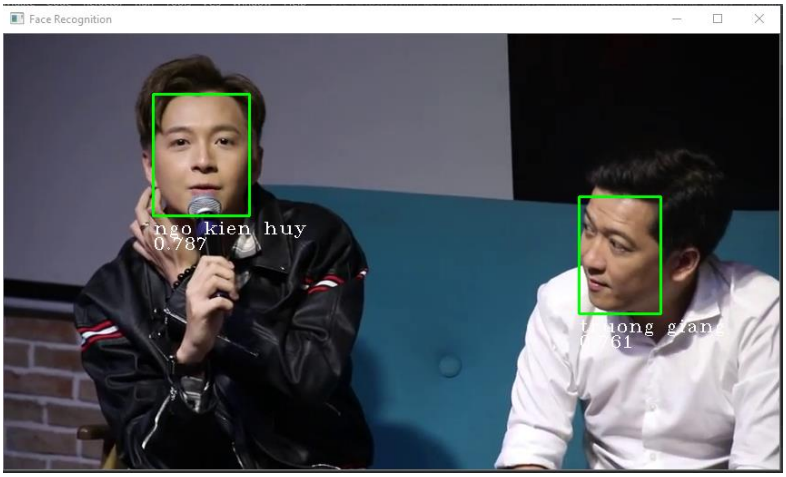
- classifier.py: sử dụng tập dữ liệu của riêng bạn để huấn luyện một trình phân loại nhận ra mọi người.
- 20180402-114759.pb: lấy từ tập dữ liệu pre-train của facenet. Với pretrained model đã được training với vài triệu ảnh nên model đã học được khả năng đặc trưng (general) là rất tốt. Khi chạy chương trình không cần training laik model.
- Facenet.py: Chức năng xây dựng mạng nhận dạng khuôn mặt.

4.5.3. Nhận diện khuôn mặt

- Nhận diện khuôn mặt qua webcam : python src/face_rec_cam.py

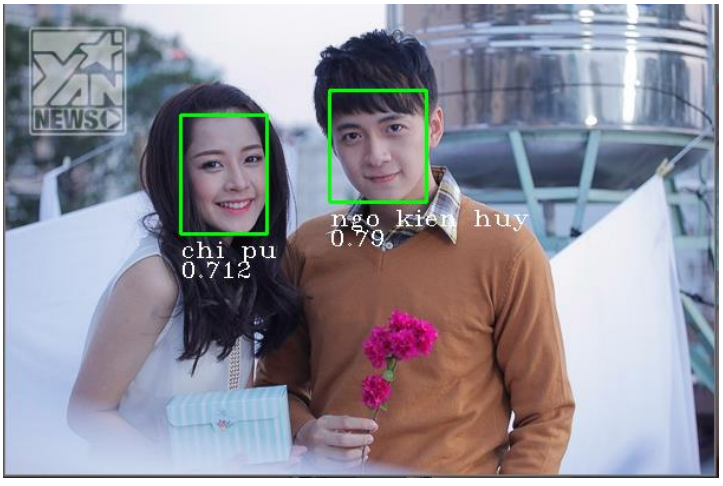
Hình 4.5. Hình ảnh sau khi nhận diện qua Webcam

- Nhận diện khuôn mặt qua video : python src/face_rec.py --path video/test1.mp4



Hình 4.6. Hình ảnh sau khi nhận diện qua video

- Nhận diện khuôn mặt qua ảnh : python src/img_dec.py



Hình 4.7. Hình ảnh sau khi nhận diện qua ảnh

KẾT LUẬN

1. Những đóng góp của bài tập lớn:

Qua nghiên cứu và thực nghiệm, bài tập lớn đã đạt được những kết quả chính như sau:

- Nghiên cứu tông quan về mạng nơron, deep learning và một số ứng dụng của deep learning.
- Nghiên cứu chi tiết mô hình học sâu DNN.
- Trên cơ sở nghiên cứu về bài toán phát hiện mặt người. Đặc biệt là thuật toán Facenet, MTCNN và SVM và mô hình *Pre-train,* áp dụng thành các thuật toán vào bài toán phát hiện mặt người
- Tuy kết quả đạt được chưa cao, nhưng đây là bước đầu để phát triển bài toán nhận dạng mặt (*face recognition*) người sau này.

2. Hướng phát triển:

Có nhiều hướng phát triển cho chương trình này, có thể phát triển cả về mặt ứng dụng và mặt thuật toán (để cải thiện hiệu quả phát hiện mặt người). Có thể xây dựng một ứng dụng chỉ cần đến phát hiện mặt người mà không cần nhận dạng. Ví dụ như một hệ thống ghép hình, ghép khuôn mặt phát hiện được vào trong một bức ảnh khác (chẳng hạn như ghép khuôn mặt của người sử dụng cho khuôn mặt của người nổi tiếng).

Ngoài ra có thể phát triển chương trình theo hướng nhận dạng khuôn mặt, xây dựng một hệ thống để học các đặc trưng của những người cần nhận dạng. Khi thực hiện, đầu tiên ta đưa qua bức ảnh qua chương trình phát hiện mặt người để phát hiện nhanh các khuôn mặt có trong ảnh, sau đấy so sách các khuôn mặt đó với các khuôn mặt mà chương trình đã được "học" từ trước, so sánh các đặc trưng của hai khuôn mặt, nếu trùng thì đưa ra thông tin về khuôn mặt được nhận dạng.

TÀI LIỆU THAM KHẢO

1. Vũ Hữu Tiệp (2018), *Machine Learning cơ bản,* NXB Khoa học kỹ thuật, Hà Nội.
2. Nils J. Nilsson (1990), *Introduction to Machine Learning*
3. C. Szegedy and e. al, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1-9, 2015.
4. F. Schroff, D. Kalenichenko and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 815-823, 2015.
5. Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, *Senior Member, IEEE,* and Yu Qiao, *Senior Member, IEEE,* "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks",pp.1604.02878