

# Ứng dụng Deep Learning trong xử lý ảnh

## 1. Cách máy tính mã hóa ảnh

Một bức ảnh trên máy tính được mã hóa dưới dạng một ma trận điểm ảnh với các kích thước:

- Width : kích thước bề ngang
- Height : kích thước bề dọc
- Depth (channel) : số kênh màu , có thể nhận một trong các giá trị:
  - 1 : ảnh gray,
  - 3 : ảnh màu RGB,
  - 4 : ảnh màu với kênh alpha (trong suốt) RGBA



Cách máy tính mã hóa hình ảnh

Mỗi điểm trong ảnh (pixel) có một giá trị màu riêng. Với ảnh đen trắng, mỗi pixel được đặc trưng bởi một giá trị cường độ sáng từ 0 đến 255, với 0 là đen hoàn toàn, 255 là trắng hoàn toàn. Với ảnh màu, mỗi pixel có 3 giá trị cường độ màu R,G,B, mỗi giá trị màu cũng nằm trong khoảng từ 0 đến 255.

Với ảnh có kênh Alpha, giá trị alpha dùng để thể hiện mức độ trong suốt của mỗi pixel. Giá trị này cũng nằm trong khoảng từ 0 đến 255, với 0 là trong suốt hoàn toàn (background), 255 là che phủ hoàn toàn. Giá trị alpha này chỉ có ý nghĩa khi vẽ ảnh lên trên một ảnh khác.

Một số định dạng ảnh phổ biến:

- Ảnh BMP : loại ảnh không nén, có nhiều định dạng con, nhưng thường sử dụng nhất là dạng 24bit tương đương với ảnh màu RGB
- Ảnh PNG : ảnh đã nén, có thể có kênh alpha (RGBA), hoặc không có kênh alpha (RGB)
- Ảnh JPG : ảnh đã nén, không có kênh alpha ( RGB)

## 2. Thư viện xử lý ảnh của Python

Python có nhiều thư viện xử lý ảnh khác nhau nhưng phổ biến nhất là PIL và OpenCV.

Cài đặt thư viện:

```
pip install pillow opencv-python
```

Một số hàm xử lý ảnh cơ bản:

- Đọc ảnh từ file :

### ■ PIL :

```
import PIL
img = PIL.Image.open('sample.jpg')
```

Hàm trên trả về một đối tượng dạng Image của PIL. Để thao tác trên dữ liệu pixel của ảnh, cần lấy được mảng numpy chứa thông tin màu của các pixel trong ảnh:

```
import numpy as np
img_data = np.array(img)
```

Theo mặc định, ảnh do PIL tải từ file sẽ có 3 (hoặc 4) kênh màu, hay mảng *img\_data* ở trên sẽ có kích thước *height x width x 3 (hoặc 4)*. Nếu cần tải ảnh và chuyển sang dạng gray, có thể sử dụng hàm *convert* của đối tượng PIL.Image

```
img = PIL.Image.open('sample.jpg').convert('L')
img_data = np.array(img) # img_data.shape ~ (height, width)
```

### ■ OpenCV:

```
import cv2
img_data = cv2.imread('sample.jpg')
```

Khác với PIL, lệnh đọc ảnh từ file của openCV trả về ngay một mảng numpy chứa giá trị màu của các pixel. Một điểm cần lưu ý là thứ tự các màu của mỗi điểm trong mảng này là BGR, trong khi phần lớn các chương trình xử lý ảnh dùng thứ tự RGB. Để chuyển đổi mảng pixel do openCV trả về sang dạng RGB, có thể dùng lệnh:

```
img_data = cv2.imread('sample.jpg')
img_data = cv2.cvtColor(img_data, cv2.COLOR_BGR2RGB)
```

Tương tự PIL, openCV cũng cho phép chuyển ảnh màu sang dạng gray:

```
img_data = cv2.imread('sample.jpg', 0)
```

Hoặc:

```
img_data = cv2.imread('sample.jpg')
```

```
img_data = cv2.cvtColor(img_data, cv2.COLOR_BGR2GRAY)
```

- Lưu mảng pixel thành file ảnh:

- PIL:

```
from PIL import Image
img = Image.fromarray(img_data)
img.save('output.jpg')
```

- OpenCV:

```
import cv2
cv2.imwrite('output.jpg', img_data)
```

- Thay đổi kích thước của ảnh:

- PIL : với PIL, việc thay đổi kích thước được thực hiện trên đối tượng Image

```
from PIL import Image
img = Image.open('sample.jpg')
new_img = img.resize((new_width, new_height))
new_img_data = np.array(new_img)
```

- OpenCV : với openCV, việc thay đổi kích thước được thực hiện ngay trên mảng pixel

```
import cv2
img_data = cv2.imread('sample.jpg', 0)
new_img_data = cv2.resize(img_data, (new_width, new_height))
```

- Hiển thị ảnh:

- PIL : PIL không hỗ trợ việc hiển thị ảnh trong chương trình. Có thể sử dụng matplotlib cho việc này.

```
import matplotlib.pyplot as plt
from PIL import Image
img = Image.open('sample.jpg')
plt.imshow(img)
plt.show()
```

- OpenCV:

```
import cv2
img_data = cv2.imread('sample.jpg')
cv2.imshow('Sample Image', img_data)
```

### 3. Convolutional Neural Network (CNN) và ứng dụng trong xử lý ảnh

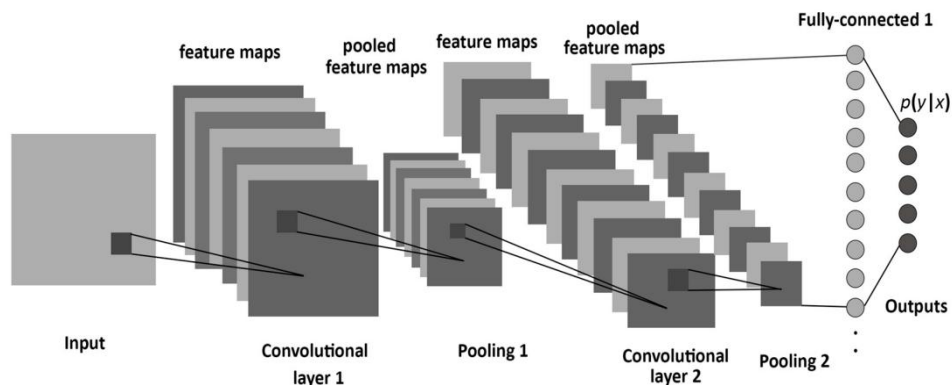
CNN là mạng neuron được dùng nhiều trong các bài toán xử lý ảnh. Các ứng dụng của CNN trong xử lý ảnh:

- Phân loại ảnh (Image Classification)
- Tìm đối tượng quan tâm trong ảnh (Object detection)
- Tách một đối tượng quan tâm ra khỏi ảnh (Image segmentation)

Giống như MLP, CNN bao gồm nhiều lớp, kết quả xử lý của lớp trước được dùng làm đầu vào cho lớp tiếp theo.

Mỗi lớp của CNN gồm các thành phần:

- 2D Convolution : đây là thành phần quan trọng nhất, có tác dụng tách ra các *feature* trong bức ảnh (sẽ được giải thích chi tiết ở phần sau)
- Activation function : tương tự như ở MLP, kết quả của 2D-Convolution được đi qua một trong các hàm activation thường dùng (relu, sigmoid, softmax, ...)
- Pooling: Có tác dụng giảm kích thước ảnh (tương tự như Down sampling), để lớp tiếp theo sẽ xử lý trên một ma trận điểm ảnh nhỏ hơn.



Cấu trúc của CNN

Ảnh được xử lý bằng CNN có kích thước đầu khá lớn (vài trăm pixel cho mỗi kích thước). Qua mỗi lớp xử lý, kích thước ảnh sẽ giảm đi (thường 2 lần). Khi kích thước ảnh giảm xuống dưới 10 pixel mỗi chiều (thường 7 hoặc 8 pixel) thì ảnh được chuyển sang dạng “Flatten”, tức toàn bộ các pixel được trải ra thành một mảng một chiều. Vector giá trị này được dùng làm đầu vào cho mạng MLP (một hoặc vài lớp), kết quả của MLP được dùng cho việc phân loại ảnh.

Chi tiết về các thành phần trong CNN:

## 2D - Convolution

Khái niệm convolution xuất phát từ trong xử lý tín hiệu. Phép tính convolution (1D) được dùng để so sánh 2 dãy số với nhau. Trong xử lý tín hiệu, người ta so sánh 2 tín hiệu (đặc trưng bởi cường độ tín hiệu theo thời gian), để xem tín hiệu thứ 2 có phải được sinh ra do hiện tượng phản xạ của tín hiệu 1 hay không.

Phép tính convolution (1D) cho phép đo sự giống nhau của 2 dãy số. Hai dãy số được xem là giống nhau (gần như) hoàn toàn nếu tồn tại một hệ số tỉ lệ chung cho từng cặp phần tử của 2 dãy:

$$x_1, x_2, \dots, x_n \sim y_1, y_2, \dots, y_n \text{ nếu } y_i = k \cdot x_i \text{ với } i = 1, 2, \dots, n$$

Định nghĩa phép tính convolution 1D như sau:

$$\text{conv}(x, y) = x_1 y_1 + x_2 y_2 + \dots + x_n y_n$$

Nếu giữ cố định  $x$  (tín hiệu nguồn), và  $y$  thay đổi sao cho

$$I = y_1^2 + y_2^2 + \dots + y_n^2 = \text{const}$$

Thì  $\text{conv}(x,y)$  sẽ lớn nhất khi tồn tại hệ số tỉ lệ  $k$  sao cho  $y_i = k.x_i$ , đây là kết quả của bất đẳng thức quen thuộc trong toán học.

Như vậy có thể dùng phép tính convolution 1D để so sánh sự giống nhau của 2 dãy số.

2D-Convolution là sự mở rộng của 1D-Convolution để dùng trong xử lý ảnh. Phép tính 2D-Convolution cho phép so sánh 2 vùng ảnh để xem chúng có phải chứa cùng một đối tượng không.

*Định nghĩa phép tính 2D-Convolution:*

Có hai mảng 2 chiều  $P, Q$  cùng kích thước  $n \times n$ . Giá trị mỗi phần tử trong 2 mảng là  $p_{ij}$  và  $q_{ij}$ , trong đó  $i, j$  là các chỉ số hàng và cột của mảng 2 chiều. Phép tính convolution 2D giữa 2 mảng được tính bằng tổng của tích các cặp phần tử tương ứng trong 2 mảng:

$$\text{conv2D}(P,Q) = \sum(p_{ij} \cdot q_{ij}) \quad \text{với } i,j = 1..n$$

Ví dụ:

$$P = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \quad Q = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$\text{conv2D}(P,Q) = 1.0 + 0.1 + 1.1 + 0.0 + 1.0 + 0.0 + 0.1 + 1.0 + 1.1 = 2$$

Phép nhân từng phần tử của 2 mảng tương tự với phép tính element-wise của numpy. Nếu  $P, Q$  là các mảng 2 chiều trong numpy thì công thức tính conv2D đơn giản là:

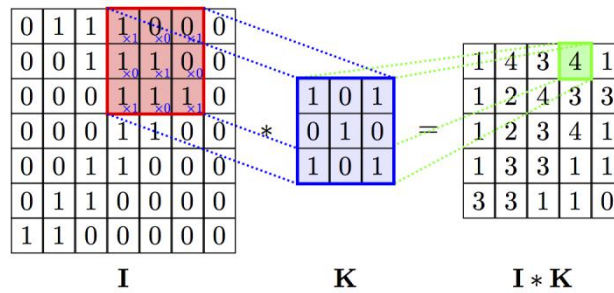
$$\text{conv2D} = \text{np.sum}(P*Q)$$

Tương tự như với 1D-Convolution, phép tính 2D-Convolution cho phép so sánh sự tương tự của 2 mảng 2 chiều. Trong xử lý ảnh, mỗi vùng ảnh được thể hiện qua một mảng 2 chiều chứa các giá trị cường độ màu của các điểm ảnh, do đó có thể dùng 2D-Convolution để so sánh 2 vùng ảnh.

Giả sử chúng ta đã có sẵn ảnh mẫu của một đối tượng (template). Chúng ta có thể kiểm tra một vùng ảnh có chứa đối tượng không bằng cách:

- Chuẩn hóa cường độ màu trung bình của vùng ảnh về giá trị tiêu chuẩn (tương đương với việc giữ  $I=\text{const}$  trong 1D-convolution)
- Thực hiện phép tính 2D-Convolution giữa ảnh mẫu và vùng ảnh đang xét. Nếu giá trị này càng lớn thì vùng ảnh đang xét càng giống ảnh mẫu.

Do ảnh đầu vào có kích thước lớn hơn so với template, việc tìm kiếm đối tượng trong ảnh phải thực hiện ở tất cả các vị trí có thể. Một cửa sổ có kích thước bằng kích thước của template sẽ trượt qua toàn bộ ảnh, tại mỗi vị trí chúng ta dùng 2D-convolution để so sánh vùng cửa sổ hiện tại với template. Nếu  $h_1, w_1$  là kích thước ảnh đầu vào,  $h_2, w_2$  là kích thước template thì chúng ta cần tính 2D-convolution tại  $(h_1-h_2+1).(w_1-w_2+1)$  vị trí.



Dùng cửa sổ trượt để tính 2D-convolution tại các vị trí trong ảnh đầu vào

Phương pháp trên trong xử lý ảnh được gọi là template-matching. Tuy nhiên phương pháp này có nhược điểm là khi vùng ảnh bị biến dạng (nghiêng, góc chụp thay đổi) thì khả năng nhận ra đối tượng trong ảnh giảm xuống đáng kể.

CNN cũng sử dụng 2D-Convolution cho việc so sánh các vùng ảnh, tuy nhiên có điểm khác:

- Mỗi phần tử 2D-Convolution chỉ so sánh một phần (bộ phận) của đối tượng trong ảnh với giá trị kernel của nó (chính là giá trị mảng 2 chiều đóng vai trò template). Nếu kết quả của phép tính 2D-Convolution lớn hơn một ngưỡng (đủ để kích hoạt hàm activation) thì giá trị của nó được truyền tiếp sang lớp sau. Trong trường hợp này, phần tử 2D-Convolution này đã tìm ra một *feature* của đối tượng.
- Mỗi lớp 2D-Convolution của CNN không chỉ có một mà có nhiều bộ giá trị kernel/template để khớp các bộ phận (*feature*) khác nhau trên đối tượng. Số kernel trong một lớp được gọi là channel, tương tự như 3 channel R,G,B của ảnh đầu vào
- CNN không chỉ có một mà có nhiều (thậm chí rất nhiều) lớp 2D-Convolution. Lớp càng về sau càng xử lý ảnh đã downsample ở kích thước nhỏ hơn, điều này tương đương với việc chúng tìm cách tách ra các feature có kích thước (gốc) lớn hơn mà các lớp đầu chưa tách ra được.

Để minh họa, chúng ta tìm hiểu một bài toán phân loại ảnh đơn giản, cần phân loại 2 nhóm ảnh trong đó nhóm thứ nhất chứa ảnh một hình vuông, nhóm thứ 2 không chứa hình vuông nào trong ảnh.



Một số ảnh chứa hình vuông

Ví dụ, một ảnh chứa hình vuông có ma trận điểm ảnh như sau:

0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0

Chúng ta cần chọn kernel (template) cho 2D-convolution để phát hiện ra các hình vuông trong ảnh. Kích thước kernel được chọn là 3x3 (phổ biến trong các model CNN). Với kích thước nhỏ như vậy, chúng ta không thể chọn kernel để phản ánh nội dung của một hình vuông, mà chỉ có thể dùng nó phản ánh một phần trên hình vuông.

Các hình vuông đều có các cạnh song song với 2 trục, do đó cách đơn giản nhất là chọn kernel thể hiện một đoạn thẳng song song với một trong 2 trục, ví dụ:

$$K = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Sử dụng phương pháp cửa sổ trượt, chúng ta tính được kết quả của 2D-convolution như sau:

2	3	3	3	3	2
2	3	3	3	3	2
2	2	1	1	2	2
2	2	1	1	2	2
2	3	3	3	3	2
2	3	3	3	3	2

Kết quả 2D-convolution giữa ảnh gốc và kernel đã chọn

Những vùng có kết quả 2D-convolution càng lớn là những vùng càng giống với kernel. Do chúng ta chọn kernel là đoạn thẳng nằm ngang nên vùng giống kernel nhất là 2 cạnh trên và dưới của hình vuông (kết quả 2D-convolution bằng 3).

Kết quả 2D-convolution sẽ được đi qua hàm activation (thường là relu), để chỉ giữ lại những vùng có kết quả cao. Ở hình trên, nếu muốn giữ lại các vùng mà kết quả 2D-convolution có giá trị bằng 3, thì cách đơn giản nhất là cho bảng kết quả trên đi qua hàm relu với offset bằng 2, kết quả thu được:

0	1	1	1	1	0
0	1	1	1	1	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	1	1	1	0
0	1	1	1	1	0

Kết quả 2D-convolution sau khi đi qua hàm activation relu với offset bằng 2

Có thể thấy vùng được giữ lại chính là 2 cạnh nằm ngang của hình vuông, do chúng có đặc điểm tương tự như với kernel. Trong trường hợp này, kernel đã tách được một *feature* của hình vuông đó là các cạnh nằm ngang.

## Pooling

Như đã nói ở phần trước, mục đích của pooling là làm giảm kích thước ảnh xuống để bước sau xử lý ít hơn. Quan sát kết quả của 2D-convolution sau khi đi qua hàm activation ở ví dụ ở trên, chúng ta nhận thấy các điểm kích hoạt thường nằm gần nhau. Điều này là do trên thực tế các pixel lân cận trong một bức ảnh thường có màu sắc gần nhau, nếu một kernel kích hoạt một vùng cửa sổ trong ảnh thì cũng có xu hướng kích hoạt cửa sổ ở vị trí lân cận. Để giảm khối lượng xử lý cho bước sau, có thể nhóm các điểm kích hoạt lân cận nhau làm một. Việc làm này gọi là *pooling*, theo đó bảng giá trị kích hoạt được giảm kích thước xuống (thường mỗi chiều 2 lần), và mỗi hình vuông 2x2 trong bảng sẽ được thay thế bằng một ô đại diện duy nhất. Giá trị ô đại diện này có thể tính theo một các cách :

- Trung bình của 4 ô trong hình vuông 2x2 . Cách này gọi là *Average Pooling*
- Giá trị của ô lớn nhất. Cách này gọi là *Max Pooling*

Ví dụ : vùng kích hoạt

0	1
0	1

Dùng Max Pooling : 4 ô trên sẽ được thay bằng 

1
---

Nếu áp dụng cho toàn bộ bảng kích hoạt:

0	1	1	1	1	0
0	1	1	1	1	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	1	1	1	0
0	1	1	1	1	0

→

1	1	1
0	0	0
1	1	1

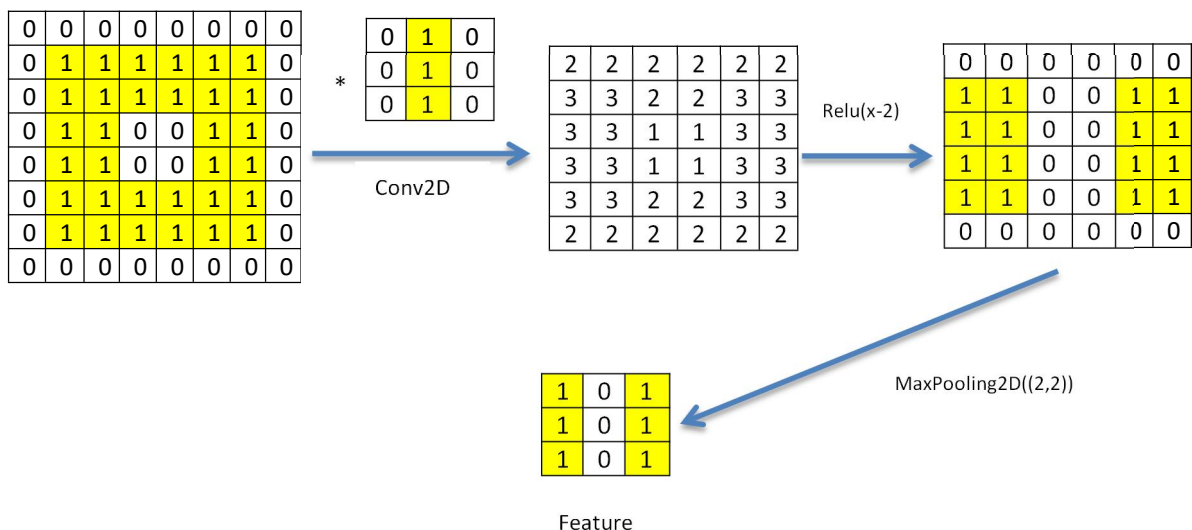
Thực hiện MaxPooling 2x2 cho bảng giá trị kích hoạt

Có thể thấy, mặc dù bảng giá trị kích hoạt giảm kích thước mỗi chiều xuống 2 lần, vùng kích hoạt (màu vàng) vẫn đại diện cho 2 cạnh nằm ngang của hình vuông, dù ở tỉ lệ nhỏ hơn.

Tương tự, chúng ta có thể dùng thêm một kernel để phát hiện 2 cạnh dọc của hình vuông như sau:

$$K_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

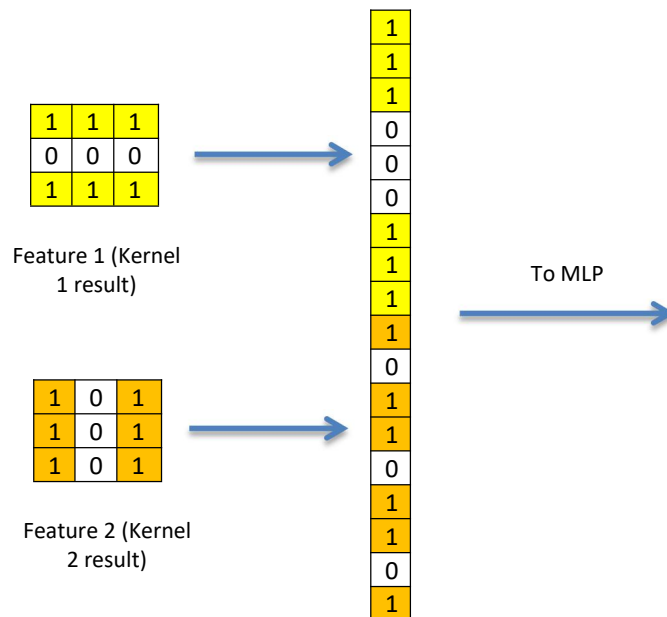
Kết quả xử lý của kernel này:



## Flatten

Khi kích thước của ảnh (hay chính xác là các bảng thuộc tính) đã giảm đủ nhỏ (thường kích thước mỗi chiều nhỏ hơn 10), toàn bộ các mảng 2 chiều này sẽ được trải thành một mảng một chiều để đi vào một mạng MLP. Quá trình chuyển dữ liệu từ các mảng 2D thành mảng 1D gọi là Flatten

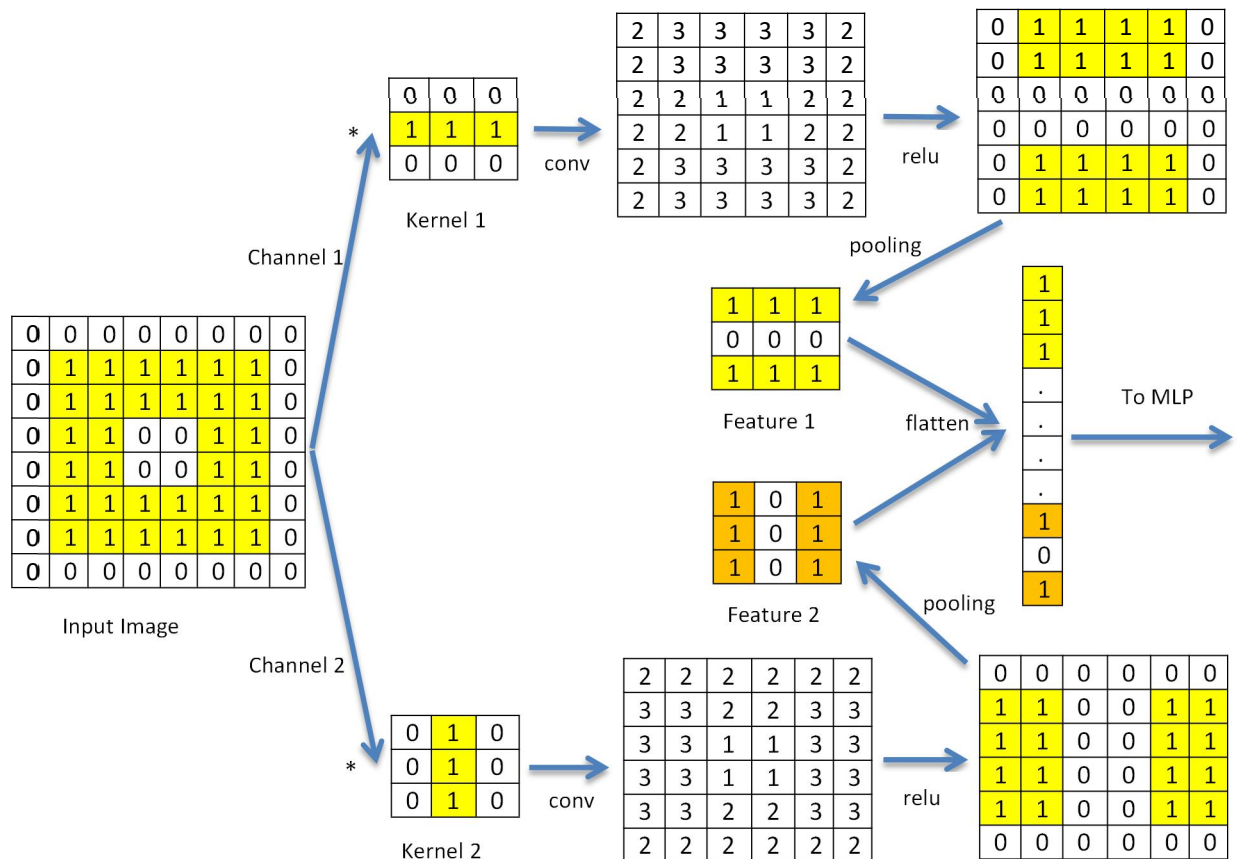




Qua trình chuyển các bảng thuộc tính thành vector 1 chiều (Flatten)

Như vậy thay vì việc đưa toàn bộ ảnh ban đầu có kích thước 8x8 vào MLP, việc dùng 2D-Convolution và Pooling giúp chúng ta tách ra 2 Feature có kích thước 3x3, vector đưa vào MLP xử lý chỉ có kích thước 2x3x3=18 thay vì 8x8=64. Trên thực tế, các ảnh thường có kích thước mỗi chiều vài trăm pixel, nếu đưa toàn bộ ảnh vào MLP thì số đầu vào có thể lên tới hàng triệu, số tham số của MLP sẽ rất lớn, quá trình training lâu và hiện tượng overfitting sẽ xảy ra nhiều.

Tóm tắt toàn bộ quá trình xử lý trong CNN với ví dụ trên có thể được minh họa bằng sơ đồ sau:



Phía trên là cách chúng ta chọn kernel theo suy nghĩ của con người. Để xem cách máy tự học kernel từ dữ liệu huấn luyện, chúng ta xem xét ví xây dựng CNN bằng keras sau: mô hình CNN tối giản này chỉ gồm 1 lớp 2D-Convolution, lớp này chỉ có 1 kernel (1 channel) duy nhất:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D

model = Sequential()

model.add(Conv2D(1, (3, 3), input_shape=(40, 40, 1), activation='relu'))

model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Flatten())
model.add(Dense(num_classes, activation='softmax'))
```

Sau khi tải ảnh vào các tập dữ liệu (Xtrain,ytrain), (Xtest, ytest), chúng ta huấn luyện mạng trên để phân biệt 2 nhóm:

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])

model.fit(Xtrain, ytrain, validation_data=(Xtest, ytest), epochs=20,
          batch_size=200, verbose=2)
```

Sau khi huấn luyện xong, chúng ta xem giá trị kernel đã được học ra sao:

```
import matplotlib.pyplot as plt
```

```

weights = model.layers[0].get_weights()
K, _ = weights
K = K.reshape((3, 3))
Kmin = np.min(K)
Kmax = np.max(K)
K = (K-Kmin)/(Kmax-Kmin)*255

kernel_img = np.array(K, dtype=np.uint8)
plt.imshow(kernel_img, cmap='gray')
plt.show()

```

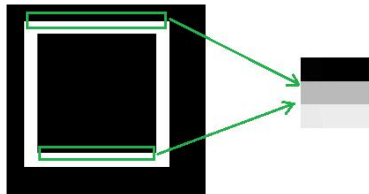
Giá trị kernel của thành phần 2D-Convolution được hiển thị bằng hình ảnh có dạng như sau:



Ảnh kernel (3x3) của mô hình CNN sau huấn luyện

Nhìn vào ảnh kernel, có thể thấy nó tìm cách phát hiện các vùng mà màu sắc tăng dần theo chiều từ trên xuống dưới. Nếu là với hình vuông trắng trên nền đen thì 2 vùng có tính chất này là:

- Nửa trên của cạnh nằm ngang phía trên
- Nửa trên của cạnh nằm ngang phía dưới



Các vùng sẽ được kích hoạt bởi kernel

Để kiểm chứng lại chúng ta sẽ hiển thị bằng giá trị kích hoạt trong mô hình CNN. Bảng này tương ứng với giá trị nằm sau lớp MaxPooling và trước lớp Fully Connected.

```

import matplotlib.pyplot as plt
from keras.models import Model

activation_layer = Model(model.input, model.layers[1].output)
activation_value = activation_layer.predict(Xtest)

value_min = np.min(activation_value)
value_max = np.max(activation_value)
activation_value = (activation_value - value_min)/(value_max - value_min)

activation_value = np.array(255*activation_value, dtype=np.uint8)
plt.imshow(activation_value[0,:,:,:])
plt.show()

```

Giá trị kích hoạt của một số ảnh chứa hình vuông sẽ như sau:



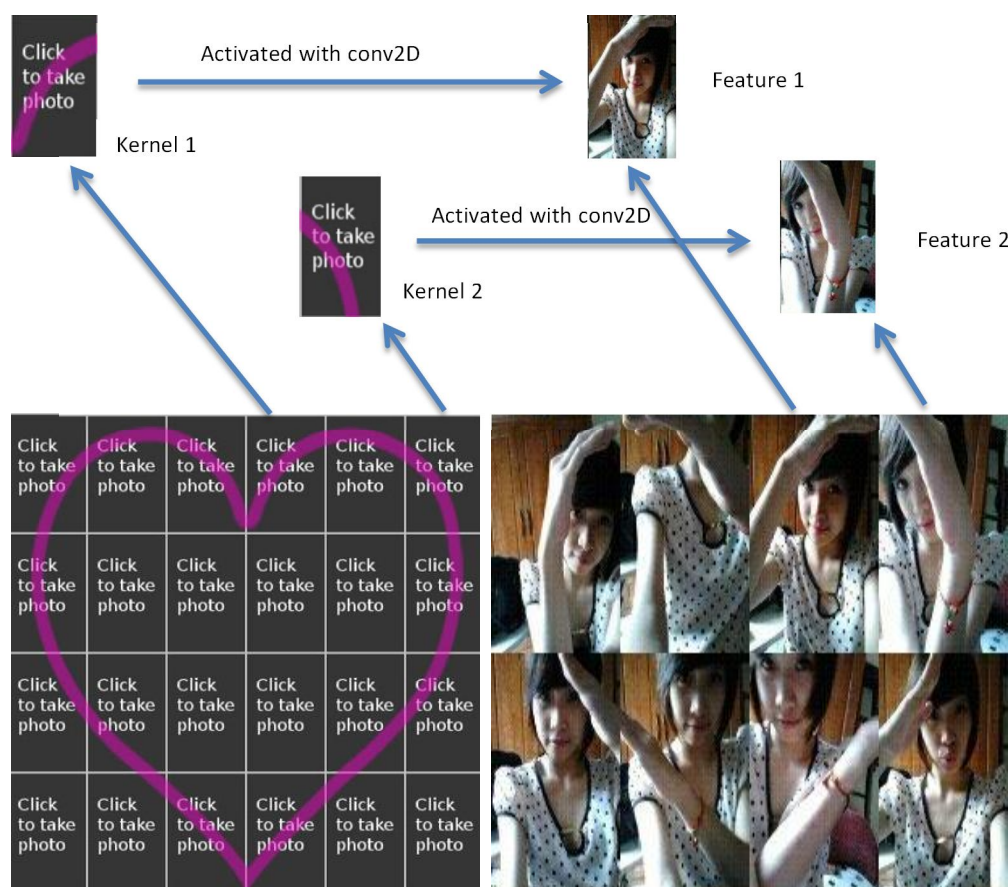
Giá trị kích hoạt sau lớp MaxPooling của một số ảnh chứa hình vuông.

Với các ảnh không chứa hình vuông, bảng giá trị kích hoạt sẽ bằng 0 tại hầu hết các điểm, do không có vùng nào có tính chất giống với kernel đã được tạo ra.

Để hiểu rõ hơn về cách CNN phát hiện các *feature* của đối tượng trong ảnh, chúng ta hãy so sánh với một ví dụ sau.

#### Trò chơi Body Symbol:

Trò chơi body symbol cho phép người chơi chụp ảnh với các cử chỉ khác nhau (dùng tay thể hiện), sau đó ghép các bức ảnh lại thành một bức ảnh chứa hình ảnh của một biểu tượng.



Có thể thấy hình ảnh biểu tượng gốc (bên trái) và bức ảnh ghép các cử chỉ (bên phải) không giống nhau hoàn toàn, nhưng chúng ta vẫn nhận ra chúng chứa cùng một nội dung. Mỗi mảnh nhỏ trong ảnh gốc giống như một kernel của CNN, mỗi ảnh nhỏ trong ảnh ghép bên phải giống như một feature. Các kernel sẽ phát hiện ra các vùng feature có đặc điểm hình học tương tự với nó nhờ phép tính 2D-convolution. Ngoài ra, vị trí tương đối của các feature với nhau cũng có vài trò quan trọng, vì nó ảnh hưởng đến các vùng giá trị tương ứng của vector 1D sau khi được flatten. Như vậy, CNN kết luận đối tượng có tồn tại trong ảnh không dựa trên cả số lượng feature được phát hiện lẫn vị trí tương đối của các feature với nhau.

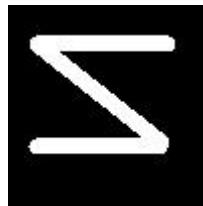
## Hiện tượng False Positive

Thông thường khi đánh giá kết quả nhận dạng của CNN (hay các mô hình Machine Learning), chúng ta quan tâm đến tỉ lệ mẫu không nhận dạng đúng trên tổng số mẫu. Sai số này là sai số âm (False

Negative). Ngoài ra còn một hiện tượng sai số khác nữa, đó là mô hình nhận ra có tồn tại của đối tượng trong khi thực tế không có. Hiện tượng này gọi là *False Positive*

Hiện tượng False Positive xảy ra khá thường xuyên với CNN. Thông thường khi phân biệt các nhóm đối tượng ảnh với nhau, CNN không phát hiện toàn bộ các feature có thể có của mỗi loại đối tượng, mà chỉ phát hiện đủ số feature để phân biệt các nhóm với nhau. Nếu số lượng ảnh mẫu dùng training không nhiều, CNN chỉ phát hiện một vài feature của đối tượng vì chỉ cần như vậy đã đủ để phân loại chính xác trên tập ảnh mẫu. Tuy nhiên khi sử dụng mô hình, một số ảnh khác dạng với các ảnh trong tập mẫu vẫn có đủ các feature này và CNN sẽ nhận nhầm nó có chứa đối tượng thật. Hiện tượng này cũng là một dạng của *overfitting*.

Ví dụ, với CNN đã được train bằng keras ở trên, chỉ cần ảnh chứa 1 feature là 2 cạnh nằm ngang của hình vuông, nó đã hiểu ảnh đó là hình vuông. Do vậy, nếu gặp một ảnh có dạng như sau:



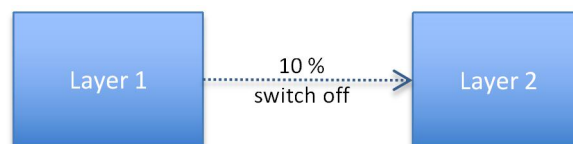
Một ảnh có thể “đánh lừa” mô hình CNN đã được huấn luyện để nó nghĩ là hình vuông

thì mô hình CNN đã được huấn luyện cũng sẽ nghĩ đây là một hình vuông. Lí do là trong tập ảnh huấn luyện không có ảnh nào tương tự như ảnh này, trong khi ảnh này có đủ feature của các hình vuông mà CNN đã học được.

#### Khắc phục hiện tượng False Positive:

Để khắc phục hiện tượng *False Postive* trong CNN có thể sử dụng một trong các biện pháp sau:

- Tăng số lượng ảnh huấn luyện : Thông thường số ảnh huấn luyện cho mỗi nhóm ít nhất phải bằng 1000 hoặc nhiều hơn
- Dùng ảnh feedback để train lại CNN : mỗi lần CNN phát hiện nhầm thì ảnh nhầm đó được giữ lại để train cho lần sau
- Sử dụng Dropout : Lớp dropout là một lớp nối trung gian giữa 2 lớp của CNN. Nó có tác dụng ngắt kết nối ngẫu nhiên giữa 2 lớp trước và sau nó theo một xác suất nhất định. Ví dụ Dropout( $p=0.1$ ) sẽ ngắt kết nối với tỉ lệ 10%, còn 90% trường hợp còn lại dữ liệu vẫn được truyền giữa 2 lớp trước & sau của Dropout



Tác dụng của Dropout : Dropout sẽ bắt CNN phải tìm ra nhiều thuộc tính đặc trưng của đối tượng hơn. Nếu không có dropout, CNN chỉ tìm ra tối thiểu các thuộc tính để phân biệt các nhóm trong tập training. Nếu có dropout, CNN sẽ phải tìm ra nhiều thuộc tính hơn, vì trong trường hợp một thuộc tính bị ngắt (do dropout ngẫu nhiên), lớp sau vẫn có thuộc tính khác để đưa ra kết quả phân loại đúng. Việc này gần giống với việc tăng sự dư thừa để có thêm sự ổn định trong các hệ thống xử lý thông tin.

## Một vài điểm thêm về kernel:

### Một số dạng kernel đặc biệt:

Bên cạnh các kernel đóng chức năng khai thác các feature trong ảnh, còn một số kernel giữ các chức năng sau:

- Identity kernel : Identity kernel không làm thay đổi giá trị của ảnh gốc qua phép tính 2D-convolution:

$$K_I =$$

0	0	0
0	1	0
0	0	0

Identity kernel thường xuất hiện ở các lớp trung gian. Tác dụng của nó là truyền những feature đã được phát hiện ở lớp trước sang lớp sau, do đó những feature đã được phát hiện này sẽ không bị mất đi do phép tính 2D-convolution

- Edge detection Kernel:

$$K_E =$$

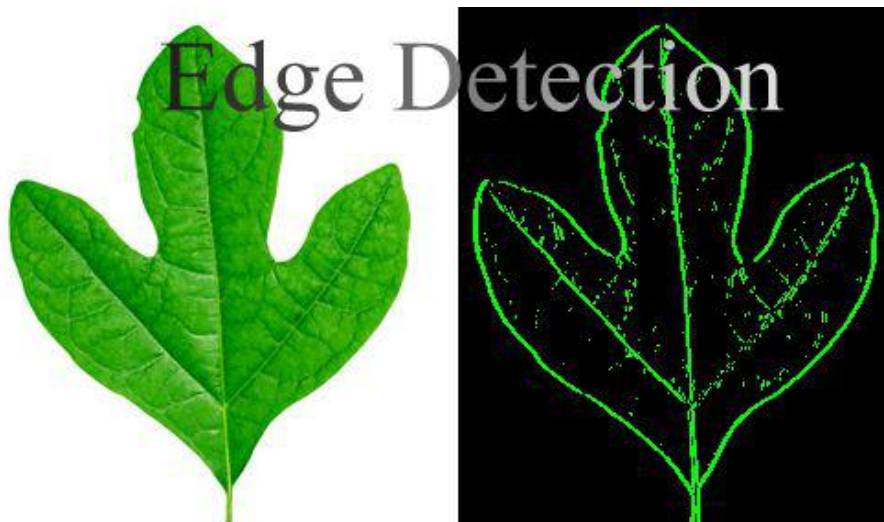
0	-1	0
-1	0	1
0	1	0

Tác dụng của Edge detection là chuyển một ảnh gốc sang dạng đường biên của ảnh. Ví dụ một hình tròn đặc sẽ chuyển thành một đường tròn. Công thức của edge detection tại một điểm có tọa độ  $(x,y)$  có thể được viết thành :

$$K_E * I = [I(x+1,y) - I(x-1,y)] + [I(x,y+1) - I(x,y-1)]$$

Biểu thức trên gồm 2 thành phần: độ thay đổi cường độ màu theo phương ngang, và độ thay đổi cường độ màu theo phương thẳng đứng. Những điểm trên đường biên là những điểm có cường độ màu khác nhiều so với các điểm lân cận do đó, biểu thức trên sẽ có giá trị lớn tại những điểm nằm trên biên của các hình.

Edge detection giúp cho việc phát hiện feature sẽ thực hiện trên các đường biên của đối tượng, bỏ đi phần đặc bên trong đối tượng vốn không chứa nhiều thông tin về đặc điểm hình học của đối tượng.



Một ví dụ về dùng kernel để phát hiện đường biên.

Lưu ý : các kernel trên được chủ ý lựa chọn cho các bài toán xử lý ảnh theo phương pháp truyền thống. Với CNN, các kernel được khởi tạo ngẫu nhiên hoàn toàn. Quá trình training, CNN sẽ tự lựa chọn kernel phù hợp cho tập dữ liệu vào.

#### Lựa chọn kích thước kernel :

Các mô hình CNN thường dùng kích thước kernel bằng 3x3 hoặc 5x5 (đôi khi có dùng 7x7). Kernel có kích thước lớn thường đặt ở các lớp đầu của CNN, các lớp sau thường dùng kernel có kích thước nhỏ hơn

#### Ghép liên tiếp nhiều Kernel:

Đôi khi trong một số mô hình CNN, người ta ghép nhiều kernel liên tiếp với nhau. Về lí thuyết, việc ghép 2 kernel K1, K2 tương đương với việc dùng một kernel:

$$K = \text{conv2D}(K_1, K_2)$$

Kích thước của kernel tương đương này bằng:

$$n = n_1 + n_2 - 1$$

Như vậy việc ghép 2 kernel 3x3 tương đương với một kernel 5x5, ghép 3 kernel 3x3 tương đương với một kernel 7x7. Tuy nhiên việc ghép kernel sẽ giúp giảm số tham số sử dụng, ví dụ

- Hai kernel 3x3 , số tham số :  $2 \times 3 \times 3 = 18$
- Một kernel 5x5, số tham số :  $5 \times 5 = 25$

## 4. Bài toán phân loại ảnh

Bài toán phân loại ảnh là bài toán cơ bản đầu tiên trong ứng dụng của CNN vào xử lý ảnh. Bài toán này yêu cầu cho biết một ảnh quan tâm có nội dung thuộc vào một trong các nhóm nào trong các danh sách cho trước.

# Xây dựng chương trình phân loại ảnh bằng CNN với Keras

Keras cho phép xây dựng mô hình CNN nhanh chóng với các lớp có sẵn tương ứng với các thành phần của CNN.

- `keras.layers.Conv2D` : tương ứng với phần tử Conv2D
- `keras.layers.MaxPooling2D` : tương ứng với phần tử MaxPooling2D
- `keras.layers.Flatten` : tương ứng với phần tử Flatten
- `keras.layers.Dropout` : tương ứng với phần tử Dropout

Một vài chi tiết về các lớp thành phần dùng trong CNN của keras:

- **Conv2D:**  
Các tham số thường sử dụng:
  - `filters` : số bộ lọc (kernel) sử dụng, tương đương với số kênh 2D-Convolution.
  - `kernel_size` : kích thước kernel, ví dụ (3,3) , (5,5) ...
  - `strides` : bước của cửa sổ trượt khi tính 2D-Convolution. Giá trị mặc định là (1,1), tức cửa sổ trượt sẽ quét qua từng điểm trong ảnh gốc. Nếu `strides` lớn hơn 1, ví dụ (2,2), cửa sổ trượt sẽ quét qua các điểm trong ảnh với bước dịch mỗi chiều bằng 2, kết quả sẽ làm giảm kích thước đầu ra mỗi chiều 2 lần.
  - `padding` : nhận một trong 2 giá trị : 'valid' (mặc định) hoặc 'same'
    - `Padding 'valid'` : (không padding), kích thước đầu ra sẽ nhỏ hơn kích thước ảnh vào một chút , cụ thể bằng  $(w_1-w_2+1, h_1-h_2+1)$  trong đó  $(w_1, h_1)$  là kích thước ảnh vào, còn  $(w_2, h_2)$  là kích thước kernel.
    - `Padding 'same'` : cho đầu ra có cùng kích thước với đầu vào. Những điểm ở vùng biên của ảnh đầu vào không đủ lân cận để tính 2D-Convolution sẽ được padding với các giá trị bằng 0 cho vùng lân cận bị thiếu.
  - `data_format` : nhận một trong 2 giá trị : 'channels\_last' (mặc định) hoặc 'channels\_first'
    - `'channels_last'` : dữ liệu vào được thể hiện bằng một mảng 4 chiều (N, height, width, channel) trong đó N là số mẫu ảnh, height là kích thước chiều dọc ảnh, width là kích thước chiều ngang ảnh, channel là số kênh màu của ảnh (3 với ảnh RGB, 1 với ảnh gray).
    - `'channels_first'` : dữ liệu vào được thể hiện bằng một mảng 4 chiều (N, channel, height, width)

Các tham số khác có thể tham khảo tại trang chủ của Keras

- **MaxPooling2D:**  
Các tham số thường sử dụng:
  - `pool_size` : kích thước vùng ô vuông sẽ được thực hiện pooling, ví dụ `pool_size = (2,2)` tương đương với giá trị 4 ô của từng hình vuông 2x2 sẽ được gộp làm một
  - `strides, padding, data_format` : tương tự như với Conv2D. Thông thường, các giá trị này để mặc định bằng None



- Flatten:  
Tham số:
  - data\_format: tương tự như trong Conv2D.
- Dropout:  
Tham số:
  - rate: tỉ lệ dropout, chọn từ 0 đến 1

Ví dụ: Xây dựng một mô hình CNN để xử lý ảnh đầu vào có kích thước 40x40, 3 channel RGB, số nhóm ảnh cần phân loại là 2. Phần xử lý gồm 2 lớp Convolution, mỗi lớp có 2 kernel kích thước 3x3.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.optimizers import Adam

model = Sequential()
model.add(Conv2D(2, (3, 3), input_shape=(40, 40, 3),
                activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(2, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(2, activation='softmax'))

adam = Adam(lr=1e-3, decay=1e-6)
model.compile(loss='categorical_crossentropy', optimizer=adam,
              metrics=['accuracy'])
```

Sau khi tạo mô hình, chúng ta có thể xem thông tin các lớp của mô hình bằng lệnh `model.summary()`:

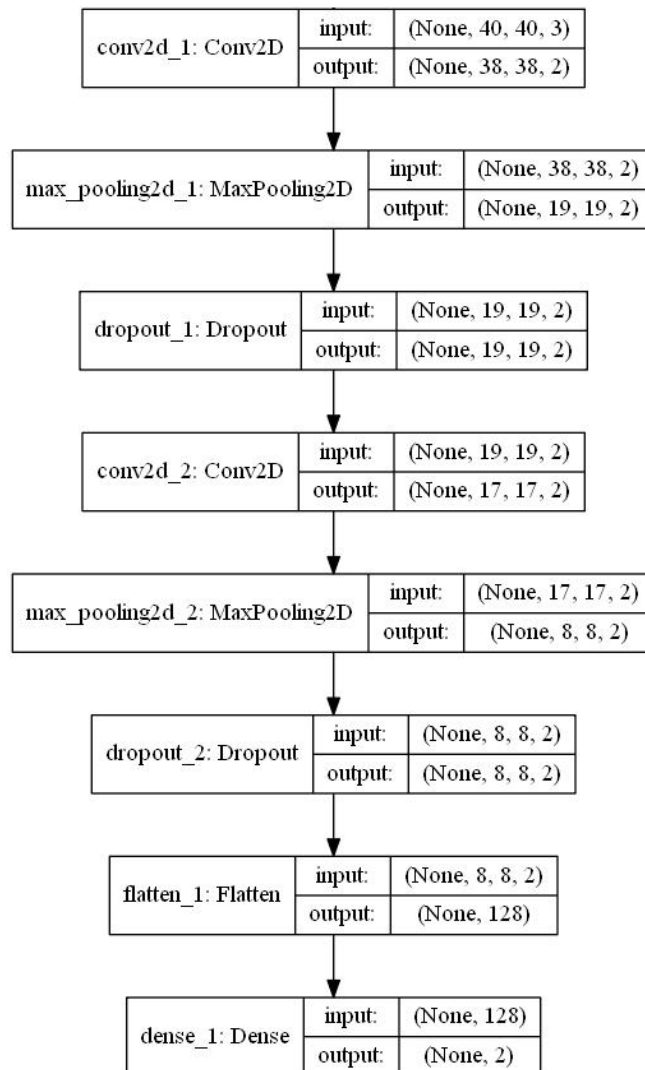
```
>> print(model.summary())
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 38, 38, 2)	56
max_pooling2d_1 (MaxPooling2D)	(None, 19, 19, 2)	0
dropout_1 (Dropout)	(None, 19, 19, 2)	0
conv2d_2 (Conv2D)	(None, 17, 17, 2)	38
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 2)	0
dropout_2 (Dropout)	(None, 8, 8, 2)	0
flatten_1 (Flatten)	(None, 128)	0
dense_1 (Dense)	(None, 2)	258

=====  
 Total params: 352  
 Trainable params: 352  
 Non-trainable params: 0

Chúng ta cũng có thể vẽ mô hình dưới dạng hình ảnh để dễ quan sát:

```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
```



Ảnh mô hình đã xây dựng

Huấn luyện mô hình:

```
model.fit(Xtrain, Ytrain, validation_data=(Xval, Yval), epochs=...)
```

Trong đó Xtrain, Ytrain là dữ liệu huấn luyện. Xval, Yval là dữ liệu phục vụ cho việc kiểm định mô hình, thường được trích một phần từ dữ liệu test (Xtest, Ytest). Mục đích của việc kiểm định này là để xem quá trình huấn luyện khi nào có thể dừng lại : Sau mỗi chu kì huấn luyện (epoch), độ chính xác của mô hình sẽ được ước tính trên bộ (Xval, Yval), nếu độ chính xác không tăng sau nhiều chu kì thì quá trình huấn luyện có thể dừng lại.

Để hiểu rõ hơn, chúng ta sẽ xây dựng một chương trình để xác định trong ảnh có người hay không.



Ảnh có người (bên trái) và ảnh không có người (bên phải)

### Chuẩn bị dữ liệu:

Dữ liệu ảnh có thể download từ file [Images.zip](#) chứa khoảng 1500 ảnh có người và 2000 ảnh không có người. File này cần download về và giải nén thành thư mục *Images* rồi đặt trong thư mục cùng với các chương trình chạy.

Dữ liệu ảnh có thể được đọc từ file jpg ngay trong chương trình huấn luyện, tuy nhiên điều này thường mất thời gian, do đó chúng ta dùng một chương trình riêng để chuyển file ảnh thành file dữ liệu của numpy. Nếu phải train lại nhiều lần thì mỗi lần sẽ không mất thời gian đọc ảnh từ file jpg, mà sẽ đọc ngay dữ liệu mảng numpy đã chuyển đổi.

Chương trình chuẩn bị dữ liệu (`prepare_data.py`):

```
# prepare_data.py
import os
import numpy as np
from PIL import Image
from keras.utils import to_categorical

data_dir = 'Images'
categories = ['Person', 'NoPerson']
num_classes = len(categories)

img_size = 224
train_factor = 0.7

Xtrain = []
Ytrain = []
Xtest = []
Ytest = []

for i, category in enumerate(categories):
    img_files = os.listdir(os.path.join(data_dir, category))

    for j, f in enumerate(img_files):
        if not f.lower().endswith('.jpg'):
            continue

        img_path = os.path.join(data_dir, category, f)
        img = Image.open(img_path).resize((img_size, img_size))

        img_data = np.array(img)
        label = to_categorical(i, num_classes)

        if j < len(img_files) * train_factor:
```

```

        Xtrain.append(img_data)
        Ytrain.append(label)
    else:
        Xtest.append(img_data)
        Ytest.append(label)

Xtrain = np.array(Xtrain, dtype=np.uint8)
Ytrain = np.array(Ytrain, dtype=np.uint8)
Xtest = np.array(Xtest, dtype=np.uint8)
Ytest = np.array(Ytest, dtype=np.uint8)

Xtrain.tofile('Xtrain.np')
Ytrain.tofile('Ytrain.np')
Xtest.tofile('Xtest.np')
Ytest.tofile('Ytest.np')

```

Chương trình chuẩn bị dữ liệu trên chỉ cần chạy 1 lần, nó sẽ tạo ra các file dữ liệu Xtrain.np, Ytrain.np, Xtest.np, Ytest.np

## Xây dựng và huấn luyện mô hình

Sau khi có dữ liệu, chúng ta xây dựng chương trình để xây dựng và huấn luyện mô hình. Việc huấn luyện CNN thường đòi hỏi máy có GPU và cài đặt các thư viện cuda, cudnn, tensorflow-gpu, ... Để biết chi tiết về cách cài đặt thư viện để sử dụng được GPU, có thể xem [hướng dẫn tại trang chủ của tensorflow](#).

```

# train.py

import os
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint

categories = ['Person', 'NoPerson']
num_classes = len(categories)

img_size = 224

def loadData():
    Xtrain = np.fromfile('Xtrain.np', dtype=np.uint8)
    Ytrain = np.fromfile('Ytrain.np', dtype=np.uint8)
    Xtest = np.fromfile('Xtest.np', dtype=np.uint8)
    Ytest = np.fromfile('Ytest.np', dtype=np.uint8)

    Xtrain = Xtrain.reshape(-1, img_size, img_size, 3)
    Xtrain = Xtrain.astype('float32')/255
    Ytrain = Ytrain.reshape(-1, 2).astype('float32')

    Xtest = Xtest.reshape(-1, img_size, img_size, 3)
    Xtest = Xtest.astype('float32')/255
    Ytest = Ytest.reshape(-1, 2).astype('float32')

    return Xtrain, Ytrain, Xtest, Ytest

def createModel():
    model = Sequential()
    model.add(Conv2D(16, (5, 5), input_shape=(img_size, img_size, 3),
        activation='relu'))

    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(0.3))

```

```

model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

opt = Adam(lr=0.001, decay=1e-6)
model.compile(loss='categorical_crossentropy', optimizer=opt,
              metrics=['accuracy'])

return model

Xtrain, Ytrain, Xtest, Ytest = loadData()
model = createModel()
print(model.summary())

early_stopping = EarlyStopping(monitor='val_acc', patience=10)

filepath = 'model-person{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'

checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
                             save_best_only=True, mode='min')

callbacks = [early_stopping, checkpoint]
model.fit(Xtrain, Ytrain, validation_data=(Xtest, Ytest), epochs=30,
         batch_size=32, callbacks=callbacks, verbose=True)

```

Chương trình sử dụng 2 callback của keras:

- **EarlyStopping** : sau mỗi epochs, chương trình đo kết quả trên tập validation, nếu kết quả không tốt hơn trong một số chu kì liên tiếp (patience=10 ở ví dụ trên), thì ngừng quá trình huấn luyện
- **ModelCheckpoint** : lưu model sau mỗi epoch, ở ví dụ trên tham số save\_best\_only được đặt bằng True, có nghĩa chỉ lưu lại model nếu kết quả (tính bằng val\_loss như được chọn trong ví dụ) tốt hơn các lần đã lưu trước đó.

Kết quả chạy chương trình trên sau 30 epochs, lần cho độ chính xác cao nhất:

```
loss: 0.0964 - acc: 0.9645 - val_loss: 0.2000 - val_acc: 0.9189
```

Độ chính xác trên tập test khoảng 91%. Kết quả này có thể tốt hơn nếu cho tăng thêm số epochs và chỉnh định các tham số trong mô hình CNN.

## Sử dụng mô hình sau quá trình huấn luyện

Chúng ta chọn lấy model có kết quả tốt nhất sau quá trình huấn luyện, ví dụ *model-person{xxxx}-loss{yyyy}-val\_loss{zzzz}.h5* và đổi tên thành *model-person.h5* để sử dụng cho việc phân loại các ảnh mới (không nằm trong tập training).

Chương trình dùng model đã train để phân loại ảnh như sau:

```
# predict.py

import numpy as np
from PIL import Image
from keras.models import load_model

img_file = 'sample.jpg'
categories = ['Person', 'NoPerson']

model = load_model('model-person.h5')
img = Image.open(img_file).resize((224,224))
img_data = np.array(img).astype('float32')/255
y = model.predict(np.array([img_data]))
categ = np.argmax(y[0])
print(categories[categ])
```

## Transfer learning với sử dụng các mô hình train từ ImageNet

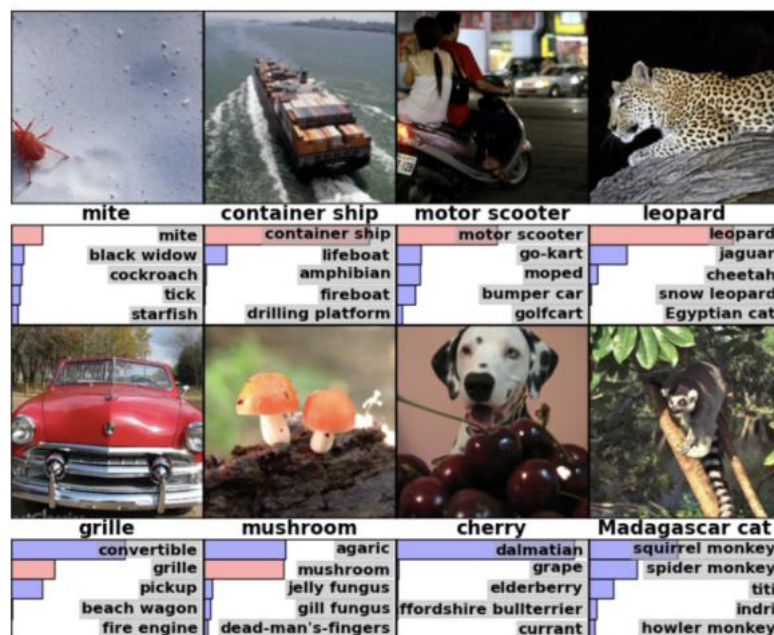
Việc xây dựng các mô hình CNN từ đầu chỉ mang tính chất minh họa hoặc cho các bài toán phân loại rất đơn giản. Với các bài toán phức tạp, các mô hình CNN thường được xây dựng theo phương pháp Transfer learning.

Transfer learning:

Là cách xây dựng mô hình dựa trên một mô hình đã có và kế thừa các kết quả đã có được từ mô hình gốc. Sự kế thừa kết quả này được thể hiện qua việc giá trị các tham số trong mô hình mới sẽ được sao chép từ mô hình gốc, thay cho việc khởi tạo ngẫu nhiên thông thường.

ImageNet:

ImageNet là tập dữ liệu ảnh để thử nghiệm các thuật toán phân loại ảnh. ImageNet bao gồm 1000 nhóm ảnh khác nhau.



Một số nhóm của ImageNet

Có nhiều mô hình CNN đã được train với tập dữ liệu của ImageNet, một số mô hình nổi tiếng là:

- [VGG](#) : có nhiều phiên bản, trong đó VGG-16 được dùng nhiều nhất. Mô hình này có cấu trúc tường minh nên thường được dùng cho transfer learning để thử nghiệm các thuật toán mới. Mô hình này có nhược điểm là khá nặng
- [Inceptionnet](#): mô hình có nguồn gốc từ Google. Kích thước nhỏ hơn VGG trong khi độ chính xác cao hơn VGG một chút
- [Resnet](#) : mô hình có nguồn gốc từ Microsoft. Kích thước và độ chính xác tương đương Inceptionnet.
- [Densenet](#): một dạng cải tiến của Resnet để cho độ chính xác cao hơn một chút
- [Mobilenet](#) : mô hình nhẹ và có thể dùng cho các nền tảng có phần cứng không mạnh, như smartphone

So sánh giữa các mô hình CNN nổi tiếng có thể xem trong bài viết [Review of Deep Learning Algorithms for Image Classification](#).

#### Sử dụng ImageNet model với keras

Keras đã dựng sẵn các mô hình ImageNet nổi tiếng. Chúng ta có thể sử dụng lại cho các mục đích riêng.

Cách tạo các mô hình ImageNet với keras:

```
# VGG
from keras.applications import vgg16
model = vgg16.VGG16()

# InceptionNet
from keras.applications import inception_v3
model = inception_v3.InceptionV3()

# ResNet
from keras.applications import resnet50
model = resnet50.ResNet50()

# DenseNet
from keras.applications import densenet
model = densenet.DenseNet169()

# MobileNet
from keras.applications import mobilenet
model = mobilenet.MobileNet()
```

Sử dụng model từ ImageNet để phân loại ảnh:

```
# predict.py

import numpy as np
from PIL import Image
from keras.applications.mobilenet import MobileNet
from keras.applications.mobilenet import preprocess_input

img_file = 'cat.jpg'
img = Image.open(img_file).resize((224,224))
img_data = np.array(img)
img_data = preprocess_input(img_data)
```

```
model = MobileNet()  
y = model.predict(np.array([img_data]))  
categ = np.argmax(y[0])  
print('categ = ', categ)
```



Dùng Pretrain model từ ImageNet để phân loại ảnh.

Nguồn : <https://github.com/tensorflow>

Với chương trình trên, ảnh đầu vào “cat.jpg” cho category lớn nhất bằng 282, tương ứng với “tiger cat”, như trong [danh sách các nhóm của ImageNet](#)

Lưu ý là kích thước đầu vào của các mô hình train trên ImageNet là 224x224

## Xây dựng và huấn luyện mô hình theo phương pháp transfer learning

ImageNet có 1000 nhóm nhưng phần lớn các ứng dụng thực tế không cần đến hết 1000 nhóm này, trong khi có những nhóm lại không có sẵn trong ImageNet. Các ứng dụng thực tế sẽ khắc phục điều này bằng cách chỉ giữ lại các lớp Convolution từ mô hình đã train trên ImageNet, bỏ đi lớp cuối cùng (lớp output ra 1000 nhóm), sau đó định nghĩa lại lớp cuối theo số nhóm của ứng dụng. Cách thực hiện như sau:

```
from keras.models import Model  
from keras.layers import Dense, GlobalAveragePooling2D  
from keras.applications.mobilenet import MobileNet  
  
num_classes = 2  
  
model = MobileNet(include_top=False)  
x = model.output  
x = GlobalAveragePooling2D()(x)  
x = Dense(512, activation='relu')(x)  
predictions = Dense(num_classes, activation='softmax')(x)  
new_model = Model(model.input, predictions)
```

Theo cách này, chúng ta xây dựng lại chương trình phân loại ảnh chứa người và không chứa người đã làm trong phần trước, nhưng theo phương pháp transfer learning.

## Huấn luyện mô hình

```
# transfer_train.py
```



```

import os
import numpy as np

from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.applications.mobilenet import MobileNet
from keras.applications.mobilenet import preprocess_input

categories = ['Person', 'NoPerson']
num_classes = len(categories)

img_size = 224

def loadData():
    Xtrain = np.fromfile('Xtrain.npy', dtype=np.uint8)
    Ytrain = np.fromfile('Ytrain.npy', dtype=np.uint8)
    Xtest = np.fromfile('Xtest.npy', dtype=np.uint8)
    Ytest = np.fromfile('Ytest.npy', dtype=np.uint8)

    Xtrain = Xtrain.reshape(-1, img_size, img_size, 3)
    Xtrain = preprocess_input(Xtrain)
    Ytrain = Ytrain.reshape(-1, 2).astype('float32')

    Xtest = Xtest.reshape(-1, img_size, img_size, 3)
    Xtest = preprocess_input(Xtest)
    Ytest = Ytest.reshape(-1, 2).astype('float32')

    return Xtrain, Ytrain, Xtest, Ytest

def createModel():
    model = MobileNet(include_top=False)
    x = model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    new_model = Model(model.input, predictions)
    opt = Adam(lr=0.001, decay=1e-6)
    new_model.compile(loss='categorical_crossentropy',
                      optimizer=opt, metrics=['accuracy'])
    return new_model

Xtrain, Ytrain, Xtest, Ytest = loadData()
model = createModel()
print(model.summary())

early_stopping = EarlyStopping(monitor='val_acc', patience=5)

filepath = 'model-person{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss',
                             verbose=1, save_best_only=True, mode='min')

callbacks = [early_stopping, checkpoint]
model.fit(Xtrain, Ytrain, validation_data=(Xtest, Ytest),
          epochs=10, batch_size=32, callbacks=callbacks, verbose=True)

```

Chương trình train dựa trên transfer learning cho tốc độ hội tụ rất nhanh, chỉ sau 1-2 epoch độ chính xác đã đạt trên 90%. Độ chính xác cao nhất trong 10 epoch:

```
loss: 2.1089e-04 - acc: 1.0000 - val_loss: 0.0247 - val_acc: 0.9953
```

Độ chính xác trên tập validation là 99.53%, cao hơn nhiều so với phương pháp tự xây dựng mô hình CNN từ đầu (91%).

## Sử dụng mô hình sau quá trình huấn luyện

```
# predict.py

import numpy as np
from PIL import Image
from keras.models import load_model
from keras.applications.mobilenet import preprocess_input

img_file = 'sample.jpg'
categories = ['Person', 'NoPerson']

model = load_model('model-person.h5')
img = Image.open(img_file).resize((224,224))
img_data = np.array(img)
img_data = preprocess_input(img_data)
y = model.predict(np.array([img_data]))
categ = np.argmax(y[0])
print(categories[categ])
```

Với các bài toán phức tạp, nếu độ chính xác của mobileNet không đủ, có thể dùng InceptionNet, ResNet hoặc DenseNet. Các mô hình này sẽ cho kết quả chính xác hơn nhưng cần máy có cấu hình (GPU) cao hơn và cần nhiều thời gian hơn.

## Training trên tập dữ liệu lớn

Với các tập dữ liệu lớn không thể tải hết vào bộ nhớ (chứa trong các mảng Xtrain, Ytrain), chúng ta phải dùng phương pháp huấn luyện theo cách tải từng phần dữ liệu vào bộ nhớ

Hàm cho phép huấn luyện theo cách trên là `model.fit_generator` :

```
model.fit_generator(
    generator=train_generator(),
    steps_per_epoch=steps_per_epoch,
    epochs=epochs,
    validation_data=test_generator(),
    validation_steps=validation_steps, ...)
```

Trong đó:

- **generator** : Hàm sinh ra dữ liệu training. Hàm này có tác dụng sinh ra một bộ dữ liệu Xtrain\_batch, Ytrain\_batch với độ dài tương đối nhỏ (batch\_size). Mỗi lần huấn luyện trên bộ dữ liệu nhỏ này gọi là một **step**
- **steps\_per\_epoch** : Số step trong một chu kì huấn luyện.
- **epochs** : Tổng số chu kì huấn luyện
- **validation\_data** : Hàm sinh ra dữ liệu validation (tương tự cách dữ liệu training được sinh ra)
- **validation\_steps** : Số step để sinh ra dữ liệu validation

Việc huấn luyện theo cách trên tương đương với cách huấn luyện quen thuộc (`model.fit`) trên tập dữ liệu huấn luyện (Xtrain, Ytrain) với độ dài `batch_size * steps_per_epoch`, và kiểm định trên tập dữ liệu validation (Xval, Yval) với độ dài `batch_size * validation_steps`

## Chuẩn bị dữ liệu

```
# prepare_batch_data.py
```

```

import os
from PIL import Image
from keras.utils import to_categorical
import numpy as np
from random import shuffle

image_dir = 'Images'
data_dir = 'data'
categories = ['Person', 'NoPerson']
num_classes = len(categories)

img_size = 224
train_factor = 0.7
chunk_size = 512

def save_data(data, tag):
    X = np.zeros((chunk_size, img_size, img_size, 3), dtype=np.uint8)
    Y = np.zeros((chunk_size, num_classes), dtype=np.uint8)

    for i, (img_path, label) in enumerate(data):
        X[i % chunk_size] = np.array(Image.open(img_path))
        Y[i % chunk_size] = label

        if (i+1) % chunk_size == 0 or i+1 == len(data):
            index = i // chunk_size
            items = i % chunk_size + 1
            print('Saving {}/{} {} items'.format(items, len(data), tag))
            Xfile = 'X{}_{}.np'.format(tag, index)
            Yfile = 'Y{}_{}.np'.format(tag, index)
            X[:items].tofile(os.path.join(data_dir, Xfile))
            Y[:items].tofile(os.path.join(data_dir, Yfile))

    data = []
    for i, category in enumerate(categories):
        img_files = os.listdir(os.path.join(image_dir, category))

        for j, f in enumerate(img_files):
            if not f.lower().endswith('.jpg'):
                continue

            img_path = os.path.join(image_dir, category, f)
            label = to_categorical(i, num_classes)
            data.append((img_path, label))

    shuffle(data)
    Ntrain = int(len(data)* train_factor)
    train_data = data[:Ntrain]
    test_data = data[Ntrain:]

    save_data(train_data, 'train')
    save_data(test_data, 'test')

```

Chương trình trên sẽ chuyển các file ảnh trong thư mục Images thành các file dữ liệu numpy (đặt trong thư mục data) với độ dài bằng chunk\_size. Các mảng này được đánh số từ 0 và tăng dần , ví dụ : Xtrain\_0.np, Xtrain\_1.np, Xtest\_0.np, Xtest\_1.np, ...

### Lưu ý :

Để tránh hiện tượng phân bố dữ liệu không đều trong quá trình huấn luyện, cần “xáo trộn” dữ liệu với hàm **random.shuffle**. Nếu huấn luyện theo các thông thường (model.fit), việc này có thể không cần thiết (vì keras đã tự động thực hiện), nhưng nếu huấn luyện theo cách sinh dữ liệu như ở đây, việc này cần thực hiện.

## Huấn luyện mô hình

```
# train_batch.py

import os
import numpy as np
import glob

from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.optimizers import SGD
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.applications.mobilenet import MobileNet
from keras.applications.mobilenet import preprocess_input

data_dir = 'data'
image_dir = 'Images'
categories = ['Person', 'NoPerson']
num_classes = len(categories)

batch_size = 32
img_size = 224
chunk_size = 512

class DataGenerator:
    def __init__(self, tag, batch_size):
        self.tag = tag
        self.batch_size = batch_size
        self.current_chunk = 0
        self.current_index = 0
        self.load_data()

    def load_data(self):

        Xfile = 'X{}_{}.np'.format(self.tag, self.current_chunk)
        if not os.path.exists(os.path.join(data_dir, Xfile)):
            self.current_chunk = 0
            Xfile = 'X{}_{}.np'.format(self.tag, self.current_chunk)

        X = np.fromfile(os.path.join(data_dir, Xfile), dtype=np.uint8)
        self.X = preprocess_input(
            X.reshape((-1, img_size, img_size, 3)))

        Yfile = 'Y{}_{}.np'.format(self.tag, self.current_chunk)
        self.Y = np.fromfile(os.path.join(data_dir, Yfile),
            dtype=np.uint8)

        self.Y = self.Y.reshape((-1, num_classes)).astype('float32')

        self.N = len(self.X)
        self.current_index = 0

        indexes = np.random.permutation(self.N)
        self.X = self.X[indexes]
        self.Y = self.Y[indexes]

    def next_batch(self):
        Xb = np.zeros((batch_size, img_size, img_size, 3),
            dtype=np.float32)

        Yb = np.zeros((batch_size, num_classes), dtype=np.float32)

        while True:
            if self.current_index + self.batch_size > self.N:
                self.current_chunk += 1
```

```

        self.load_data()

        for i in range(batch_size):
            Xb[i] = self.X[self.current_index % self.N]
            Yb[i] = self.Y[self.current_index % self.N]
            self.current_index += 1

        yield Xb, Yb

def createModel():
    model = MobileNet(include_top=False)
    x = model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    new_model = Model(model.input, predictions)
    opt = SGD(lr=1e-3, momentum=0.9, decay=1e-6, nesterov=True)
    new_model.compile(loss='categorical_crossentropy',
                      optimizer=opt, metrics=['accuracy'])
    return new_model

Ntrain = chunk_size * len(glob.glob(os.path.join(data_dir, 'Xtrain*.np')))
Ntest = chunk_size * len(glob.glob(os.path.join(data_dir, 'Xtest*.np')))

train_generator = DataGenerator('train', batch_size)
test_generator = DataGenerator('test', batch_size)

model = createModel()

early_stopping = EarlyStopping(monitor='val_acc', patience=5)
filepath = 'model-person{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'

checkpoint = ModelCheckpoint(filepath, monitor='val_loss',
                             verbose=1, save_best_only=True, mode='min')

callbacks = [early_stopping, checkpoint]

model.fit_generator(generator=train_generator.next_batch(),
                    steps_per_epoch=Ntrain//batch_size,
                    epochs=10,
                    validation_data=test_generator.next_batch(),
                    validation_steps=Ntest//batch_size,
                    callbacks=callbacks, verbose=True)

```

Chương trình sử dụng class `DataGenerator` để sinh ra các bộ dữ liệu training và validation. Hàm `next_batch` sinh ra các mẻ dữ liệu với độ dài bằng `batch_size`, bằng cách đọc lần lượt các mảng dữ liệu `Xtrain*.np`, `Ytrain*.np`, `Xtest*.np`, `Ytest*.np` vào bộ nhớ. Mỗi khi các mảng này được tải vào bộ nhớ, chương trình thực hiện xáo trộn dữ liệu (mặc dù chương trình chuẩn bị dữ liệu đã thực hiện điều này), để tránh tình trạng việc huấn luyện rơi vào cực tiểu địa phương:

```

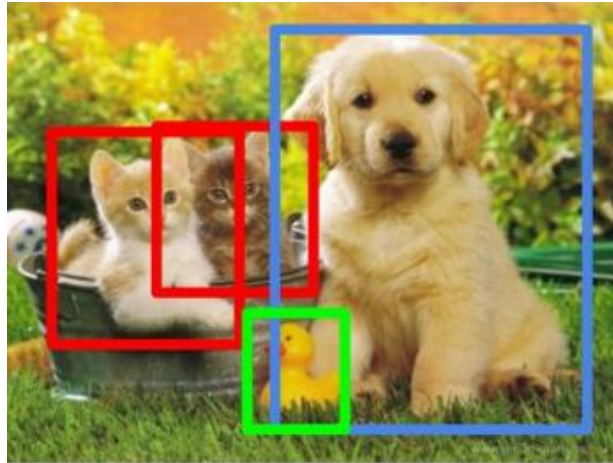
indexes = np.random.permutation(self.N)
self.X = self.X[indexes]
self.Y = self.Y[indexes]

```

Ngoài ra, chương trình sử dụng bộ tối ưu SGD thay cho Adam. Về cơ bản, Adam tốt hơn SGD cho việc huấn luyện. Tuy nhiên điều này chỉ đúng nếu các mẻ dữ liệu tương đối đồng nhất (lí tưởng nhất là các mẻ hoàn giống nhất  $\sim \text{step\_per\_epoch}=1$ ). Với dữ liệu ảnh, do kích thước một mẫu dữ liệu lớn nên `batch_size` thường khá nhỏ, các mẻ dữ liệu khác nhau khá nhiều, việc dùng Adam sẽ không hiệu quả trong nhiều trường hợp.

## 5. Bài toán phát hiện đối tượng trong ảnh (Object detection)

So với bài toán phân loại ảnh, bài toán phát hiện đối tượng trong ảnh có thêm yêu cầu là xác định vị trí của đối tượng trong ảnh. Ngoài ra trong một ảnh có thể có chứa nhiều đối tượng



Yêu cầu của bài toán phát hiện đối tượng trong ảnh. Nguồn : medium.com

Chi tiết về các phương pháp phát hiện đối tượng trong ảnh có thể tham khảo trong bài viết [Review of Deep Learning Algorithms for Object Detection](#)

Về cơ bản, Object Detection là sự mở rộng của Image Classification có thêm phần xác định vị trí các đối tượng. Do đó, Object Detection thường dựa trên các model về phân loại ảnh đã có (VGG, InceptionNet, ResNet, MobileNet, ....) sau đó bổ sung thêm phần xác định tọa độ đối tượng.

Một số tóm tắt về nội dung các phương pháp Object Detection:

- Phương pháp 2 giai đoạn (Two phase detector):
  - Region-based Convolutional Network (R-CNN): Sử dụng phương pháp tìm kiếm chọn lọc (Selective Search) dựa trên không gian màu và một số chỉ tiêu so sánh để phát hiện các vùng có thể chứa đối tượng (proposal regions), sau đó dùng CNN để xem các vùng đó có là ảnh của đối tượng không, theo cách như Image Classification đã làm
  - Fast Region-based Convolutional Network (Fast R-CNN) : cải tiến của R-CNN, dùng một mô hình CNN chung để tính toán feature map cho toàn bộ các vùng proposal, thay cho việc tính CNN trên từng vùng như ở R-CNN
  - Faster Region-based Convolutional Network (Faster R-CNN): Cải tiến của R-CNN, dùng một CNN riêng để tìm các vùng proposal, chứ không cần đến selective search
- Phương pháp một giai đoạn (One phase detector):
  - YOLO (You only look once) : chia ảnh thành một lưới (SxS ô vuông), mỗi ô vuông được gán 2 thành phần dự đoán : thành phần thứ nhất dự đoán một số hình chữ nhật có thể là vùng ảnh đối tượng (tâm các hình chữ nhật sẽ thuộc ô vuông này), thành phần thứ 2 dự đoán nhóm đối tượng nằm trong các hình chữ nhật. Độ tin cậy về khả năng chứa đối tượng của hình chữ nhật và độ tin cậy dự đoán nhóm đối tượng nằm trong hình chữ nhật được nhân

với nhau, chỉ những hình chữ nhật có độ tin cậy tổng hợp này lớn hơn một ngưỡng mới được dùng làm kết quả cuối cùng.

- Single-Shot Detector (SSD) : Dùng một mô hình CNN duy nhất để dự đoán đồng thời tọa độ hình chữ nhật chứa đối tượng và nhóm đối tượng bên trong hình chữ nhật đó. Thông tin về tọa độ hình chữ nhật được dự đoán bằng cách khai thác thông tin các feature map ở các lớp giữa CNN (trong khi nhóm đối tượng chủ yếu dựa trên thông tin của feature map lớp cuối)

Các phương pháp một giai đoạn cho tốc độ xử lý nhanh hơn nhiều so với các phương pháp 2 giai đoạn, trong khi độ chính xác không thấp hơn, do đó hiện nay trong object detection, người ta chủ yếu dùng phương pháp một giai đoạn. Trong các phương pháp 2 giai đoạn, chỉ có Faster-RCNN được dùng trong một số trường hợp vì có độ chính xác cao hơn một chút.

## Object Detection của tensorflow

Có nhiều project trên github đã xây dựng chương trình cho các phương pháp Object Detection. Khác với Image Classification, Object Detection vẫn được cải tiến liên tục, do đó các project thường do các cá nhân phát triển và có thể không được duy trì. Hiện thư viện keras chuẩn chưa có các mô hình Object Detection dựng sẵn. Trong các project trên github về Object Detection, tin cậy hơn cả là bản thân project tensorflow (được Google hỗ trợ). Phần object detection trong tensorflow nằm tại thư mục [object\\_detection](#)

## Sử dụng model đã có sẵn để phát hiện đối tượng trong ảnh

Các pretrained model trong project tensorflow đã được train trên [tập ảnh CoCo](#) với 91 nhóm đối tượng.

Chương trình ví dụ về cách sử dụng pretrained model cho việc phát hiện đối tượng có thể tham khảo tại [link](#).

Nếu sử dụng các model ssd thì có thể dùng cách đơn giản sau:

- Download hoặc checkout [tensorflow\\_model](#), di chuyển vào thư mục research/object\_detection
- Download một trong các pretrained model tại [model zoo](#). Các model thông dụng là :  
ssd\_mobilenet\_v1\_coco, ssd\_inception\_v2\_coco, ssd\_resnet\_50\_fpn\_coco
- Giải nén model đã download về và đặt trong thư mục research/object\_detection
- Vào trong thư mục research/object\_detection và chạy chương trình như sau:

```
# sample_object_detect.py

import numpy as np
import sys
import tensorflow as tf

from matplotlib import pyplot as plt
import cv2

sys.path.append("..")
from object_detection.utils import ops as utils_ops
```

```

IMG_PATH = 'sample.jpg'
PATH_TO_FROZEN_GRAPH = \
    'ssd_mobilenet_v1_coco_2018_01_28/frozen_inference_graph.pb'

image = cv2.imread(IMG_PATH)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_expand = np.array([image])
h, w, _ = image.shape

detection_graph = tf.Graph()
with detection_graph.as_default():
    graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
        serialized_graph = fid.read()
        graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(graph_def, name='')

    with tf.Session() as sess:
        ops = tf.get_default_graph().get_operations()
        tensor_dict = {}
        for key in ['num_detections', 'detection_boxes',
                    'detection_scores', 'detection_classes']:
            tensor_dict[key] = \
                tf.get_default_graph() \
                    .get_tensor_by_name(key + ':0')

        image_tensor = tf.get_default_graph() \
            .get_tensor_by_name('image_tensor:0')

        output_dict = sess.run(tensor_dict,
                                feed_dict={image_tensor: image_expand})

        num_detections = int(output_dict['num_detections'][0])
        classes = output_dict['detection_classes'][0].astype(np.uint8)
        boxes = output_dict['detection_boxes'][0]
        scores = output_dict['detection_scores'][0]

        for i in range(num_detections):
            if scores[i] > 0.5 and classes[i] == 1:
                y1, x1, y2, x2 = boxes[i]
                x1 = int(x1 * w)
                x2 = int(x2 * w)
                y1 = int(y1 * h)
                y2 = int(y2 * h)
                cv2.rectangle(image, (x1, y1), (x2, y2),
                              (0,255,0), 2)

        plt.imshow(image)
        plt.show()

```

Biến `PATH_TO_FROZEN_GRAPH` cần đặt để trỏ đến file model đã download và giải nén ra. Biến `IMG_PATH` là file ảnh cần tìm các đối tượng. Chương trình ví dụ trên sẽ tìm các đối tượng có class bằng 1 trong tập ảnh COCO - chính là các hình người, sau đó sẽ vẽ hình chữ nhật bao quanh mỗi ảnh người tìm được.





Kết quả phát hiện người trong ảnh sử dụng model của tensorflow

## Training Object Detection với tập dữ liệu riêng

Tương tự như trong phân loại ảnh, có thể dùng transfer learning để tạo các model cho các tập dữ liệu riêng. Chi tiết thực hiện có thể tham khảo tại [link](#).

Tóm tắt các bước thực hiện như sau.

### ● Cài đặt các thư viện liên quan:

Để Object Detection của tensorflow hoạt động đầy đủ các chức năng, cần cài đặt các thư viện liên quan theo hướng dẫn chi tiết tại [link](#). Hướng dẫn sau được áp dụng cho Ubuntu 64 bit.

- Cài đặt python3 : hiện python3 đang thay thế python2 trong hầu hết các ứng dụng. Để cài đặt python3 trên ubuntu:

```
sudo apt-get install python3 python3-pip python3-tk
```

- Cài đặt các thư viện cần thiết:

```
pip3 install tensorflow tensorflow-gpu matplotlib cython contextlib2
```

- Cài đặt COCO API.

Trước hết cần đảm bảo build tool của Ubuntu đã cài đặt:

```
sudo apt-get install gcc g++ make
```

Checkout COCO API từ github:

```
git clone https://github.com/cocodataset/cocoapi.git
```

Vào trong thư mục cocoapi/PythonAPI:

```
cd cocoapi/PythonAPI
```

Chỉnh sửa file *Makefile*, đổi các lệnh **python** thành **python3**, sau đó build thư viện bằng lệnh:

```
make
```

File thư viện sẽ được tạo ra trong thư mục `pycocotools`. Chúng ta copy thư mục này vào thư mục `models/research` của tensorflow (đã checkout từ github về):

```
cp -r pycocotools <path_to_tensorflow>/models/research
```

#### ■ Biên dịch các thư viện Protobuf.

Các thư viện protobuf nằm trong thư mục `object_detection/protos` của tensorflow. Để biên dịch các file này, cần sử dụng công cụ build là `protoc`:

```
cd <path_to_tensorflow>/models/research
```

```
wget https://github.com/google/protobuf/releases/download/v3.0.0/protoc-3.0.0-linux-x86\_64.zip
```

```
unzip protobuf.zip
```

```
./bin/protoc object_detection/protos/*.proto --python_out=.
```

#### ■ Thiết lập đường dẫn để python tìm các thư viện:

```
cd <path_to_tensorflow>/models/research  
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

Lệnh trên cần thực hiện mỗi lần mở cửa sổ terminal của Ubuntu.

#### ■ Kiểm tra xem cài đặt đã thành công hay chưa bằng lệnh:

```
python3 object_detection/builders/model_builder_test.py
```

### ● Chuẩn bị dữ liệu

Dữ liệu để train Object Detection gồm các ảnh và tọa độ các hình chữ nhật bao các đối tượng trong ảnh. Trước hết, từ thư mục `models/research` của tensorflow, chúng ta tạo ra một thư mục để chứa các dữ liệu riêng:

```
cd <path_to_tensorflow>/models/research  
mkdir custom_data
```

Trong thư mục `custom_data`, tạo ra các thư mục con theo cấu trúc sau:

```
custom_data  
├── data  
│   ├── images  
│   ├── annotations.csv  
│   └── label_map.pbtxt  
├── records  
└── model_dir
```

Các thư mục được thể hiện bằng chữ đậm, các file được thể hiện bằng chữ thường.

Trong đó:

- Thư mục `images` chứa các file ảnh cho training, ví dụ `1.jpg`, `2.jpg`, ....
- File `label_map.pbtxt` chứa danh sách các nhóm đối tượng theo cấu trúc sau:

```
item {  
  id: 1  
  name: 'Object_Class_1'  
}  
  
item {  
  id: 2  
  name: 'Object_Class_2'  
}  
....
```

- File `annotations.csv` chứa tọa độ các vùng đối tượng trong các ảnh theo cấu trúc sau:

```
1.jpg, xmin1,ymin1,xmax1,ymax1,obj_class1  
2.jpg, xmin2,ymin2,xmax2,ymax2,obj_class2
```

Mỗi dòng mô tả một đối tượng trong một file ảnh, bao gồm : tên file ảnh, 4 tọa độ của hình chữ nhật bao đối tượng, tên nhóm đối tượng.

Lưu ý: Một file ảnh có thể có nhiều đối tượng, được thể hiện bằng nhiều dòng dữ liệu theo cấu trúc trên (một file ảnh sẽ xuất hiện ở nhiều dòng, mỗi dòng ứng với một đối tượng trong ảnh).

- Thư mục `records` : chứa các file dữ liệu theo định dạng `tf_record` của tensorflow. Các file này là mã hóa nhị phân của các file ảnh và vùng tọa độ đối tượng ( tương tự như cách dùng file dữ liệu numpy trong phần Image classification). Các file record này sẽ được tạo ra nhờ một chương trình `prepare_data.py` (xem ở phần dưới đây). Chương trình huấn luyện sẽ đọc các file record này để làm dữ liệu training.
- Thư mục `model_dir` : chứa các file model được sinh ra trong quá trình training.

Để dễ hình dung, có thể download dữ liệu [data.zip](#), sau đó giải nén thành thư mục `data` với 3 thành phần : `images`, `annotations.csv`, `label_map.pbtxt` như ở trên. Bộ dữ liệu này gồm các ảnh người và các đánh dấu vị trí khuôn mặt, đối tượng cần xác định vị trí chỉ có một nhóm là 'face'.



Một ảnh mẫu trong tập dữ liệu tải về

Sau khi đã có thư mục data, chúng ta dùng chương trình sau để sinh ra các bản ghi tf\_record. Chương trình được đặt tên là `prepare_data.py` và đặt trong thư mục `models/research/object_detection` :

```
# prepare_data.py

import os
import hashlib
import contextlib2
import tensorflow as tf

from random import shuffle
from os.path import join
from PIL import Image

from object_detection.utils.dataset_util import *
from object_detection.utils.label_map_util import get_label_map_dict
from object_detection.dataset_tools.tf_record_creation_util \
    import open_sharded_output_tfrecords

data_dir = 'custom_data/data'
output_dir = 'custom_data/records'
label_map_file = 'label_map.pbtxt'
annotation_file = 'annotations.csv'
num_shards = 10

def create_tf_example(img_file, boxes):
    img_path = join(data_dir, 'images', img_file)

    with open(img_path, 'rb') as fid:
        encoded_jpg = fid.read()

    image = Image.open(img_path)

    if image.format != 'JPEG':
        raise ValueError('Image format not JPEG')

    key = hashlib.sha256(encoded_jpg).hexdigest()
    width, height = image.size
    xmin, ymin, xmax, ymax, classes = [], [], [], [], []

    for box in boxes:
        x1, y1, x2, y2, class_name = box
        xmin.append(float(x1) / width)
        ymin.append(float(y1) / height)
        xmax.append(float(x2) / width)
        ymax.append(float(y2) / height)
        classes.append(label_map_dict[class_name])

    feature_dict = {
        'image/height': int64_feature(height),
        'image/width': int64_feature(width),
        'image/filename': bytes_feature(img_file.encode('utf8')),
        'image/source_id': bytes_feature(img_file.encode('utf8')),
        'image/key/sha256': bytes_feature(key.encode('utf8')),
        'image/encoded': bytes_feature(encoded_jpg),
        'image/format': bytes_feature('jpeg'.encode('utf8')),
        'image/object/bbox/xmin': float_list_feature(xmin),
        'image/object/bbox/xmax': float_list_feature(xmax),
        'image/object/bbox/ymin': float_list_feature(ymin),
        'image/object/bbox/ymax': float_list_feature(ymax),
        'image/object/class/label': int64_list_feature(classes),
    }

    features = tf.train.Features(feature=feature_dict)
    return tf.train.Example(features=features)
```

```

def create_tf_records(output_prefix, examples):
    map_boxes = {}

    for example in examples:
        img_file, x1, y1, x2, y2, class_name = example

        if img_file not in map_boxes:
            map_boxes[img_file] = []

        map_boxes[img_file].append((x1, y1, x2, y2, class_name))

    data_list = list(map_boxes.items())

    with contextlib2.ExitStack() as tf_record_close_stack:
        output_tfrecords = open_sharded_output_tfrecords(
            tf_record_close_stack, output_prefix, num_shards)

        for idx, item in enumerate(data_list):
            if idx % 100 == 0:
                print('On item {} of {}'.format(idx, len(examples)))

            img_file, boxes = item
            tf_example = create_tf_example(img_file, boxes)

            if not tf_example:
                continue

            shard_idx = idx % num_shards
            output_tfrecords[shard_idx] \
                .write(tf_example.SerializeToString())

if __name__ == '__main__':
    label_map_dict = get_label_map_dict(join(data_dir, label_map_file))

    with open(join(data_dir, annotation_file)) as fid:
        examples_list = [line.strip().split(',') for line in fid]

    shuffle(examples_list)
    num_examples = len(examples_list)
    num_train = int(0.7 * num_examples)
    train_examples = examples_list[:num_train]
    val_examples = examples_list[num_train:]

    train_output_prefix = join(output_dir, 'train.record')
    val_output_prefix = join(output_dir, 'val.record')

    print('Creating training records ....')
    create_tf_records(train_output_prefix, train_examples)

    print('Creating validation records ....')
    create_tf_records(val_output_prefix, val_examples)

```

Chúng ta chạy lệnh sau để tạo ra các bản ghi tf\_record trong thư mục *custom\_data/records*:

```

cd <path_to_tensorflow>/models/research
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
python3 object_detection/prepare_data.py

```

Chương trình sẽ in ra số bản ghi cho mỗi tập training và validation. Chúng ta cần ghi lại số bản ghi cho tập validation để dùng cho bước sau.

## ● Tạo cấu hình training

Với Object Detection, có thể chọn training dựa trên các phương pháp khác nhau (Faster-RCNN, SSD, ...) và dùng các backend khác nhau (MobileNet, ResNet, ...) Mỗi cấu hình training sẽ có một file mẫu tương ứng nằm trong thư mục *object\_detection/samples/configs*.

Trong ví dụ này, chúng ta sẽ sử dụng SSD với backend là Mobilenet. Cách làm ở dưới đây có thể áp dụng cho các backend khác (Resnet, InceptionNet, ...)

File cấu hình sử dụng là *ssd\_mobilenet\_v1\_coco.config*. Chúng ta copy file này từ thư mục *samples* sang thư mục *custom\_data* :

```
cp object_detection/samples/configs/ssd_mobilenet_v1_coco.config custom_data
```

Sau đó, cần chỉnh sửa lại nội dung của file này tại một số điểm:

- Sửa số nhóm đối tượng trong thuộc tính *model/ssd/num\_classes* thành số đối tượng của tập dữ liệu training (ví dụ hiện tại này là 1, vì chỉ có đối tượng *face*)

- Sửa phần *fine\_tune\_checkpoint* thành:

```
fine_tune_checkpoint: "custom_data/model.ckpt"
```

- Sửa phần *train\_input\_reader* thành:

```
train_input_reader: {
  tf_record_input_reader {
    input_path: "custom_data/records/train.record-?????-of-00010"
  }
  label_map_path: "custom_data/data/label_map.pbtxt"
}
```

- Sửa phần *eval\_input\_reader* thành:

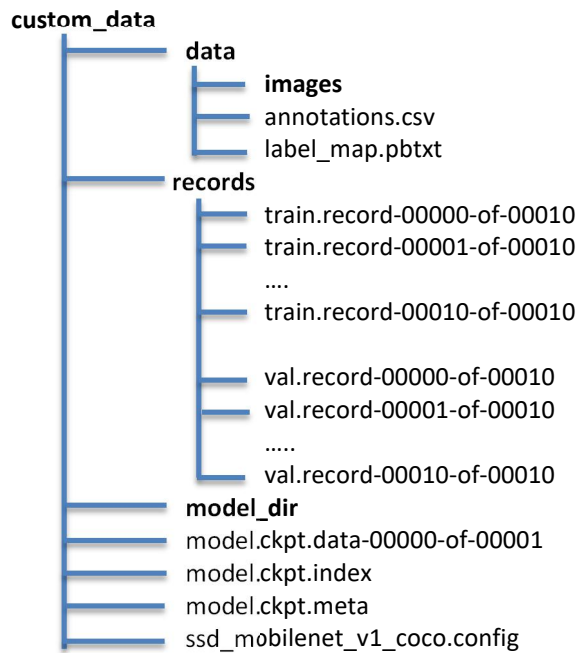
```
eval_input_reader: {
  tf_record_input_reader {
    input_path: "custom_data/records/val.record-?????-of-00010"
  }
  label_map_path: "custom_data/data/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

- Sửa số mẫu validation trong thuộc tính *eval\_config/num\_examples* (gần cuối file) thành số mẫu validation thật của tập dữ liệu (đã ghi lại ở bước chuẩn bị dữ liệu)

## ● Download pretrained model để làm giá trị khởi tạo cho model (transfer learning)

Model cần download là *ssd\_mobilenet\_v1\_coco*, có thể được download tại [link](#). Sau khi download file zip về, chúng ta giải nén, sau đó copy các file **model.ckpt.\*** vào thư mục *custom\_data*.

Sau khi thực hiện xong các bước, chúng ta kiểm tra lại để đảm bảo thư mục *custom\_data* có đủ các thành phần sau:



## ● Bắt đầu quá trình training

Sử dụng lệnh sau để bắt đầu quá trình training:

```
cd <path_to_tensorflow>/models/research
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

python3 object_detection/model_main.py \
  --model_dir=custom_data/model_dir \
  --pipeline_config_path=custom_data/ssd_mobilenet_v1_coco.config \
  >/dev/null &
```

Quá trình training, các model sẽ được tạo ra trong thư mục custom\_data/model\_dir

## ● Sử dụng mô hình sau quá trình training

Chuyển model từ dạng checkpoint sang dạng frozen:

```
cd <path_to_tensorflow>/models/research
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

python3 object_detection/export_inference_graph.py \
  --input_type image_tensor \
  --pipeline_config_path custom_data/ssd_mobilenet_v1_coco.config \
  --trained_checkpoint_prefix custom_data/model_dir/model.ckpt-xxxx \
  --output_directory final_model
```

Model dạng frozen sẽ được tạo ra trong thư mục có tên final\_model. Kiểm tra thư mục này để đảm bảo có các file sau:

- frozen\_inference\_graph.pb
- model.ckpt.data-00000-of-00001
- model.ckpt.index
- model.ckpt.meta
- pipeline.config
- saved\_model

Cách sử dụng model đã tạo ra : Thư mục **final\_model** có tác dụng giống như các thư mục giải nén từ các pretrained model đã dùng trong phần detect object ở đầu phần này. Xem lại chương trình ở phần đó, chúng ta thấy có dòng lệnh:

```
PATH_TO_FROZEN_GRAPH = \
    'ssd_mobilenet_v1_coco_2018_01_28/frozen_inference_graph.pb'
```

Nếu muốn dùng model mới được tạo ra, chúng ta thay dòng trên bằng:

```
PATH_TO_FROZEN_GRAPH = 'final_model/frozen_inference_graph.pb'
```

## 6. Nhận dạng khuôn mặt

Bài toán nhận dạng khuôn mặt đã có từ khá lâu trong xử lý ảnh. Các phương pháp xử lý ảnh truyền thống cho kết quả khá tốt, nhưng thường mất khá nhiều công sức cho việc train mô hình. Mỗi khi có thêm một người mới trong nhóm, toàn bộ dữ liệu sẽ phải train lại hoàn toàn.

Deep learning xử lý bài toán theo hướng đơn giản hơn, theo đó một mô hình duy nhất được train trên một lượng rất lớn người tham gia. Đầu ra của mô hình không phải là identity của mỗi người mà là một vector có độ dài 128 byte. Cách làm này được gọi là **Embedding**, có nghĩa một đối tượng dữ liệu có kích thước lớn được thay bằng một vector có kích thước nhỏ hơn. Vector này giống như lớp ngay sát lớp cuối trong các mô hình phân loại ảnh mà phương pháp transfer learning muốn tách lấy để từ đó bổ sung thêm lớp mới cho mục đích riêng của mình.

Các pretrained model thường được công khai trên các project github, chúng ta có thể download về và sử dụng luôn. Có thể sử dụng model từ các project sau:

- [OpenFace](#) : các model được xây dựng dựa trên framework pytorch
- [Keras-OpenFace](#), [Deep face recognition with Keras](#) : bản port của OpenFace sang keras
- [FaceRecognition](#) : Sử dụng [Dlib](#)

Việc sử dụng mô hình pretrain để nhận dạng khuôn mặt khá đơn giản. Cho ảnh khuôn mặt đi qua model có sẵn, chúng ta có được kết quả là vector Embedding với độ dài 128 byte. Vector này được dùng làm đầu vào cho mô hình phân loại, có thể sử dụng SVM, MLP hay cả K-nearest neighbour.

### Xây dựng chương trình check-in bằng khuôn mặt:

Bài toán check-in có hơi khác với bài toán phân loại. Với bài toán phân loại, chúng ta cần cho biết một ảnh cho trước là ai trong một số người đã biết. Với bài toán check-in, chúng ta còn phải cho biết liệu ảnh đó có là ảnh của người nào trong danh sách đăng kí không, hay là ảnh của một người lạ. Với bài toán check-in, chúng ta không xây dựng mô hình phân loại, mà tính độ tương tự của ảnh cần kiểm tra với với các ảnh của các thành viên đã đăng kí. Nếu độ tương tự cao hơn một ngưỡng thì coi ảnh đó là ảnh của thành viên, nếu không ảnh đó xem là ảnh của người lạ.

Cách đo độ tương tự giữa 2 ảnh là tính khoảng cách giữa 2 vector Embedding của 2 ảnh đó :

$$d^2(x, y) = \sum (x_i - y_i)^2$$



Trong đó  $x, y$  là 2 vector Embedding của 2 ảnh khuôn mặt (mỗi vector có độ dài bằng 128).

Chi tiết về cách xây dựng một chương trình check-in trên khuôn mặt có thể tham khảo tại [link](#). Tóm tắt cách thực hiện như sau:

## Bước 1 : Tách lấy vùng ảnh khuôn mặt từ bức ảnh toàn thân

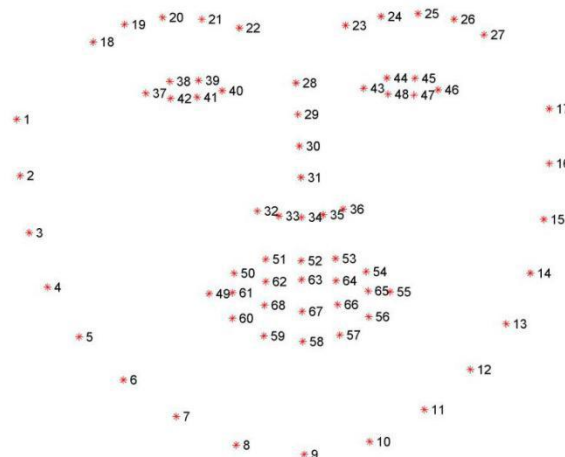
Trước hết, chúng ta cần tách lấy vùng ảnh khuôn mặt từ ảnh toàn thân. Để thực hiện việc này, cách thông dụng là dùng thư viện [dlib](#). Dlib là thư viện viết bằng C++ nhưng có hỗ trợ python. Để sử dụng dlib từ python, chúng ta dùng pip để cài đặt:

```
pip install dlib
```

Quá trình cài đặt có thể sẽ cần build thư viện từ file mã nguồn (C++). Trên windows, việc này thường phức tạp. Để tránh phải build từ mã nguồn, nên chọn phiên bản dlib đã được build thành thư viện nhị phân. Danh sách các phiên bản dlib có thể xem tại [link](#). Các phiên bản với tên mở rộng .whl là các file đã được build thành thư viện nhị phân, nếu cài đặt phiên bản này sẽ không cần build lại từ mã nguồn. Ví dụ để cài đặt file dlib-19.8.1-cp36-cp36m-win\_amd64.whl cho python 3.6, chúng ta dùng lệnh:

```
pip install dlib==19.8.1
```

Dlib có nhiều pretrained model cho phép xác định khuôn mặt trong ảnh. Một trong các model phổ biến là [shape\\_predictor\\_68\\_face\\_landmarks](#). Model này cho phép xác định 68 điểm trên khuôn mặt theo thứ tự sau:



68 điểm trên khuôn mặt xác định nhờ model `shape_predictor_68_face_landmarks`. Nguồn : [pyimagesearch.com](http://pyimagesearch.com)

Trong 68 điểm này, Các điểm từ 1 đến 17 là đường biên khuôn mặt, các điểm từ 18 đến 22 là lông mày trái, từ 23 đến 27 là lông mày phải, 28 đến 36 là mũi, 37 đến 42 là mắt trái, 43 đến 48 là mắt phải, 49 đến 68 là miệng. Thứ tự các điểm tuân theo đúng vị trí tương đối như trong hình minh họa.

Chương trình sau sẽ tách ảnh vùng mặt từ ảnh toàn thân, sau đó quay ảnh và căn chỉnh để khuôn mặt về tư thế thẳng đứng ( 2 mắt cùng ở trên một đường nằm ngang), vị trí mũi nằm ở tâm của bức ảnh. Để chạy chương trình, cần dowload file [shape\\_predictor\\_68\\_face\\_landmarks.dat.bz2](#), giải nén và để trong cùng thư mục với chương trình.

```
# crop_face.py

import numpy as np
from PIL import Image
from align import AlignDlib
```

```

alignment = AlignDlib('shape_predictor_68_face_landmarks.dat')

def get_face_image(img, output_size):
    box = alignment.getLargestFaceBoundingBox(img)
    face_img = alignment.align(output_size, img, box,
                               landmarkIndices=AlignDlib. OUTER_EYES_AND_NOSE)

    return face_img

if __name__ == '__main__':
    img = Image.open('person1.jpg')
    img_data = np.array(img)
    crop_img_data = get_face_image(img_data, 160)
    crop_img = Image.fromarray(crop_img_data)
    crop_img.save('face1.jpg')

```

Chương trình sẽ tách lấy ảnh khuôn mặt từ ảnh gốc và lưu thành một file ảnh mới.



Tách ảnh vùng khuôn mặt từ ảnh người

## Bước 2: So sánh 2 khuôn mặt

### Sử dụng pretrained model của OpenFace

Model của OpenFace (đã chuyển sang định dạng keras) có thể download về từ link [nn4.small2.v1.h5](#)

Để sử dụng model này, chúng ta cần download các file [model.py](#) và [utils.py](#) và để chúng trong thư mục chạy chương trình (cùng cả file model [nn4.small2.v1.h5](#))

Để so sánh 2 khuôn mặt, chúng ta tính vector Embedding của chúng, sau đó đo khoảng cách giữa 2 vector này. Nếu khoảng cách này nhỏ hơn một ngưỡng (thường chọn bằng 0.7 với model của OpenFace), thì kết luận chúng là của cùng một người, ngược lại thì kết luận là 2 người khác nhau.

```

import numpy as np
from model import create_model

distance_thresh = 0.7

model = create_model()
model.load_weights('nn4.small2.v1.h5')

def get_embedding(face_img):

```

```

    predict = model.predict(np.array([face_img]))
    return predict[0]

def is_one_person(face_img1, face_img2):
    emb1 = get_embedding(face_img1)
    emb2 = get_embedding(face_img2)
    distance = np.sqrt(np.sum((emb1 - emb2) ** 2))
    return distance < distance_thresh

```

Chương trình đầy đủ để kiểm tra 2 bức ảnh có cùng là của một người:

```

# compare_face.py

import sys
import numpy as np
from model import create_model
from PIL import Image
from crop_face import get_face_image

distance_thresh = 0.7
face_img_size = 96

model = create_model()
model.load_weights('nn4.small2.v1.h5')

def get_embedding(face_img):
    predict = model.predict(np.array([face_img]))
    return predict[0]

def is_one_person(face_img1, face_img2):
    if (face_img1 is None) or (face_img2 is None):
        return False

    emb1 = get_embedding(face_img1)
    emb2 = get_embedding(face_img2)
    distance = np.sqrt(np.sum((emb1 - emb2) ** 2))
    return distance < distance_thresh

img1 = np.array(Image.open(sys.argv[1]))
img2 = np.array(Image.open(sys.argv[2]))

face_img1 = get_face_image(img1, face_img_size)
face_img2 = get_face_image(img2, face_img_size)

if is_one_person(face_img1, face_img2):
    print('Same person')
else:
    print('Two people')

```

Cách sử dụng chương trình:

```
python face_compare.py img1.jpg img2.jpg
```



Image 1



Image 2



Embedding distance = 0.4



So sánh khuôn mặt trong 2 ảnh

### Sử dụng pretrained model của Dlib

Ngoài khả năng xác định vị trí khuôn mặt trong ảnh, Dlib còn cung cấp cả model để tính Face Embedding. Model này được train dựa trên resnet và có thể download từ [link](#)

So với OpenFace, việc tính Embedding cho ảnh khuôn mặt của Dlib có điểm khác là cần cung cấp vị trí các bộ phận trên mặt (mũi, mắt trái, mắt phải).

Chương trình sử dụng model của Dlib để kiểm tra 2 bức ảnh có là của một người không như sau:

```
# compare_face_w_dlib.py

import sys
import numpy as np
import dlib
from PIL import Image

distance_thresh = 0.6

face_detector = dlib.get_frontal_face_detector()
face_encoder = dlib.face_recognition_model_v1(
    'dlib_face_recognition_resnet_model_v1.dat')
pose_predictor = dlib.shape_predictor(
    'shape_predictor_68_face_landmarks.dat')
```

```

def get_face_location(img):
    locations = face_detector(img)
    if len(locations) > 0:
        return locations[0]
    return None

def get_face_landmark(img):
    face_location = get_face_location(img)
    if face_location:
        return pose_predictor(img, face_location)
    return None

def get_face_encode(img):
    landmark = get_face_landmark(img)
    if landmark:
        return np.array(face_encoder.compute_face_descriptor(
            img, landmark))
    return None

def is_one_person(img1, img2):
    emb1 = get_face_encode(img1)
    emb2 = get_face_encode(img2)

    if (emb1 is None) or (emb2 is None):
        return False

    distance = np.sqrt(np.sum((emb1 - emb2) ** 2))
    return distance < distance_thresh

img1 = np.array(Image.open(sys.argv[1]))
img2 = np.array(Image.open(sys.argv[2]))

if is_one_person(img1, img2):
    print('Same person')
else:
    print('Two people')

```

Dựa trên các chương trình trên chúng ta có thể xây dựng ứng dụng cho phép check-in bằng khuôn mặt:

- Chụp một số ảnh mẫu của các thành viên, để nâng cao độ chính xác, yêu cầu chụp khuôn mặt ở các góc khác nhau. Tính vector Embedding của các ảnh này và lưu trong cơ sở dữ liệu
- Khi có một người check-in, tách lấy ảnh khuôn mặt người đó, tính Embedding của ảnh mặt đó, tìm vector Embedding trong cơ sở dữ liệu có khoảng cách nhỏ nhất đến vector Embedding vừa tính. Nếu khoảng cách này nhỏ hơn ngưỡng chọn trước, thì xác nhận người đó là một thành viên đã đăng kí, nếu ngược lại thì xác định là người lạ.

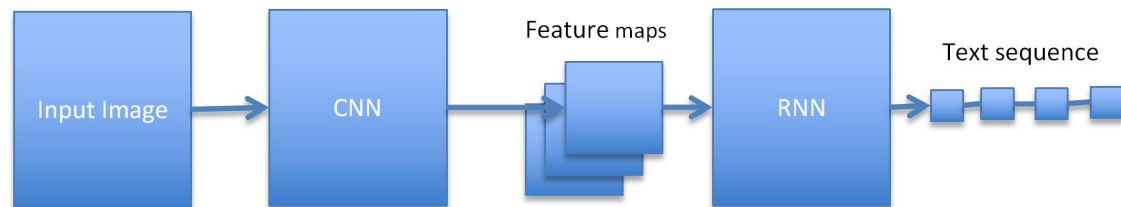
## 7. Nhận dạng kí tự (OCR)

Nhận dạng kí tự (OCR) là một lĩnh vực đã được nghiên cứu nhiều. Quá trình xử lý OCR gồm nhiều bước, trong đó có việc xác định layout tài liệu, phát hiện các vùng chứa chữ, đưa ảnh các vùng chữ qua hệ thống nhận dạng, tổng hợp kết quả để đưa ra văn bản cuối cùng.

Trong phần này, chúng ta không tìm hiểu chi tiết về các bước xử lý trong OCR mà xem xét một ứng dụng của Deep Learning có khả năng đưa ra kết quả text trong ảnh chỉ với một bước xử lý duy nhất. Chúng ta sẽ xây dựng một mô hình Deep Learning có khả năng nhận đầu vào là một ảnh và đưa ra kết quả là đoạn text trong ảnh đó.

Các loại ảnh có thể xử lý một bước như vậy thường là các ảnh chỉ chứa một dòng chữ, nhưng dòng chữ đó thường lẫn vào trong môi trường xung quanh, ví dụ như : biển số xe ô tô, xe máy, số seri trên bao bì sản phẩm, số thẻ tín dụng.

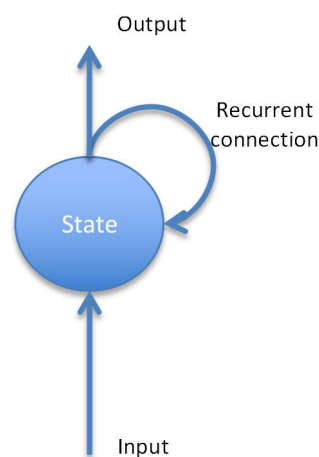
Trước hết , chúng ta tìm hiểu qua nguyên lý nhận dạng OCR với Deep Learning. Về cơ bản, OCR Engine dựa trên Deep Learning gồm 2 phần nối tiếp nhau : Convolutional Neural Network (CNN) có tác dụng tách các feature của các chữ từ ảnh, ghép với một Recurrent Neural Network (RNN) có tác dụng sinh ra dãy kí tự ở đầu ra.



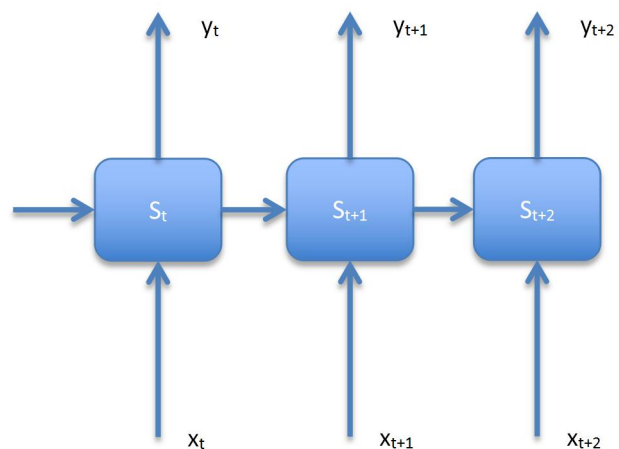
Cấu trúc của OCR dựa trên Deep Learning

CNN đã được trình bày ở phần đầu, trong phần này chúng ta tìm hiểu một chút về Recurrent Neural Network (RNN) và ứng dụng của nó.

So với các mạng neuron thông thường, RNN có điểm khác là có khả năng “ghi nhớ”. Đầu ra của RNN không chỉ phụ thuộc đầu vào mà còn phụ thuộc các thông tin trong quá khứ. RNN có khả năng lưu trữ kết quả xử lý của các bước trước để “tham khảo” khi đưa ra tính toán kết quả ở bước hiện tại.



Cấu trúc của RNN



Cấu trúc dạng khai triển của RNN

RNN thường được dùng trong các bài toán xử lý ngôn ngữ tự nhiên, khi đầu vào là một chuỗi các từ có thứ tự (một câu), và đầu ra cũng là chuỗi từ có thứ tự. Ý nghĩa của câu không chỉ phụ thuộc các từ trong câu mà còn phụ thuộc vào trình tự xuất hiện của các từ.

#### Cách xây dựng RNN:

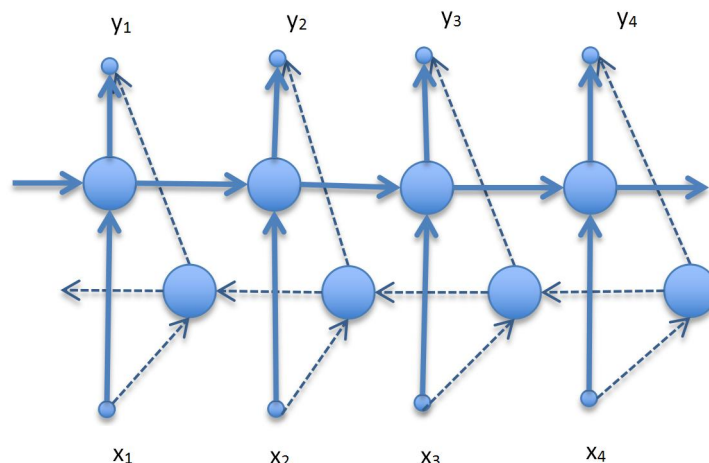
Một mạng neuron dạng RNN được dựng từ nhiều phần tử nhớ. Mỗi phần tử lưu trữ một trạng thái ứng với đầu ra tại một thời điểm trong quá khứ. Số phần tử càng nhiều thì mức độ ghi nhớ của RNN càng lớn. Tuy nhiên, khi độ dài RNN lớn, một khó khăn gặp phải là trong quá trình huấn luyện RNN, đạo hàm của sai số đầu ra theo các tham số của các phần tử ở xa sẽ rất nhỏ (*Vanishing Gradient*), do đó việc huấn luyện dựa trên phương pháp Gradient Descent không thực hiện được. Để khắc phục hiện tượng này, các phần tử của RNN được lựa chọn theo một trong 2 cấu trúc:

- LSTM (Long Short Term Memory)
- GRU (Gated Recurrent Unit)

Chi tiết về cấu tạo của LSTM và GRU có thể tìm trên internet, mục tiêu của chúng là để giải quyết vấn đề *Vanishing Gradient* trong RNN.

#### Bidirectional RNN:

Trong một số trường hợp, người ta ghép 2 RNN ngược chiều nhau thành một mạng, gọi là RNN hai chiều (Bidirectional RNN). Trong Bidirectional RNN, không chỉ trạng thái quá khứ tác động đến đầu ra hiện tại, mà trạng thái hiện tại cũng có thể làm hiệu chỉnh các giá đầu ra đã được tính trong quá khứ.



Cấu trúc của Bidirectional RNN

Một ví dụ về ứng dụng của Bidirectional RNN là hệ thống dịch giữa 2 ngôn ngữ. Khi cần dịch một câu, từng từ của câu sẽ được đưa vào hệ thống dịch, tại mỗi bước hệ thống sẽ lựa chọn từ đầu ra tương ứng với từ đầu vào sao cho phù hợp nhất với văn cảnh (chính là các trạng thái của quá khứ). Tuy nhiên, có trường hợp một phần của câu đã được dịch nhưng đến giữa câu, xuất hiện một từ, mà phần đã dịch tỏ ra không phù hợp với từ mới xuất hiện, lúc đó hệ thống phải hiệu chỉnh lại phần đã dịch. Trong trường hợp đó nhánh ngược của hệ thống Bidirectional RNN sẽ tác động ngược trở lại các đầu ra đã tích toán trong quá khứ, yêu cầu chúng cần phải được hiệu chỉnh lại.

Trong hệ thống OCR, RNN có tác dụng tạo ra các ký tự ở đầu ra. Trình tự xuất hiện các chữ cái trong một ngôn ngữ thường tuân theo các quy tắc (ví dụ nếu biết các chữ cái đã xuất hiện thì có thể biết khả năng chữ cái nào có thể xuất hiện tiếp theo dựa trên quy tắc ghép chữ trong ngôn ngữ của văn bản đang xét). RNN sẽ khai thác thông tin này để đưa ra chuỗi ký tự ở đầu ra một cách hợp lý nhất.

Sau khi biết một số điểm về cách dùng Deep Learning trong OCR, chúng ta xây dựng một ví dụ minh họa để tách lấy một dòng chữ từ ảnh.

Chương trình minh họa được xây dựng dựa trên hướng dẫn về cách xây dựng OCR Engine ở [bài viết](#). Ứng dụng OCR trong bài viết này cho phép tách ra các biển số xe từ các ảnh có kích thước 128x64 pixel.

Predicted: X835KX31  
True: X835KX31

Input img



Dùng Deep Learning để đọc biển số xe. Nguồn : hackernoon.com

Tuy nhiên, Deep Learning có thể xử lý được trong những trường hợp phức tạp hơn như vậy. Chúng ta sẽ xây dựng một mô hình cho phép đọc số thẻ tín dụng trực tiếp từ ảnh thẻ.



Xây dựng ứng dụng Deep Learning để đọc số thẻ tín dụng

## Bước 1: Chuẩn bị dữ liệu

Dữ liệu ảnh các thẻ tín dụng có thể download từ file [creditcard\\_images.zip](#) chứa 1000 ảnh train và 200 ảnh test, mỗi ảnh có kích thước 512x384.

Tương tự các chương trình xử lý ảnh, chúng ta sử dụng một chương trình `prepare_data.py` để chuyển ảnh từ dạng jpg sang file dữ liệu của numpy, để tiết kiệm thời gian tải dữ liệu cho chương trình training.

```
# prepare_data.py

import os
from PIL import Image
import numpy as np
```



```

data_dir = 'creditcard_images'
letters = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

def text_to_label(text):
    return [letters.index(x) for x in text]

def read_data(img_dir):
    imgs = []
    labels = []
    for filename in os.listdir(img_dir):
        name, ext = os.path.splitext(filename)
        img = Image.open(os.path.join(img_dir, filename))
        img = img.convert('L')
        imgs.append(np.array(img))
        labels.append(text_to_label(name))

    imgs = np.array(imgs, dtype=np.uint8)
    labels = np.array(labels, dtype=np.uint8)
    return imgs, labels

if __name__ == '__main__':
    print('Creating training data ...')
    imgs_train, labels_train = read_data(os.path.join(data_dir, 'train'))
    imgs_train.tofile('imgs_train.npy')
    labels_train.tofile('labels_train.npy')

    print('Creating test data ...')
    imgs_test, labels_test = read_data(os.path.join(data_dir, 'test'))
    imgs_test.tofile('imgs_test.npy')
    labels_test.tofile('labels_test.npy')

```

Chương trình sẽ tạo ra các file dữ liệu imgs\_train.npy, labels\_train.npy, imgs\_test.npy, labels\_test.npy

## Bước 2 : Xây dựng mô hình

File model.py dưới đây dùng để tạo mô hình OCR

```

# model.py

import numpy as np
from keras import backend as K
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, Dropout
from keras.layers import Reshape, Lambda
from keras.layers.merge import add, concatenate
from keras.models import Model
from keras.layers.recurrent import GRU
from keras.optimizers import SGD

img_h, img_w = 320, 512
num_letter = 10
max_text_len = 16

num_cnn_layer = 4
downsample_factor = 2 ** num_cnn_layer

def ctc_lambda_func(args):
    y_pred, label, input_length, label_length = args
    y_pred = y_pred[:, 2:, :]
    return K.ctc_batch_cost(label, y_pred, input_length, label_length)

def create_model():
    input_data = Input(shape=(img_w, img_h, 1), dtype='float32', name='img')

```

```

inner = input_data

for _ in range(num_cnn_layer):
    inner = Conv2D(16, (3,3), padding='same', activation='relu')(inner)
    inner = MaxPooling2D(pool_size=(2, 2))(inner)
    inner = Dropout(0.2)(inner)

downsample_w = img_w // downsample_factor
inner = Reshape(target_shape=(downsample_w, -1))(inner)
inner = Dense(32, activation='relu')(inner)

gru_1 = GRU(512, return_sequences=True)(inner)
gru_1back = GRU(512, return_sequences=True, go_backwards=True)(inner)
gru1_merged = add([gru_1, gru_1back])
gru_2 = GRU(512, return_sequences=True)(gru1_merged)
gru_2back = GRU(512, return_sequences=True,
                go_backwards=True)(gru1_merged)

gru2_merged = concatenate([gru_2, gru_2back])
y_pred = Dense(1+num_letter, activation='softmax',
               name='softmax')(gru2_merged)

label = Input(shape=[max_text_len], dtype='float32', name='label')
input_length = Input(shape=[1], dtype='int64', name='input_length')
label_length = Input(shape=[1], dtype='int64', name='label_length')

args = [y_pred, label, input_length, label_length]
loss_out = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')(args)

model = Model(inputs=[input_data, label, input_length, label_length],
              outputs=loss_out)

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True, clipnorm=5)
model.compile(loss={'ctc': lambda y_true, y_pred: y_pred},
              optimizer=sgd)

return model

```

Trong mô hình được xây dựng, nửa đầu là CNN với 4 lớp 2D-Convolution. Sau đó kết quả của CNN được đi qua hai Bidirectional RNN : RNN thứ nhất được tạo thành từ thành phần gru1 (forward) và gru1\_back (backward), mỗi thành phần gồm 512 phần tử GRU. RNN thứ 2 được tạo từ 2 thành phần là gru2 và gru2\_back. Lớp với tên y\_pred là lớp đưa ra kết quả dòng ký tự trong ảnh. Nếu in ra thông tin về các lớp trong mô hình (print(model.summary())), thì kích thước của đầu ra của y\_pred là 32x11, trong đó 11 là số nhóm ký tự (10 ký tự từ 0 đến 9 và 1 ký tự trống), còn 32 là kích thước được lựa chọn để đủ chứa chuỗi ký tự đầu ra (tính cả khả năng các ký tự trống xuất hiện ở giữa)

Mô hình đáng lẽ sẽ kết thúc ở lớp y\_pred, tuy nhiên nếu dừng lại tại đây thì không lựa chọn được hàm mục tiêu để đánh giá độ khớp giữa y\_pred so với chuỗi ký tự thực trong ảnh (y\_true) (ít nhất keras 2.2 vẫn chưa hỗ trợ). Do đó, các thành phần y\_pred, label (chính là giá trị của y\_true), input\_length (độ dài của y\_pred), label\_length (độ dài của label) được gộp lại để tính hàm mục tiêu với tên **ctc** (Connectionist temporal classification). Đây là hàm được dùng để đánh ra độ chính khớp giữa chuỗi kết quả dự đoán của mô hình so với chuỗi kết quả thực, do đó thường được dùng để huấn luyện RNN.

### Bước 3 : Huấn luyện mô hình

Chương trình train.py dưới đây dùng để huấn luyện mô hình

```

# train.py

import numpy as np
from model import create_model

```

```

img_h, img_w = 320, 512
num_letter = 10
max_text_len = 16

num_cnn_layer = 4
downsample_factor = 2 ** num_cnn_layer

def create_data(imgs, labels):
    N = len(imgs)
    input_lengths = np.ones((N, 1)) * (img_w // downsample_factor - 2)
    label_lengths = np.array([[len(label)] for label in labels])
    imgs = [np.expand_dims(img.T, -1).astype('float32')/255
             for img in imgs]
    imgs = np.array(imgs)

    X = {'img' : imgs, 'label' : labels,
         'input_length' : input_lengths,
         'label_length' : label_lengths }

    Y = { 'ctc' : np.zeros(N)}

    return X, Y

imgs_train = np.fromfile('imgs_train.npy', dtype=np.uint8)
imgs_train = imgs_train.reshape(-1, img_h, img_w)
labels_train = np.fromfile('labels_train.npy', dtype=np.uint8)
labels_train = labels_train.reshape(-1, max_text_len)
Xtrain, Ytrain = create_data(imgs_train, labels_train)

imgs_test = np.fromfile('imgs_test.npy', dtype=np.uint8)
imgs_test = imgs_test.reshape(-1, img_h, img_w)
labels_test = np.fromfile('labels_test.npy', dtype=np.uint8)
labels_test = labels_test.reshape(-1, max_text_len)
Xtest, Ytest = create_data(imgs_test, labels_test)

model = create_model()
model.fit(Xtrain, Ytrain, validation_data=(Xtest, Ytest), epochs=40,
         batch_size=32, verbose=True)

model.save_weights('model_ocr_weights.h5')

```

#### Bước 4 : Sử dụng mô hình sau quá trình huấn luyện

Sau khi đã kết thúc quá trình huấn luyện, có thể dùng mô hình để đọc số từ các ảnh thẻ tín dụng

```

# predict.py

import os
import itertools
import numpy as np
from model import create_model
from PIL import Image
import tensorflow as tf
from keras import backend as K

letters = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

sess = tf.Session()
K.set_session(sess)

model = create_model()
model.load_weights('model_ocr_weights.h5')
net_inp = model.get_layer(name='img').input
net_out = model.get_layer(name='softmax').output

```

```
def decode(out):
    out_best = list(np.argmax(out[2:], 1))
    out_best = [k for k, g in itertools.groupby(out_best)]
    outstr = ''
    for c in out_best:
        if c < len(letters):
            outstr += letters[c]
    return outstr

def getImageText(img_path):
    img = Image.open(img_path)
    img = img.convert('L')
    img = np.array(img)
    X_data = np.array([np.expand_dims(img.T, -1).astype('float32')/255])
    net_out_value = sess.run(net_out, feed_dict={net_inp:X_data})
    pred_text = decode(net_out_value[0])
    return pred_text

print(getImageText('creditcard_images/test/4024007102907408.jpg'))
```

Như trong phần xây dựng mô hình đã nói, đầu ra của mô hình được dùng để tính hàm mục tiêu *ctc*, phục vụ quá trình huấn luyện, do đó muốn lấy kết quả của chuỗi kí tự cần lấy thông tin ở đầu ra của lớp *y\_pred*. Đầu ra này có kích thước 11x32, tương đương tối đa 32 kí tự, mỗi kí tự nhận một trong 11 giá trị - từ 0 đến 9 và 1 kí tự trống.

Với ảnh test 4024007102907408.jpg, giá trị của chuỗi 32 kí tự như sau:

```
4,4,4,4,10,0,2,10,4,10,0,10,0,7,1,10,10,0,2,9,10,0,10,7,4,4,0,8,10,10,10,10
```

Một kí tự trong chuỗi gốc có thể xuất hiện một vài lần trong chuỗi kết quả, để xử lý sự dư thừa này, hàm *groupby* của *itertools* được dùng để nhóm các kí tự giống nhau và nằm cạnh nhau làm một, kết quả chuỗi đầu ra trở thành:

```
4,10,0,2,10,4,10,0,10,0,7,1,10,0,2,9,10,0,10,7,4,0,8,10
```

Cuối cùng, kí tự với mã 10 được xem là kí tự trống (hoặc đơn giản là token phân cách 2 kí tự), do đó chúng được bỏ đi, kết quả còn lại là :

```
4024007102907408
```

Đây chính là chuỗi số trên ảnh của thẻ.

Nếu thử trên toàn bộ 200 ảnh test thì có khoảng 180 ảnh cho kết quả chính xác. Các trường hợp sai thường do bị mất một kí tự nằm lẫn vào trong ảnh nền. Có thể nâng cao độ chính xác nếu có nhiều ảnh training hơn, cho quá trình training lâu hơn, hoặc sử dụng CNN với 3 kênh màu, thay vì chỉ dùng 1 kênh gray như trong ví dụ.

## 8. Tìm kiếm bằng hình ảnh

Mục đích của việc tìm kiếm bằng hình ảnh là từ một ảnh mẫu (query image) tìm ra trong kho dữ liệu các ảnh giống với ảnh mẫu nhất (giống như tìm kiếm nội dung bằng văn bản)

Để thực hiện được việc tìm kiếm bằng hình ảnh, cần xác định được mức độ tương tự giữa 2 ảnh, hay cần có một hàm khoảng cách để đo mức độ giống nhau của 2 ảnh. Ngoài ra việc tính toán khoảng cách này phải thực hiện rất nhanh vì lượng dữ liệu trong kho lưu trữ rất nhiều.

Cách xây dựng một hàm khoảng cách như vậy tương tự cách làm với ảnh khuôn mặt trong phần nhận dạng khuôn mặt (hay nói đúng, so sánh 2 ảnh khuôn mặt là trường hợp riêng của bài toán tổng quát chúng ta đang xem xét ở đây). Theo đó, chúng ta cần xây dựng một model để chuyển một ảnh có kích thước dữ liệu lớn thành một vector có độ dài nhỏ hơn nhiều (vector Embedding, như đã trình bày trong phần nhận dạng khuôn mặt). Sau đó, việc so sánh các ảnh được thực hiện qua việc đo khoảng cách Euclide giữa các vector Embedding.

## Sử dụng pretrained model từ ImageNet để tính Embedding

Các pretrained model trên ImageNet có thể được sử dụng cho việc tạo ra vector Embedding của một ảnh. ImageNet có 1000 nhóm đối tượng, lớp cuối của các mô hình ImageNet luôn có 1000 đầu ra tương ứng với 1000 nhóm đối tượng đó. Thông tin ở lớp này ít có tác dụng cho các mục đích tổng quát, vì đã được “ép” vào một trong các vector *onehot* của 1000 nhóm đối tượng. Tuy nhiên, lớp liền trước của lớp cuối chứa đựng nhiều thông tin tổng quát về các cấu trúc hình học và màu sắc, do đó có thể dùng để làm vector Embedding cho ảnh.

Bất kỳ mô hình nào từ ImageNet (VGG, Resnet, InceptionNet ...) đều có thể sử dụng cho mục đích “khai thác thuộc tính” ở trên. Ví dụ dưới đây minh họa cách dùng VGG để tạo vector Embedding:

```
from keras.applications import VGG16
from keras.models import Model

model = VGG16()
feat_extractor = Model(inputs=model.input,
                       outputs=model.layers[-2].output)
```

Với đoạn chương trình trên, chúng ta đã có được mô hình `feat_extractor` cho phép tính Embedding của một ảnh. Đầu vào của `feat_extractor` chính là đầu vào của VGG16 (kích thước 224x224x3), đầu ra của `feat_extractor` là đầu ra của lớp gần cuối của VGG (`layers[-2]`), đầu ra này có kích thước bằng 4096. Như vậy vector Embedding ở đây có độ dài bằng 4096.

Để rõ hơn, chúng ta sẽ xây dựng ứng dụng tìm kiếm hình ảnh trên tập dữ liệu ảnh mẫu [flower\\_images.zip](#). Tập ảnh này chứa các ảnh hoa theo từng nhóm, mỗi nhóm có từ 5-7 ảnh giống nhau. Mỗi nhóm hoa được gán một số hiệu, trong đó chữ số hàng trăm là màu sắc của hoa (Ví dụ : 101, 102 - đỏ, 201, 202 - hồng, ...)

Chương trình sau tính Embedding của toàn bộ ảnh trong tập dữ liệu và ghi vào file nhị phân numpy (để phục vụ việc tìm kiếm nhanh):

```
# features_extraction.py

import os
import numpy as np
from PIL import Image
from keras.applications import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.models import Model

model = VGG16()
feat_extractor = Model(inputs=model.input,
                       outputs=model.layers[-2].output)

img_dir = 'flower_images'
image_paths = []
features = []

for root, dir, files in os.walk(img_dir):
```

```

for file in files:
    img_path = os.path.join(root, file)
    img = Image.open(img_path).resize((224, 224))
    X = np.array(img)
    X = preprocess_input(np.array([X]))
    feature = feat_extractor.predict(X)[0]
    features.append(feature)
    image_paths.append(img_path)

features = np.array(features, dtype=np.float32)
features.tofile('features.npy')

f = open('img_list.txt', 'w')
for img_path in image_paths:
    f.write(img_path + '\n')
f.close()

```

Chương trình sau cho phép tìm kiếm ảnh giống nhất với một ảnh mẫu:

```

# find_similar_images.py

import numpy as np
import random

f = open('img_list.txt')
image_paths = [line.strip() for line in f]
f.close()
N = len(image_paths)

features = np.fromfile('features.npy', dtype=np.float32)
features = features.reshape((N, -1))

i = random.randint(0, N-1)
print('Query image : ', image_paths[i])































distances = [(np.linalg.norm(features[j] - features[i]), j)
              for j in range(N) if j != i]

distances = sorted(distances)

print('Top 10 Similar Images : ')
for _, j in distances[:10]:
    print(image_paths[j])

```

Kết quả một số ví dụ tìm kiếm :

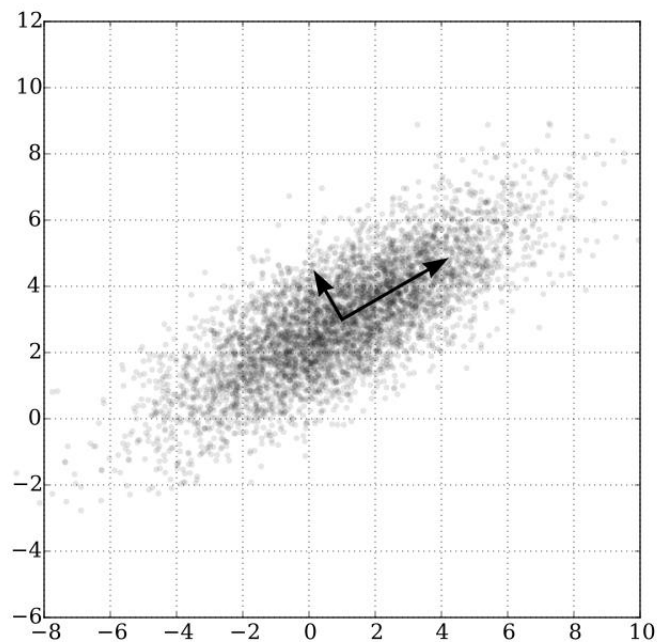
Query Image	Top similar Images				
					
					
					
					
					

Kết quả tìm kiếm bằng hình ảnh dựa trên feature lấy từ VGG16

## Giảm kích thước vector Embedding bằng Principal Component Analysis (PCA)

Principal Component Analysis (PCA) là phép phân tích để tìm ra các thuộc tính quan trọng của một tập dữ liệu. Trong một tập dữ liệu, mỗi mẫu dữ liệu có nhiều thuộc tính, nhưng để phân biệt các mẫu với nhau, chỉ một số thuộc tính giữ vai trò quan trọng.

Về mặt toán học, PCA giống như một phép chiếu từ không gian nhiều chiều lên không gian với số chiều ít hơn (ví dụ chiếu từ không gian 3 chiều lên một mặt phẳng). Không gian chiếu được chọn sao cho mức độ biến thiên (variance) của hình chiếu các điểm thuộc tính là lớn nhất. Các chiều của không gian chiếu (ít hơn so với số chiều của không gian thuộc tính gốc) chính là các thuộc tính quan trọng mà PCA đã giữ lại.



Minh họa phép chiếu trong không gian thuộc tính qua PCA . Nguồn : Wikipedia

Quay lại bài toán tìm kiếm ảnh, chúng ta đã có vector Embedding với kích thước 4096. Tuy nhiên, chỉ một số thành phần trong 4096 thuộc tính này quyết định đến sự khác biệt của các bức ảnh. Do đó chúng ta dùng PCA để giảm kích thước của vector Embedding. Việc này giúp tăng tốc độ tính toán và giảm kích thước dữ liệu (các thuộc tính của các ảnh trong kho ảnh) cần lưu trữ để phục vụ tìm kiếm nhanh.

Chương trình sử dụng PCA để giảm kích thước vector Embedding:

```
# pca_dimension_reduction.py

import numpy as np
from sklearn.decomposition import PCA

features = np.fromfile('features.npy', dtype=np.float32)
features = features.reshape((-1, 4096))

pca = PCA(n_components=256)
pca.fit(features)
pca_features = pca.transform(features)
pca_features.tofile('features_pca.npy')
```

Sau khi chạy chương trình trên, từ thuộc tính gốc (`features.npy`) với kích thước 4096 cho 1 mẫu dữ liệu, thuộc tính mới được sinh ra (`features_pca.npy`) với kích thước 256 cho 1 mẫu dữ liệu. Sau đó chúng ta có thể dùng file thuộc tính mới cho chương trình tìm kiếm ảnh bằng cách thay dòng:

```
features = np.fromfile('features.npy', dtype=np.float32)
```

bằng:

```
features = np.fromfile('features_pca.npy', dtype=np.float32)
```

Kết quả tìm kiếm cơ bản không thay đổi, nhưng tốc độ tìm kiếm sẽ nhanh hơn.

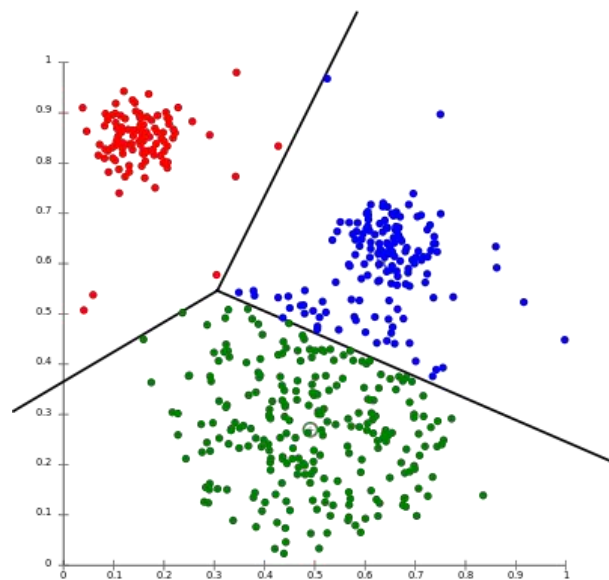


## Phân nhóm dữ liệu với K-Mean để tìm kiếm nhanh

Nếu kho dữ liệu có hàng triệu ảnh, khi tìm kiếm, chương trình sẽ phải tính khoảng cách giữa thuộc tính ảnh đầu vào với thuộc tính của tất cả các ảnh trong kho dữ liệu, quá trình này sẽ rất lâu.

Thay vì tìm trong toàn kho dữ liệu, có thể phân nhóm kho dữ liệu thành các “cụm” với thuật toán K-mean, sau đó với mỗi ảnh đầu vào, việc tìm kiếm chỉ thực hiện trên các “cụm” có thuộc tính gần với thuộc tính của ảnh đầu vào nhất.

K-Mean là thuật toán phân nhóm theo cách “tự tổ chức” (self-organization), một hình thức phân nhóm tự động không cần dữ liệu huấn luyện của con người. Cách thức thực hiện của K-Mean như sau : Giả sử đã có một tập dữ liệu với các vector thuộc tính, cần chia tập này thành K nhóm (số K cần được chọn bởi con người), K-Mean sẽ tạo ra K nhóm dữ liệu sao cho sự biến động về giá trị của các thuộc tính trong mỗi nhóm là nhỏ nhất, trong khi sự biến động về giá trị của các thuộc tính giữa các nhóm là lớn nhất.



Minh họa các K-Mean phân chia các nhóm dữ liệu

Chương trình sau dùng K-Mean để phân chia tập dữ liệu Embedding trong phần tìm kiếm ảnh ở phần trên thành K=5 nhóm:

```
# kmean_clustering.py

from sklearn.cluster import KMeans
import numpy as np

K = 5

features = np.fromfile('features_pca.npy', dtype=np.float32)
features = features.reshape((-1, 256))

kmeans = KMeans(n_clusters=K, random_state=0).fit(features)
labels = kmeans.labels_
labels.tofile('labels.npy')

cluster_centers = kmeans.cluster_centers_
cluster_centers.tofile('cluster_centers.npy')
```

Chương trình sau dùng các nhóm do K-Mean tạo ra để tìm kiếm ảnh tương tự với một ảnh cho trước. Với mỗi ảnh đầu vào, chương trình sẽ xác định khoảng cách từ thuộc tính của ảnh đó tới tâm của các nhóm (`cluster_centers`), sau đó quá trình tìm kiếm sẽ chỉ thực hiện trong nhóm gần nhất với thuộc tính ảnh đầu vào:

```
# find_similar_images_w_kmean.py

import numpy as np
import random
import sys

K = 5

f = open('img_list.txt')
image_paths = [line.strip() for line in f]
f.close()
N = len(image_paths)

labels = np.fromfile('labels.npy', dtype=np.int32)
cluster_centers = np.fromfile('cluster_centers.npy', dtype=np.float32)
cluster_centers = cluster_centers.reshape((K, -1))

features = np.fromfile('features_pca.npy', dtype=np.float32)
features = features.reshape((N, -1))

i = random.randint(0, N-1)
print('Query image : ', image_paths[i])































dist_to_centers = np.sum((cluster_centers - features[i]) ** 2, axis=1)
kmin = np.argmin(dist_to_centers)
indexes = np.where(labels == kmin)[0]

distances = [(np.linalg.norm(features[j] - features[i]), j)
              for j in indexes if j != i]

distances = sorted(distances)

print('Top 10 Similar Images : ')
for _, j in distances[:10]:
    print(image_paths[j])
```

Kết quả một số ví dụ tìm kiếm

Query Image	Top similar Images					
						
						
						
						
						

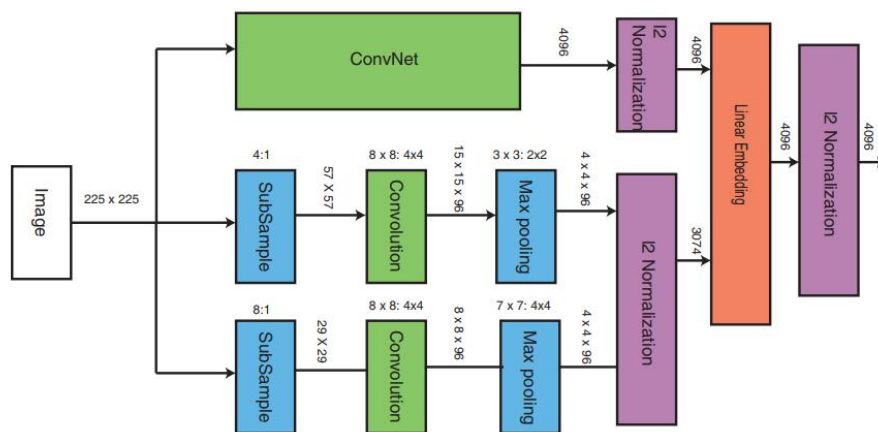
Một số kết quả tìm kiếm sau khi phân nhóm dữ liệu với K-mean

Có thể thấy việc dùng K-Mean giúp cho kết quả tìm kiếm đồng nhất hơn về màu sắc, do các ảnh thuộc cùng nhóm thường có màu giống nhau.

## Training trên tập dữ liệu riêng

Việc dùng pretrained model từ ImageNet khá thuận tiện và tiết kiệm thời gian do không cần xây dựng & huấn luyện mô hình riêng. Tuy nhiên, khi có tập dữ liệu riêng và tiêu chí giống nhau của các ảnh trong tập dữ liệu cũng đặc thù, chúng ta có thể xây dựng mô hình riêng cho tập dữ liệu này.

Một trong các mô hình được sử dụng cho việc tính Embedding là [DeepRanking](#). Về cơ bản, mô hình này cũng dựa trên các mô hình từ ImageNet (cũng giữ lại lớp gần cuối), tuy nhiên có bổ sung thêm thành phần để khai thác các macro-feature trong ảnh (các feature ở lớp đầu trong mạng CNN). Các feature này có tác dụng quan trọng trong việc so sánh sự giống nhau giữa các ảnh, (trong khi các feature từ ImageNet model thường chỉ tốt cho việc phân loại ảnh)



Cấu trúc của mô hình Deep Ranking.

Nguồn : [Learning Fine-grained Image Similarity with Deep Ranking](#)

Nhìn vào cấu trúc của mô hình Deep Ranking, có thể thấy model gồm 3 kênh song song : Kênh phía trên cùng (ConvNet) chính là mô hình từ ImageNet. Hai kênh dưới chính là các kênh để khai thác các macro-feature trong ảnh.

### Triplet:

Khái niệm triplet được dùng trong Deep Ranking (và trong các mô hình cần so sánh sự giống nhau giữa các đối tượng). Một triplet là một bộ 3 (q,p,n) trong đó q (query) là đối tượng gốc , p (positive) là mẫu giống với q, n (negative) là mẫu khác với q. Hàm mục tiêu (loss function) dựa trên triplet được định nghĩa:

$$L(q, p, n) = D(q, p) - D(q, n)$$

Trong đó D(q,p) là khoảng cách từ mẫu gốc đến mẫu tương tự, D(q,n) là khoảng cách từ mẫu gốc đến mẫu khác biệt. Tập dữ liệu huấn luyện gồm các bộ (p,q,n), quá trình huấn luyện sẽ tối thiểu hóa hàm mục tiêu ở trên. Việc này tương đương với làm cho khoảng cách giữa các mẫu giống nhau nhỏ đi, trong khi khoảng cách giữa các mẫu khác nhau tăng lên.

## Xây dựng mô hình Deep Ranking cho bài toán tìm kiếm ảnh

Cách xây dựng mô hình Deep Ranking dựa trên keras có thể tham khảo từ [link](#).

### Chuẩn bị dữ liệu:

```
# prepare_data.py

import os
import numpy as np
from PIL import Image

img_size = 224
image_dir = 'flower_images'

def load_data(tag):
    data = []
```

```

group_ids = []
image_paths = []
for root, dirs, files in os.walk(os.path.join(image_dir, tag)):
    for file in files:
        group_id = int(os.path.split(root)[-1])
        img_path = os.path.join(root, file)
        image_paths.append(img_path)
        img = Image.open(img_path).resize((img_size, img_size))
        data.append(np.array(img))
        group_ids.append(group_id)

data = np.array(data)
group_ids = np.array(group_ids, dtype=np.int32)

data.tofile(tag + '_data.npy')
group_ids.tofile(tag + '_group_ids.npy')

with open(tag + '_img_list.txt', 'w') as fid:
    for img_path in image_paths:
        fid.write(img_path + '\n')

load_data('train')
load_data('test')

```

Chương trình tạo ra các file dữ liệu :

- train\_data.npy, test\_data.npy : Dữ liệu ảnh
- train\_group\_ids.npy, test\_group\_ids.npy : id nhóm của các ảnh
- train\_img\_list.txt, test\_img\_list.txt : danh sách đường dẫn tới các ảnh

## Xây dựng mô hình

```

# model.py

from keras import backend as K
from keras.models import Model
from keras.layers import Dense, Flatten, Conv2D, \
    MaxPool2D, GlobalAveragePooling2D, Dropout, Lambda, Reshape
from keras.layers.merge import concatenate
from keras.optimizers import SGD
from keras.applications.mobilenet import MobileNet

img_size = 224
batch_size = 16 * 3

def _loss_tensor(y_true, y_pred):
    epsilon = 1e-7
    y_pred = K.clip(y_pred, epsilon, 1.0-epsilon)
    loss = 0

    for i in range(0, batch_size//3):
        q_embedding = y_pred[3*i+0]
        p_embedding = y_pred[3*i+1]
        n_embedding = y_pred[3*i+2]
        D_q_p = K.sqrt(K.sum((q_embedding - p_embedding)**2))
        D_q_n = K.sqrt(K.sum((q_embedding - n_embedding)**2))
        loss = (loss + D_q_p - D_q_n )

    return 2 + loss/(batch_size//3)

def create_model():
    mb_model = MobileNet(weights=None, include_top=False)

```

```

input = mb_model.input
x = mb_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.6)(x)
x = Lambda(lambda _x: K.l2_normalize(_x, axis=1))(x)

input = Reshape(target_shape=(img_size, img_size, 3))(input)

chan1 = Conv2D(64, kernel_size=(8, 8),strides=(16,16),
               padding='same')(input)
chan1 = MaxPool2D(pool_size=(3,3),strides = (4,4),
                  padding='same')(chan1)
chan1 = Flatten()(chan1)
chan1 = Lambda(lambda x: K.l2_normalize(x,axis=1))(chan1)

chan2 = Conv2D(64, kernel_size=(8, 8),strides=(32,32),
               padding='same')(input)
chan2 = MaxPool2D(pool_size=(7,7),strides = (2,2),
                  padding='same')(chan2)
chan2 = Flatten()(chan2)
chan2 = Lambda(lambda x: K.l2_normalize(x,axis=1))(chan2)

merge= concatenate([x, chan1, chan2])
emb = Dense(256)(merge)
l2_norm_final = Lambda(lambda x: K.l2_normalize(x,axis=1))(emb)

model = Model(inputs=mb_model.input, outputs=l2_norm_final)
sgd = SGD(lr=0.001, momentum=0.9, nesterov=True, decay=1e-6)
model.compile(loss=_loss_tensor, optimizer=sgd)
return model

```

Mô hình được xây dựng dựa trên MobileNet, có bổ sung thêm các thành phần chan1, chan2 cho việc khai thác macro-feature. Đầu ra của model là 1 vector Embedding có độ dài bằng 256. Hàm mục tiêu được xây dựng bằng cách tính hiệu ( $D_{q,p} - D_{q,n}$ ) - như đã trình bày trong phần trước, cộng thêm một hằng số (chọn bằng 2, để làm cho giá trị loss không âm). Dữ liệu huấn luyện cần được gộp theo từng bộ 3 (q,p,n) liên tiếp nhau, để có thể tính hàm mục tiêu theo cách trên.

## Huấn luyện mô hình

```

# train.py

import numpy as np
import random
from model import create_model
from keras.callbacks import EarlyStopping, ModelCheckpoint

img_size = 224
train_factor = 0.8
batch_size = 16 * 3

class DataGenerator:
    def __init__(self, tag, batch_size):
        self.data = np.fromfile(tag + '_data.npy', dtype=np.uint8)
        self.data = self.data.reshape((-1, img_size, img_size, 3))
        self.data = self.data.astype('float32')/255.0
        self.batch_size = batch_size
        self.N = len(self.data)
        self.current_index = 0
        self.group_ids = np.fromfile(tag + '_group_ids.npy', dtype=np.int32)
        self.group_indexes = {}

        for gid in np.unique(self.group_ids):

```

```

        self.group_indexes[gid] = np.where(self.group_ids == gid)[0]

    self.color_group_ids = self.group_ids // 100 - 1
    self.color_group_indexes = {}
    for cgid in np.unique(self.color_group_ids):
        self.color_group_indexes[cgid] = \
            np.where(self.color_group_ids == cgid)[0]

    def get_num_samples(self):
        return self.N

    def next_batch(self):
        Xb = np.zeros((self.batch_size, img_size, img_size, 3))
        Yb = np.zeros(self.batch_size)
        while True:
            for i in range(batch_size//3):
                i_q = self.current_index
                gid = self.group_ids[i_q]
                cgid = self.color_group_ids[i_q]
                p_indexes = set(self.group_indexes[gid]) - set([i_q])

                if random.random() < 0.3:
                    n_indexes = set(self.color_group_indexes[cgid]) \
                        - p_indexes - set([i_q])
                else:
                    n_indexes = set(range(self.N)) - p_indexes - set([i_q])

                i_p = random.choice(list(p_indexes))
                i_n = random.choice(list(n_indexes))

                Xb[3*i] = self.data[i_q]
                Xb[3*i + 1] = self.data[i_p]
                Xb[3*i + 2] = self.data[i_n]
                self.current_index += 1

            if self.current_index == self.N:
                self.current_index = 0
                self.data_indexes = np.random.permutation(self.N)

        yield Xb, Yb

train_generator = DataGenerator('train', batch_size)
test_generator = DataGenerator('test', batch_size)
Ntrain = train_generator.get_num_samples()
Ntest = test_generator.get_num_samples()

model = create_model()

early_stopping = EarlyStopping(monitor='val_loss', patience=5)
filepath = 'model-{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss',
                             verbose=1, save_best_only=True, mode='min')
callbacks = [early_stopping, checkpoint]

model.fit_generator(generator=train_generator.next_batch(),
                    steps_per_epoch=20*Ntrain//batch_size,
                    epochs=10,
                    validation_data=test_generator.next_batch(),
                    validation_steps=10*Ntest//batch_size,
                    callbacks=callbacks, verbose=True)

```

Class DataGenerator được dùng để sinh ra các bộ triplet theo nguyên tắc:

- q : được chọn tuần tự từ các mẫu dữ liệu

- p : chọn ngẫu nhiên một mẫu trong cùng nhóm với q
- n : chọn một mẫu ngẫu nhiên không cùng nhóm với q

Nếu chọn n ngẫu nhiên từ các mẫu không cùng nhóm với q thì khả năng cao là q & n sẽ không cùng màu sắc, khi đó  $D(q,n)$  sẽ lớn nhưng không phản ánh đúng sự khác biệt về hình học, do đó chương trình dành 30% số mẫu để chọn n và q có cùng màu sắc:

```
if random.random() < 0.3:
    n_indexes = set(self.color_group_indexes[cgid]) - p_indexes - set([i_q])
```

Việc này giúp tránh tình trạng model được tạo ra sẽ chỉ phân biệt các ảnh chủ yếu dựa trên màu sắc.

## Sử dụng mô hình sau quá trình huấn luyện

Trước hết, chúng ta dùng mô hình để sinh ra các feature cho các ảnh, như cách đã làm với các pretrained model từ ImageNet:

```
# gen_features.py

import numpy as np
from keras import backend as K
from keras.models import load_model

img_size = 224

def _loss_tensor(y_true, y_pred):
    return K.sqrt(K.sum((y_true - y_pred)**2))

model = load_model('model.h5',
                   custom_objects={'_loss_tensor': _loss_tensor})

train_data = np.fromfile('train_data.npy', dtype=np.uint8)
train_data = train_data.reshape((-1, img_size, img_size, 3))
test_data = np.fromfile('test_data.npy', dtype=np.uint8)
test_data = test_data.reshape((-1, img_size, img_size, 3))
data = np.concatenate((train_data, test_data))

data = data.astype('float32')/255.0
features = model.predict(data, batch_size=32, verbose=True)
features.tofile('features.npy')
```

Sau khi có feature của các ảnh, chúng ta có thể dùng để tìm kiếm ảnh như cách làm với các pretrained model từ ImageNet. Một điểm khác nhỏ là các ảnh trong thư mục train đã được dùng trong huấn luyện nên việc tìm kiếm chỉ thử nghiệm trên các ảnh trong thư mục test.

```
# find_similar_images

import numpy as np
import random
import sys

train_image_paths = []

with open('train_img_list.txt') as fid:
    for line in fid:
        train_image_paths.append(line.strip())

test_image_paths = []
with open('test_img_list.txt') as fid:
    for line in fid:
        test_image_paths.append(line.strip())
```



```

image_paths = train_image_paths + test_image_paths
N = len(image_paths)
Ntrain = len(train_image_paths)

features = np.fromfile('features.npy', dtype=np.float32)
features = features.reshape((N, -1))

i = random.randint(Ntrain, N-1)
print('Query image : ', image_paths[i])



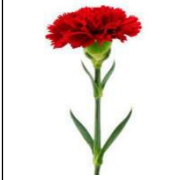


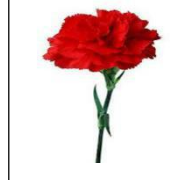


















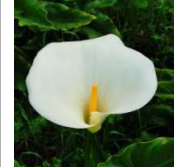





distances = [(np.linalg.norm(features[j] - features[i]), j)
              for j in range(N) if j != i]

distances = sorted(distances)

print('Top 10 Similar Images : ')
for _, j in distances[:10]:
    print(image_paths[j])

```

Kết quả một số ví dụ tìm kiếm:

Query Image	Top Similar Images				
					
					
					
					
					

Kết quả tìm kiếm dựa trên mô hình train theo phương pháp Deep Ranking

Việc tìm kiếm có thể chưa chính xác do số lượng ảnh train khá ít (chỉ khoảng 1000 ảnh). Ngoài ra có nhiều cách chọn các bộ triplet để kết quả mô hình tốt hơn. Chi tiết có thể tham khảo trong tài liệu gốc về Deep Ranking.