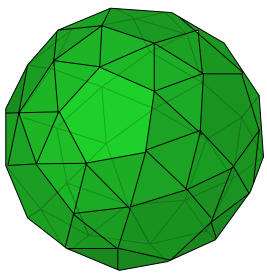
Projects in Mathematics and Applications

OBJECT DETECTION

Ngày 29 tháng 8 năm 2019

Nguyễn Đắc Khôi Nguyên*
Phạm Nguyễn Hằng Vân ‡
†Trương Minh Hoàng
§Lê Đình Hải



*Trường THPT Nguyễn Hữu Huân, TP.HCM
†Northfield Mount Hermon School, MA, USA
‡Trường THPT chuyên Trần Phú, TP.Hải Phòng
§Trường THPT chuyên Lê Quý Đôn, Khánh Hòa

Lời cảm ơn

Lời đầu tiên, chúng tôi xin gửi đến Ban tổ chức trại hè PiMA và nhà sáng lập - anh Lê Việt Hải, anh Trần Hoàng Bảo Linh, anh Cấn Trần Thành Trung - lời cảm ơn chân thành nhất. Bên cạnh việc tạo điều kiện để tất cả các trại sinh đến từ mọi miền đất nước được tham gia, giao lưu, học hỏi, các anh còn tạo nên một cộng đồng những học sinh yêu thích toán sẵn sàng giúp đỡ chúng tôi.

Chúng tôi cũng xin được gửi lời cảm ơn các anh chị quản lí, người hướng dẫn và các thầy cô, khách mời mà chúng tôi có cơ hội tiếp xúc trong thời gian trại hè. Đặc biệt, xin cảm ơn anh Thế Anh, anh Long - những đàn anh cố vấn nhóm về kiến thức đề tài và kĩ năng lập trình; chị Thục Như, chị Ngọc - những người chị đã động viên và chỉ bảo chúng tôi cách làm việc nhóm, cách viết báo cáo, ... Sự quan tâm, theo sát của anh chị từ những ngày đầu tiên là nguồn động lực lớn của chúng tôi.

Đồng thời, nhóm xin cảm ơn các nhà tài trợ đã tạo điều kiện về cơ sở vật chất để trại hè có thể diễn ra thuận lợi, an toàn.

Đến năm thứ 4 của PiMA, với những cải tiến về nội dung giảng dạy và quy mô trại hè, PiMA đã và đang lan tỏa niềm đam mê toán học tới nhiều học sinh THPT trên toàn quốc. Niềm đam mê đó không dừng lại ở sự vui thích mà còn là sự truyền cảm hứng, sức mạnh và tầm nhìn để chúng tôi mạnh mẽ hơn trong quãng đường sắp tới của mình.

Trại hè PiMA 2019 sắp kết thúc. Nhưng chúng tôi tin rằng tinh thần chia sẻ từ những con người nơi đây sẽ thúc đẩy PiMA phát triển trong tương lai. Về phần mình, chúng tôi tin rằng những bài học từ đây sẽ giúp chúng tôi đón đầu các thử thách mới - một cách chủ động và nhiệt tình.

Một lần nữa, xin chân thành cảm ơn PiMA, vì 12 ngày trại vô cùng quý giá.

Tóm tắt nội dung

Vốn là khả năng mà con người thuần thục một cách tự nhiên, máy tính giờ đây có thể xác định các vật thể trong một bức ảnh nhờ sự phát triển vượt bậc của AI. Đây được gọi là Object Detection, một trong những bài toán cơ bản của Computer Vision cùng với một số bài toán kinh điển khác là Image Classification và Image Segmentation. Ở bài báo cáo này, chúng tôi xin trình bày mô hình tổng quan mạng CNN, những biến thể của CNN giúp giải quyết vấn đề hiệu quả hơn, và demo lập trình xác định hình ảnh những con chó trên tập dữ liệu Stanford Dogs. Mong bài viết có thể giúp người đọc hình dung sự tương đồng cũng như khác biệt giữa mạng Neural Network/CNN truyền thống trong Machine Learning và các phiên bản CNN nâng cấp trong Deep Learning, từ đó có cái nhìn định hướng rõ ràng hơn về hai khái niệm quan trọng này.

Mục lục

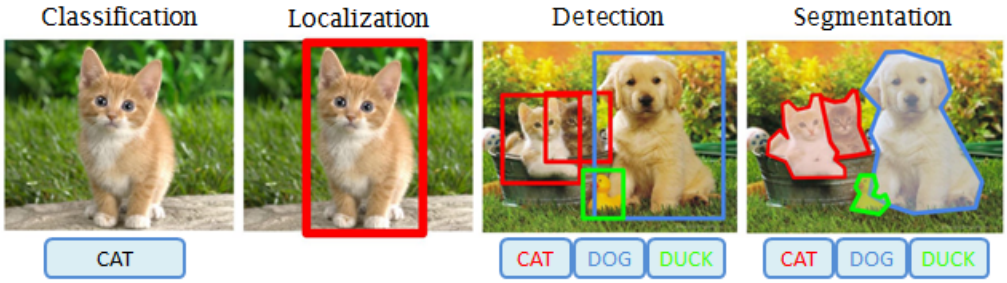
1 Đặt vấn đề

1.1 Computer Vision

Computer Vision (CV) – ngành thị giác máy tính là gì?

Đây là một lĩnh vực trong Trí Tuệ Nhân Tạo và Khoa Học Máy Tính, bao gồm các phương pháp thu nhận, xử lý ảnh kỹ thuật số, trích xuất thông tin có ý nghĩa từ nội dung của hình ảnh. Việc phát triển lĩnh vực này có bối cảnh từ việc sao chép các khả năng thị giác con người bởi sự nhận diện và hiểu biết một hình ảnh mang tính điện tử. Các thuật toán CV có thể được

mô tả là sự kết hợp giữa xử lý hình ảnh và học máy. Một số ứng dụng tiêu biểu có thể kể đến là: phân loại hình ảnh (**Image Classification**), phát hiện hình ảnh vật thể (**Image Detection**), tái tạo cảnh 3D từ hình ảnh 2D, truy xuất hình ảnh, tự động hóa trong giao thông.



Hình 1: Một số bài toán điển hình của Computer Vision

Hình 2: Ứng dụng của Computer Vision trong tự động hóa giao thông

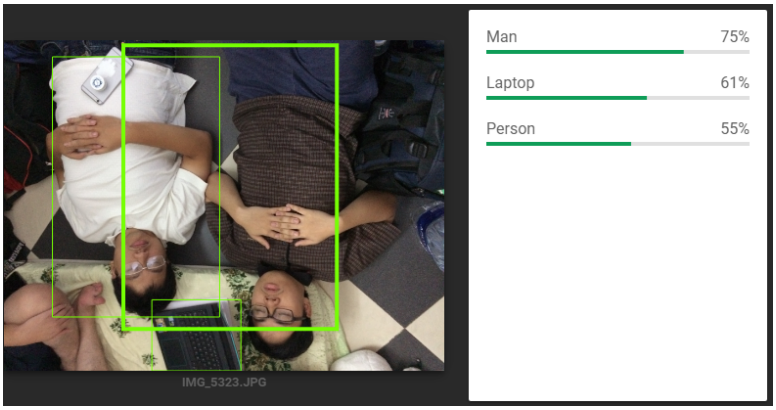
1.2 Object Detection

Object Detection (Nhận Diện Vật Thể) - bài toán kinh điển của thị giác máy tính

Đúng như tên gọi của nó, ở bài toán này ta mong muốn xác định bức ảnh này có những đối tượng nào, ở đâu. Đầu vào (**input**) là một hình ảnh chứa vật thể nào đó mà ta chưa biết

trước, đầu ra (**output**) là vị trí của những vật thể đó được thể hiện bằng cách đóng khung (**bounding box**) và nhãn loại vật thể (**label**). Chẳng hạn, bức ảnh dưới đây có 2 chàng trai trẻ, 1 cái laptop: các chàng trai được dán nhãn ***men***, chiếc laptop được dán nhãn ***laptop***, đóng trong khung xanh riêng. Dưới đây là ví dụ minh họa cho đầu vào và đầu ra của bài toán nhận diện vật thể sử dụng API của Google Vision.

Hình 3: Input



Hình 4: Output

Giải bài toán Object Detection/CV nói chung hay DL/ML nói riêng thường phức tạp và đa dạng tùy vào yêu cầu và điều kiện đặt ra. Bài viết dưới đây tập trung vào cốt lõi là mạng CNN và các phương pháp biến thể của nó để giải quyết vấn đề tối ưu hơn.

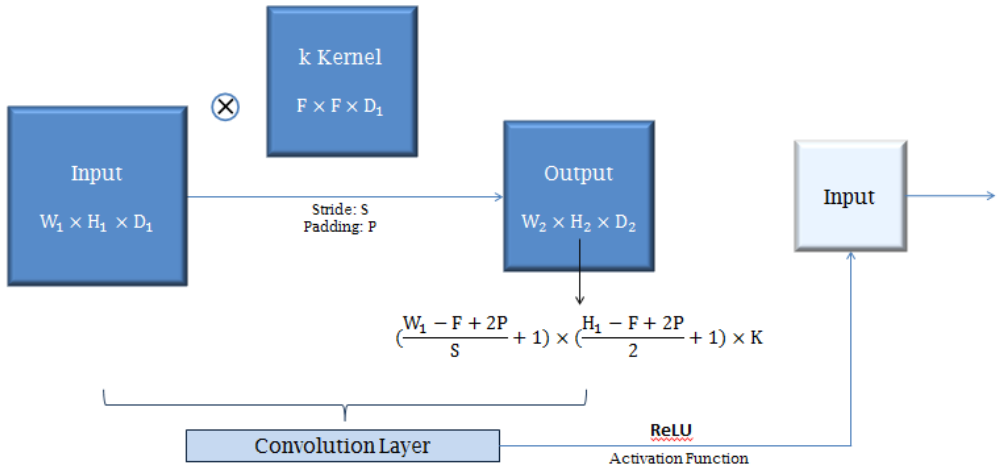
2 Kiến trúc mạng cơ bản của CNN

2.1 Các lớp cơ bản trong CNN

2.1.1 Convolution Layer

1. Mô hình tổng quan

   - Input $W_1 \times H_1 \times D_1$
   - Hyperparameter
     - Số bộ lọc - filter/kernel K
     - Kích thước bộ lọc F
     - Số bước - stride S
     - Số padding P



centering

Hình 5: Hình ảnh Convolution Layer

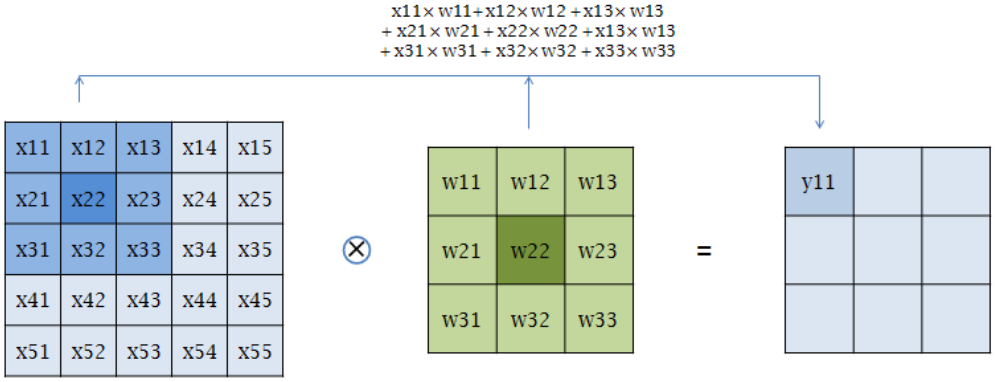
- Output $W_2 \times H_2 \times D_2$
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$
- Parameter
  - Số tham số/filter: $F \cdot F \cdot D_1$
  - Số tham số/layer: $(F \cdot F \cdot D_1) \cdot K$
  - Số bias: K
- Output của layer trước qua hàm kích hoạt trở thành Input của layer tiếp theo.

2. Phép tích chập $X \otimes W = Y$ Một cách trực quan, ta có thể hình dung phép tích chập như

   "trượt"bộ lọc W lên ma trận đầu vào X cho đến khi quét hết các vùng diện tích $F \times F$. Tổng quát hơn, đối với đầu vào là vector 3D, thay vì trượt mặt phẳng ta trượt khối lọc hình hộp lên hết phần thể tích $F \times F \times D$.

   Giả định S = 1, P = 0. Ta thực hiện các bước tính sau:

   - Với mỗi phần tử x_{ijk} trong ma trận X lấy ra một ma trận có kích thước bằng kích thước của kernel W có phần tử x_{ijk} làm trung tâm.
   - Nhân các phần tử tương ứng, sau đó tính tổng rồi viết kết quả vào ma trận Y.



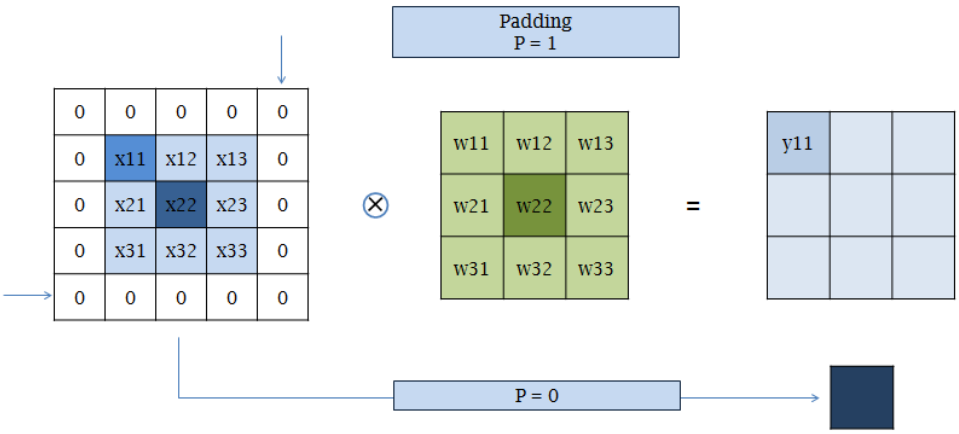
Hình 6: Cách tính tích chập

- Mục đích
  - Phát hiện và trích xuất đặc trưng. Mỗi lớp tích chập chứa các bộ lọc phát hiện đặc trưng (filter - feature detector) cho phép phát hiện và trích xuất những đặc trưng khác nhau của đầu vào, từ đó chuyển đổi để tạo ra dữ liệu đầu vào cho lớp kế tiếp.
  - Sắp xếp bộ lọc ở lớp tích chập để phát hiện các đặc trưng từ đơn giản đến phức tạp. Các lớp đầu tiên dùng bộ lọc hình học (geometric filters) để phát hiện cạnh ngang, dọc, chéo → chi tiết như mắt, mũi, tóc, → lớp tích chập sâu nhất dùng để phát hiện đối tượng hoàn hỉnh như: chó, mèo, chim, ô tô, đèn giao thông, ...

Hình 7: Ứng dụng bộ lọc trong phát hiện đặc trưng cạnh

3. Padding

- P = 1: Chèn thêm 1 vector 0 vào viền
- P = k: Chèn thêm k vector 0 vào viền

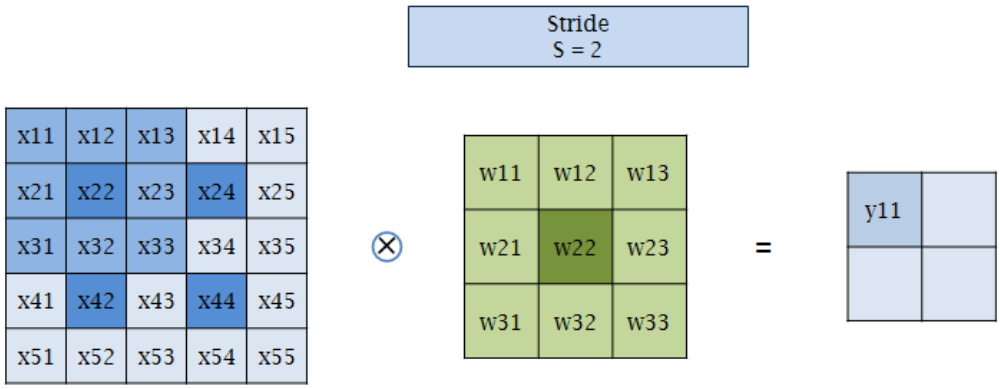


Hình 8: Ví dụ về Padding = 1 tức chèn 1 lớp 0 ở viền

- Mục đích
  - Tạo ra kích thước mong muốn của ma trận đầu ra. Khắc phục việc kích thước ma trận trở nên quá nhỏ sau nhiều phép tích chập.
  - Xử lí các phần tử tại biên. Khắc phục việc mất mát thông tin tại những điểm ảnh ở góc (1 lần) hoặc cạnh (2 lần) được bao phủ ít hơn.

4. Stride

- S = 1: Di chuyển ô trung tâm 1 pixel



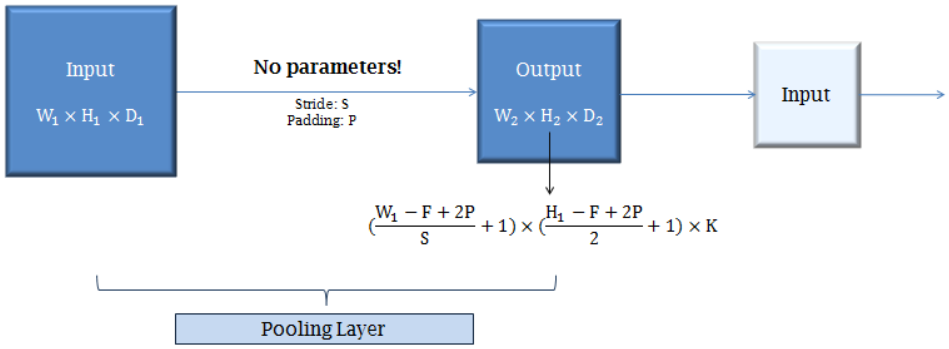
Hình 9: Ví dụ Stride = 2 tức là mỗi bước di chuyển ô trung tâm 2 pixel

- S = k: Di chuyển ô trung tâm k pixel
- Mục đích
  - Tạo ra ma trận đầu ra có kích thước nhỏ hơn ma trận đầu vào.

5. Tự học bộ lọc
   Các giá trị của ma trận lọc được coi như tham số của một mạng nơ-ron và huấn luyện (ví dụ như back-propagation) để có một tập giá trị tối ưu trích xuất thông tin không khi theo chiều cố định mà còn cả các góc lẻ và chi tiết tương đối phức tạp.

2.1.2 Pooling Layer

1. Mô hình tổng quan



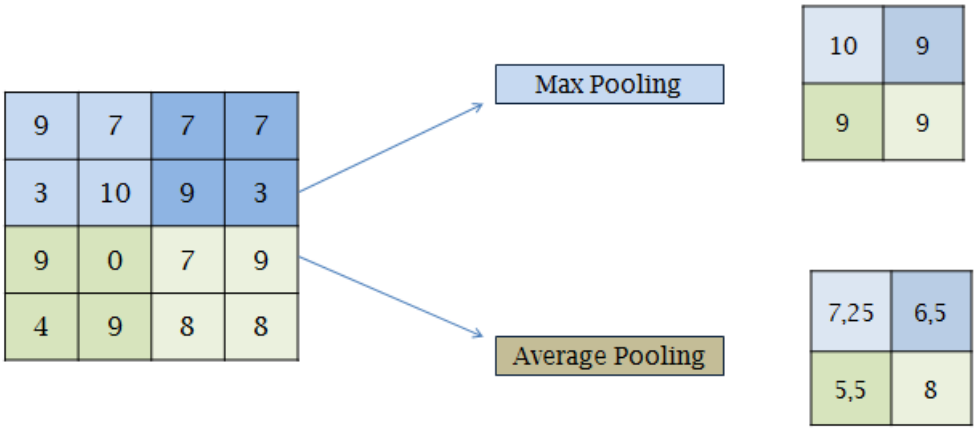
Hình 10: Hình ảnh Pooling Layer

- Input $W_1 \times H_1 \times D_1$
- Hyperparameter
  - Kích thước bộ lọc F
  - Số bước - stride S
  - Số padding P
- Output $W_2 \times H_2 \times D_2$
  - $W_2 = (W_1 - F + 2P)/S + 1$

  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = D_1$
- Các công thức trong Pooling Layer không đổi so với Convolution Layer, các thông số thường sử dụng là $F = 2, S = 2, P = 0$.

2. Các kiểu Pooling phổ biến

- Max Pooling (hay dùng)
- Average Pooling (ít dùng)



Pooling.PNG

Hình 11: Ví dụ Max Pooling và Average Pooling

3. Ý nghĩa

- Tập trung vào các phần "được coi"là đặc trưng của vùng được bao phủ.
- Giảm kích thước đầu vào → Tăng tốc độ tính toán hiệu suất phát hiện đặc trưng.

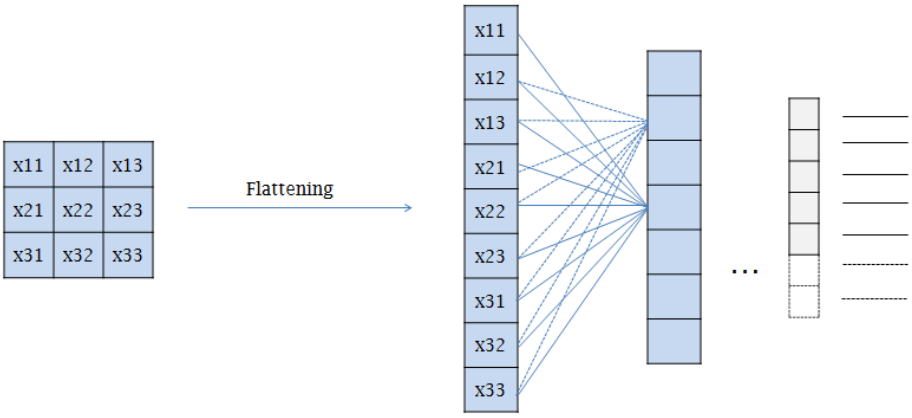
2.1.3 Fully-Connected Layer

Sau khi model đã học được tương đối các đặc điểm của ảnh qua các hidden layer thì tensor của đầu ra layer cuối cùng (kích thước $W_n \times H_n \times D_n$) sẽ được chuyển về 1 vector cùng bộ số. Quá trình chuyển các tham số về dạng vector này gọi là làm phẳng (flatten). Sau đó vector được đưa vào các lớp liên kết đầy đủ (Fully-Connected Layer) để tính toán dựa trên thông tin học được trước đó và đưa ra output cuối cùng.

2.2 Cấu trúc cơ bản mạng CNN

1. Mô hình CNN thường gặp

- INPUT: Pixel của ảnh ứng với điểm trên ma trận.
- CONV: Lớp tích chập trích xuất đặc trưng ảnh.
- Hàm kích hoạt phi tuyến: Phép chập với nhiều bộ lọc trình bày ở trên có thể chuyển thành CNN một lớp bằng cách cộng thêm vào mỗi ma trận ra một số thực và đưa chúng qua một hàm kích hoạt phi tuyến. Trong CNN, hàm $\text{ReLU}(x) = \max(x, 0)$ thường được sử dụng.
- POOLING: Tạo ra các thành phần đầu ra có kích thước nhỏ.



Hình 12: Hình ảnh Fully-Connected Layer

- FC: Sử dụng lớp FC tính toán dữ liệu đầu ra.
- Một số hàm phân loại, tuyến tính, xác suất, ... tùy vào yêu cầu bài toán và hiệu quả thực nghiệm. Ví dụ bài toán Classification chủ yếu sử dụng hàm SOFTMAX $s_i = \frac{\exp(x_i)}{\sum_{j=1} \exp(x_j)}$.
- OUTPUT: Tên vật thể nhận dạng + Vị trí vật (bao quanh bởi bounding box). Biểu diễn dưới dạng vector y với các hệ số p_i (Xác suất vật là I) và b_i (tọa độ, chiều rộng, chiều dài ô vuông).
- Ví dụ mô hình CNN thường gặp:
  → INPUT
  → [[CONV + RELU]*N → POOL?]*M
  → [FC + RELU]*K
  → [FC + SOFTMAX]
  → OUTPUT

2. So sánh CNN và NN
   Đặc điểm khác biệt lớn nhất giữa CNN và NN là CNN có lớp CONV với bộ lọc trích xuất thông tin.

   - Chia sẻ tham số: Tham số ở đây chính là hệ số trong kernel dùng để quét qua ma trận đầu vào. Vì cùng được quét chung bằng bộ kernel nên xảy ra sự dùng chung tham số giữa các vùng gần nhau.
   - Liên kết thưa: Trong CNN một thành phần đầu ra (output unit) chỉ bị ảnh hưởng bởi các điểm lân cận, trái với NN tất cả các nút đều có quan hệ với nhau.
   - Từ hai điều trên, có thể thấy số tham số trong CNN giảm hơn nhiều so với NN.

3 Phát triển CNN để giải quyết bài toán Object Detection

Mô hình mạng CNN là model trợ giúp đắc lực trong việc giải quyết bài toán định dạng hình ảnh. Tuy nhiên, chỉ dùng mạng CNN truyền thống không giải được bài toán. Như chúng ta đã biết, bài toán Detection khác Classification ở việc đòi hỏi output là những vật thể được khoanh vùng tại đúng vị trí của chúng trên ảnh input. Trong khi đó, bài toán phân loại chỉ yêu cầu máy tính phân biệt hình nào chứa vật thể gì. Vấn đề nảy sinh ở chỗ lớp output của bài toán xác định luôn thay đổi bởi vì tần số xuất hiện của một vật thể nào đó là khác nhau

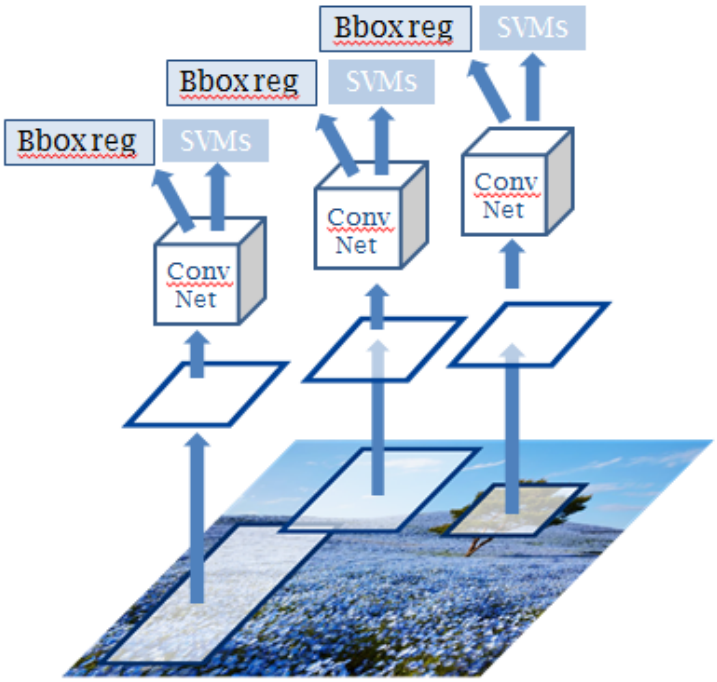
tùy thuộc vào bức ảnh. Chính vì thế, dùng mạng CNN thông thường để tìm vật sẽ hao tốn một lượng bộ nhớ khổng lồ bởi các spatial pixel của các vật trong cùng một ảnh sẽ chồng lên nhau. Các phương pháp giới thiệu sau đây sẽ giúp ta giải quyết bài toán xác định hình ảnh trong thời gian ngắn hơn mà kho tiêu tốn bộ nhớ nhiều.

3.1 R-CNN

Để giải quyết bài toán trên, vào năm 2012 nhà nghiên cứu Ross Girshick đã đề xuất một phương pháp để phát triển mạng CNN truyền thống: **Regional Convoluntional Neural Network (R-CNN)**.

Ý tưởng của R-CNN:

1. Đề xuất những vùng quan tâm (**region proposal**) bằng thuật toán **Selective Search Algorithm (SSA)** để lấy ra 2000 vùng là các khung hình chữ nhật có khả năng chứa vật thể.
2. 2000 region proposal được đưa qua một model CNN (có thể pre-trained) để thực hiện chiết xuất vector đặc trưng (**feature vector**).
3. Vector đặc trưng sau đó qua **thuật toán SVM** để phân loại xem nó có nằm trong lớp background (lớp k+1) hay không, nếu có thì ta không cần phân loại.
4. Cuối cùng ta dùng vector đặc trưng làm dữ liệu train một mô hình hồi quy (**regression model)** làm cho bounding box chính xác hơn cho mỗi vật thể trong mỗi ảnh.



Hình 13: R-CNN

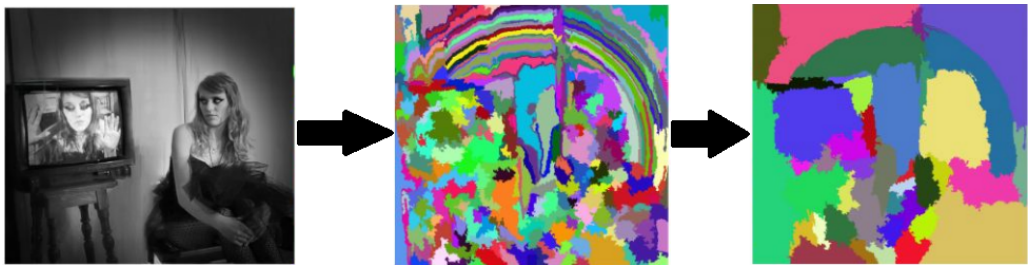
3.1.1 Selective Search Algorithm

Thuật toán thỏa một số tiêu chí sau được coi là hiệu quả:

- Bao quát được toàn bộ bức hình input, nắm bắt được gần như toàn bộ tỉ lệ trong bức hình dù vật thể có nhiều tỉ lệ khác nhau trong bức hình thậm chí là rất nhỏ.
- Tính toán nhanh chóng.

Ý tưởng thuật toán:

- Tạo các **segmentation** phụ ban đầu, chúng ta sẽ tạo ra nhiều region ứng cử viên.
- Sử dụng thuật toán tham lam (**greedy algorithm)** để kết hợp các vùng tương tự nhau thành vùng lớn hơn một cách đệ quy.
- Sử dụng các **region** vừa được tạo ra để đưa ra **region** cuối cùng.



Hình 14: Minh họa ý tưởng thuật toán SSA theo từng bước

3.1.2 CNN để rút trích vector đặc trưng

Qua thuật toán SSA, ta đã có được các region proposal. Bây giờ, ta phân loại các region proposal này thành các tập hợp đã được dán sẵn nhãn. Do thuật toán SSA cho ta tới 2000 region proposal nên có nhiều vùng không chứa vật gì, nên ta phải tạo thêm 1 lớp background. Ta cần huấn luyện một mạng CNN có nhiệm vụ phân loại hình ảnh. Trong đó: input là các region proposal; output là vector $k+1$ chiều chứa thông tin xác suất để vật chứa trong region proposal nằm trong $k+1$ tập hợp trên. Các region chọn ra được resize lại theo yêu cầu của CNN. Ta sẽ giữ lại phần ConvNet của CNN (bao gồm lớp convolutional và lớp pooling) và bỏ đi các lớp fully-connected. Sau đó output của ConvNet được đưa vào input của mô hình Logistic Regression, trong đó hàm kích hoạt ở output layer là hàm Softmax:

$$f(s)_i = \frac{e^{s_i}}{\sum_{i=1}^{k+1} e^{s_j}}$$

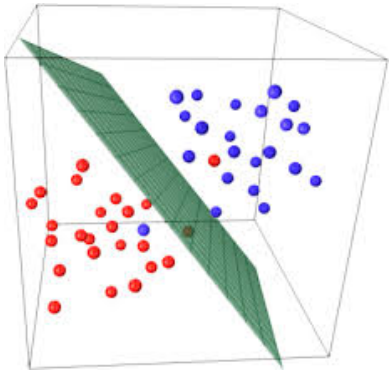
Ta sẽ train model này bằng tối ưu **hàm loss** của Softmax là một hàm được định nghĩa:

$$Cross - EntropyLoss = -\sum_{i=1}^{k+1} t_i log(f(s)_i)$$

Sau khi train thành công ta lấy ra các vector đặc trưng của input để sử dụng trong các bước tiếp theo.

3.1.3 Support Vector Machine

Support Vector Machine (SVM) giúp hỗ trợ giải quyết một trong những nhiệm vụ phổ biến nhất của ML:***Phân chia tập dữ liệu***. SVM giúp ta phân loại điểm dữ liệu mới vào một trong hai lớp dữ liệu được hình thành từ những điểm dữ liệu cũ. Giả sử những điểm dữ liệu cũ nằm trong không gian n chiều thì SVM sẽ giúp ta tìm ra siêu phẳng $n-1$ chiều để phân cách các điểm dữ liệu một cách tối ưu nhất. SVM sẽ giúp ta phân loại liệu region proposal có nằm trong lớp background hay không.



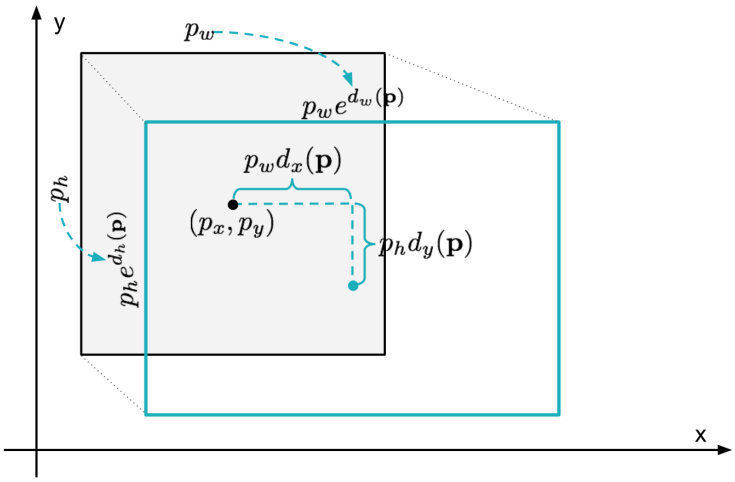
Hình 15: Minh họa cách phân chia các điểm dữ liệu nhờ SVM

3.1.4 Bounding Box Regression

Trong mục này, để tiện lợi ta sử dụng thuật ngữ bounding box thay cho region proposal, ý nghĩa của chúng là như nhau. Chúng ta sử dụng mô hình hồi quy cho các bounding box để làm chúng vừa vặn hơn với vật, tức là thêm một lớp hồi quy nữa. Đầu vào của tập huấn luyện là n cặp $(P^i, G^i)_{i=1,\ldots,n}$. Trong đó:

- $P^i = (P^i_x, P^i_y, P^i_w, P^i_h)$ là một vector 4 chiều với các thành phần tương ứng là tọa độ tâm, chiều rộng, chiều cao của bounding box được các bước trên đưa ra.
- $G^i = (G^i_x, G^i_y, G^i_w, G^i_h)$ là các tọa độ thực tế cần đạt được của bounding box.

Chúng ta cần biến đổi sao cho P càng gần với G càng tốt. Chúng ta tham số hóa phép biến



Hình 16: Minh họa quá trình biến đổi từ bounding box dự đoán thành bounding box thực tế

đổi 4 thành phần của vector P thành 4 hàm $d_x(P)$, $d_y(P)$, $d_w(P)$, $d_h(P)$, P sau khi qua phép biến đổi sẽ thành:

$$\hat{g}_x = P_w d_x(P) + P_x$$
$$\hat{g}_y = P_h d_y(P) + P_y$$
$$\hat{g}_w = P_w exp(d_w(P))$$
$$\hat{g}_h = P_h exp(d_h(P))$$

Mỗi $d_i(P)$ với i ∈ x,y,w,h, có thể nhận giá trị từ $[-\infty, \infty]$. Hàm muc tiêu ta cần tìm là:

$$t_x = \frac{g_x - p_x}{p_w}$$

$$t_y = \frac{g_y - p_y}{p_h}$$
$$t_w = \log(\frac{g_w}{p_w})$$
$$t_h = \log(\frac{g_h}{p_h})$$

Hàm mất mát của bài toán hồi quy mà ta cần tối ưu hóa:

$$L_{reg} = \sum_{i \in x,y,w,h} (t_i - d_i(P))^2 + \lambda ||w||^2$$

3.1.5 Vấn đề với R-CNN

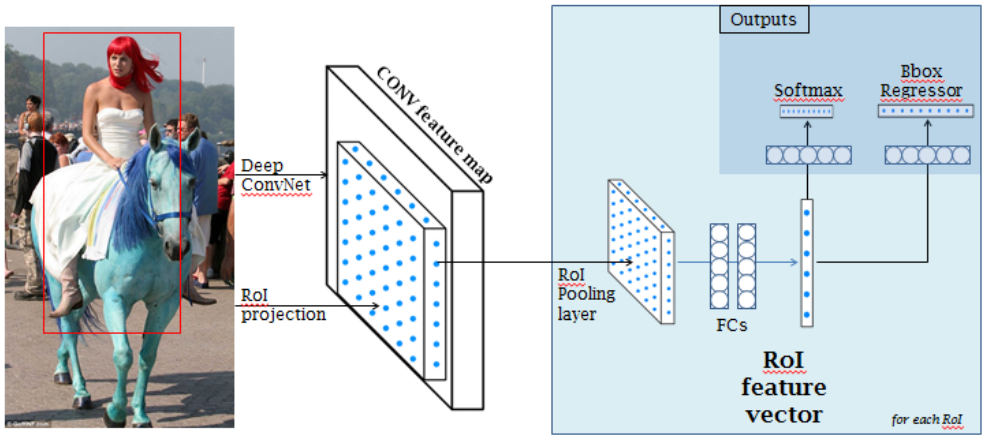
Ta biết rằng RCNN giúp ta giải quyết bài toán detection tuy nhiên việc sử dụng biến thể này còn nhiều hạn chế như:

- Với mỗi bức hình ta cần cho chạy CNN cho cả 2000 region. Tưởng tượng rằng với số lượng bức hình khổng lồ thì con số này sẽ tăng gấp nhiều lần, điều này dẫn tới tốn kém thời gian và tiền bạc.
- RCNN phải chạy tới 3 model: CNN để chiết suất các đặc trưng của regions, SVM để phân loại, BBR để làm cho bounding box được fix hơn.

Lúc này lại xuất hiện câu hỏi: Có thể không cần tách tới 2000 region proposal không? Có thể dùng ít hơn 3 model được không?

3.2 Fast R-CNN

Chưa thỏa mãn với tốc độ xử lý của R-CNN, nhà nghiên cứu Ross Girshick một lần nữa cải tiến mô hình này, gọi là **Fast R-CNN**. Cách vận hành của Fast R-CNN gần giống với R-CNN. Tuy nhiên, thay vì lọc ra khoảng 2000 region proposal và phân tích cả 2000 vùng đó trong ConvNet, Fast R-CNN đưa cả bức ảnh vào mạng CNN (bao gồm nhiều lớp convolutional và max-pooling) để tạo ra **convolutional feature map**. Tiếp sau đó, các region proposal tương ứng sẽ được chiết xuất ra từ feature map, làm phẳng hóa và đưa vào 2 lớp fully-connected để tìm ra lớp của vùng và giá trị offset value của bounding box. Lý do fast R-CNN xử lý nhanh



Hình 17: Quy trình hoạt động của Fast R-CNN

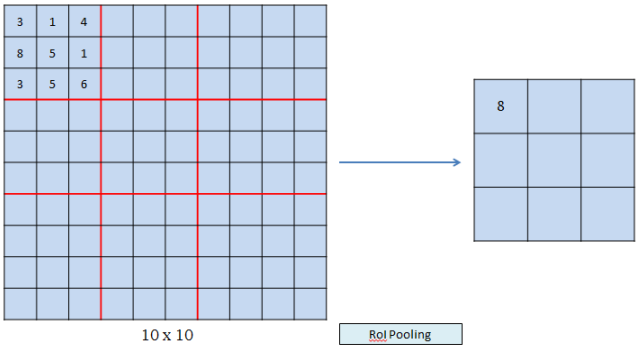
hơn R-CNN là bởi mạng CNN chỉ cần xử lý ảnh 1 lần để tạo ra convolutional feature map *thay vì xử lý cả 2000 region proposal.*

3.2.1 ROI pooling

Trong R-CNN, trước khi đưa vào mạng CNN hệ thống đã resize các region proposal nên quy trình tính toán mới thực hiện được. Trong fast R-CNN, do size của các region proposal trong feature map khác nhau, phẳng hóa chúng sẽ cho ra các output vector kích thước khác nhau. Đó là lý do ta cần thêm lớp **Region of Interest (RoI) pooling** với nhiệm vụ đưa các vùng về cùng kích thước.
RoI pooling bản chất giống như max pooling hay average pooling. Tuy nhiên, ROI pooling sẽ luôn cho ra một ma trận output có kích thước cố định. Gọi input của RoI pooling kích thước m*n và output có kích thước h*k (thông thường h, k nhỏ ví dụ 7*7).

- Ta chia chiều rộng thành h phần, (h-1) phần có kích thước m/h, phần cuối có kích thước m/h + m mod h.
- Tương tự ta chia chiều dài thành k phần, (k-1) phần có kích thước n/ k, phần cuối có kích thước n/k + n mod k.

Ví dụ $m = n = 10, h = k = 3, do\text{m/h} = 3$ và $mmodh = 1$, nên ta sẽ chia chiều rộng thành 3 phần, 2 phần có kích thước 3, và 1 phần có kích thước 4.



Trên mỗi vùng nhỏ tìm được, ta thực hiện max pooling. Kết quả tìm được là ma trận chiết xuất size cố định cần tìm.

3.2.2 Fine-tuning for detection

Điểm đặc biệt của Fast R-CNN là trong quá trình training, hệ thống đi qua một bước fine-tuning làm tối ưu hóa hàm phân loại softmax và bounding-box regressors cùng một lúc thay vì train hàm softmax, SVMs, và regressors trong ba quá trình khác nhau. Những thành phần của quá trình này bao gồm *hàm loss, mini-batch sampling strategy, back-propagation through RoI pooling layers, và SGD hyper-parameters*. Chúng ta sẽ xem xét **hàm Multi-task Loss** của Fast R-CNN. Các kí hiệu sử dụng:

- u: các lớp tập hợp được đánh số ($u \in [1, k] \cap \mathbb{N}$). Ta quy ước lớp background có $u = 0$.
- p: tập hợp phân phối xác suất rời rạc của k+1 lớp ($p = p_0, ..., p_k$), p được tính qua hàm softmax sau khi qua một lớp fully connected layer.
- v: các region (bounding box) đã được được biểu diễn dưới dạng vector 4 chiều $v = (v_x, v_y, v_w, v_h)$.
- t^k: bounding box sau khi được dự đoán k lần ($t^k = (t^k_x, t^k_y, t^k_w, t^k_h)$)

Do lớp cuối có 2 lớp: classification và bounding box nên hàm Loss của mô hình Fast R-CNN được gọi là Multi-task Loss:

$$L = L_{cls} + L_{box}$$

Các khung chỉ chứa background không được nhận diện nên $L_{box} = 0$ tại những chỗ đó. Ta cần 1 hàm bỏ qua background.

$$\chi[u \geqslant 1] = \begin{cases} 1 & \text{nếu } u \geqslant 1 \\ 0 & \text{trong trường hợp khác} \end{cases}$$

$$L(p, u, t^u, v) = L_{cls}(p, u) + \chi[u \geqslant 1] L_{box}(t^u, v)$$

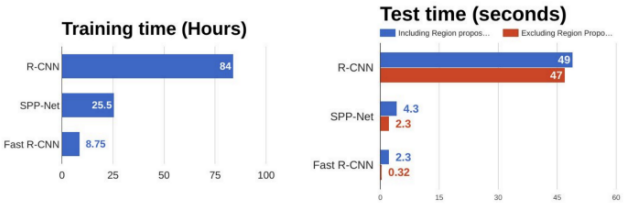
log loss cho class u:

$$L_{cls}(p, u) = \log(p_u)$$

$$L_{box}(t^u, v) = \sum_{i \in x,y,w,h} (t_i^w - v_i) L_1^{smooth}$$

3.3 Faster R-CNN

Khi kiểm tra tốc độ xử lý của R-CNN và Fast R-CNN, người ta phát hiện ra region proposal làm chậm tốc độ test của cả hai model dù Fast R-CNN xử lý nhanh hơn R-CNN gấp 9 lần (xem biểu đồ). Từ đây, các nhà nghiên cứu suy luận thuật toán SSA là tác nhân làm chậm model giải quyết bài toán Object Detection. Theo suy nghĩ đó, một phương pháp phân vùng mới là điều tất yếu cần được tạo ra.

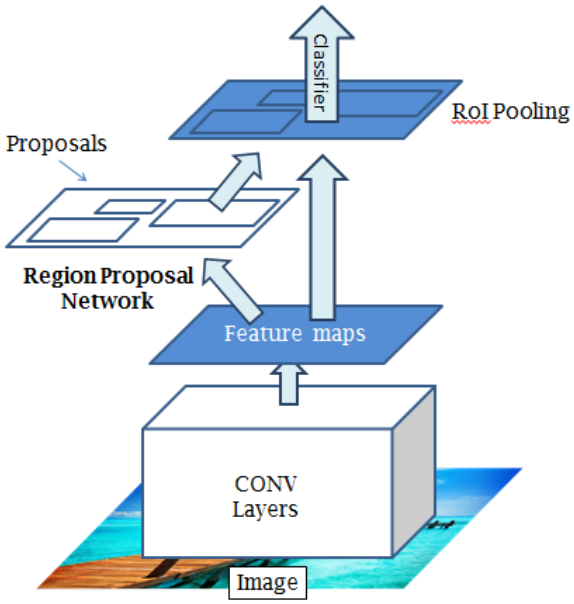


Phương pháp đó là **Region Proposal Network (RPN)** - một mạng CNN mới riêng biệt dùng để tìm region proposal. Với RPN, mô hình mới hoạt động gần giống như Fast R-CNN. Tuy nhiên, khi mạng CNN chiết xuất ra convolutional feature map sẽ đưa vào RPN để tìm region proposal song song với RoI pooling. Phần còn lại diễn ra tương tự như fast R-CNN. Ý tưởng làm faster R-CNN nhanh là mạng RPN chạy cùng lúc với bước RoI pooling. Quy trình các bước của faster R-CNN được biểu diễn ở hình bên dưới.

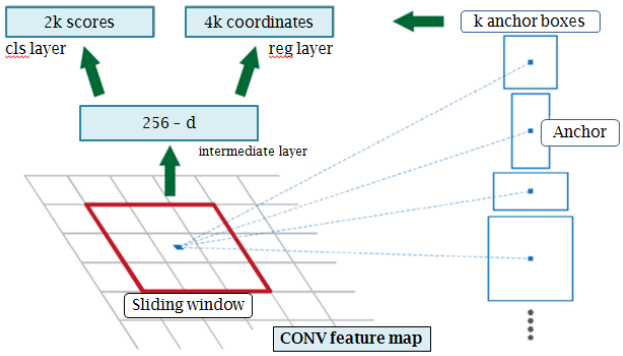
3.3.1 Region Proposal Network

Mạng RPN có input là ảnh feature map và output là những hình chữ nhật region proposal. Bởi vì chúng ta muốn rút ngắn tính toán bằng cách cho chạy mạng RPN và fast R-CNN cùng lúc trong model faster R-CNN, chúng ta cho rằng hai mạng có chung một số lớp convolutional. RPN hình thành bằng cách cho một cửa sổ (sliding window) n x n chiếu feature map vào một

lớp d trung gian có chiều đơn giản hơn. Sau đó, feature map sẽ được đưa vào hai lớp fully-connected họ hàng — lớp box-regression (reg) và lớp box-classification (cls). Chưa hết, tại mỗi vị trí trên sliding window máy tính cũng dự đoán tối đa k region proposal, vốn là những hình chữ nhật có kích thước dài rộng khác nhau. Reg layer do đó có output là 4k tọa độ của k hình chữ nhật proposal và cls layer có ouput là 2k điểm xác suất vật-hay-không-vật cho k hình trên.



Hình 18: Quy trình các bước của faster R-CNN



Hình 19: Nguyên lý vận hành của mạng RPN bằng hình ảnh

3.3.2 Anchor

Các ô chữ nhật region proposal mà mạng RPN tìm ra ở trên gọi là các Anchor. Tại mỗi vị trí trên sliding window, ta định nghĩa anchor có tâm tại sliding window đó kèm thêm kích thước hình chữ nhật bao quanh nó. Như vậy mỗi anchor có output là vector 4 tham số $(x_{center}, y_{center}, width, height)$. Ta mặc định 3 kích thước và 3 tỉ lệ bất kì để có k=9 anchor tại bất kì điểm nào trên sliding window. Qua RPN, chỉ những anchor chắc chắn chứa vật thể mới được giữ lại.

3.3.3 Loss Function của Faster R-CNN

Để train mạng RPN, chúng ta dán nhãn binary label cho mỗi anchor tìm được (vật hay không vật). Anchor dương gồm hai loại: (i) anchor có phần giao Intersection-over-Union (IoU) cao nhất với ground-truth box, hoặc (ii) anchor có độ trùng lặp IoU lớn hơn 0.7 với 5 ground-truth box bất kì. Để ý rằng một ground-truth box có thể assign nhiều anchor dương. Tiếp đó, ta chọn anchor âm là các anchor không dương mà có tỉ lệ trùng lặp IoU thấp hơn 0.3 trên tất cả ground-truth box. Những anchor không âm và không dương sẽ bị loại bỏ khỏi quy trình training. Với những định nghĩa trên, ta thu được hàm loss function cần tối ưu gần giống như hàm multi-task loss trong Fast R-CNN. Hàm loss này là:

$$L = L_{cls} + L_{reg}$$

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \frac{\lambda}{N_{reg}} \sum_i (p_i^*) L_1^{smooth}(t_i - t_i^*)$$

Trong đó:

- p_i: xác suất dự đoán của một anchor là vật thể
- p_i^*: nhãn binary ground-truth của một anchor là vật thể hay không
- t_i: 4 tham số dự đoán cho tọa độ của anchor
- t_i^*: tọa độ thực tế (ground-truth coordinates) của bounding boxes
- N_{cls}: hệ số chuẩn hóa cho độ lớn lớp classification
- N_{reg}: hệ số chuẩn hóa cho số lượng anchor trên 1 bức ảnh
- λ: hệ số cân bằng, ở đây cho là 10 để L_{cls} và L_{reg} chiếm tầm quan trọng như nhau trong công thức

Hàm classification loss L_{cls} là hàm log trên 2 giá trị (vật thể hoặc không vật thể). Hàm regression loss là hàm $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$ trong đó R là hàm mất mát Smooth L1 định nghĩa trong fast R-CNN.

4 Thực nghiệm

4.1 Nhận xét

4.1.1 Sự ảnh hưởng của Initialization

Như đã biết, phần lớn hàm loss của nhiều bài toán mà ta cần tối ưu hóa đều không phải là hàm tuyến tính, vậy nên nó sẽ có sự thay đổi đạo hàm, dẫn tới tồn tại những giá trị cực tiểu toàn phần trong một tập đóng. Với những hàm lồi, việc **initialization** gần như là không quan trọng, vì *khả năng "mắc kẹt"tại saddle point gần như là rất thấp*. Tất nhiên với những hàm loss function rắc rối hơn, việc chỉ tối ưu hóa bằng các phương pháp thông thường như sử dụng **SGD** là chưa đủ, vì cũng tồn tại khả năng để chúng ta "mắc kẹt"tại những cực tiểu địa phương. Và hàm loss function cần tối ưu của bài toán **Object Recognition** là một trong số đó, vậy nên ta cần quan tâm về **Object Recognition**. Sau đây, chúng mình sẽ giới thiệu về một phương

pháp **initialization** khá đơn giản nhưng cũng có hiệu quả cao có tên là **initialization**. Việc hiểu nó khá là đơn giản. Xét một phương trình tuyến tính đơn giản:

$$\mathbf{y} = \mathbf{W.x} + \mathbf{b}$$

Qua chứng minh[1], ta cần *phương sai* của $\mathbf{x}$ và y vẫn giữ nguyên khi đi qua hết các layer trong *Neural Network*, nghĩa là:

$$var(\mathbf{y}) = var(\mathbf{W.x}) = E(x_1)^2.var(W_1) + ... + E(x_n)^2.var(W_n) = var(W_1.x_1) + ... + var(W_n.x_n)$$

Ở đây, ta thấy rằng $\mathbf{b}$ coi như phương sai bằng 0. Ta lại thấy rằng từng đơn thức vế phải đều được phân phối như nhau, nên ta sẽ coi như là

$$\text{Var}(\mathbf{y}) = N * \text{Var}(W_i) * \text{Var}(x_i).$$

[1]http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

Và như đã nói, điều chúng ta cần là

$$\text{Var}(\mathbf{y}) = \text{Var}(\mathbf{x})$$

nên ta có thể kết luận:

$$\text{Var}(\mathbf{W}) = \frac{1}{N}$$

Từ đó, ta dễ dàng thấy cách initialize có khả năng cho ra hiệu quả tốt nhất là đặt các W_i ngẫu nhiên theo phân phối chuẩn $N(0, \frac{1}{N})$, với N là số lượng neuron đầu vào, tuy nhiên trong bài viết được dẫn nguồn ở trên thì tác giả cho N là trung bình cộng số neuron đầu vào và đầu ra, nhưng chúng mình sẽ vờ không thấy và cài đặt với cách đơn giản hơn.

4.1.2 Sự ảnh hưởng của số lớp và số lượng bias

Điều đầu tiên mà chúng ta thấy được là càng nhiều layer, độ *linh hoạt* của NN càng cao, nghĩa là độ chính xác cực đại mà nó đạt tới sẽ càng tiệm cận về 1. Tuy nhiên, có một vấn đề, **overfitting**. Như bài toán về *Linear Regression*, ta dễ dàng giải nó bằng phương pháp nhân tử Lagrange nhưng đồng nghĩa là model sẽ càng phức tạp và nó fit quá mức khiến nó không nhận ra độ lỗi trong bộ test. Đây cũng vậy, NN càng nhiều layer sẽ khiến nó chỉ làm được việc trong tập training, khi đem ra ngoài nó sẽ vô dụng. Và việc có quá nhiều layer cũng làm mọi thứ trở nên rắc rối hơn khi số lượng trọng số cũng tăng lên theo, nghĩa là việc tính toán sẽ chậm hơn rất nhiều. Vậy nên, khi muốn tìm ra một NN tốt nhất cho một bài toán thì người ta thường sẽ quan tâm đến việc bố trí cấu trúc của các layer thay vì cố tăng số layer và nhìn CPU 100% trong vô vọng.

Về **bias** thì mọi thứ có vẻ khác biệt một chút. **Bias** giúp cho NN trở nên *linh hoạt* hơn về mặt tịnh tiến. Nghĩa là càng có thêm **bias** thì ta dễ dàng dịch chuyển tập dự đoán gần hơn với tập train mà không có sự thay đổi về phương sai. Và đó là lý do mà thông thường người ta vẫn luôn đặt số lượng **bias** bằng với số **weight** trong NN, vì việc ta có tăng thêm hơn con số đó thì nó vẫn chỉ có tác dụng tương tự nhưng lại mất thêm dữ liệu để lưu thêm trọng số.

4.1.3 Tối ưu hóa sử dụng SGD

Như đã đề cập, điều ta cần làm cho bài toán là tối ưu hóa hàm loss. Và không may thay, làm loss trong bài toán **Object Recognition** chẳng phải hàm tuyến tính. Vậy nên cách tốt nhất mà mọi người vẫn thường làm đó là tối ưu bằng **SGD**, hay còn gọi là **Stochastic Gradient Descend**.

Nôm na về **GD** thì nó là việc chúng ta đi ngược lại với hướng của *gradient* của hảm loss để có thể đi về nơi có giá trị nhỏ hơn, nghĩa là

$$\mathbf{X'} = \mathbf{X} - \eta * \nabla_X f(\mathbf{X}).$$

Trong đó, ta coi giá trị η là một số thực gọi là *learning rate*. Vì đôi khi giá trị của Gradient tại $\mathbf{X}$ có thể sẽ khiến bộ tham số $\mathbf{X}$ sẽ chỉ liên tục đi xung quanh giá trị cực tiểu và tốn nhiều thời gian để từ từ đạt tới nó.

SGD là một phương pháp tối ưu về mặt thời gian cho việc tối ưu nhưng không gây ảnh hướng đáng kể cho độ chính xác. Nguyên tắc là thay vì tính hàm loss dựa trên toàn tập train, chúng ta chỉ tính một phần được chọn ngẫu nhiên trong tập, thường thì con số này khá nhỏ so với tập train. Qua thực nghiệm cho thấy rằng việc này tuy tăng số lần *epoch* nhưng thời gian lại được giảm đi đáng kể và hội tụ nhanh về điểm cực tiểu.

4.2 Thực hiện

Chúng mình thực hiện dựa trên github repository [2] và có sửa đổi để phù hợp với dataset [3] được yêu cầu. Và vì như đã đề cập, **FRCNN** đạt hiệu suất tốt hơn hẳn **RCNN, Fast RCNN** nên chúng mình sẽ sử dụng model này, ngoài ra các **NN** đều sẽ dùng model của *resnet*.

Vì bộ dữ liệu gồm 2 folder *Annotations* chứa các file .xml gồm thông tin bounding boxes, classes và *Images* chứa các ảnh, và định dạng mà code trên repo này yêu cầu phải có một file .txt chứa đường dẫn tới các ảnh và cả bounding boxes kèm theo, vậy nên cần phải tinh chỉnh dữ liệu đôi chút.

```
# Tiền xử lý dữ liệu
from glob import glob
import xml.etree.ElementTree as ET
import pandas as pd

num = int(6) #Giảm số class

data = []
cnt = int(0)

classes = glob('datasets/Annotation/*')
for cls in classes:
  print(cls, cnt)
  if(cnt > num): break
  anno = glob(cls + '/*')
  for file in anno:
    row = []
    xml = ET.parse(file)
    for e in xml.getroot().iter('object'):
      dog_breed = e.find('name').text
      xmin = int(e.find('bndbox/xmin').text)
      ymin = int(e.find('bndbox/ymin').text)
      xmax = int(e.find('bndbox/xmax').text)
      ymax = int(e.find('bndbox/ymax').text)
    row = [file, dog_breed, xmin, xmax, ymin, ymax]
    print(row)
    data.append(row)
  cnt += 1

data = pd.DataFrame(data, columns=['filename', 'dog_breed', 'xmin', 'xmax',
 ↪  'ymin', 'ymax'])
data[['filename', 'dog_breed', 'xmin', 'xmax', 'ymin',
 ↪  'ymax']].to_csv('dog_breed.csv', index=False)

train = pd.read_csv('dog_breed.csv')
train.head()

data = pd.DataFrame()
```

[2] Github
[3] Kaggle

```
data['format'] = train['filename']

for i in range(data.shape[0]):
  data['format'][i] = data['format'][i].replace('Annotation', 'Images') +
  ↪ '.jpg,' + str(train['xmin'][i]) + ',' + str(train['ymin'][i]) + ',' +
  ↪ str(train['xmax'][i]) + ',' + str(train['ymax'][i]) + ',' +
  ↪ train['dog_breed'][i]

data.to_csv('annotate.txt', header=None, index=None, sep=' ')

print('ok')
```

Chúng mình khởi tạo các layer chính trong **FRCNN** như sau:

```
#Ví dụ về 1 Convolution block

x = Convolution2D(nb_filter1, (1, 1), strides=strides, name=conv_name_base +
↪ '2a', trainable=trainable)(input_tensor)
x = FixedBatchNormalization(axis=bn_axis, name=bn_name_base + '2a')(x)
x = Activation('relu')(x)

x = Convolution2D(nb_filter2, (kernel_size, kernel_size), padding='same',
↪ name=conv_name_base + '2b', trainable=trainable)(x)
x = FixedBatchNormalization(axis=bn_axis, name=bn_name_base + '2b')(x)
x = Activation('relu')(x)

x = Convolution2D(nb_filter3, (1, 1), name=conv_name_base + '2c',
↪ trainable=trainable)(x)
x = FixedBatchNormalization(axis=bn_axis, name=bn_name_base + '2c')(x)

shortcut = Convolution2D(nb_filter3, (1, 1), strides=strides,
↪ name=conv_name_base + '1', trainable=trainable)(input_tensor)
shortcut = FixedBatchNormalization(axis=bn_axis, name=bn_name_base +
↪ '1')(shortcut)

x = Add()([x, shortcut])
x = Activation('relu')(x)

#Ví dụ về 1 layer neural network dùng cho RPN và Classification NN

x = ZeroPadding2D((3, 3))(img_input)

x = Convolution2D(64, (7, 7), strides=(2, 2), name='conv1', trainable =
↪ trainable)(x)
x = FixedBatchNormalization(axis=bn_axis, name='bn_conv1')(x)
x = Activation('relu')(x)
x = MaxPooling2D((3, 3), strides=(2, 2))(x)

x = conv_block(x, 3, [64, 64, 256], stage=2, block='a', strides=(1, 1),
↪ trainable = trainable)
x = identity_block(x, 3, [64, 64, 256], stage=2, block='b', trainable =
↪ trainable)
```

```
x = identity_block(x, 3, [64, 64, 256], stage=2, block='c', trainable =
 ↪  trainable)

x = conv_block(x, 3, [128, 128, 512], stage=3, block='a', trainable =
 ↪  trainable)
x = identity_block(x, 3, [128, 128, 512], stage=3, block='b', trainable =
 ↪  trainable)
x = identity_block(x, 3, [128, 128, 512], stage=3, block='c', trainable =
 ↪  trainable)
x = identity_block(x, 3, [128, 128, 512], stage=3, block='d', trainable =
 ↪  trainable)

x = conv_block(x, 3, [256, 256, 1024], stage=4, block='a', trainable =
 ↪  trainable)
x = identity_block(x, 3, [256, 256, 1024], stage=4, block='b', trainable =
 ↪  trainable)
x = identity_block(x, 3, [256, 256, 1024], stage=4, block='c', trainable =
 ↪  trainable)
x = identity_block(x, 3, [256, 256, 1024], stage=4, block='d', trainable =
 ↪  trainable)
x = identity_block(x, 3, [256, 256, 1024], stage=4, block='e', trainable =
 ↪  trainable)
x = identity_block(x, 3, [256, 256, 1024], stage=4, block='f', trainable =
 ↪  trainable)

#Tuy nhiên, với từng loại RPN và Classification NN thì sẽ được modify đôi
 ↪  chút để phù hợp với công việc của chúng

#Sau đó, mình sẽ xây dựng model chính

optimizer = Adam(lr=1e-5)
optimizer_classifier = Adam(lr=1e-5)
model_rpn.compile(optimizer=optimizer, loss=[losses.rpn_loss_cls(num_anchors),
 ↪  losses.rpn_loss_regr(num_anchors)])
model_classifier.compile(optimizer=optimizer_classifier,
 ↪  loss=[losses.class_loss_cls,
 ↪  losses.class_loss_regr(len(classes_count)-1)],
 ↪  metrics={'dense_class_{}'.format(len(classes_count)): 'accuracy'})
model_all.compile(optimizer='sgd', loss='mae')
```

Sau đó, mọi thứ còn lại sẽ phụ thuộc vào **Google Colab**. Nhưng ngặt là *runtime* sẽ tự động reset sau 12 giờ, vậy nên mình cần phải lưu lại model của những lần giảm giá trị của *loss*, và sau mỗi lần reset thì *weight* của model vẫn còn trong Drive(nếu chúng ta mount) thì dễ dàng tiếp tục huấn luyện model.

```
if curr_loss < best_loss:
        best_loss = curr_loss
        model_all.save_weights(C.model_path)
```

Trong quá trình train, có thể thấy rằng việc xảy ra *overfit* là có khả năng, vì chưa hiểu rõ cách hoạt động của việc train trong code này nên chúng mình vẫn chưa thể thêm *validation*

để tránh tình trạng đó. Tuy nhiên, theo đánh giá thì bộ dữ liệu khá tốt vì có rất ít ảnh trông tương tự nhau (đến cả mình còn không biết phân biệt mấy loài chó này như thế nào). Nhưng để đảm bảo, có một cách khác để tăng khả năng tránh *overfit* mà thiếu đi việc *validate*, đó là *data augmentation*.

```
#Quay theo trục dọc

img = cv2.flip(img, 1)
for bbox in img_data_aug['bboxes']:
        x1 = bbox['x1']
        x2 = bbox['x2']
        bbox['x2'] = cols - x1
        bbox['x1'] = cols - x2

#Quay theo trục ngang

img = cv2.flip(img, 0)
for bbox in img_data_aug['bboxes']:
        y1 = bbox['y1']
        y2 = bbox['y2']
        bbox['y2'] = rows - y1
        bbox['y1'] = rows - y2
```

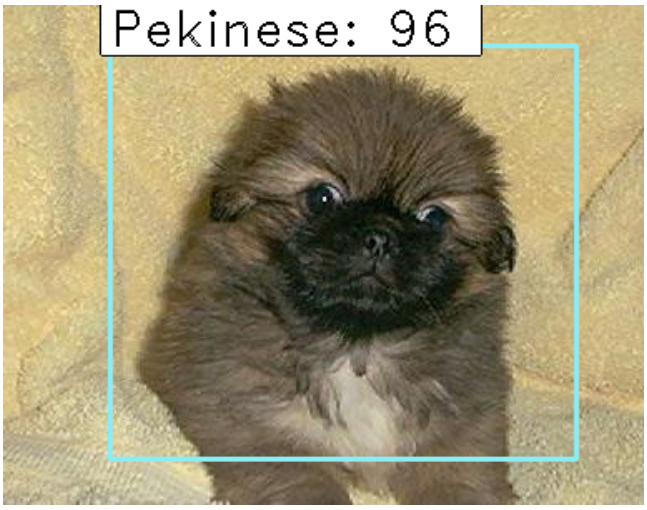
Chúng mình có một nhận xét không hợp lý nhưng lại rất thuyết phục, đó là trong mạng lưới, chỉ có lớp *NN* để classify các loài chó là phụ thuộc vào số lượng class, vì các khối *NN* còn lại thì đảm nhiệm những vai trò khác: **Convolutional NN** để tách các *feature*, **RPN** để chọn ra các *anchor box*. Vậy nên nếu chúng ta áp dụng *transfer learning* thì sao?

Câu trả lời là có thể. Vậy nên ban đầu chúng mình chỉ chọn ra 7 class để huấn luyện tập model ban đầu, tới khi độ chính xác của model đạt ở một ngưỡng cho phép (như là 85% chẳng hạn) thì sau đó chúng mình tăng số lượng class lên, nhập thêm dữ liệu vào file *annotation.txt* và từ đó làm việc được với nhiều giống chó hơn.

4.3 Kết quả

Sau khi train, chúng mình tìm được một số output trông khá là mãn nguyện

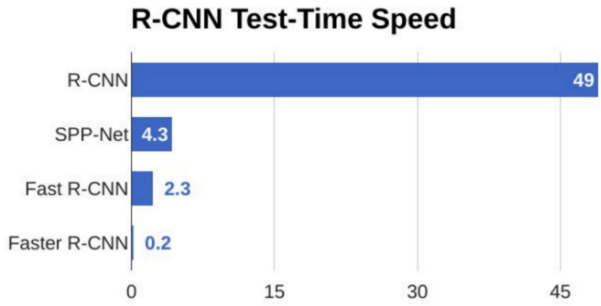
Hình 20: Chó Chihuahua



Hình 21: Chó Pekinese

5 Kết luận đánh giá

RCNN hay cụ thể hơn là model **FRCNN** là một trong những phương pháp dễ dàng để làm việc với bài toán **Object Recognition**, nó thừa hưởng khả năng tìm ra *bounding box* của vật và cải tiến tốc độ train cũng như là test hơn hẳn với model **RCNN** cơ bản.



Hình 22: So sánh hiệu năng của các model

Ngoài **FRCNN**, có những model khác hay được sử dụng cho bài toán này như là **YOLOv3** hoặc là **SSD**(Single Shot Detector), và với tùy bài toán, tùy bộ dữ liệu và yêu cầu mà chúng ta có thể chọn lựa phương pháp phù hợp.

Hình 23: So sánh hiệu năng của các model

Trong hiện tại, chúng mình vẫn còn thiếu *validation* trong model, vậy nên việc thêm nó vào là điều đầu tiên cần làm. Ngoài ra, thử nghiệm và tìm ra *activation function, loss function* phù hợp cũng giúp mô hình sau khi huấn luyện chính xác hơn và chạy nhanh hơn trong thực tế.

Sau khi thành thục, chúng mình sẽ tự viết một model **FRCNN** riêng để dễ dàng chỉnh sửa đúng với ý mình hơn, vì hiện tại việc không config một số tham số là lý do mà mô hình tuy đã được huấn luyện nhưng vẫn còn nhiều hạn chế.

6 Hướng phát triển trong tương lai

6.1 Optical Character Recognition

Đây là phương thức chuyển đổi từ những ảnh chứa dữ liệu viết tay, in,... trở thành dạng dữ liệu số. Về cơ bản nó chính là bài toán **Object Recognition** để tìm các *bounding box* chứa dữ liệu ảnh mà chúng tav huấn luyện trước, và *classify* chúng, hay còn gọi là **"đọc"**. Điều này có thể ứng dụng cho việc số hóa các văn bản vật lý và ứng dụng cho vô số công việc khác nhau mà những người sao kê tốn rất nhiều thời gian để làm.

6.2 Tracking Object

Khi sử dụng các model có hiệu năng cao và cho chúng chạy real-time, nó sẽ có thể liên tục theo dõi đối tượng mà nó tìm ra, nhờ vậy mà có thể ứng dụng trong an ninh, robotics,...

6.3 Activity Recognition

Một bước tiếp theo của **Activity Recognition**, đó là chúng ta sẽ không chỉ nhận diện hoạt động của một người mà còn tìm ra các đặc điểm về hoạt động của họ. Nếu ứng dụng cho các bài toán về tương tác giữa người và máy thì nó sẽ là công cụ đắc lực và giúp việc giao tiếp trở nên thuận lợi hơn.

6.4 Iris recognition

Mống mắt có sự độc nhất với mỗi người cũng như dấu vân tay, vậy nên việc áp dụng bài toán vào việc nhận diện mống mắt cũng là một ứng dụng khá tốt trong việc bảo mật bằng sinh trắc học. Trong thực tế, người ta cho rằng việc sinh trắc bằng mống mắt tới giờ chính là cách chính xác nhất.

6.5 Digital Watermarking

Watermark là một phương thức để đánh dấu bản quyền của một sản phẩm về mặt trí tuệ dưới dạng số. Tuy nhiên, đôi khi việc *watermark* tùy tiện có thể gây khó dễ cho người dùng, và đôi khi gây hại đến phần dữ liệu quan trọng. Vậy nên một lần nữa **Object Recognition** sẽ có vai trò để nhận diện đâu là nơi tốt nhất đặt *watermark* để tránh rủi ro trên.

Tài liệu

[1] Ross Girshick. Fast r-cnn.

[2] Trevor Darrell Jitendra Malik Ross Girshick, Jeff Donahue. Rich feature hierarchies for accurate object detection and semantic segmentation.

[3] Ross Girshick Jian Sun Shaoqing Ren, Kaiming He. Faster r-cnn: Towards real-time object detection with region proposal networks.

[4] Nguyễn Thanh Tuấn. Deep learning cơ bản.