

NHẬN DIỆN DẤU HIỆU NHỊP TIM BẤT THƯỜNG DỰA TRÊN VIDEO MRI SỬ DỤNG PHƯƠNG PHÁP 3D CNN VÀ LSTM

ĐỀ CƯƠNG LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC

Đặng Lâm Tùng – 1713856
Giảng viên hướng dẫn
TS. Phạm Quang Thái



ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ, BỘ MÔN VIỄN THÔNG

Số: _____/BKĐT

Khoa: Điện – Điện tử

Bộ môn: Viễn thông

NHIỆM VỤ LUẬN VĂN TỐT NGHIỆP

1. Họ và tên: Đặng Lâm Tùng, MSSV: 1713856
2. Ngành: Điện – Điện tử, Chuyên ngành: Kỹ thuật Điện tử – Truyền thông
3. Đề tài: Nhận diện dấu hiệu nhịp tim bất thường dựa trên video MRI sử dụng phương pháp 3D CNN và LSTM
4. Nhiệm vụ:
 - Nhiệm vụ 1
 - Nhiệm vụ 2
 - Nhiệm vụ 3
5. Ngày giao nhiệm vụ: 2-10-2019
6. Ngày hoàn thành nhiệm vụ: 2-10-2020
7. Họ và tên người hướng dẫn:
TS. Phạm Quang Thái
BM Viễn Thông, Khoa Điện – Điện Tử

Phần hướng dẫn:

100%

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

Tp. HCM, Ngày ___ tháng ___ năm 20___
CHỦ NHIỆM BỘ MÔN

Tp. HCM, Ngày ___ tháng ___ năm 20___
NGƯỜI HƯỚNG DẪN CHÍNH

PGS. TS. Hà Hoàng Kha

TS. Phạm Quang Thái

PHẦN DÀNH CHO KHOA, BỘ MÔN:

Người duyệt (chấm sơ bộ): _____

Đơn vị: _____

Ngày bảo vệ: _____

Điểm tổng kết: _____

Nơi lưu trữ luận văn: _____

LỜI CÁM ƠN

Sinh viên viết phần lời cảm ơn vào đây. Lưu ý giới hạn không quá 1 trang.

Tp. HCM, Ngày 15 tháng 1 năm 2021 Dặng

Lâm Tùng

LỜI CAM ĐOAN

Tôi tên: Đặng Lâm Tùng (MSSV: 1713856), là sinh viên chuyên ngành Kỹ thuật Điện tử - Truyền thông, tại Trường Đại học Bách Khoa, Đại học Quốc gia thành phố Hồ Chí Minh. Tôi xin cam đoan những nội dung sau đều là sự thật:

- Công trình nghiên cứu này hoàn toàn do chính tôi thực hiện;
- Các tài liệu và trích dẫn trong luận văn này được tham khảo từ các nguồn thực tế, có uy tín và độ chính xác cao;
- Các số liệu và kết quả của công trình này được tôi tự thực hiện một cách độc lập và trung thực.

Tp. HCM, Ngày 15 tháng 1 năm 2021 Đặng

Lâm Tùng

TÓM TẮT NỘI DUNG

Trong cuộc sống hiện đại ngày nay, chất lượng cuộc sống ngày càng được nâng cao, thì vấn đề sức khỏe, đặc biệt là về các bệnh tim mạch ngày càng được quan tâm, những phương pháp mới tiến bộ hơn được sử dụng vào trong chẩn đoán và điều trị. Một phương pháp chẩn đoán hiện nay được quan tâm nghiên cứu rất nhiều đó là phương pháp chụp cắt lớp MRI. Khi chụp MRI khu vực tim sẽ được chụp nhiều lát cắt (slide), số slide này có số lượng tầm khoảng 8 - 16 nhát cắt, và để chẩn đoán thì cần phải chụp nhiều chuỗi ảnh như vậy để có thể quan sát quy trình tim đập. Để xử lý loại dữ liệu này, em sử dụng phương pháp 3D CNN kết hợp với LSTM để phân loại các chuỗi ảnh này. Đề cương này em đã cài đặt một thuật toán và train nó trên một tập dataset chuỗi ảnh MRI tim nhỏ. Thuật toán này có thể phân loại chuỗi ảnh tim là bình thường hay không với độ chính xác 98.4 % sau 10 epochs

ABSTRACT

Tóm tắt luận văn bằng tiếng Anh. Giới hạn trong 1 trang. Nội dung tóm tắt tham khảo phần hướng dẫn và ví dụ của tóm tắt tiếng Việt.

MỤC LỤC

1 Giới thiệu	1
1.1 Đặt vấn đề	1
1.2 Phạm vi và phương pháp nghiên cứu	1
2 Cơ sở lý thuyết	3
2.1 Lý thuyết về ảnh MRI và ảnh MRI chụp tim:	3
2.1.1 Giới thiệu MRI	3
2.1.2 Cơ sở vật lý	4
2.1.3 Giải phẫu tim	5
2.2 Lý thuyết về xử lý ảnh:	6
2.2.1 Mạng Neuron tích chập (CNN):	6
2.3 Long sort term memory	12
2.3.1 Recurrent Neural Network	12
2.3.2 Long Short Term Memory	14
3 Kết quả nghiên cứu	17
3.1 Phương pháp tiếp cận	17
3.1.1 Heart Detection	17
3.1.2 Heart Classification	22
3.2 Kết quả và phân tích	25
3.3 Kết luận chương	25
4 Kết luận	27
4.1 Tóm tắt và kết luận chung	27
4.2 Mục tiêu và tiến độ thực hiện	27

DANH SÁCH HÌNH VẼ

Hình 2.1	Máy chụp MRI	3
Hình 2.2	Giải phẫu tim: 1. Tâm nhĩ phải; 2. Tâm nhĩ trái; 3. Tĩnh mạch chủ trên; 4. Động mạch chủ; 5. Động mạch phổi; 6. Tĩnh mạch phổi; 7. Van hai lá; 8. Van động mạch chủ; 9. Tâm thất trái; 10. Tâm thất phải; 11. Tĩnh mạch chủ dưới; 12. Van ba lá; 13. Van động mạch phổi	5
Hình 2.3	Công thức tích chập	6
Hình 2.4	Biểu diễn tích chập	7
Hình 2.5	Biểu diễn tích chập	8
Hình 2.6	Biểu diễn tích chập	8
Hình 2.7	Biểu diễn tích chập	9
Hình 2.8	Lớp RPN của Faster RCNN	10
Hình 2.9	Lớp RPN của Faster RCNN	11
Hình 2.10	RNN unrolled	12
Hình 2.11	RNN unrolled	13
Hình 2.12	Biểu diễn RNN theo thời gian (thứ tự trong sequence)	13
Hình 2.13	Long term memory ảnh hưởng đến RNN	13
Hình 2.14	Biểu diễn RNN theo thời gian (thứ tự trong sequence)	14
Hình 2.15	Ví dụ 1 cell của LSTM	14
Hình 2.16	Ví dụ 1 cell của LSTM	14
Hình 2.17	Ví dụ 1 cell của LSTM	15
Hình 2.18	Ví dụ 1 cell của LSTM	15
Hình 2.19	Ví dụ 1 cell của LSTM	15
Hình 2.20	Ví dụ 1 cell của LSTM	16
Hình 3.1	Video MRI tim bình thường	17
Hình 3.2	Video MRI tim bình thường	17
Hình 3.3	Video MRI tim bình thường và bất bình thường	18
Hình 3.4	Data ban đầu	18
Hình 3.5	Data sau khi được đánh dấu	18
Hình 3.6	Tool labelimg để tạo dataset	19
Hình 3.7	Tool labelimg để tạo dataset	19
Hình 3.8	File xml mà labelimg sinh ra	20
Hình 3.9	File config của Faster R-CNN	20
Hình 3.10	Code phần test model	21
Hình 3.11	Kết quả của việc train model detect tim	21
Hình 3.12	Hình tim sau khi đã preprocess	22
Hình 3.13	Hình tim ngay sau khi detect được crop ra	22
Hình 3.14	Code của mô hình	23
Hình 3.15	Code model classifications	23

Hình 3.16 Model classifier	24
Hình 3.17 Code training mô hình	25
Hình 3.18 Kết quả training sau khoảng 10 epoch	25

DANH SÁCH BẢNG

Bảng 4.1 Tiết độ thực hiện luận văn	27
---	----

DANH SÁCH TỪ VIẾT TẮT

Chương 1. GIỚI THIỆU

1.1 Đặt vấn đề

Trong cuộc sống hiện đại ngày nay, chất lượng cuộc sống ngày càng được nâng cao, thì vấn đề sức khỏe, đặc biệt là về các bệnh tim mạch ngày càng được quan tâm, những phương pháp mới tiến bộ hơn được sử dụng vào trong chẩn đoán và điều trị. Một phương pháp chẩn đoán hiện nay được quan tâm nghiên cứu rất nhiều đó là phương pháp chụp cắt lớp MRI.

Thông thường, khi chụp MRI khu vực tim sẽ được chụp nhiều lát cắt (slide), số slide này có số lượng tầm khoảng 8 - 16 nhát cắt, và để chẩn đoán thì cần phải chụp nhiều chuỗi ảnh như vậy để có thể quan sát quy trình tim đập. Data này gồm một số video là video chụp MRI tim trong thời gian 1s - 30 khung hình, gồm 2 class là normal và abnormal, chất lượng video 4K, dung lượng > 700 MB/file, được cung cấp bởi CLB AI đại học Bách Khoa, do thầy Quản Thành Thơ, khoa khoa học máy tính quản lý.

Em hiểu rõ bản thân không có nhiều kiến thức về lĩnh vực này, nên em chỉ coi bài toán này như một bài toán về deep learning, hoàn toàn chưa có đủ giá trị thực hiện như một mô hình chẩn đoán hoàn chỉnh có thể áp dụng được trong thực tế.

Nhiệm vụ của đề cương này là:

- + Tìm hiểu phương pháp xử lý hiệu quả với bài toán, bài toán này là bài toán phân loại (classification) một video MRI.
- + Cài đặt thuật toán xử lý bài toán, thực hiện kiểm chứng hiệu quả của phương pháp.

1.2 Phạm vi và phương pháp nghiên cứu

Trong mục này, sinh viên liệt kê các phương pháp đã dùng để kiểm nghiệm giả thiết trong bài toán nghiên cứu (chẳng hạn mô phỏng hay thực nghiệm). Đồng thời, sinh viên liệt kê các giới hạn trong quá trình thực hiện các phương pháp (chẳng hạn mô phỏng trong điều kiện gì, đo đạc trong điều kiện gì).

Phạm vi và phương pháp nghiên cứu trong đề cương luận văn như sau:

- Tìm hiểu dữ liệu bài toán,
- Xây dựng mô hình nhận diện (detection) tim trong ảnh chụp MRI sử dụng Tensorflow Object Detection.
- Xây dựng mô hình phân loại (classification) chuỗi ảnh tim (video) sử dụng 3D convolution và LSTM,
- Thủ nghiệm mô hình.

Chương 2. CƠ SỞ LÝ THUYẾT

2.1 Lý thuyết về ảnh MRI và ảnh MRI chụp tim:

2.1.1 Giới thiệu MRI

Chụp cộng hưởng từ (Magnetic resonance imaging) là một phương pháp thu hình ảnh của các cơ quan trong cơ thể sống và quan sát lượng nước bên trong các cấu trúc của các cơ quan. Ảnh cộng hưởng từ hạt nhân dựa trên một hiện tượng vật lý là hiện tượng cộng hưởng từ hạt nhân.

Chụp cộng hưởng từ gọi đây là "chụp cộng hưởng từ hạt nhân" bắt đầu được dùng để chẩn đoán bệnh từ năm 1982. Hiện tượng cộng hưởng từ hạt nhân bắt đầu được 2 tác giả Bloch và Purcell phát hiện năm 1952. Sự khác nhau cơ bản giữa chụp cộng hưởng từ và chụp X quang là năng lượng dùng trong chụp X quang là năng lượng phóng xạ tia X còn trong chụp cộng hưởng từ là năng lượng vô tuyến điện. [1]



Hình 2.1: Máy chụp MRI

2.1.2 Cơ sở vật lý

[1] Gồm 4 giai đoạn:

Giai đoạn 1: Sắp hàng hạt nhân

Mỗi hạt nhân trong môi trường vật chất đều có một mômen từ tạo ra bởi spin (sự xoay) nội tại của nó.

Các hạt nhân đều sắp xếp một cách ngẫu nhiên và từ trường của chúng triệt tiêu lẫn nhau do đó không có từ trường dư ra để ghi nhận được.

Khi có một từ trường mạnh tác động từ bên ngoài các mômen từ của hạt nhân sẽ sắp hàng song song cùng hướng hoặc ngược hướng của từ trường, ngoài ra chúng còn chuyển động dần chung quanh hướng của từ trường bên ngoài nó.

Các vectơ từ hạt nhân sắp hàng song song cùng chiều với hướng từ trường bên ngoài có số lượng lớn hơn các vectơ từ hạt nhân sắp hàng ngược chiều và chúng không thể triệt tiêu cho nhau hết, do đó có mạng lưới từ hoá theo hướng của từ trường bên ngoài.

Các vectơ tạo ra hiện tượng từ hoá chủ yếu theo hướng của từ trường bên ngoài; đó là trạng thái cân bằng.

Trong trạng thái cân bằng không có một tín hiệu nào có thể được ghi nhận. Khi trạng thái cân bằng đó bị xáo trộn sẽ có tín hiệu được hình thành.

Giai đoạn 2: Kích thích hạt nhân

Hiện tượng sắp hàng hạt nhân kết thúc thì các hạt nhân hydrogen tức proton sẽ phóng thích năng lượng dùng để sắp hàng chúng để trở về vị trí ban đầu. Tốc độ phóng thích các photon này dựa vào năng lượng được phóng thích. Thời gian cần thiết cho 63% vectơ khôi phục theo chiều dọc gọi là T1. Thời gian cần thiết để cho 63% vectơ khôi phục theo chiều ngang gọi là T2.

Giai đoạn 3: Ghi nhận tín hiệu

Khi các proton trở lại sắp hàng như cũ do ảnh hưởng từ trường bên ngoài chúng phóng thích năng lượng dưới dạng tín hiệu tần số vô tuyến. Cường độ phát ra từ một đơn vị khối lượng mô được thể hiện trên một thang màu từ trắng đến đen, trên đó màu trắng là cường độ tín hiệu cao, màu đen là không có tín hiệu. Cường độ tín hiệu của một loại mô phụ thuộc vào thời gian khôi phục lại từ tính T1 và T2, mật độ photon của nó.

Giai đoạn 4: Tạo hình ảnh

T1 tạo ra tín hiệu MRI mạnh và cho thấy hình ảnh các cấu trúc giải phẫu với T1 dịch não tuỷ, lớp vỏ xương, không khí và máu lưu thông với tốc độ cao tạo ra những tín hiệu không đáng kể và thể hiện màu sẫm. Chất trắng và chất xám biểu hiện bằng màu xám khác nhau và chất xám đậm hơn. Với T1 thì mô mỡ có màu sáng đó là lợi thế lớn nhất để ghi hình mô mỡ trong hốc mắt, ngoài màng cứng tuỷ xương và cột sống. Máu tụ mạn tĩnh có hình ảnh tín hiệu cao và thể hiện ảnh màu trắng. Tuy nhiên sự khác biệt giữa hàm lượng nước trong mô không lớn thì độ nhạy hình ảnh T1 không cao. Do đó không thể ghi hình được ở những tổn thương nhỏ không đè đầy cấu trúc giải phẫu.

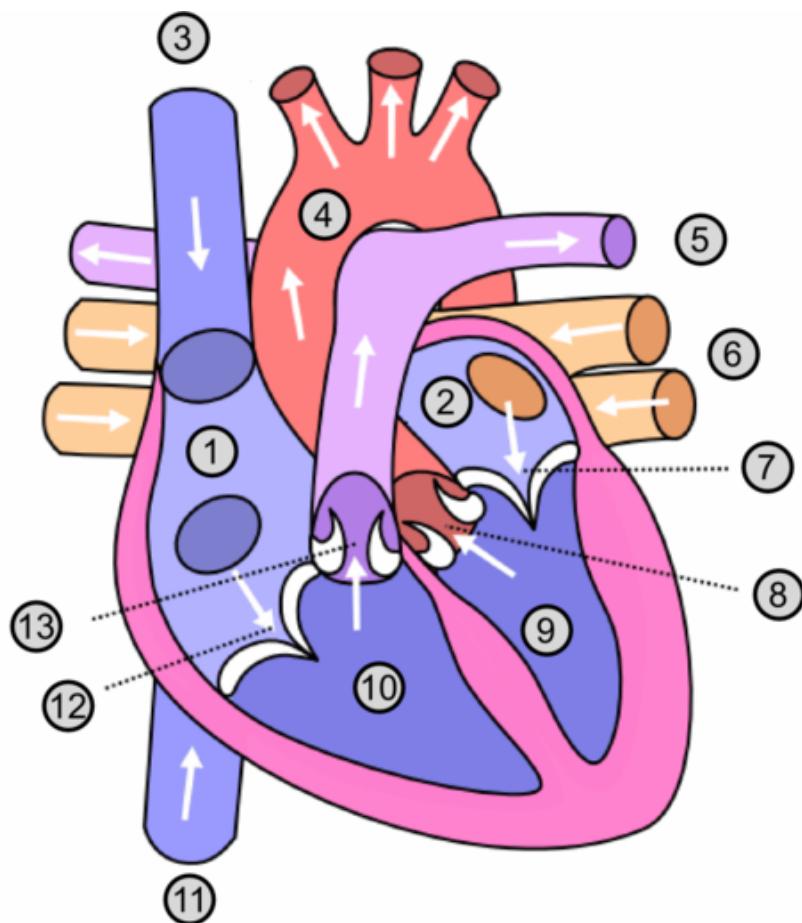
2.1.3 Giải phẫu tim

[2]

Tim nằm ở trung thất, mặt trên cơ hoành. Tim gồm 4 buồng, nằm trên là 2 buồng nhĩ, nằm dưới là 2 buồng thất. Vách gian thất và vách gian nhĩ phân tim theo trực dọc. Bên ngoài cơ tim được bao bọc bởi màng ngoài tim và bên trong được lót bởi màng trong tim. Hai động mạch vành xuất phát từ gốc động mạch chủ có nhiệm vụ cung cấp máu nuôi dưỡng tim.

Tâm nhĩ của tim có thành mỏng hơn tâm thất, nằm ở đáy tim. Tâm nhĩ phải có tĩnh mạch chủ từ dưới đổ vào, tâm nhĩ trái có 4 tĩnh mạch từ phổi đổ vào; [3]

Tâm thất, nhất là tâm thất trái có thành cơ dày hơn tâm nhĩ. Trong tâm thất có các cơ nhú, đầu tự do các cơ có các thùng gân nối vào các lá của van nhĩ thất.



Hình 2.2: Giải phẫu tim: 1. Tâm nhĩ phải; 2. Tâm nhĩ trái; 3. Tĩnh mạch chủ trên; 4. Động mạch chủ; 5. Động mạch phổi; 6. Tĩnh mạch phổi; 7. Van hai lá; 8. Van động mạch chủ; 9. Tâm thất trái; 10. Tâm thất phải; 11. Tĩnh mạch chủ dưới; 12. Van ba lá; 13. Van động mạch phổi

Chu trình tim đập như sau:

Một hệ thống tuần hoàn sẽ được lặp đi lặp lại nhiều lần, giúp máu chảy liên tục đến tim, phổi và các bộ phận khác của cơ thể. Chu trình này diễn ra như sau:

Bên phải tim

Máu được đưa vào tim thông qua hai tĩnh mạch lớn là tĩnh mạch chủ trên và tĩnh mạch chủ dưới. Sau đó, làm trống máu nghèo oxy từ cơ thể vào tâm nhĩ phải của tim. Khi tâm nhĩ co lại, van ba lá sẽ mở ra và máu được chảy từ tâm nhĩ phải vào tâm thất phải. Khi tâm thất đã đầy máu, van ba lá sẽ đóng lại để ngăn không cho máu chảy ngược lại tâm nhĩ trong khoảng thời gian tâm thất co lại. Sau khi tâm thất co lại, van động mạch phổi sẽ đưa máu ra khỏi tim để vào phổi. Tại đây, máu được oxy hóa, thải CO₂ và nhận O₂, sau đó quay trở lại tâm nhĩ trái thông qua các tĩnh mạch phổi.

Bên trái tim

Lúc này, các tĩnh mạch phổi rỗng máu giàu oxy từ phổi vào tâm nhĩ trái của tim. Khi tâm nhĩ co lại, van hai lá mở ra giúp máu chảy từ tâm nhĩ trái vào tâm thất trái. Sau khi tâm thất đã đầy, van hai lá đóng lại, ngăn không cho máu chảy ngược vào tâm nhĩ trong lúc tâm thất co lại.

Sau khi tâm thất đã co lại, van động mạch chủ sẽ đưa máu ra khỏi tim để đến khắp các bộ phận của cơ thể.

Khi máu đi qua van động mạch phổi, nó sẽ tiếp tục đi vào trong phổi. Quá trình này được gọi là tuần hoàn phổi. Máu xuất phát từ van động mạch phổi, sau đó đi đến động mạch phổi và các mao mạch nhỏ bên trong phổi. Tại đây, oxy đi từ các túi khí nhỏ gọi là các phế nang trong phổi và qua các thành của mao mạch vào máu. Trong khi đó, chất thải carbon dioxide được tạo ra qua quá trình trao đổi chất sẽ đi từ máu vào các túi khí và rời khỏi cơ thể khi bạn thở ra. Sau khi máu đã được thanh lọc và oxy hóa, nó sẽ quay trở lại tâm nhĩ trái thông qua các tĩnh mạch phổi.

Trong đề cương này, do không có chuyên môn về lĩnh vực nên em chỉ tập trung vào phân loại dataset có sẵn (nguồn data được cung cấp từ dự án của khoa Máy tính).

2.2 Lý thuyết về xử lý ảnh:

2.2.1 Mạng Neuron tích chập (CNN):

Tích chập (Convolution)

Trong xử lý tín hiệu, tích chập là một khái niệm hết sức quen thuộc, nó là một phép toán cho 2 hàm số để ra một hàm số khác phép tích chập có thể thể hiện nhiều kiểu biến đổi (có thể là lọc tín hiệu, xét xem tín hiệu có theo mẫu nào không,...)

Tích chập của hàm số f và g được viết là $f * g$, là 1 phép biến đổi tích phân đặc biệt:

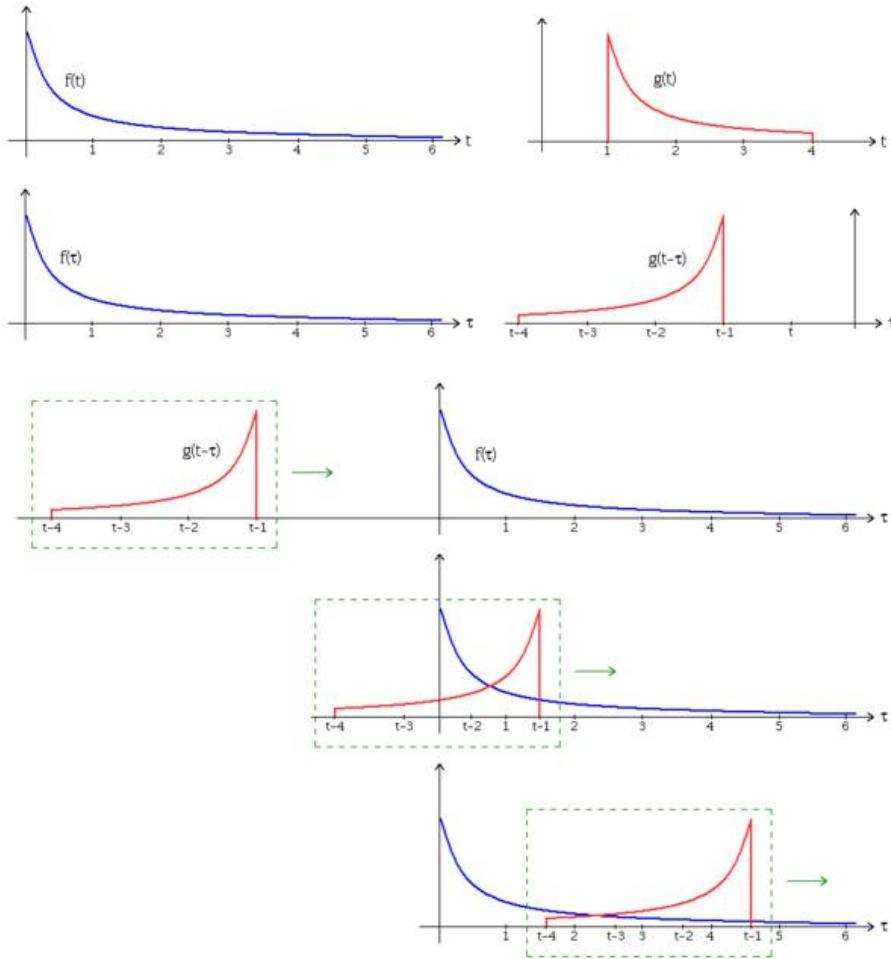
$$\begin{aligned} (f * g)(t) &\stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \\ &= \int_{-\infty}^{\infty} f(t - \tau) g(\tau) d\tau. \quad (\text{giao hoán}) \end{aligned}$$

Hình 2.3: Công thức tích chập

Công thức của tích chập thể hiện tính chất của tích chập đó là khi thực hiện, ta phải trừ ngược lại đáp ứng xung của hệ thống để tích chập thực hiện trước khi tín hiệu xuất

hiện (thể hiện việc tín hiệu có thể xuất hiện ở trước điểm 0)

Trong đó ta thực hiện trừ ngược và lật hàm g lại sau đó tích phân trên cả miền của tín hiệu



Hình 2.4: Biểu diễn tích chập

Ta hoàn toàn có thể coi đây như là tín hiệu 2 chiều vậy, chính vì vậy để lọc các tính chất của ảnh ta có thể áp dụng được các công cụ như tích chập. Chính vì thế dẫn đến các bộ lọc được dùng rất nhiều trong computer vision. Tuy nhiên ở đây tín hiệu có 2 chiều, và hơn nữa lại không có trường hợp tín hiệu xuất hiện trước $t = 0$ nên tích chập ở đây không cần phải lật kernel như tín hiệu 1 chiều. Chính vì vậy nó giống một khái niệm khác là tương quan chéo, mặc dù bản chất không giống nhau, phép này vẫn là tích chập.

Công thức của tích chập 2 chiều:

Ở đây có thể thấy phép convolution được thực hiện giống với 1 chiều, và phép tích phân được thay thế bằng phép tổng, phù hợp với tín hiệu rời rạc

Mạng neuron tích chập

Mạng Nơ-ron Tích Chập có kiến trúc khác với Mạng Nơ-ron thông thường. Mạng Nơ-ron bình thường chuyển đổi đầu vào thông qua hàng loạt các tầng ẩn. Mỗi tầng là một tập các nơ-ron và các tầng được liên kết đầy đủ với các nơ-ron ở tầng trước đó. Và ở tầng cuối cùng sẽ là tầng kết quả đại diện cho dự đoán của mạng.

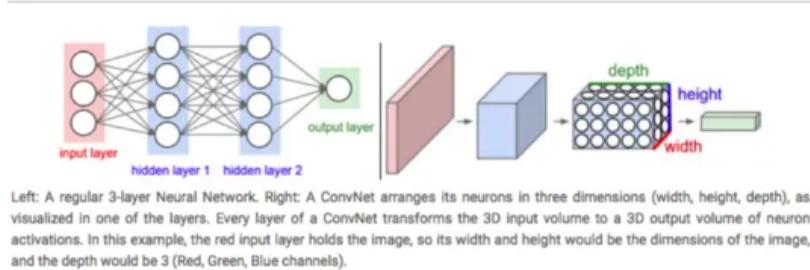
$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i-m, j-n)K(m, n) \quad (2)$$

$$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n)K(-m, -n) \quad (3)$$

Hình 2.5: Biểu diễn tích chập

Đầu tiên, mạng Nơ-ron Tích Chập được chia thành 3 chiều: rộng, cao, và sâu. Kế đến, các nơ-ron trong mạng không liên kết hoàn toàn với toàn bộ nơ-ron kế đến nhưng chỉ liên kết tới một vùng nhỏ. Cuối cùng, một tầng đầu ra được tối giản thành véc-tơ của giá trị xác suất.

CNNs gồm hai thành phần:



Hình 2.6: Biểu diễn tích chập

- Phần tầng ẩn hay phần rút trích đặc trưng: trong phần này, mạng sẽ tiến hành tính toán hàng loạt phép tích chập và phép hợp nhất (pooling) để phát hiện các đặc trưng. Ví dụ: nếu ta có hình ảnh con ngựa vằn, thì trong phần này mạng sẽ nhận diện các sọc vằn, hai tai, và bốn chân của nó.

- Phần phân lớp: tại phần này, một lớp với các liên kết đầy đủ sẽ đóng vai trò như một bộ phân lớp các đặc trưng đã rút trích được trước đó. Tầng này sẽ đưa ra xác suất của một đối tượng trong hình.

Tích chập là một khái quan trọng trong CNN. Thuật ngữ tích chập được dựa trên một phép hợp nhất toán học của hai hàm tạo thành hàm thứ ba. Phép toán này kết hợp hai tập thông tin khác nhau. Trong trường hợp CNN, tích chập được thực hiện trên giá trị đầu vào của dữ liệu và kernel/filter (thuật ngữ này được sử dụng khác nhau tùy tình huống) để tạo ra một bản đồ đặc trưng (feature map).

Ta thực hiện phép tích chập bằng cách trượt kernel/filter theo dữ liệu đầu vào. Tại mỗi vị trí, ta tiến hành phép nhân ma trận và tính tổng các giá trị để đưa vào bản đồ đặc trưng.

Trong hình dưới đây, thành phần kernel/filter (màu xanh lá) trượt trên đầu vào (màu xanh dương) và kết quả được trả về bản đồ đặc trưng (màu đỏ). Kernel/filter có kích thước là 3×3 trong ví dụ này.

Trong thực tế, tích chập được thực hiện trên không gian 3 chiều. Vì mỗi hình ảnh được biểu diễn dưới dạng 3 chiều: rộng, cao, và sâu. Chiều sâu ở đây chính là giá trị màu sắc của hình (RGB). Ta thực hiện phép tích chập trên đầu vào nhiều lần khác nhau.

Mỗi lần sử dụng một kernel/filter khác nhau. Kết quả ta sẽ thu được những bản đồ đặc trưng khác nhau. Cuối cùng, ta kết hợp toàn bộ bản đồ đặc trưng này thành kết quả cuối cùng của tầng tích chập.

Tương tự như mạng nơ-ron thông thường, ta sử dụng một hàm kích hoạt (activate function) để có đầu ra dưới dạng phi tuyến. Trong trường hợp CNN, đầu ra của phép tích chập sẽ đi qua hàm kích hoạt nào đó ví dụ như hàm ReLU (rectified linear units).

Trong quá trình trượt kernel/filter trên dữ liệu đầu vào, ta sẽ quy định một bước nhảy (stride) với mỗi lần di chuyển. Thông thường ta lựa chọn thường chọn bước nhảy là 1. Nếu kích thước bước nhảy tăng, kernel/filter sẽ có ít ô trùng lặp.

Bởi vì kích thước đầu ra luôn nhỏ hơn đầu vào nên ta cần một phép xử lí đầu vào để đầu ra không bị co giãn. Đơn giản ta chỉ cần thêm một lề nhỏ vào đầu vào. Một lề với giá trị 0 sẽ được thêm vào xung quanh đầu vào trước khi thực hiện phép tích chập.

Thông thường, sau mỗi tầng tích chập, ta sẽ cho kết quả đi qua một tầng hợp nhất (pooling layer). Mục đích của tầng này là để nhanh chóng giảm số chiều. Việc này giúp giảm thời gian học và hạn chế việc overfitting.

Một phép hợp nhất đơn giản thường được dùng đó là max pooling, phép này lấy giá trị lớn nhất của một vùng để đại diện cho vùng đó. Kích thước của vùng sẽ được xác định trước để giảm kích thước của bản đồ đặc trưng nhanh chóng nhưng vẫn giữ được thông tin cần thiết.

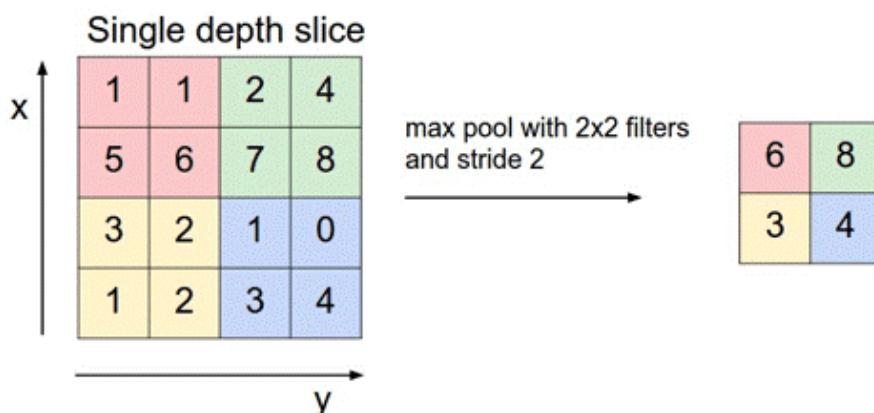
Tổng kết lại khi sử dụng CNN, ta cần chú ý đến 4 tham số quan trọng:

- Kích thước kernel/filter

- Số lượng kernel/filter
- Kích thước bước nhảy (stride)
- Kích thước lề (padding)

Phân lớp

Trong phần phân lớp, ta sử dụng một vài tầng với kết nối đầy đủ để xử lí kết quả của phần tích chập. Vì đầu vào của mạng liên kết đầy đủ là 1 chiều, ta cần làm phẳng đầu vào trước khi phân lớp. Tầng cuối cùng trong mạng CNN là một tầng liên kết đầy đủ, phân này hoạt động tương tự như mạng nơ-ron thông thường.



Hình 2.7: Biểu diễn tích chập

Kết quả thu được cuối cùng sẽ là một véc-tơ với các giá trị xác suất cho việc dự đoán như mạng nơ-ron thông thường.

Nhận diện vật thể (Object Detection)

Phần này em trình bày về Faster RCNN, phương pháp sử dụng để xử lý ảnh tim trong đề cương này. Phần này tham khảo các bài báo [4] [5]

Nguyên lý làm việc của Faster RCNN như sau:

1. Hình ảnh đưa vào sẽ được đi qua một lớp CNN lớn để thực hiện extract feature
2. Sau khi có feature map, sẽ có một lớp RPN (Regional Proposal Network) tìm các khu vực cần quan tâm để thực hiện bounding box.
3. Từ các khu vực đã được tách ra, một feature vector có chiều dài cố định sẽ được tách ra từ mỗi vùng sử dụng lớp pooling ROI
4. Feature vector sau đó được phân loại sử dụng Fast R-CNN.

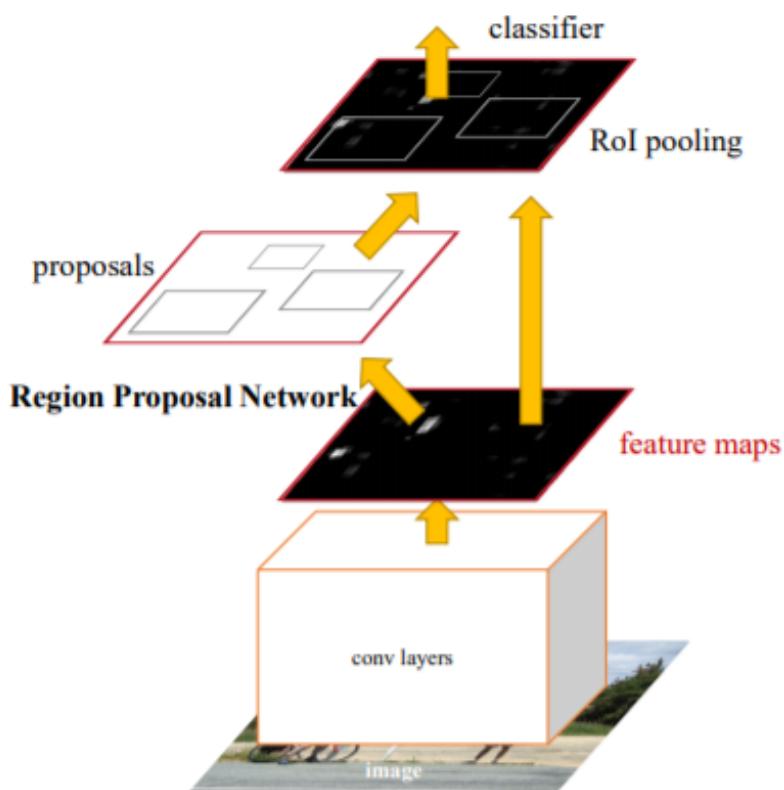


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the ‘attention’ of this unified network.

Hình 2.8: Lớp RPN của Faster RCNN

1. Region Proposal Network (RPN)

Khác biệt so với Fast R-CNN là ở mạng RPN này, tác dụng của nó là thay thế cho phương pháp tìm chọn lọc của RCNN và Fast-RCNN bằng một mạng nơ-ron, và vì

là một mạng nơ-ron, RPN có thể được huấn luyện trong cùng một lần train, và vì đây là một CNN sử dụng chung lớp convolution với Fast R-CNN nên nó không tốn thời gian chạy như tìm kiếm chọn lọc

Dựa trên một cửa sổ kích thước $n \times n$ quét trên cả feature map, các vùng quan tâm sẽ được chọn, các vùng này không phải là các vùng cuối cùng mà chúng sẽ được chọn dựa trên điểm "đối tượng"

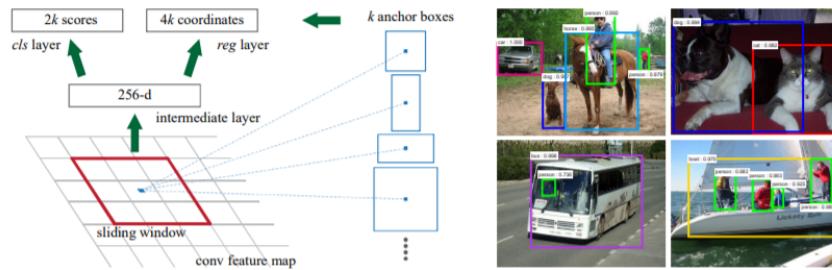


Figure 3: **Left:** Region Proposal Network (RPN). **Right:** Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

Hình 2.9: Lớp RPN của Faster RCNN

2. Anchor

Theo hình tiếp theo, feature map của lớp convolution cuối cùng được đưa qua một cửa sổ trượt có kích thước n^*n , $n = 3$ với lớp convolution cơ sở là VGG-16. Với mỗi cửa sổ trượt, có K vùng đề xuất được tạo, mỗi vùng này được biểu diễn bằng một hình chữ nhật bao quanh vật (anchor). Các thông số của các hộp này là:

- Scale
- Tỉ lệ chiều rộng/cao

Bằng cách sử dụng "neo" (anchor) với hệ số tỉ lệ và tỉ lệ chiều rộng/cao, ta có thu được tính chất scale-invariant (không phụ thuộc tỉ lệ) cho thuật toán mà không cần thu phóng ảnh như các thuật toán cổ điển hoặc phải sử dụng nhiều filters hơn.

Với mỗi n^*n vùng đề xuất, ta tách một feature vector có kích thước xác định để đưa vào 2 lớp fully connected:

Lớp đầu tiên là lớp CLS, lớp này có 2 output, output đầu tiên sẽ sinh ra chỉ số "vật thể", thể hiện khu vực này là vật hay chỉ là background, output thứ 2 là để phân loại vật thể trong khu vực này là gì.

Lớp thứ 2 gọi là reg, output là một vector 4 chiều có chứa bounding box của khu vực.

3. Objectness Score

Để train RPN, mỗi anchor được cho điểm đối tượng dương hoặc âm dựa trên Intersection-over-Union (IoU).

IoU là tỉ số giữa diện tích của anchor box và ground truth box. IoU có thể giao động từ 0.0 đến 1.0, trong đó 1.0 là anchor box trùng hoàn toàn ground truth box, 0.0 là không hề trùng nhau.

4 điều kiện dưới đây sử dụng IoU để định nghĩa điểm đối tượng:

Anchor có IoU lớn hơn 0.7 so với ground truth box là positive .

Nếu không có anchor nào có IoU lớn hơn 0.7, thì gán nhãn dương cho anchor có giá trị IoU cao nhất mà trùng với ground truth.

Điểm đối tượng âm được gán cho anchor khi IoU cho tất cả các ground truth box nhỏ hơn 0.3. Điểm âm là anchor box này là background.

Anchors không đóng góp cho quá trình training được đánh là không liên đến quá trình training khi $0.3 < \text{IoU} < 0.5$

4. Training Faster R-CNN

Theo paper Faster R-CNN có 3 cách training:

Cách đầu tiên gọi là alternating training, trong đó RPN được train để tạo ra các vùng đề xuất. Trọng số của các lớp convolutional dùng chung được khởi tạo dựa trên một mô hình pre-trained trên ImageNet. Các trọng số khác của RPN được khởi tạo ngẫu nhiên.

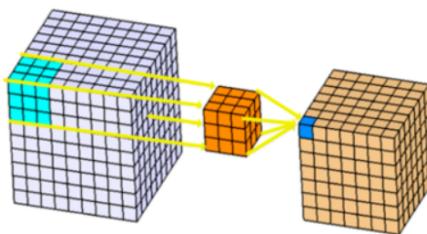
Sau khi RPN tạo ra anchor box, trọng số của cả RPN và lớp convolution dùng chung sẽ được tune lại.

Các vùng đề xuất tạo ra bởi RPN sẽ được dùng để train Fast R-CNN.

3D convolution

[6] Lớp convolution chúng ta thường thấy và sử dụng được gọi là convolution 2D, mỗi lớp này cho ra output chỉ gồm 2 chiều, vậy nếu trường hợp cần lớn hơn 2 chiều thì sao?

3D convolutions thực hiện một bộ lọc 3 chiều lên dataset và filter đi theo 3 chiều (x, y, z). Output shape của lớp này là một ma trận 3 chiều, do đó lớp này có thể lấy được nhiều thông tin về liên hệ giữa các điểm trong không gian 3 chiều hơn, và do đó hợp hơn với ảnh cắt lớp tim như bài toán đề ra.



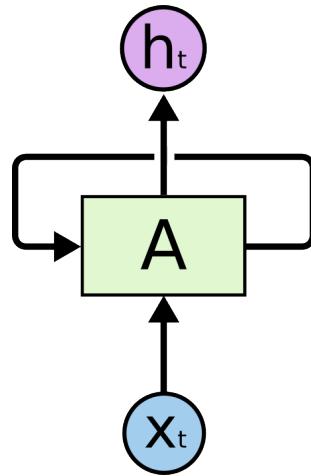
Hình 2.10: RNN unrolled

2.3 Long short term memory

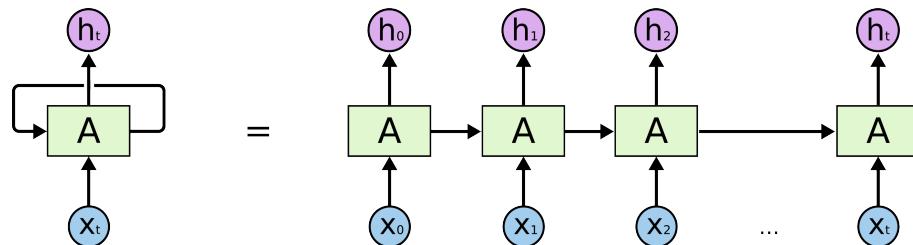
2.3.1 Recurrent Neural Network

Để thực hiện xử lý các giá trị có tính chất chuỗi (sequence) với mạng nơ-ron, kiến trúc Recurrent Neural Network được sử dụng. Phần này được tham khảo từ blog [7]

Hình trên là biểu diễn ý tưởng cơ bản của mạng nơ-ron RNN: dữ liệu từ các thời điểm khác nhau trong thời gian đều có ảnh hưởng đến dữ liệu ở thời điểm tiếp sau:



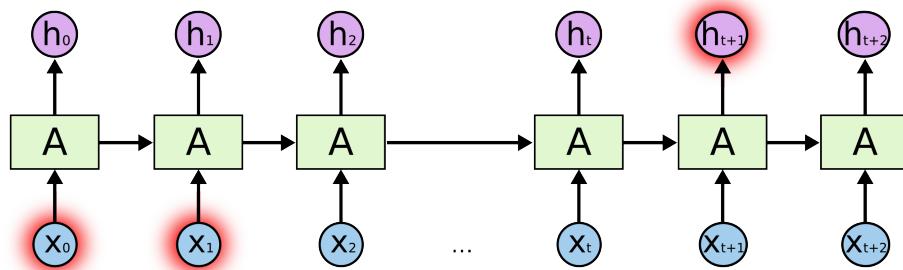
Hình 2.11: RNN unrolled



Hình 2.12: Biểu diễn RNN theo thời gian (thứ tự trong sequence)

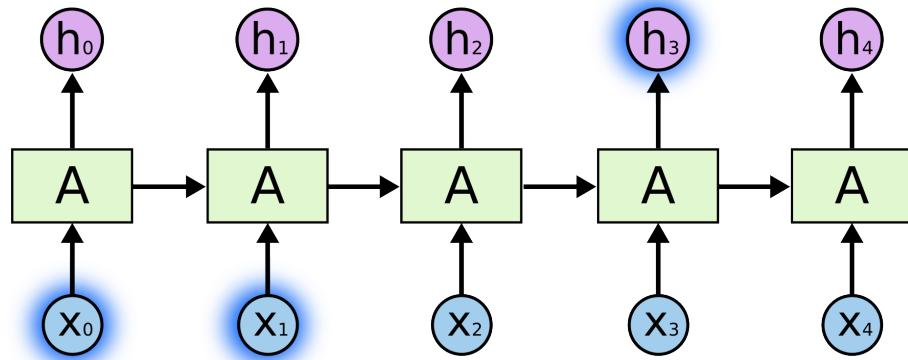
Bài blog rất nổi tiếng của Andrew Karpathy đã nói về tính hiệu quả của RNN, tuy nhiên RNN có một số vấn đề, nên thực tế biến thể của nó là LSTM được sử dụng nhiều hơn.

Ví dụ trong trường hợp chuỗi dữ liệu cho RNN quá dài, những dữ liệu ban đầu cũng ảnh hưởng đến đầu ra cho dù những dữ liệu ấy có thể không còn liên quan đến dữ liệu hiện tại:



Hình 2.13: Long term memory ảnh hưởng đến RNN

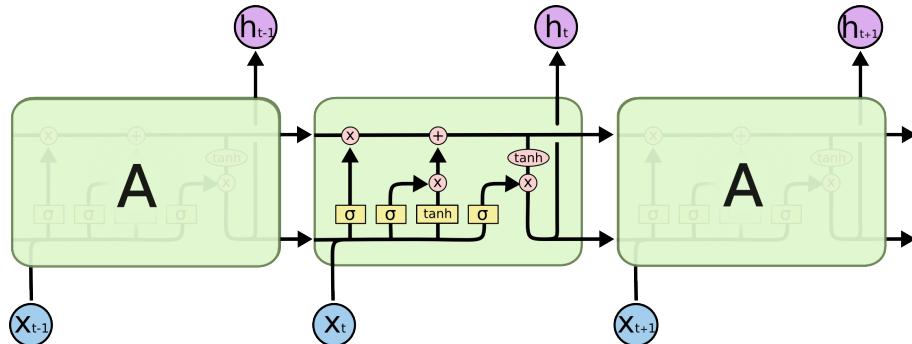
Trường hợp chúng ta mong muốn đó là dữ liệu đầu vào của RNN chỉ ảnh hưởng đến đầu ra ở một thời điểm xác định như sau:



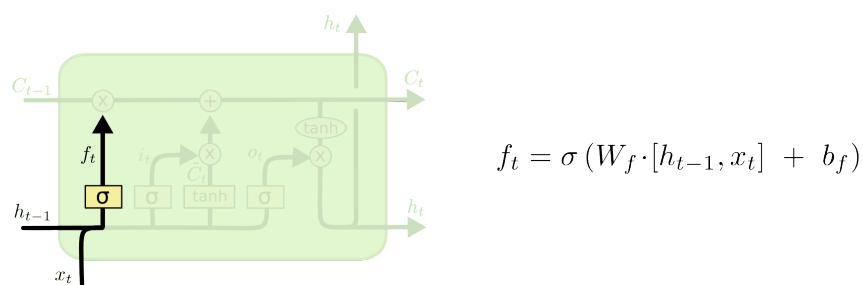
Hình 2.14: Biểu diễn RNN theo thời gian (thứ tự trong sequence)

2.3.2 Long Short Term Memory

LSTM là một biến thể của RNN, trong đó các mạng nơ-ron không chỉ sử dụng dữ liệu từ các điểm dữ liệu trước mà còn có cổng "quên" để các giá trị quá xa trong quá khứ không ảnh hưởng đến các giá trị hiện tại.



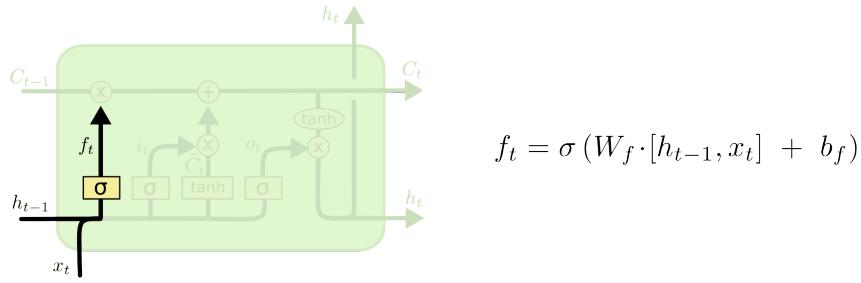
Hình 2.15: Ví dụ 1 cell của LSTM



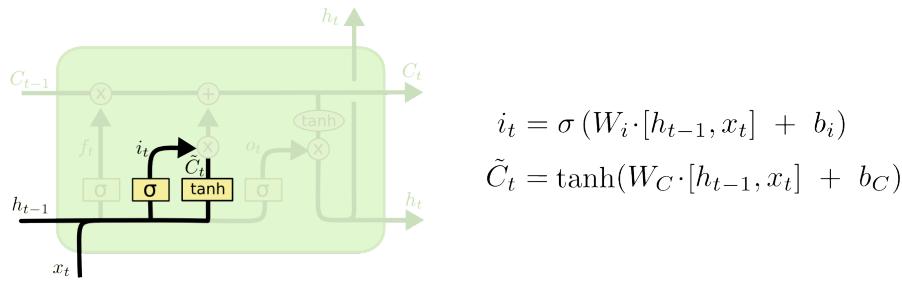
Hình 2.16: Cổng quên của LSTM

Hình dưới đây hiển thị cổng "quên" của 1 cell LSTM, đây là một lớp nơ-ron sigmoid có input là đầu ra của cell trước đó h_{t-1} và trọng số W_f .

Cổng quên này có thể có giá trị giữa 0 và 1, 0 tức cell sau hoàn toàn "quên" các giá trị từ cell trước, 1 tức là cell sau nhận toàn bộ giá trị của cell trước.

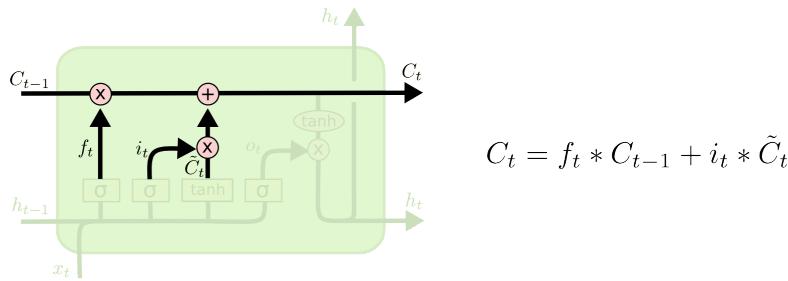


Hình 2.17: Cổng input của LSTM



Hình 2.18: Ví dụ 1 cell của LSTM

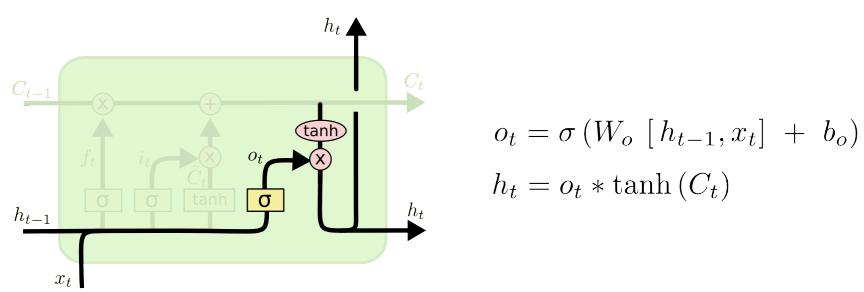
Tiếp theo ta có cổng input, giá trị trạng thái tiềm năng \hat{C}_t của cell sẽ được cập nhật theo công thức như hình, giá trị này được tính bằng một lớp nơ-ron tanh với input là giá trị đầu vào \mathbf{h}_{t-1} và giá trị input của cell hiện tại là \mathbf{x}_t , đầu ra của cổng này được nhân pointwise với giá trị của output cell trước với một lớp sigmoid.

Hình 2.19: Tính giá trị \hat{C}_t của cell LSTM thứ t

Tiếp tục với bước cập nhật giá trị C_t theo công thức như hình trên, ta nhận giá trị \hat{C}_t với giá trị quên đã tính ở trước và cộng với giá trị

Cuối cùng, cổng output của cell như sau:

Giá trị output \mathbf{h}_t được tính dựa trên trạng thái của cell C_t sau khi qua một lớp mạng nơ-ron tanh (để đưa giá trị về đoạn -1 đến 1), giá trị này được nhân với giá trị đầu ra của lớp trước sau khi qua một lớp sigmoid.

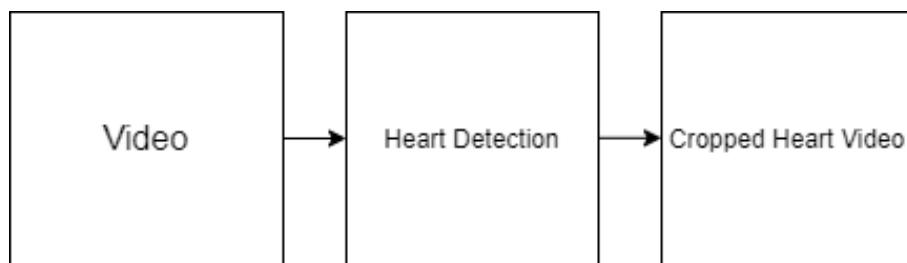


Hình 2.20: Cổng output của LSTM

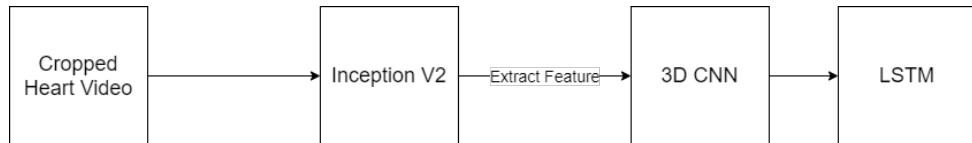
Chương 3. KẾT QUẢ NGHIÊN CỨU

3.1 Phương pháp tiếp cận

Sơ đồ khối phần mềm:



Hình 3.1: Video MRI tim bình thường



Hình 3.2: Video MRI tim bình thường

3.1.1 Heart Detection

Xây dựng và test mô hình heart detection, em sử dụng Tensorflow Object Detection API, cụ thể hơn là sử dụng mô hình Faster RCNN.

Tập train > 1200 tấm

Tập test > 300 tấm

Loss 0.01

Phương pháp train mạng nơ-ron như sau:

Trước tiên em phải đánh dấu dataset bằng tool labelimg: [8]

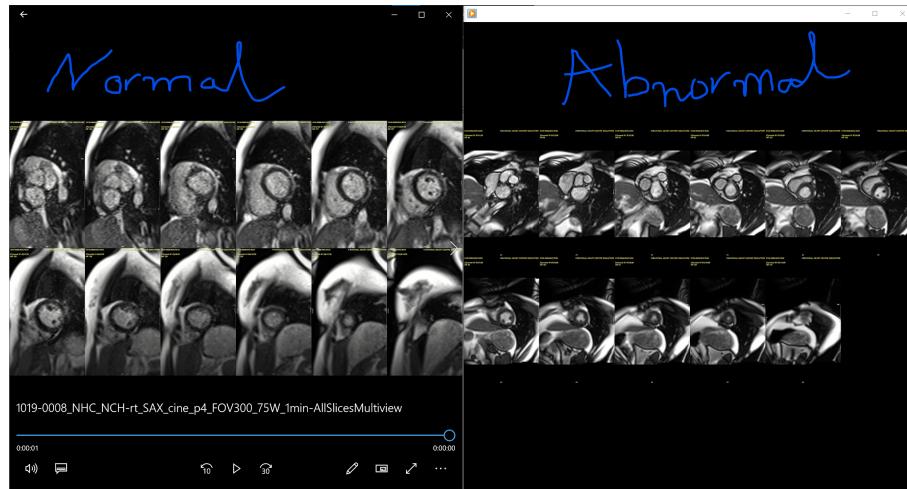
Sau khi sử dụng tool này để tạo dataset, ta sẽ có dataset có dạng như sau:

Sau khi đã đánh dấu dataset, việc tiếp theo là tool này sẽ tự động sinh ra các file .xml chứa thông tin ảnh và địa chỉ ảnh:

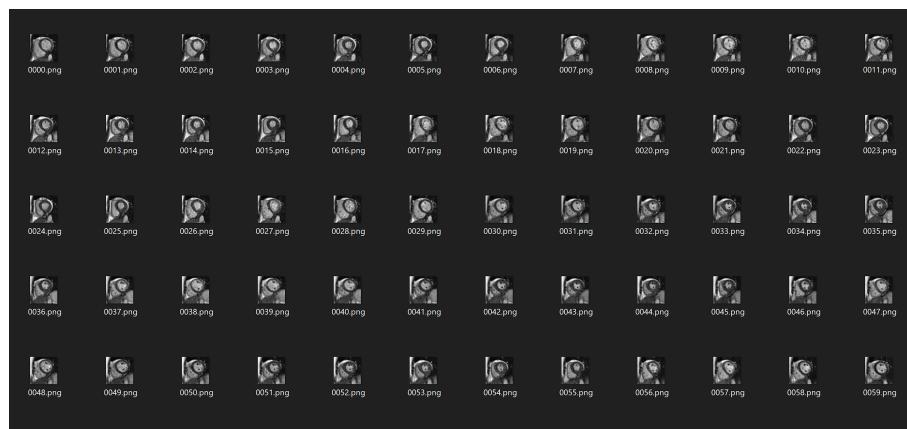
Sau khi đánh dấu tất cả dataset, ta cần convert data này sang dạng tfrecord để thuận tiện training hơn:

Trước tiên ta phải cài đặt Tensorflow Object Detection API [9]

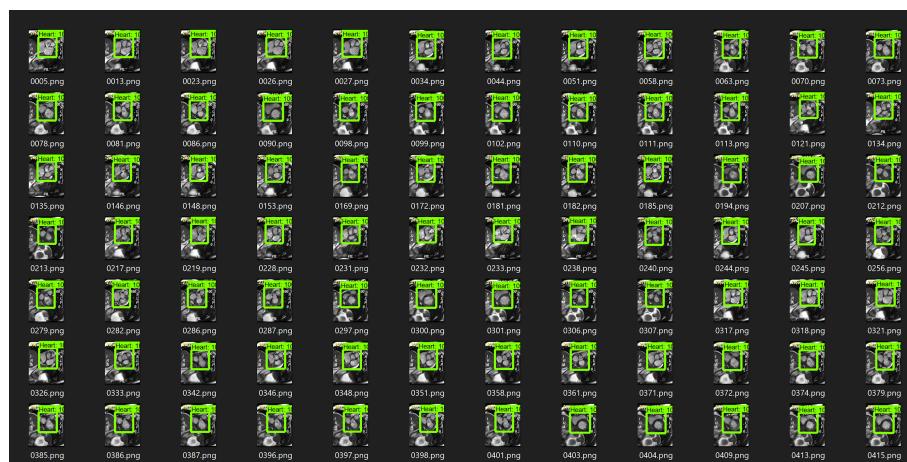
Sau khi cài xong ta cần chạy file python sau để convert file thành dạng tfrecord.



Hình 3.3: Video MRI tim bình thường và bất bình thường

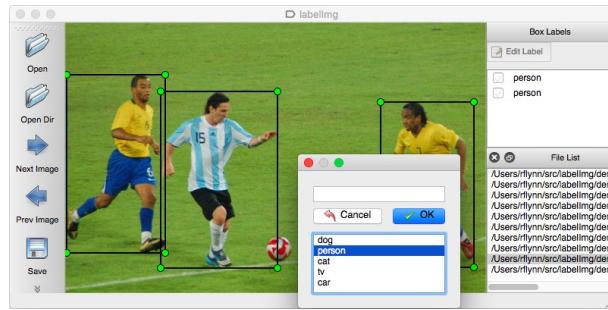


Hình 3.4: Data ban đầu

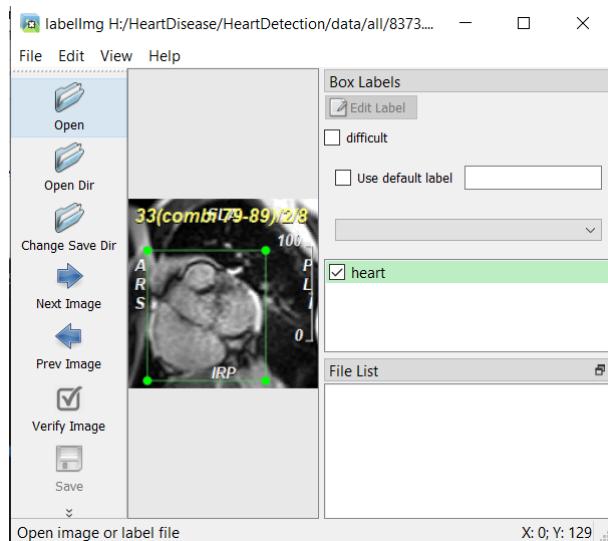


Hình 3.5: Data sau khi được đánh dấu

```
python object_detection/dataset_tools/create_pascal_tf_record.py \
--label_map_path=object_detection/data/pascal_label_map.pbtxt \
```



Hình 3.6: Tool labelImg để tạo dataset



Hình 3.7: Tool labelImg để tạo dataset

```
--data_dir=VOCdevkit --year=VOC2012 --set=train \
--output_path=pascal_train.record
python object_detection/dataset_tools/create_pascal_tf_record.py \
--label_map_path=object_detection/data/pascal_label_map.pbtxt \
--data_dir=VOCdevkit --year=VOC2012 --set=val \
--output_path=pascal_val.record
```

Sau khi đã có file tfrecord, em sử train model object detection như hướng dẫn của tensorflow: [9]

```
! python /content/models/research/object_detection/model_main.py \
--pipeline_config_path={pipeline_fname} \
--model_dir={model_dir} \
--alsologtostderr \
--num_train_steps={num_steps} \
--num_eval_steps={num_eval_steps}
```

File config như sau:

```

H: > HeartDisease > HeartDetection > data > all > 8373.xml
1   <annotation>
2     <folder>all</folder>
3     <filename>8373.png</filename>
4     <path>H:\HeartDisease\HeartDetection\data\all\8373.png</path>
5     <source>
6       <database>Unknown</database>
7     </source>
8     <size>
9       <width>160</width>
10      <height>160</height>
11      <depth>3</depth>
12    </size>
13    <segmented>0</segmented>
14    <object>
15      <name>heart</name>
16      <pose>Unspecified</pose>
17      <truncated>0</truncated>
18      <difficult>0</difficult>
19      <bndbox>
20        <xmin>15</xmin>
21        <ymin>43</ymin>
22        <xmax>115</xmax>
23        <ymax>153</ymax>
24      </bndbox>
25    </object>
26  </annotation>
27

```

Hình 3.8: File xml mà labelimg sinh ra

```

H: > HeartDisease > HeartDetection > faster_rcnn_inception_v2.config
1  # Faster R-CNN with Inception v2, configured for Oxford-IIIT Pets Dataset.
2  # Users should configure the fine_tune_checkpoint field in the train config as
3  # well as the label_map_path and input_path fields in the train_input_reader and
4  # eval_input_reader. Search for "PATH_TO_BE_CONFIGURED" to find the fields that
5  # should be configured.
6
7  model {
8    faster_rcnn {
9      num_classes: 1
10     image_resizer {
11       keep_aspect_ratio_resizer {
12         min_dimension: 600
13         max_dimension: 1024
14       }
15     }
16     feature_extractor {
17       type: 'faster_rcnn_inception_v2'
18       first_stage_features_stride: 16
19     }
20     first_stage_anchor_generator {
21       grid_anchor_generator {
22         scales: [0.25, 0.5, 1.0, 2.0]
23         aspect_ratios: [0.5, 1.0, 2.0]
24         height_stride: 16
25         width_stride: 16
26       }
27     }
28     first_stage_box_predictor_conv_hyperparams {
29       op: CONV
30       regularizer {
31         l2_regularizer {
32           weight: 0.0
33         }
34       }
35       initializer {
36         truncated_normal_initializer {
37           stddev: 0.01
38         }
39     }
40   }
41   train_config {
42     data {
43       tfrecord {
44         files: "train.record"
45       }
46     }
47     fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED/fine_tuned_inception_v2.ckpt"
48     batch_size: 2
49     num_steps: 100000
50     data_augmentation_options {
51       random_hue {
52       }
53     }
54     optimizer {
55       momentum_optimizer {
56         learning_rate {
57           fixed_lr {
58             lr: 0.001
59           }
60         }
61       }
62     }
63   }
64   eval_config {
65     num_examples: 100
66     batch_size: 1
67   }
68 }

```

Hình 3.9: File config của Faster R-CNN

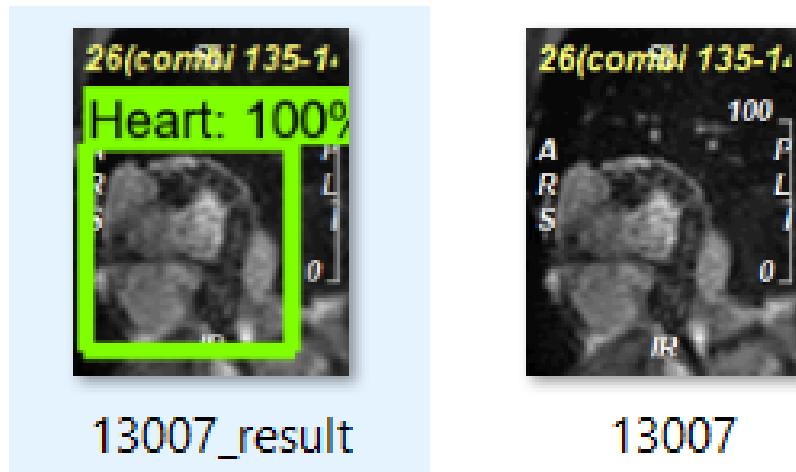
Model sử dụng là Faster - RCNN, việc tranining được thực hiện trên google colab và tồn tầm 3 tiếng.

Để chạy model, em load lại mô hình đã lưu ở dạng file .pb và chạy detection trên tập data test:

```
130
131     # Actual detection.
132     output_dict = self.run_inference(image_np_expanded)
133
134     potential_box_index = np.argmax(output_dict['detection_scores'])
135     if (output_dict['detection_scores'][potential_box_index] > self.HEART_THRESHOLD and
136         output_dict['detection_classes'][potential_box_index] == self.LABEL_MAP_DICT['Heart']):
137
138         ymin, xmin, ymax, xmax = tuple(output_dict['detection_boxes'][potential_box_index].tolist())
139         (left, right, top, bottom) = (xmin * im_width, xmax * im_width,
140                                       ymin * im_height, ymax * im_height)
141         if return_crop:
142             crop_heart = self.padding_crop_image(image_np, top, bottom, left, right)
143             return crop_heart, (top, bottom, left, right)
144
145     return (top, bottom, left, right)
146 else:
147     assert 0, "No cardiac found!"
148
149 if __name__ == '__main__':
150     detection = CardiacDetection()
151
152     # To test visualize
153     imgs = os.listdir('data/test')
154     imgs = list(filter(lambda x: x[-3:].lower() == 'png', imgs))
155
156     for img in imgs:
157         print('Processing: ' + img)
158         image = Image.open('data/test/' + img)
159         image_np = detection.run_visualize(image)
160         plt.imsave('data/test_result/' + img, image_np)
```

Hình 3.10: Code phần test model

Kết quả:



Hình 3.11: Kết quả của việc train model detect tim

Trong thực tế, chỉ khu vực tim được detect sẽ được sử dụng cho các tác vụ tiếp theo.

3.1.2 Heart Classification

Bước classification này là bước sau khi đã tiền xử lý video bằng cách detect và cắt phần ảnh tim cần thiết ra. Để thử nghiệm thuật toán, em tạo một dataset có 12 video, mỗi video 30s, trong đó 6 video nhịp tim bình thường và 6 video nhịp tim bất thường.

Bài toán classify này có thể xem như bài toán sequence (frame) classification, nhưng khác biệt là bài toán sequence classification thông thường (bên lĩnh vực NLP) thì input chỉ là 1 từ, còn với bài toán này lại là 1 frame chứa 11 channel.

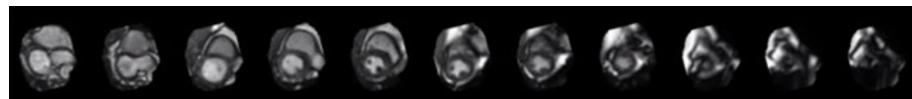
Đầu tiên dùng các mô hình có sẵn (cụ thể là Inception Resnet V2) để extract các feature trên 1 channel của 1 frame, sau đó ghép chúng lại.

Để mô hình có thể học được những đặc trưng của 1 sequence, thì chúng ta cần phải đưa nó qua những tầng CNN 3D (vì 1 sequence ảnh là một không gian 3D)

Sau khi đã qua các tầng CNN 3D thì chúng ta sẽ dùng 1 tầng LSTM để giúp mô hình học được sự liên kết giữa các frame

Cuối cùng, chúng ta cho output từ LSTM qua các tầng Fully Connected để classify.

Với tập dữ liệu đã segment tim và xoay cùng hướng: Mô hình không thể hội tụ, mặc dù đã train gần 20 epochs nhưng loss không thể giảm và accuracy không thể tăng



Hình 3.12: Hình tim sau khi đã preprocess



Hình 3.13: Hình tim ngay sau khi detect được crop ra

Mô hình classification này gồm 2 phần: - Feature Extractor

- Classifier Model

Mô hình feature extractor là mô hình Inception V2, trong quá trình training phần này chỉ đóng vai trò extract feature cho các tác vụ training classifier sau này.

Data đầu vào là video có 11 lát cắt tim, nên phần này sẽ nhận vào vào 11 ảnh RGB, output ra sẽ là nối 11 output này lại thành 1 lớp để đưa vào phần classifier phía sau.

Mô hình classifier này gồm các lớp 3D CNN và LSTM, video này có khoảng 500 frame, do đó quá trình train sử dụng LSTM 512 cell để nhận diện sự liên hệ giữa các frame.

Do input là các feature có độ sâu lớn và mối liên hệ theo không gian 3 chiều nên cần sử dụng lớp 3D convolution.

```

205 class MainModel():
206     def __init__(self, batch_size=1, seq_frame_length=6, channels=11, num_class=2, ckpt_path='./model/ckpt'):
207         self.IRN2 = IR2.InceptionResNetV2(weights='imagenet', include_top=False,
208                                         pooling='max', input_tensor=Input(shape=(299, 299, 3)))
209         self.feature_extractor = Model(inputs=self.IRN2.input, outputs=self.IRN2.get_layer('activation_75').output)
210         self.feature_extractor._make_predict_function()
211
212         self.classifier_model = ClassifierModel(batch_input_shape=(batch_size, seq_frame_length, self.feature_extractor.output.shape[1].value,
213                                                 channels * self.feature_extractor.output.shape[2].value, self.feature_extractor.output.shape[3].value),
214                                                 num_class=num_class)
215         self.classifier_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
216         self.checkpoint = None
217         self.continue_training = False
218
219         if os.path.exists(os.path.join(ckpt_path, 'model_best_ckpt.h5')):
220             self.load_model(os.path.join(ckpt_path, 'model_best_ckpt.h5'))
221             self.continue_training = True
222         else:
223             ckpt = os.listdir(ckpt_path)
224             ckpt = [x for x in ckpt if x[-2:] == 'h5']
225
226             if ckpt:
227                 self.load_model(os.path.join(ckpt_path, ckpt[-1]))
228                 self.continue_training = True
229
230         self.seq_frame_length = seq_frame_length
231         self.batch_size = batch_size
232         self.channels = channels
233         self.num_class = num_class
234         self.ckpt_path = ckpt_path

```

Hình 3.14: Code của mô hình

```

87     def ClassifierModel(batch_input_shape, num_class):
88         model = Sequential()
89         model.add(ZeroPadding3D(padding=(1,1,1), input_shape=batch_input_shape[1:]))
90         model.add(Convolution3D(128, 3, activation='relu'))
91         model.add(ZeroPadding3D(padding=(1,1,1)))
92         model.add(Convolution3D(128, 3, activation='relu'))
93         model.add(MaxPooling3D((1, 3, 3), strides=(1, 1, 1)))
94
95         model.add(ZeroPadding3D(padding=(1,1,1)))
96         model.add(Convolution3D(256, 3, activation='relu'))
97         model.add(ZeroPadding3D(padding=(1,1,1)))
98         model.add(Convolution3D(256, 3, activation='relu'))
99         model.add(MaxPooling3D((1, 3, 3), strides=(1, 1, 1)))
100
101        model.add(Lambda(lambda x: K.mean(x, axis=4)))
102        model.add(Dense(128, activation='relu'))
103        model.add(Dense(32, activation='relu'))
104        model.add(Reshape((batch_input_shape[1], 992)))
105        model.add(LSTM(512))
106        model.add(Dense(256, activation='relu'))
107        model.add(Dropout(0.2))
108        model.add(Dense(num_class, activation='softmax'))
109
110    return model

```

Hình 3.15: Code model classifications

Để tính ra chỉ số phần trăm của

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
zero_padding3d_1 (ZeroPaddin	(None, 8, 37, 387, 256)	0
conv3d_1 (Conv3D)	(None, 6, 35, 385, 128)	884864
zero_padding3d_2 (ZeroPaddin	(None, 8, 37, 387, 128)	0
conv3d_2 (Conv3D)	(None, 6, 35, 385, 128)	442496
max_pooling3d_1 (MaxPooling3	(None, 6, 33, 383, 128)	0
zero_padding3d_3 (ZeroPaddin	(None, 8, 35, 385, 128)	0
conv3d_3 (Conv3D)	(None, 6, 33, 383, 256)	884992
zero_padding3d_4 (ZeroPaddin	(None, 8, 35, 385, 256)	0
conv3d_4 (Conv3D)	(None, 6, 33, 383, 256)	1769728
max_pooling3d_2 (MaxPooling3	(None, 6, 31, 381, 256)	0
lambda_1 (Lambda)	(None, 6, 31, 381)	0
dense_1 (Dense)	(None, 6, 31, 128)	48896
dense_2 (Dense)	(None, 6, 31, 32)	4128
reshape_1 (Reshape)	(None, 6, 992)	0
lstm_1 (LSTM)	(None, 512)	3082240
dense_3 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 2)	514
=====		
Total params: 7,249,186		
Trainable params: 7,249,186		
Non-trainable params: 0		

Hình 3.16: Model classifier

Quá trình train: Trong quá trình train em chỉ train phần model classification, quá trình train sử dụng lớp callback của Keras để dừng khi loss không cải thiện.

```

def fit(self, X_train, Y_train, X_val=None, Y_val=None, epochs=100):
    """
    x_train: (videos, frames, channels, row, col, color_channels)
    y_train: (videos, label)
    """
    training_generator = DataGenerator(len(X_train), self.seq_frame_length, self.channels, self.num_class,
                                       X_train, Y_train, self.feature_extractor, batch_size=1)

    early_stop = EarlyStopping(monitor='val_loss', patience=5)

    if (X_val and Y_val):
        validation_generator = DataGenerator(len(X_val), self.seq_frame_length, self.channels, self.num_class,
                                             X_val, Y_val, self.feature_extractor, batch_size=1)
        checkpoint = ModelCheckpoint(os.path.join(self.ckpt_path, 'model_best_ckpt.h5'), save_best_only=True, mode='min', save_weights_only=True)

        H = self.classifier_model.fit_generator(generator=training_generator,
                                                validation_data=validation_generator,
                                                callbacks=[checkpoint, early_stop],
                                                epochs=epochs)
    else:
        checkpoint = ModelCheckpoint(os.path.join(self.ckpt_path, 'model_{epoch:04d}_ckpt.h5'), period=5, save_weights_only=True)

        H = self.classifier_model.fit_generator(generator=training_generator,
                                                callbacks=[checkpoint, early_stop],
                                                epochs=epochs)

    del modelVGG, self.ckpt_path

```

Hình 3.17: Code training mô hình

3.2 Kết quả và phân tích

Kết quả training sau khoảng 10 epoch: Mô hình hội tụ khá nhanh, hiện tại em mô hình bị overfit, hướng giải quyết là thêm nhiều data vào mô hình.

```

Preprocessing 12/13 ...
Epoch 1/15
  1/313 [=====] - ETA: 48:40 - loss: 0.0063 - accuracy: 1.00002021-01-14 16:19:09.
2021-01-14 16:19:09.900080: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
  2/313 [=====] - ETA: 29:20 - loss: 0.1287 - accuracy: 1.00002021-01-14 16:19:11.
  313/313 [=====] - 586s 2s/step - loss: 0.0720 - accuracy: 0.9808
/tensorflow-1.15.2/python3.6/keras/callbacks/callbacks.py:846: RuntimeWarning: Early stopping conditioned on
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning
Epoch 2/15
  313/313 [=====] - 579s 2s/step - loss: 0.0883 - accuracy: 0.9840
Epoch 3/15
  313/313 [=====] - 579s 2s/step - loss: 0.0682 - accuracy: 0.9808
Epoch 4/15
  313/313 [=====] - 575s 2s/step - loss: 0.0694 - accuracy: 0.9808
Epoch 5/15
  313/313 [=====] - 572s 2s/step - loss: 0.0747 - accuracy: 0.9840
Epoch 6/15
  313/313 [=====] - 576s 2s/step - loss: 0.0610 - accuracy: 0.9776
Epoch 7/15
  313/313 [=====] - 570s 2s/step - loss: 0.0882 - accuracy: 0.9808
Epoch 8/15
  313/313 [=====] - 575s 2s/step - loss: 0.0709 - accuracy: 0.9840
Epoch 9/15
  313/313 [=====] - 571s 2s/step - loss: 0.0630 - accuracy: 0.9872
Epoch 10/15
  166/313 [=====] - ETA: 4:28 - loss: 0.0926 - accuracy: 0.9639^C

```

Hình 3.18: Kết quả training sau khoảng 10 epoch

3.3 Kết luận chương

Do thiếu sót các kiến thức nền tảng về bài toán và giới hạn phần cứng google colab, hiện tại em chưa thể thực hiện thí nghiệm kiểm chứng sự hiệu quả của thuật toán trên data thật.

Chương 4. KẾT LUẬN

4.1 Tóm tắt và kết luận chung

Các điểm đã đạt được

- Tìm hiểu lý thuyết xử lý ảnh, CNN, Object Detection, LSTM,...
- Training Faster RCNN để thực hiện nhận diện tim,
- Thủ thực hiện mô hình classification video tim dựa trên 3D convolution và LSTM

4.2 Mục tiêu và tiến độ thực hiện

Mục tiêu dự kiến thực hiện khi hoàn thành luận văn tốt nghiệp bao gồm:

1. Hoàn chỉnh chương trình xử lý video MRI tim.
2. Thủ thí nghiệm với Attention và so sánh với LSTM.
3. Ước lượng và phân tích hiệu quả hệ thống.

Kế hoạch thực hiện như Bảng 4.1:

Bảng 4.1: Tiến độ thực hiện luận văn

Nội dung	Tháng			
	1	2	3	4
Tìm hiểu lý thuyết	o			
Viết chương trình	o	o		
Chạy mô phỏng		o	o	
Phân tích kết quả			o	o
Viết báo cáo				o

TÀI LIỆU THAM KHẢO

- [1] Wikipedia contributors, “Chụp cộng hưởng từ,” 2021, [Online; accessed 14-January-2021]. [Online]. Available: https://vi.wikipedia.org/wiki/Ch%e1%bb%a5p_c%e1%bb%99ng_h%C6%B0%E1%BB%9Fng_t%E1%BB%AB
- [2] ——, “Tim,” 2021, [Online; accessed 14-January-2021]. [Online]. Available: <https://vi.wikipedia.org/wiki/Tim>
- [3] ——, “Cardiac cycle,” 2021, [Online; accessed 14-January-2021]. [Online]. Available: https://en.wikipedia.org/wiki/Cardiac_cycle
- [4] R. Girshick, “Fast r-cnn,” 2015.
- [5] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” 2015.
- [6] S. Bansal. 3d convolutions : Understanding + use case. [Online]. Available: <https://www.kaggle.com/shivamb/3d-convolutions-understanding-use-case>
- [7] C. Olah. 3d convolutions : Understanding + use case. [Online]. Available: <https://colah.github.io/posts/2015-08-Understanding-LSTMs>
- [8] Tzutalin. LabelImg. [Online]. Available: <https://github.com/tzutalin/labelImg>
- [9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “Quick start: Distributed training on the oxford-iiit pets dataset on google cloud,” 2015. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_pets.md