ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA VẬT LÝ

NHÓM 6

HUMAN ACTIVITY RECOGNITION

Học phần Thị Giác Máy Tính

Ngành Kỹ thuật Điện tử và Tin học

(Chương trình đạo tạo chuẩn)

Hà Nội - 2023

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA VẬT LÝ

NHÓM 6

HUMAN ACTIVITY RECOGNITION

Học phần Thị Giác Máy Tính

Ngành Kỹ thuật Điện tử và Tin học

(Chương trình đạo tạo chuẩn)

Giảng viên hướng dẫn: TS. Phạm Tiến Lâm

Thầy Đặng Văn Báu

Hà Nội - 2023

Lời cảm ơn

Trên thực tế không có sự thành công nào mà không gắn liền với những sự giúp đỡ dù ít hay nhiều, dù trực tiếp hay gián tiếp từ những người xung quanh. Trong suốt quá trình học tập và làm việc tại giảng đường tới nay, nhóm 6 đã nhận được nhiều sự quan tâm, giúp đỡ từ các thầy cô, gia đình và bạn bè. Nhóm 6 xin gửi lời cảm ơn chân thành nhất đến **TS. Phạm Tiến Lâm** và **Thầy Đặng Văn Báu** đã tận tâm hướng dẫn nhóm qua những buổi thảo luận, thực hành về môn học. Thầy đã giúp nhóm tích lũy thêm nhiều kiến thức, kĩ năng để có cái nhìn toàn diện ,sâu sắc và tính thiết thực mà môn học mang đến.

Kiến thức là vô hạn mà sự tiếp thu tri thức của mỗi người là khác nhau và còn tồn tại những hạn chế nhất định. Do đó, trong quá trình hoàn thành bài báo cáo này, chắc chắn không tránh khỏi những thiếu sót. Nhóm 6 rất mong nhận được những ý kiến đóng góp quý báu từ thầy để bài báo cáo của nhóm được hoàn thiện hơn. Thời gian trôi qua thật nhanh, mới ngày nào chúng em còn bỡ ngỡ bước vào môn học , được thầy dìu dắt, dạy bảo. Dưới sự giảng dạy tận tình, tâm huyết của thầy, chúng em đã được tiếp thu những kiến thức bổ ích, quý giá, giúp chúng em mở mang tầm hiểu biết, hoàn thiện bản thân và chuẩn bị hành trang vững vàng cho tương lai.Chúng em xin hứa sẽ tiếp tục học tập, rèn luyện, phấn đấu để xứng đáng với sự kỳ vọng của thầy. Nhóm 6 xin kính chúc thầy sức khỏe và thành công trên sự nghiệp giảng dạy.

Nhóm 6 xin chân thành cảm ơn!

Nguyễn Đình Quang - 20002155

Nguyễn Mạnh Trung - 20002171

Đào Ánh Dương - 20002112

Mục lục

Lời cảm ơn **i**

Danh sách hình vẽ **v**

Danh sách tên viết tắt **vii**

MỞ ĐẦU **1**

1 TỔNG QUAN VỀ ĐỀ TÀI **3**

1.1 Khái niệm Thị giác máy tính (Computer Vision) 3

1.2 Các trường hợp sử dụng của thị giác máy tính là gì? 4

1.3 Human Activity Recognition là gì? 4

1.4 HAR hoạt động như thế nào? 5

1.4.1 Mô tả bài toán thực tế 5

1.4.2 Hướng tiếp cận bài toán 5

2 CƠ SỞ LÝ THUYẾT DÙNG ĐỂ GIẢI QUYẾT BÀI TOÁN **6**

2.1 Mạng neural tích chập (Convolutional Neural Network - CNN) 6

2.1.1 Khái niệm 6

2.1.2 Feature trong CNN 6

2.1.3 Những lớp cơ bản của mạng CNN 7

Convolutional Layer 7

ReLU Layer 8

Pooling Layer 8

Fully Connected Layer 8

2.1.4 Kiến trúc của mạng CNN 9

2.2 Mạng hồi quy RNN 9

2.3 Mạng LSTM 11

Tổng quan về LSTM 12

Ý tưởng cốt lõi của LSTM 13

Thành phần của LSTM 14

Bên trong LSTM 14

Ưu điểm của LSTM 16

2.4 Các thư viện sử dụng 17

Thư viện Numpy 17

Thư viện OpenCV 18

Thư viện Tensorflow 18
Thư viện Sklearn 19
Thư viện Matplotlib 20
Thư viện Keras 21
2.5 Thư viện MediaPipe 22
2.6 Chỉ số đánh giá hiệu suất 23

3 HUMAN ACTIVITY RECOGNITION 26
3.1 Sử dụng công cụ MediaPipe và mạng LSTM 26
3.1.1 Tiền xử lý dữ liệu 26
Thu thập dữ liệu 27
Chuẩn hóa dữ liệu 29
3.2 Xây dựng mô hình 30
3.2.1 Tiêu chí ảnh hưởng đến mô hình 31
3.2.2 Tiêu chí đánh giá mô hình 32
Kết quả huất luyện mô hình 33
3.2.3 Thử nghiệm với thời gian thực webcam 34
3.3 Xây dựng mô hình sử dụngConvLSTM2D layer và mô hình LRCN . . . 37
3.3.1 Tiền xử lý dữ liệu 37
Thu thập dữ liệu 37
Tăng cường dữ liệu 38
Chuẩn hóa dữ liệu 38
3.3.2 Xây dựng mô hình sử dụng ConvLSTM2D layer 40
Kiến trúc mô hình 41
3.3.3 Huấn luyện mô hình 44
Đánh giá mô hình 45
3.3.4 Xây dựng mô hình LRCN 46
Kiến trúc mô hình 48
Đánh giá mô hình 51
3.3.5 Thử nghiệm với video download từ youtube 53

4 Demo sản phẩm và đánh giá các phương pháp 55
4.1 DEMO SẢN PHẨM 55
4.1.1 Sử dụng công cụ MediaPipe và xây dựng mạng LSTM 55
4.1.2 Mô hình ConvLSTM 57
4.1.3 Mô hình LRCN 64
4.2 Đánh giá các phương pháp 72
4.2.1 Cách tiếp cận sử dụng ConvLSTM2D 73
4.2.2 Cách tiếp cận sử dụng LRCN 73

4.3 Hướng phát triển trong tương lai 74

KẾT LUẬN **75**

Tài liệu tham khảo **76**

Code **77**

Danh sách hình vẽ

1.1 Computer Vision. 3
1.2 Flow Chart. 5

2.1 Toàn bộ luồng CNN để xử lý hình ảnh đầu vào và phân loại các đối tượng dựa trên giá trị. 6
2.2 Mô hình cấu trúc của mạng CNN. 7
2.3 Convolutional Layer. 8
2.4 Một đoạn RNN. 10
2.5 Mạng RNN. 11
2.6 Mô hình LSTM. 12
2.7 Mạng RNN chuẩn. 12
2.8 Mạng LSTM. 13
2.9 Mạng LSTM line. 13
2.10 Cổng Gate. 14
2.11 Forget Gate. 15
2.12 Input Gate. 15
2.13 Cell Gate. 15
2.14 Output Gate. 16
2.15 Confusion matrix. 24
2.16 Accuracy. 24
2.17 Precision. 25
2.18 Recall. 25
2.19 F1-Score. 25

3.1 Luồng bài toán 26
3.2 MediaPipe Pose 27
3.3 Model Accuracy và Model Loss. 33
3.4 Đánh giá hiệu suất mô hình. 34
3.5 Confusion Matrix. 34
3.6 Visualize the Data with its Labels. 37
3.7 Visualize the numbers. 38
3.8 ConvLSTM2D layer 40
3.9 Model ConvLSTM2D. 44
3.10 Model Accuracy và Model Loss. 45
3.11 Đánh giá hiệu suất mô hình. 45
3.12 Confusion Matrix. 46

3.13 LRCN model 46
3.14 Kiến trúc model LRCN. 50
3.15 Model Accuracy và Model Loss. 51
3.16 Đánh giá hiệu suất mô hình. 52
3.17 Confusion Matrix. 52

4.1 Swing output 55
4.2 Hand Swing output 55
4.3 Love output 56
4.4 Clap output 56
4.5 Doze output 57
4.6 Billards output 57
4.7 Soccer Juggling output 58
4.8 Swing output 59
4.9 HorseRace output 60
4.10 PlayingPiano output 61
4.11 PushUps output 62
4.12 Skijet output 63
4.13 Drumming output 64
4.14 Billards output 65
4.15 Soccer Juggling output 66
4.16 Swing output 67
4.17 HorseRace output 68
4.18 PlayingPiano output 69
4.19 PushUps output 70
4.20 Skijet output 71
4.21 Drumming output 72

Danh sách tên viết tắt

CNN	**C**onvolutional **N**eural **N**etwork
I/O	**I**nput **O**utput
ROM	**R**ead **O**nly **M**emory
RAM	**R**andom **A**ccess **M**emory
CPU	**C**entral **P**rocessing **U**nit

MỞ ĐẦU

Computer vision đã và đang đóng góp một vai trò quan trọng trong tất cả các lĩnh vực trên toàn thế giới, mỗi năm có hàng ngàn báo cáo khoa học về đề tài này. Các công ty công nghệ hàng đầu như Google, Facebook, Amazon và Microsoft đang đầu tư mạnh mẽ vào học máy để phát triển các sản phẩm và dịch vụ mới. Ngoài ra, học máy cũng đang được sử dụng rộng rãi trong các lĩnh vực như y tế, tài chính, sản xuất và nhiều lĩnh vực khác để cải thiện hiệu suất và tối ưu hóa quy trình.

Những năm gần đây, khi mà khả năng tính toán của các máy tính được nâng lên một tầm cao mới và lượng dữ liệu khổng lồ được thu thập, Machine Learning đã tiến thêm một bước dài và Deep Learning ra đời. Deep Learning đã giúp máy tính thực thi những công việc phức tạp hơn như: phân loại cả ngàn vật thể trong các bức ảnh, tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người...

Nhận dạng hoạt động của con người (Human Activity Recognition – HAR) là một lĩnh vực nghiên cứu thú vị về thị giác máy tính và tương tác giữa người với máy.

HAR mang lại nhiều ứng dụng có ích cho cuộc sống con người. Theo dõi sức khỏe có thể được thực hiện thông qua các thiết bị đeo theo dõi hoạt động thể chất, nhịp tim và chất lượng giấc ngủ. Trong nhà thông minh, các giải pháp dựa trên HAR cho phép tiết kiệm năng lượng và tạo sự thoải mái cho cá nhân bằng cách phát hiện khi một người ra vào phòng và điều chỉnh ánh sáng hoặc nhiệt độ. Các thiết bị an toàn cá nhân có thể tự động cảnh báo các dịch vụ khẩn cấp hoặc một số liên lạc được chỉ định. Và đó chỉ là phần nổi của tảng băng chìm.

Với nhiều bộ dữ liệu có sẵn được chia sẻ công khai, việc tìm kiếm dữ liệu phục vụ mục đích nghiên cứu và phát triển HAR là rất đơn giản. Trong bài này này, Nhóm 6 sẽ cùng bạn tìm hiểu thêm về công nghệ tiên tiến nhất hiện nay của HAR, cùng với các phương pháp học sâu và bộ dữ liệu mở hỗ trợ tác vụ.

Nội dung chính của báo cáo có 5 chương:

Chương 1: Tổng quan về đề tài

Chương 2: Cơ sở lý thuyết dùng để giải quyết bài toán

Chương 3: Human Activity Recognition

Chương 4: Demo sản phẩm và đánh giá các phương pháp

Chương 5: Kết luận

Chương 1 TỔNG QUAN VỀ ĐỀ TÀI

1.1 Khái niệm Thị giác máy tính (Computer Vision)

hị giác máy tính là một công nghệ mà máy sử dụng để tự động nhận biết và mô tả hình ảnh một cách chính xác và hiệu quả. Ngày nay, các hệ thống máy tính có quyền truy cập vào khối lượng lớn hình ảnh và dữ liệu video bắt nguồn từ hoặc được tạo bằng điện thoại thông minh, camera giao thông, hệ thống bảo mật và các thiết bị khác. Ứng dụng thị giác máy tính sử dụng trí tuệ nhân tạo và máy học (AI/ML) để xử lý những dữ liệu này một cách chính xác nhằm xác định đối tượng và nhận diện khuôn mặt, cũng như phân loại, đề xuất, giám sát và phát hiện.

Hình 1.1: Computer Vision.

Tuy rằng công nghệ xử lý thông tin hình ảnh đã xuất hiện từ lâu nhưng phần lớn quy trình vẫn đòi hỏi sự can thiệp của con người, tốn nhiều thời giờ và dễ bị lỗi. Ví dụ: việc triển khai hệ thống nhận diện khuôn mặt trước đây yêu cầu nhà phát triển phải gắn thẻ thủ công hàng ngàn hình ảnh bằng các điểm dữ liệu chính, chẳng hạn như chiều rộng sống mũi và khoảng cách giữa hai mắt. Tự động hóa các tác vụ này đòi hỏi sức mạnh điện toán rộng lớn vì dữ liệu hình ảnh không có cấu trúc và phức tạp để máy tính có thể sắp xếp. Do đó, ứng dụng thị giác tốn kém và hầu hết các tổ chức không thể tiếp cận.

Ngày nay, tiến bộ trong lĩnh vực này kết hợp với sự tăng cường đáng kể của sức mạnh điện toán đã cải thiện cả quy mô và độ chính xác của quy trình xử lý dữ liệu hình ảnh.

Các hệ thống thị giác máy tính được hỗ trợ bởi tài nguyên điện toán đám mây hiện giờ trở nên dễ tiếp cận với tất cả mọi người. Bất kỳ tổ chức nào cũng có thể sử dụng công nghệ này để xác minh danh tính, kiểm duyệt nội dung, phân tích video phát trực tuyến, phát hiện lỗi và nhiều tính năng khác.

1.2 Các trường hợp sử dụng của thị giác máy tính là gì?

Nhiều ứng dụng thị giác máy tính được sử dụng trong lĩnh vực giải trí, kinh doanh, chăm sóc sức khỏe, giao thông vận tải và cuộc sống hàng ngày. Hãy cùng xem xét một số trường hợp sử dụng dưới đây:

- Bảo mật và an toàn
- Hiệu quả hoạt động
- Chăm sóc sức khỏe
- Phương tiện tự hành

1.3 Human Activity Recognition là gì?

Human Activity Recognition (HAR) là một nhánh của ngành khoa học máy tính, với mục tiêu là tạo ra các hệ thống và kỹ thuật có khả năng tự động nhận dạng và phân loại các hành động của con người dựa trên dữ liệu cảm biến. HAR sử dụng các cảm biến để giải thích các cử chỉ hoặc chuyển động của cơ thể con người và xác định hoạt động hoặc chuyển động của con người.

Các hệ thống HAR thường được sử dụng trong nhiều ứng dụng khác nhau, bao gồm chăm sóc sức khỏe, vận động, an ninh, biểu diễn thể thao, v.v.

Trong khi xây dựng mô hình, mục tiêu của hệ thống HAR là dự báo nhãn hành động của một người trong hình ảnh hoặc video, thường được thực hiện thông qua nhận dạng hoạt động dựa trên video và nhận dạng hoạt động dựa trên hình ảnh.

1.4 HAR hoạt động như thế nào?

Hình 1.2: Flow Chart.

1.4.1 Mô tả bài toán thực tế

- Sử dụng mạng CNN và LSTM để huấn luyện một mô hình có thể nhận biết hành động dựa vào webcam hoặc video
- Đưa hành động vào webcam hoặc video chứa hành hành động vào model để dự đoán

1.4.2 Hướng tiếp cận bài toán

- Sử dụng thư viện MediaPipe và mạng LSTM với đầu vào là hành động trước webcam và đầu ra là phân loại hành động đó
- Sử dụng lớp layer ConvLSTM
- Xây dựng mô hình LRCN

Chương 2 CƠ SỞ LÝ THUYẾT DÙNG ĐỂ GIẢI QUYẾT BÀI TOÁN

2.1 Mạng neural tích chập (Convolutional Neural Network - CNN)

2.1.1 Khái niệm

Mạng neural tích chập (Convolutional Neural Network - CNN) là một loại mạng neural sử dụng trong lĩnh vực xử lý ảnh và nhận dạng. Nó sử dụng các lớp tích chập để trích xuất đặc trưng từ ảnh đầu vào và áp dụng các lớp kết nối đầy đủ để phân loại ảnh.

Hình 2.1: Toàn bộ luồng CNN để xử lý hình ảnh đầu vào và phân loại các đối tượng dựa trên giá trị.

Về kỹ thuật, mô hình CNN để training và kiểm tra, mỗi hình ảnh đầu vào sẽ chuyển nó qua 1 loạt các lớp tích chập với các bộ lọc (Kernel), tổng hợp lại các lớp được kết nối đầy đủ (Full Connected) và áp dụng hàm Softmax để phân loại đối tượng có giá trị xác suất giữa 0 và 1. Hình dưới đây là toàn bộ luồng CNN để xử lý hình ảnh đầu vào và phân loại các đối tượng dựa trên giá trị.

2.1.2 Feature trong CNN

Các feature (đặc trưng) được sử dụng để trích xuất thông tin từ ảnh đầu vào. Các feature này được học tự động từ dữ liệu và được sử dụng để giảm thiểu số lượng thông tin cần xử lý trong quá trình huấn luyện mạng.

Các feature này có thể là các đường cạnh, các vùng sáng tối, các đường cong và các đối tượng phức tạp hơn.

Hình 2.2: Mô hình cấu trúc của mạng CNN.

Các feature này được trích xuất thông qua các bộ lọc tích chập và được kết hợp lại để tạo thành các feature maps, đóng vai trò quan trọng trong việc phân loại ảnh. Các feature maps này được đưa vào các lớp kết nối đầy đủ để phân loại ảnh đầu vào.

2.1.3 Những lớp cơ bản của mạng CNN

Convolutional Layer

Phần quan trọng nhất của toàn mạng CNN, các yếu tố quan trọng trong lớp Convolutional là: padding, stride, feature map và filter map.

- Mạng CNN sử dụng filter để áp dụng vào các vùng của ma trận hình ảnh. Các filter map là các ma trận 3 chiều, bên trong đó là những tham số và chúng được gọi là parameters.
- Stride: dịch chuyển filter map theo từng pixel dựa vào các giá trị từ trái qua phải.
- Padding: Giá trị viền xung quanh của ma trận hình ảnh sẽ được gán các giá trị 0 để có thể tiến hành nhân tích chập ma trận mà không làm giảm kích thước ma trận ảnh ban đầu.
- Feature map: Biểu diễn kết quả sau mỗi lần feature map quét qua ma trận ảnh đầu vào. Sau mỗi lần quét thì lớp Convolutional sẽ tiến hành tính toán.

Hình 2.3: Convolutional Layer.

ReLU Layer

Lớp ReLU là hàm kích hoạt trong mạng CNN, được coi là activation function. Nó có tác dụng mô phỏng những nơ ron có tỷ lệ truyền xung qua axon và hỗ trợ tính toán nhanh hơn.

Trong quá trình dùng hàm ReLU, chú ý đến việc tùy chỉnh learning rate và dead unit. Những lướp ReLU được dùng sau khi filter map được tính và áp dụng ReLU lên các giá trị của filter map.

Pooling Layer

Khi ma trận ảnh đầu vào có kích thước quá lớn, các lớp Pooling layer sẽ được đặt vào giữa những lớp Convolutional để làm giảm những parameters. Hai loại Pooling được sửu dụng phổ biến: Max pooling và Average.

Fully Connected Layer

Lớp có nhiệm vụ đưa ra kết quả sau khi 2 lớp Convolutional và Pooling đã nhận được ảnh truyền, khi này, ta sẽ thu được một model đọc được thông tin của ảnh.

2.1.4 Kiến trúc của mạng CNN

Mạng CNN là tập hợp những Convolutional layer xếp chồng lên nhau, đồng thời mạng sử dụng những hàm như ReLU và Tanh để kích hoạt các trọng số trong các node. Các lớp này sau khi qua các hàm activation sẽ có trọng số trong những node và có thể tạo ra những thông tin trừu tượng hơn đến với các lớp kế tiếp trong mạng.

Cấu trúc cơ bản của một mô hình CNN:

- Local receptive: Lớp này sử dụng để tách lọc dữ liệu, thông tin hình ảnh để từ đó có thể lựa chọn các vùng có giá trị sử dụng hiệu quả cao nhất.
- Shared weights field: Lớp này hỗ trợ làm giảm các tham số đến mức tối thiểu trong mạng CNN. Trong từng lớp convolution sẽ chứa các feature map riêng và từng feature thì sẽ có khả năng phát hiện một vài feature trong hình ảnh.
- Pooling layer: Lớp cuối cùng và sử dụng để làm đơn giản các thông tin output. Sau khi tính toán xong và quét qua các layer trong mạng thì pooling layer sẽ được dùng để lược bỏ các thông tin không hữu ích.

2.2 Mạng hồi quy RNN

Con người không bắt đầu suy nghĩ của họ từ đầu tại tất cả các thời điểm. Cũng như bạn đang đọc bài viết này, bạn hiểu mỗi chữ ở đây dựa vào từ bạn đã hiểu các chữ trước đó chứ không phải là đọc tới đâu ném hết đi tới đó, rồi lại bắt đầu suy nghĩ lại từ đầu tới chữ bạn đang đọc. Tức là tư duy đã có một bộ nhớ để lưu lại những gì diễn ra trước đó.

Tuy nhiên các mô hình mạng nơ-ron truyền thống thì không thể làm được việc đó, đó có thể coi là một khuyết điểm chính của mạng nơ-ron truyền thống. Ví dụ, bạn muốn phân loại các bối cảnh xảy ra ở tất cả các thời điểm trong một bộ phim, thì đúng là không rõ làm thế nào để có thể hiểu được một tình huống trong phim mà lại phụ thuộc vào các tình huống trước đó nếusử dụng các mạng nơ-ron truyền thống.

Mạng nơ-ron hồi quy (Recurrent Neural Network) sinh ra để giải quyết vấn đề đó. Mạng này chứa các vòng lặp bên trong cho phép thông tin có thể lưu lại được. Hình vẽ trên mô tả một đoạn của mạng nơ-ron hồi quy A với đầu vào là x_t và đầu ra là h_t.Một

Hình 2.4: Một đoạn RNN.

vòng lặp cho phép thông tin có thể được truyền từ bước này qua bước này qua bước khác của mạng nơ-ron.

Các vòng lặp này khiến cho mạng nơ-ron hồi quy trông có vẻ khó hiểu. Tuy nhiên, nếu bạn để ý một chút thì nó không khác mấy so với các mạng nơ-ron thuần. Một mạng nơ-ron hồi quy có thể được coi là nhiều bản sao chép của cùng một mạng, trong đó mỗi đầu ra của mạng này là đầu vào của một mạng sao chép khác. Nói thì hơi khó hiểu, nhưng

bạn hãy xem hình mô tả sau: Chuỗi lặp lại các mạng này chính là phân giải của mạng

Hình 2.5: Mạng RNN.

nơ-ron hồi quy, các vòng lặp khiến chúng tạo thành một chuỗi danh sách các mạng sao chép nhau. Bạn có thấy nó khác gì một mạng nơ-ron thuần không? Không khác gì phải không? Các nút của mạng vẫn nhận đầu vào và có đầu ra hệt như mạng nơ-ron thuần.

Trong vài năm gần đây, việc ứng dụng RNN đã đưa ra được nhiều kết quả không thể tin nổi trong nhiều lĩnh vực: nhận dạng giọng nói, mô hình hóa ngôn ngữ, dịch máy, mô tả ảnh,... Danh sách vẫn còn đang được mở rộng tiếp. Anh Andrej Karpathy đã đề cập tới một số kêt quả mà RNN mang lại tại bài viết này, nên tôi sẽ không bàn luận thêm nữa. Nhưng tôi vẫn muốn thốt lên rằng chúng thật là quá tuyệt vời.

Đằng sau sự thành công này chính là sự đóng góp của LSTM. LSTM là một dạng đặc biệt của mạng nơ-ron hồi quy, với nhiều bài toán thì nó tốt hơn mạng hồi quy thuần. Hầu hết các kết quả thú vị thu được từ mạng RNN là được sử dụng với LSTM. Trong bài viết này, ta sẽ cùng khám phá xem mạng LSTM là cái gì nhé.

2.3 Mạng LSTM

Mạng nơ-ron LSTM(Long Short-Term Memory là một loại mạng nơ ron hồi quy RNN được thiết kế đặc biệt để giải quyết vấn đề biến mất đường hồi quy (vanishing grandient) trong việc xử lý chuỗi liên tục . Biến mất đường hồi quy là hiện tượng khi gradient trong quá trình lan truyền ngược giảm đi rất nhanh , gây khó khăn trong việc học các phụ thuộc xa trong dữ liệu chuỗi . LSTM đã được giới thiệu vào năm 1997 bởi Sep Hochreiter và Jurgen Schmidhuber và từ đó trở thành một trong những mô hình RNN phổ biến nhất và hiệu quả nhất .

Hình 2.6: Mô hình LSTM.

Tổng quan về LSTM

LSTM được thiết kế để tránh được vấn đề phụ thuộc xa (long-term dependency). Việc nhớ thông tin trong suốt thời gian dài là đặc tính mặc định của chúng, chứ ta không cần phải huấn luyện nó để có thể nhớ được. Tức là ngay nội tại của nó đã có thể ghi nhớ được mà không cần bất kì can thiệp nào.

Mọi mạng hồi quy đều có dạng là một chuỗi các mô-đun lặp đi lặp lại của mạng nơ-ron. Với mạng RNN chuẩn, các mô-dun này có cấu trúc rất đơn giản, thường là một tầng tanh LSTM cũng có kiến trúc dạng chuỗi như vậy, nhưng các mô-đun trong nó có cấu

Hình 2.7: Mạng RNN chuẩn.

trúc khác với mạng RNN chuẩn. Thay vì chỉ có một tầng mạng nơ-ron, chúng có tới 4 tầng tương tác với nhau một cách rất đặc biệt.

Hình 2.8: Mạng LSTM.

Ý tưởng cốt lõi của LSTM

Chìa khóa của LSTM là trạng thái tế bào (cell state) - chính đường chạy thông ngang phía trên của sơ đồ hình vẽ.

Trạng thái tế bào là một dạng giống như băng truyền. Nó chạy xuyên suốt tất cả các mắt xích (các nút mạng) và chỉ tương tác tuyến tính đôi chút. Vì vậy mà các thông tin có thể dễ dàng truyền đi thông suốt mà không sợ bị thay đổi.

Hình 2.9: Mạng LSTM line.

LSTM có khả năng bỏ đi hoặc thêm vào các thông tin cần thiết cho trạng thái tế báo, chúng được điều chỉnh cẩn thận bởi các nhóm được gọi là cổng (gate).

Các cổng là nơi sàng lọc thông tin đi qua nó, chúng được kết hợp bởi một tầng mạng sigmoid và một phép nhân.

Tầng sigmoid sẽ cho đầu ra là một số trong khoản [0,1], mô tả có bao nhiêu thông tin có thể được thông qua. Khi đầu ra là 0 thì có nghĩa là không cho thông tin nào qua cả, còn khi là 1 thì có nghĩa là cho tất cả các thông tin đi qua nó. Một LSTM gồm có 3 cổng như vậy để duy trì và điều hành trạng thái của tế bào.

Hình 2.10: Cổng Gate.

Thành phần của LSTM

Bên trong LSTM

- Bước đầu tiên của LSTM là quyết định xem thông tin nào cần bỏ đi từ trạng thái tế bào. Quyết định này được đưa ra bởi tầng sigmoid - gọi là "tầng cổng quên" (forget gate layer) . Nó sẽ lấy đầu vào là h_{t-1} và x_t rồi đưa kết quả là một trong khoảng [0,1] cho mỗi trạng thái tế bào C_{t-1} . Đầu ra là 1 thể hiện rằng nó giữ toàn bộ thông tin lại . còn 0 chỉ là toàn bộ thông tin sẽ bị bỏ đi

- Bước tiếp theo là quyết định xem thông tin mới nào ta sẽ lưu vào trạng thái tế bào. Việc này gồm 2 phần. Đầu tiên là sử dụng một tầng sigmoid được gọi là "tầng cổng vào" (input gate layer) để quyết định giá trị nào ta sẽ cập nhập. Tiếp theo là một

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Hình 2.11: Forget Gate.

tầng tanh để tạo ra một vec-to cho giá trị mới C_t nhằm thêm vào cho trạng thái. Trong bước tiếp theo, ta sẽ kết hợp 2 giá trị đó lại để tạo ra một cập nhập cho trạng thái.

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Hình 2.12: Input Gate.

- Sau đó , LSTM cập nhật trạng thái của Cell State C_{t-1} thành trạng thái mới C_t bằng cách kết hợp thông tin mới và thông tin cũ . Ta sẽ nhân trạng thái cũ với f_t để bỏ đi những thông tin ta quyết định quên lúc trước. Sau đó cộng thêm $i_t * C_t$ Trạng thái mơi thu được này phụ thuộc vào việc ta quyết định cập nhập mỗi giá trị trạng thái ra sao.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Hình 2.13: Cell Gate.

- Cuối cùng ,LSTM tính toán giá trị của cổng đầu ra và sử dụng nó để tính toán đầu ra của đơn vị LSTM hiện tại và truyền trạng thái ẩn tới đơn vị LSTM tiếp theo

trong chuỗi .Đầu tiên, ta chạy một tầng sigmoid để quyết định phần nào của trạng thái tế bào ta muốn xuất ra. Sau đó, ta đưa nó trạng thái tế bảo qua một hàm tanh để co giá trị nó về khoảng [-1,1],và nhân nó với đầu ra của cổng sigmoid để được giá trị đầu ra ta mong muốn.

$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Hình 2.14: Output Gate.

Ưu điểm của LSTM

LSTM là một bước lớn trong việc sử dụng RNN. Ý tưởng của nó giúp cho tất cả các bước của RNN có thể truy vấn được thông tin từ một tập thông tin lớn hơn. Ví dụ, nếu bạn sử dụng RNN để tạo mô tả cho một bức ảnh, nó có thể lấy một phần ảnh để dự đoán mô tả từ tất cả các từ đầu vào. Bằng chứng là Xu, et al. (2015) đã thực hiện được chính xác việc này. Hiện nay cũng đã có nhiều kết qua thực sự rất thú vị được chú ý và dường như có nhiều kết quả hơn chúng ta vẫn biết. LSTM có những ưu điểm quan trọng áp dụng vào bài toán nhận diện hành động con người và xử lý chuỗi thời gian

- LSTM có khả năng lưu trữ thông tin lâu dài và xử lý các chuỗi dữ liệu dài và phức tạp hơn so với các kiến trúc mạng nơ-ron tái phát khác như mạng Elman hay mạng Jordan.
- LSTM được sử dụng để giải quyết các vấn đề trong xử lý ngôn ngữ tự nhiên như: phân loại văn bản, dịch máy, sinh văn bản tự động, tổng hợp giọng nói, và xử lý dữ liệu ngôn ngữ tự nhiên khác.
- LSTM có thể được sử dụng để giải quyết các vấn đề trong xử lý âm thanh và hình ảnh như: nhận dạng giọng nói dạng (voice recognition), phân loại hình ảnh, nhận diện khuôn mặt, phát hiện đối tượng và các ứng dụng khác trong lĩnh vực thị giác máy tính.

2.4 Các thư viện sử dụng

Thư viện Numpy

Thư viện NumPy (Numerical Python) là một thư viện Python phổ biến được sử dụng để làm việc với mảng đa chiều và các phép toán số học trên chúng. NumPy cung cấp một cấu trúc dữ liệu mảng mạnh mẽ và các hàm tiện ích để thực hiện các phép toán số học, xử lý mảng và thao tác trên dữ liệu số. Dưới đây là một số khái niệm và tính năng chính của NumPy:

- Mảng NumPy (ndarray): Mảng NumPy là cấu trúc dữ liệu chính trong NumPy, cho phép lưu trữ và xử lý các mảng đa chiều. Mảng NumPy có kích thước cố định và các phần tử trong mảng cùng kiểu dữ liệu.
- Các phép toán số học: NumPy cung cấp các hàm và toán tử cho phép thực hiện các phép toán số học trên mảng như cộng, trừ, nhân, chia, lũy thừa, căn bậc hai, logarit...
- Truy cập phần tử: Bạn có thể truy cập và thay đổi giá trị các phần tử trong mảng NumPy bằng cách sử dụng chỉ mục và cắt mảng (slicing).
- Hàm toán học và thống kê: NumPy cung cấp nhiều hàm toán học và thống kê tiện ích như min, max, mean, sum, std,... để thao tác trên mảng.
- Hàm điều kiện: NumPy cung cấp các hàm và phương thức cho phép kiểm tra và lựa chọn các phần tử trong mảng dựa trên các điều kiện như np.where, np.logical_and, np.logical_or,...
- Thao tác trên mảng: NumPy cung cấp các hàm và phương thức để thực hiện các phép biến đổi mảng như reshape, transpose, flatten, concatenate,...
- Tích hợp C/C++: NumPy cho phép tích hợp mã C/C++ vào mã Python thông qua giao diện C API của nó, giúp tăng tốc độ xử lý các phép toán số học.
- NumPy được sử dụng rộng rãi trong nhiều lĩnh vực như khoa học dữ liệu, máy học, tính toán khoa học, xử lý ảnh và âm thanh, mô phỏng vật lý,... Thư viện này là một phần quan trọng của hệ sinh thái Python cho tính toán số và xử lý dữ liệu.

Thư viện OpenCV

Thư viện cv2 (OpenCV) là một thư viện mã nguồn mở được sử dụng rộng rãi trong xử lý ảnh và thị giác máy tính trong ngôn ngữ lập trình Python. OpenCV cung cấp nhiều chức năng và công cụ mạnh mẽ để xử lý, phân tích và trực quan hóa ảnh và video. Dưới đây là một số chức năng chính của thư viện cv2:

- Đọc và ghi ảnh và video: OpenCV cung cấp các hàm để đọc và ghi các tệp tin ảnh và video từ các nguồn khác nhau, bao gồm ổ đĩa, camera,...
- Xử lý ảnh: OpenCV cung cấp nhiều chức năng để xử lý ảnh, bao gồm chuyển đổi màu sắc, cắt, xoay, thu phóng, lật, v.v. Bạn cũng có thể áp dụng các bộ lọc, làm mờ, lọc nhiễu, phát hiện cạnh và nhiều phép biến đổi khác cho ảnh.
- Xử lý video: OpenCV hỗ trợ xử lý video, bao gồm khung hình theo thời gian, trích xuất khung hình, ghi lại video và áp dụng các hiệu ứng video.
- Xử lý thị giác máy tính: OpenCV cung cấp các chức năng và thuật toán để phân tích và nhận dạng các đối tượng trong ảnh, bao gồm nhận dạng khuôn mặt, phát hiện vật thể, trích xuất đặc trưng, theo dõi đối tượng,...
- Xử lý điểm ảnh: OpenCV cho phép bạn truy cập và chỉnh sửa các điểm ảnh trực tiếp, bao gồm xử lý pixel, tạo hiệu ứng đồ họa và thực hiện các phép tính điểm ảnh phức tạp.
- Hiển thị ảnh: OpenCV cung cấp các chức năng để hiển thị ảnh và video trực tiếp trên màn hình, tạo cửa sổ đồ họa và tương tác với ảnh.

Thư viện cv2 có tính ổn định, tốc độ xử lý nhanh và rất phổ biến trong cộng đồng xử lý ảnh và thị giác máy tính.

Thư viện Tensorflow

Thư viện TensorFlow là một thư viện mã nguồn mở phát triển bởi Google AI và được sử dụng rộng rãi trong lĩnh vực học máy và trí tuệ nhân tạo. TensorFlow cung cấp một cấu trúc dữ liệu gọi là "đồ thị tính toán" (computational graph) để biểu diễn và thực thi

các phép tính toán số trên dữ liệu. Đồ thị tính toán trong TensorFlow bao gồm các nút (nodes) đại diện cho các phép tính toán và các cung (edges) đại diện cho luồng dữ liệu giữa các nút. Dưới đây là một số khái niệm và tính năng chính của TensorFlow:

- Đồ thị tính toán (Computational graph): TensorFlow sử dụng đồ thị tính toán để biểu diễn và thực thi các phép tính toán. Đồ thị tính toán giúp tối ưu hóa và tận dụng được tính song song của phần cứng để thực hiện các phép tính nhanh chóng.
- Tensors: TensorFlow sử dụng cấu trúc dữ liệu tensor để lưu trữ và xử lý dữ liệu. Tensor có thể là một vector, ma trận, hay một mảng đa chiều với các phần tử cùng kiểu dữ liệu.
- Các lớp và phép tính: TensorFlow cung cấp một loạt các lớp và phép tính để xây dựng các mô hình học máy. Các lớp bao gồm các lớp mạng nơ-ron, lớp tích chập, lớp tổng hợp, lớp tái tạo,... Phép tính bao gồm các phép tính toán như cộng, trừ, nhân, chia, lũy thừa,...
- Tối ưu hóa và tạo mô hình: TensorFlow cung cấp các tối ưu hóa để tối đa hóa hiệu suất và tăng tốc độ huấn luyện mô hình. Ngoài ra, TensorFlow cũng cung cấp khả năng tạo và lưu trữ các mô hình đã huấn luyện để sử dụng sau này.
- Tích hợp và mở rộng: TensorFlow cho phép tích hợp và mở rộng với các công cụ và thư viện khác như Keras, scikit-learn, OpenCV, v.v. Điều này giúp thực hiện các tác vụ phức tạp và kết hợp các công cụ và thuật toán khác nhau.
- Tính tương thích và di động: TensorFlow hỗ trợ nhiều phiên bản và cung cấp tính tương thích đa nền tảng, cho phép chạy trên các thiết bị di động, máy tính cá nhân, máy chủ, và cụm máy tính phân tán.

TensorFlow là một công cụ mạnh mẽ trong lĩnh vực học máy và trí tuệ nhân tạo, cho phép xây dựng và huấn luyện các mô hình phức tạp và giải quyết các bài toán thực tế.

Thư viện Sklearn

Scikit-learn, hay còn được gọi là sklearn, là một thư viện mã nguồn mở phổ biến trong ngôn ngữ lập trình Python được sử dụng cho Machine Learning và Data Mining.

Scikit-learn cung cấp nhiều công cụ và thuật toán tiện ích để xây dựng và đánh giá các mô hình học máy, phân tích dữ liệu và thực hiện các tác vụ tiền xử lý dữ liệu. Dưới đây là một số chức năng và tính năng chính của thư viện scikit-learn:

- Cung cấp các thuật toán học máy tiêu chuẩn: Scikit-learn cung cấp các thuật toán học máy phổ biến như hồi quy tuyến tính, hồi quy logistic, máy vector hỗ trợ (SVM), cây quyết định, Random Forest, K-means clustering,... Các thuật toán này được triển khai một cách hiệu quả và dễ sử dụng.
- Tích hợp các công cụ tiền xử lý dữ liệu: Scikit-learn cung cấp nhiều công cụ tiền xử lý dữ liệu như chuẩn hóa dữ liệu, mã hóa biến định danh, xử lý giá trị thiếu, trích xuất đặc trưng,... Điều này giúp chuẩn bị dữ liệu cho việc huấn luyện mô hình.
- Đánh giá và tối ưu hóa mô hình: Scikit-learn cung cấp các phép đo và công cụ để đánh giá hiệu suất của mô hình học máy, bao gồm chia dữ liệu thành tập huấn luyện và tập kiểm tra, cross-validation, tính toán độ chính xác, độ phân loại, mất mát,... Ngoài ra, scikit-learn cũng cung cấp các công cụ tối ưu hóa để điều chỉnh các siêu tham số của mô hình.
- Tích hợp với các thư viện khác: Scikit-learn tích hợp tốt với các thư viện khác trong hệ sinh thái của Python như NumPy, Pandas và Matplotlib, tạo điều kiện thuận lợi cho xử lý dữ liệu và trực quan hóa kết quả.

Scikit-learn là một công cụ mạnh mẽ và phổ biến trong lĩnh vực Machine Learning, đặc biệt là trong các tác vụ phân loại, hồi quy và gom cụm dữ liệu. Với tư cách là một thư viện mã nguồn mở, scikit-learn đang tiếp tục được phát triển và cung cấp cập nhật mới để phục vụ cộng đồng Machine Learning ngày càng lớn.

Thư viện Matplotlib

Matplotlib là một thư viện trong Python được sử dụng để tạo và hiển thị đồ thị, biểu đồ, hình ảnh và các loại visualizations khác. Nó cung cấp các công cụ mạnh mẽ để tạo ra các biểu đồ chất lượng cao, giúp bạn trực quan hóa dữ liệu một cách dễ dàng và linh hoạt. Dưới đây là một số thành phần chính trong thư viện matplotlib:

- pyplot: Giao diện API cơ bản của Matplotlib, cung cấp các hàm để tạo và tùy chỉnh đồ thị và biểu đồ.
- Figure: Đại diện cho một hình ảnh hoặc một tệp tin hình ảnh.
- Axes: Cung cấp các phương thức để tạo các đối tượng đồ thị như các trục, điểm, đường.
- Subplots: Cho phép tạo và quản lý nhiều đồ thị trong cùng một hình ảnh.
- Plot: Hàm để tạo các loại đồ thị và biểu đồ, bao gồm đồ thị đường (line plot), biểu đồ cột (bar plot), biểu đồ hộp (box plot), đồ thị điểm (scatter plot).
- Colorbar: Hiển thị thanh màu để giải thích giá trị của màu sắc trong đồ thị.
- Title, Label, Legend‘: Cung cấp các phương thức để thêm tiêu đề, nhãn và chú giải vào đồ thị.

Matplotlib cũng hỗ trợ nhiều kiểu đồ thị và biểu đồ, cho phép bạn tùy chỉnh màu sắc, kích thước, kiểu đường, điểm, các đánh dấu trục, v.v. Thư viện này rất linh hoạt và mạnh mẽ, cho phép bạn tạo ra các visualizations phức tạp và tùy chỉnh chúng theo ý muốn.

Thư viện Keras

Keras là một thư viện mã nguồn mở cho Python được sử dụng để xây dựng các mô hình học máy và mạng nơ-ron. Nó được thiết kế để làm cho việc xây dựng mô hình học máy trở nên dễ dàng và nhanh chóng hơn. Keras có nhiều tính năng hữu ích, bao gồm:

- Dễ dàng sử dụng: Keras được thiết kế để làm cho việc xây dựng mô hình học máy trở nên dễ dàng và trực quan hơn.
- Tích hợp với các thư viện tính toán số: Keras hỗ trợ các thư viện tính toán số như TensorFlow, Theano và CNTK, cho phép bạn chọn thư viện tính toán số phù hợp nhất cho dự án của mình.
- Hỗ trợ nhiều loại mô hình: Keras hỗ trợ nhiều loại mô hình, bao gồm mạng nơ-ron tiêu chuẩn, mạng nơ-ron tích chập và mạng nơ-ron tái tạo.

- Tích hợp với các công cụ tối ưu hóa: Keras cung cấp các công cụ tối ưu hóa để giúp bạn tìm kiếm các tham số tốt nhất cho mô hình của mình.
- Hỗ trợ các lớp và hàm kích hoạt tiêu chuẩn: Keras cung cấp các lớp và hàm kích hoạt tiêu chuẩn để giúp bạn xây dựng mô hình học máy nhanh chóng và dễ dàng.

Keras là một trong những thư viện phổ biến nhất cho việc xây dựng các mô hình học máy và mạng nơ-ron.

2.5 Thư viện MediaPipe

Đây là một công cụ do google thiết kế ra, nó cung cấp nhiều tính năng cho các bài toán AI/ML chạy trên nhiều nền tảng khác nhau MediaPipe là tập hợp của một loạt các giải pháp Machine Learning đa nền tảng, có thể can thiệp được và cực kỳ lightweight.

Một số ưu điểm có thể kể tới của giải pháp này bao gồm:

- Cung cấp một giải pháp inference nhanh chóng: Google khẳng định rằng bộ công cụ này có thể chạy ổn định trên hầu hết các cấu hình phần cứng thông dụng.
- Mã nguồn mở và miễn phí: Toàn bộ source code được công khai trên MediaPipe, người dùng hoàn toàn có thể sử dụng và tùy chỉnh trực tiếp để phù hợp với bài toán của mình.
- Dễ dàng cài đặt và triển khai: Việc cài đặt cực kỳ dễ dàng và tiện lợi, có thể triển khai trên nhiều nền tảng khác nhau như Mobile (Android/iOS), Desktop/Cloud, Web và IoT devices.

Hầu hết các bài toán nổi bật trong lĩnh vực Computer Vision - Thị giác máy tính, đều được Google cài đặt trong MediaPipe. Trong bài toán mình sẽ sử dụng Human Pose Estimation Mở rộng từ bài toán Hands Detection, Human Pose Estimation cung cấp một mô hình skeleton 3D cho cả cơ thể, với các khớp quan trọng được định nghĩa sẵn và được nối với nhau để tạo thành khung của người. Chiến thuật được đặt ra cho bài toán này tương tự như bài Hands Detection và Face Mesh. BlazeFace, một lần nữa, được sử dụng làm tư tưởng chính cho thuật toán xử lý bài này.

Các ứng dụng của mediapipe

- Phát hiện khuôn mặt
- Tìm lưới khuôn mặt: dùng cho các ứng dụng biến đổi khuôn mặt như Tiktok
- Iris: tìm khoảng cách từ đồng tử của mắt đến camera mà không cần Depth webcam
- Detect cử chỉ bàn tay
- Tìm hình dáng của cơ thể
- Object detection , Box tracking
- Theo dõi chuyển động của vật thể

2.6 Chỉ số đánh giá hiệu suất

Để đánh giá phân loại ảnh được chính xác, các mô hình trong phần trước sữ được chạy bằng cách sử dụng bộ dữ liệu dưới dạng hình ảnh, vì vậy cần có sự điều chỉnh để nâng cấp độ chính xác của chúng. Đối với mỗi mô hình 3 kết quả điển hình được hiển thị trong CNN là:

- Đồ thị độ chính xác (model ACC): ACC cho biết sự thay đổi độ chính xác của mô hình qua các vòng lặp/epochs trong quá trình huấn luyện.
- Đồ thị loss (model loss): Đồ thị loss biểu thị sự thay đổi của hàm loss của mô hình qua các vòng lặp hoặc epochs trong quá trình huấn luyện. Hàm loss tính toán sự sai khác giữa giá trị dự đoán của mô hình và giá trị thực tế.
- Confusion matrix: Về cơ bản, Confusion matrix thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một class, và được dự đoán là rơi vào một class.
  - TP (True Positive): Số lượng dự đoán chính xác.
  - TN (True Negative): Số lương dự đoán chính xác một cách gián tiếp. Là khi mô hình dự đoán đúng một người không bị bệnh tức là việc không chọn trường hợp bị bệnh là chính xác.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Hình 2.15: Confusion matrix.

- FP (False Positive - Type 1 Error): Số lượng các dự đoán sai lệch. Là khi mô hình dự đoán một người bị bệnh và người đó hoàn toàn khỏe mạnh.
- FN (False Negative - Type 2 Error): Số lượng các dự đoán sai lệch một cách gián tiếp. Là khi mô hình dự đoán một người không bị bệnh nhưng người đó bị bệnh, tức là việc không chọn trường hợp bị bệnh là sai.

$\Longrightarrow$ Từ 4 chỉ số TP, TN, FP, FN, ta có 4 chỉ số để đánh giá mức độ tin cậy của một mô hình:

- Accuracy: Được tính bằng cách chia tổng số dự đoán đúng cho tất cả các dự đoán.

$$\boldsymbol{ACC} = \frac{TP + TN}{TP + FP + TN + FN}$$

Hình 2.16: Accuracy.

- Precision: Trong tất cả các dự đoán Positive được đưa ra, bao nhiêu dự đoán là chính xác.
- Recall: Trong tất cả các trường hợp Positive, bao nhiêu trường hợp đã được dự đoán chính xác.

$$\textit{Precision} = \frac{TP}{TP + FP}$$

Hình 2.17: Precision.

$$\textit{Recall} = \frac{TP}{TP + FN}$$

Hình 2.18: Recall.

- F1-Score: Dùng để đánh giá mức độ tin cậy chung của mô hình

$$\textit{F - measure} = \frac{2*Recall*Precision}{Recall + Precision}$$

Hình 2.19: F1-Score.

Chương 3 HUMAN ACTIVITY RECOGNITION

Bài toán của chúng ta xây dựng đó là bài toán nhận dạng các hành động con người qua webcam hoặc video. Các hành động con người thì rất nhiều vì vậy trong bài viết này mình chỉ nhận dạng một vài hành động thường ngày qua con người. Thuật toán chúng tôi xử dụng trong bài này là CNN và mạng LSTM kết hợp với công cụ MediaPipe . Chúng tôi sẽ tiếp cận để xây dựng mô hình nhận biết theo 3 hướng :

- Sử dụng công cụ MediaPipe và xây dựng mạng LSTM
- Xây dựng mô hình ConvLSTM
- Xây dựng mô hình LRCN

3.1 Sử dụng công cụ MediaPipe và mạng LSTM

Hình 3.1: Luồng bài toán.

3.1.1 Tiền xử lý dữ liệu

Tiền xử lý dữ liệu là một bước rất quan trọng trong việc giải quyết bất kỳ vấn đề nào trong lĩnh vực học máy. Hầu hết các bộ dữ liệu được sử dụng trong các vấn đề liên quan

đến học máy cần được xử lý, làm sạch và biến đổi trước khi một thuật toán có thể được huấn luyện trên những bộ dữ liệu này.

Thu thập dữ liệu

Hình 3.2: MediaPipe Pose .

Trước khi chuẩn bị dữ liệu cho đầu vào tôi sẽ giới thiệu về một công cụ mạnh mẽ MediaPipe Pose Mediapipe Pose để biết cách vẽ các điểm keypoints và cách hoạt động của nó như thế nào . Mỗi vị trí trong 33 keypoints đó đều có 4 điểm tương ứng hoặc có thể nói là tọa độ tương ứng đó là: X,Y,Z,VISIBILITY. Phần này chính là input của mạng LSTM. Chúng ta sẽ đưa 4 điểm tương ứng này với từng điểm keypoints(có 33 điểm) sẽ tao thành array, sau đó chúng ta làm phẳng nó ra 33*4 sẽ là 132 điểm. Thêm 1 vấn đề nữa là nếu như trong 1 khung hình mà không thể hết được 33 điểm thì các vị trí không

thể vẽ được được gán bằng 0 hoặc xấp xỉ không nếu không nhìn rõ . Đầu tiên cần khởi tạo thư viện mediapipe .

```
    import cv2
import mediapipe as mp
import pandas as pd

cap = cv2.VideoCapture(0)

mpPose = mp.solutions.pose
pose = mpPose.Pose()
mpDraw = mp.solutions.drawing_utils
```

Bây giờ sử dụng thư viện để tìm các điểm , vẽ điểm nút và sau đó vẽ đường nối các điểm nút lại . Ghi lại các thông số data vào list . Sử dụng thư viện pandas để lưu dữ liệu thu được vao file txt

```
lm_list = []
label = "CLAP"
no_of_frames = 600

def make_landmark_timestep(results):
    print(results.pose_landmarks.landmark)
    c_lm = []
    for id, lm in enumerate(results.pose_landmarks.landmark):
        c_lm.append(lm.x)
        c_lm.append(lm.y)
        c_lm.append(lm.z)
        c_lm.append(lm.visibility)
    return c_lm

def draw_landmark_on_image(mpDraw, results, img):
    mpDraw.draw_landmarks(img, results.pose_landmarks, mpPose.POSE_CONNECTIONS)

    for id, lm in enumerate(results.pose_landmarks.landmark):
        h, w, c = img.shape
        print(id, lm)
        cx, cy = int(lm.x * w), int(lm.y * h)
        cv2.circle(img, (cx, cy), 10, (0, 0, 255), cv2.FILLED)
    return img


while len(lm_list) <= no_of_frames:
    ret, frame = cap.read()
    if ret:
        frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = pose.process(frameRGB)

        if results.pose_landmarks:
            lm = make_landmark_timestep(results)
            lm_list.append(lm)
            frame = draw_landmark_on_image(mpDraw, results, frame)

        cv2.imshow("image", frame)
        if cv2.waitKey(1) == ord('q'):
            break

df  = pd.DataFrame(lm_list)
df.to_csv(label + ".txt")
cap.release()
cv2.destroyAllWindows()
```

Chuẩn hóa dữ liệu

Trong học máy , chuẩn hóa dữ liệu là một bước rất quan trọng dường như là không thể thiếu trong việc xây dựng mô hình .Việc chuẩn hóa dữ liệu sẽ giúp cho mô hình đạt được kết quả tốt . Trong phần này nhóm sẽ trình bày phương pháp nhóm sử dụng để chuẩn hóa dữ liệu . Đầu tiên sử dụng thư viện pandas để đọc dữ liệu thu được

```
import numpy as np
import pandas as pd

from keras.layers import LSTM, Dense, Dropout, BatchNormalization
from keras.models import Sequential
from keras.utils import to_categorical

from sklearn.model_selection import train_test_split

bodyswing_df = pd.read_csv("../data/SWING.txt")
handswing_df = pd.read_csv("../data/HANDSWING.txt")
doze_df = pd.read_csv("../data/DOZE.txt")
love_df = pd.read_csv("../data/LOVE.txt")
clap_df = pd.read_csv("../data/CLAP.txt")
X = []
y = []
no_of_timesteps = 10

dataset_body = bodyswing_df.iloc[:,1:].values
n_sample = len(dataset_body)
for i in range(no_of_timesteps, n_sample):
    X.append(dataset_body[i-no_of_timesteps:i,:])
    y.append(0)

dataset_hand = handswing_df.iloc[:,1:].values
n_sample = len(dataset_hand)
for i in range(no_of_timesteps, n_sample):
    X.append(dataset_hand[i-no_of_timesteps:i,:])
    y.append(1)


dataset_doze = doze_df.iloc[:,1:].values
n_sample = len(dataset_doze)
for i in range(no_of_timesteps, n_sample):
    X.append(dataset_doze[i-no_of_timesteps:i,:])
    y.append(2)

dataset_love = love_df.iloc[:,1:].values
n_sample = len(dataset_love)
for i in range(no_of_timesteps, n_sample):
    X.append(dataset_love[i-no_of_timesteps:i,:])
    y.append(3)

dataset_clap = clap_df.iloc[:,1:].values
n_sample = len(dataset_clap)
for i in range(no_of_timesteps, n_sample):
    X.append(dataset_clap[i-no_of_timesteps:i,:])
    y.append(4)
```

Dữ liệu trong file CSV sẽ được load bằng thư viện pandas. Đối với dữ liệu đầu vào nhóm chia ra thành các lô gồm 10 dữ liệu với mỗi dữ liệu là 1 lần lấy được các điểm , mỗi lần sẽ lấy được 132 giá trị .

```
    X, y = np.array(X), np.array(y)

print(X.shape, y.shape)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# 132 = 4 * 33
# 10 = no_of_number


# one hot encoding

y_train_onehot = to_categorical(y_train)
y_test_onehot = to_categorical(y_test)
print(y_train_onehot.shape , y_test_onehot.shape)
n_features = X.shape[2]
```

Do mô hình là mô hình phân loại nhiều lớp nên các trạng thái thời tiết sẽ phải được one hot encode bằng hàm one_hot_encode() như trên.

Dữ liệu được chia thành tập train và tập test: Sau đó ta tạo 4 biến, gồm X_train, y_train và X_test, y_test. Với đối số truyền vào là giá trị X, y ta đã lấy từ dữ liệu bên trên, test_size trả về cho ta phần trăm dữ liệu được chia, ví dụ 0.2 tương ứng với dữ liệu được chia thành 20% giá trị là test, còn lại là dữ liệu train. random_state bằng một số tương ứng nào đó để đảm bảo mỗi lần ta chạy lại mô hình, giá trị phân tách ngẫu nhiên nhận được là giống nhau, bạn có thể cho số nào bất kỳ.

3.2 Xây dựng mô hình

```
    model = Sequential()
    model.add(LSTM(128, return_sequences=True, activation='relu', input_shape=(
        no_of_timesteps, n_features)))
    model.add(Dropout(0.2))
    model.add(LSTM(256, return_sequences=True, activation='relu'))
    model.add(Dropout(0.2))
    model.add(LSTM(256, return_sequences=False, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(5, activation='softmax'))

    model.compile(optimizer="adam", metrics = ['accuracy'], loss = "
        categorical_crossentropy")
    model.fit(X_train, y_train_onehot, epochs=20, batch_size=32,validation_data=(
        X_test, y_test_onehot))
    model.save("model_16.h5")
```

Đối với mô hình LSTM, dữ liệu sẽ được reshape lại thành dạng (samples, no_of_timesteps, features) với samples là số lượng mẫu, time steps là số lượng data liền trước và features là số lượng đặc trưng của mỗi mẫu . Mô hình được xây dựng bởi 3 lớp LSTM và 4 lớp kết nối đầy đủ. Thư viện tensorflow cung cấp đầy đủ chức năng cần thiết để xây dựng

mô hình như trên. Ở đây hai lớp LSTM đầu tiên sẽ đặt tham số 'return_sequences=True' để cho phép LSTM layer sẽ trả về chuỗi kết quả cho mỗi bước thời gian trong chuỗi đầu vào.

- Thêm một lớp LSTM với 128 đơn vị, trả về các chuỗi, sử dụng hàm kích hoạt ReLU và chỉ định hình dạng đầu vào.
- Sau mỗi lớp LSTM, Thêm một lớp dropout với tỷ lệ dropout là 0.2
- Thêm một lớp LSTM khác với 256 đơn vị, trả về các chuỗi, sử dụng hàm kích hoạt ReLU.
- Thêm một lớp LSTM khác với 256 đơn vị, không trả về chuỗi, sử dụng hàm kích hoạt ReLU.
- Thêm một lớp chuẩn hóa theo batch.
- Thêm các lớp kết nối đầy đủ với hàm kích hoạt ReLU.
- Cuối cùng là 1 lớp kết nối đầy đủ với hàm kích hoạt softmax

Sau khi xây dựng mô hình, mô hình sẽ được biên dịch với hàm mất mát là categorical_crossentropy và hàm tối ưu hóa là adam. Sau đó, mô hình sẽ được huấn luyện với 20 epochs và batch size là 32.

*** Sau khi đã xây dựng được kiến trúc của mô hình . Bước tiếp theo là thực hiện huấn luyện mô hình

3.2.1 Tiêu chí ảnh hưởng đến mô hình

Có nhiều yếu tố ảnh hưởng đến độ hiệu quả của một mô hình máy học. Dưới đây là một số tiêu chí quan trọng:

- Độ rõ ràng và độ phong phú của dữ liệu: Mô hình sẽ hoạt động tốt hơn khi dữ liệu huấn luyện rõ ràng, có độ phân loại cao và đủ đa dạng. Dữ liệu không rõ ràng, nhiễu và thiếu thông tin có thể làm giảm độ hiệu quả của mô hình.

- Số lượng và chất lượng dữ liệu: Một số lượng dữ liệu huấn luyện đủ lớn có thể giúp mô hình học được các mẫu và mối quan hệ phức tạp. Đồng thời, chất lượng dữ liệu cũng quan trọng, bao gồm độ chính xác, đồng nhất và đại diện cho phân phối dữ liệu thực tế.

- Chọn mô hình phù hợp: Một mô hình phải được lựa chọn dựa trên yêu cầu cụ thể của bài toán. Mô hình phải có khả năng học và biểu diễn các mẫu và quan hệ trong dữ liệu một cách hiệu quả.

- Quá trình huấn luyện: Thời gian huấn luyện, tỷ lệ học tập, thuật toán tối ưu hóa và các siêu tham số khác cũng có thể ảnh hưởng đến hiệu quả của mô hình. Quá trình huấn luyện phải được điều chỉnh và tối ưu để đạt được kết quả tốt.

- Tính diễn giải và khả năng áp dụng: Một mô hình có khả năng diễn giải tốt và có thể áp dụng vào thực tế sẽ có hiệu quả cao hơn. Sự diễn giải giúp người dùng hiểu rõ quyết định của mô hình và tin tưởng vào kết quả dự đoán.

3.2.2 Tiêu chí đánh giá mô hình

Khi đánh giá huấn luyện một mô hình máy học, có một số tiêu chí quan trọng cần xem xét. Dưới đây là một số tiêu chí phổ biến để đánh giá mô hình:

- Độ chính xác (Accuracy): Đây là tiêu chí cơ bản để đánh giá khả năng dự đoán chính xác của mô hình trên tập dữ liệu kiểm tra. Độ chính xác được tính bằng tỷ lệ giữa số lượng dự đoán đúng và tổng số mẫu trong tập kiểm tra.

- Mất mát (Loss): Mất mát đo lường mức độ sai khác giữa các dự đoán của mô hình và giá trị thực tế tương ứng. Một hàm mất mát được sử dụng để đo lường sự sai khác này và mục tiêu là tìm cách giảm mất mát trong quá trình huấn luyện.

- Đồng nhất (Consistency): Mô hình cần cho ra kết quả tương tự khi được huấn luyện lại trên các tập dữ liệu khác nhau hoặc khi được huấn luyện nhiều lần trên cùng một tập dữ liệu. Sự đồng nhất giúp đảm bảo tính ổn định và tin cậy của mô hình.

- Quá khớp (Overfitting) và thiếu khớp (Underfitting): Quá khớp xảy ra khi mô hình đã học nhớ quá mức từ dữ liệu huấn luyện, nhưng không thể tổng quát hóa tốt cho

các dữ liệu mới. Thiếu khớp xảy ra khi mô hình không học được đủ thông tin từ dữ liệu huấn luyện và không thể dự đoán chính xác trên dữ liệu mới. Đánh giá sự quá khớp và thiếu khớp là một tiêu chí quan trọng để đảm bảo mô hình có khả năng tổng quát hóa tốt.

- Thời gian huấn luyện (Training time): Thời gian huấn luyện là thời gian mô hình cần để học từ dữ liệu huấn luyện. Đánh giá thời gian huấn luyện là quan trọng đặc biệt khi xem xét các mô hình phức tạp hoặc dữ liệu lớn.
- Khả năng diễn giải (Interpretability): Khả năng diễn giải của mô hình đo lường khả năng hiểu được lý do tại sao một dự đoán được thực hiện. Một mô hình diễn giải tốt có thể cung cấp giải thích rõ ràng và logic cho quá trình ra quyết định.
- Hiệu suất tính toán (Computational performance): Đánh giá hiệu suất tính toán của mô hình là quan trọng, đặc biệt khi triển khai mô hình trên các hệ thống có tài nguyên hạn chế. Thời gian dự đoán và khả năng làm việc với dữ liệu lớn là những yếu tố quan trọng để xem xét.

$\Longrightarrow$ Những tiêu chí trên là chỉ một số ví dụ phổ biến. Đánh giá mô hình cần tuân thủ các tiêu chí phù hợp với bài toán cụ thể và ngữ cảnh sử dụng mô hình đó.

Kết quả huất luyện mô hình

Hình 3.3: Model Accuracy và Model Loss.

Đồ thị ACC, độ chính xác (Accuracy) lên đến 98% và bắt đầu ổn định khi epoch = 20, cho cả tập dữ liệu huấn luyện và tập test, đồ thị Loss của cũng cho thấy sau mức này,

Loss của cả tập train và validation. Kết quả cho thấy mô hình có độ phù hợp tốt và không bị quá khớp (over-fitting).

- Đánh giá hiệu suất mô hình:

```
29/29 [==============================] - 0s 9ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       191
           1       0.99      0.99      0.99       193
           2       1.00      1.00      1.00       110
           3       0.99      1.00      0.99       213
           4       1.00      1.00      1.00       204

    accuracy                           1.00       911
   macro avg       1.00      1.00      1.00       911
weighted avg       1.00      1.00      1.00       911
```

Hình 3.4: Đánh giá hiệu suất mô hình.

- Confusion Matrix:

Hình 3.5: Confusion Matrix.

3.2.3 Thử nghiệm với thời gian thực webcam

*** Sử dụng thư viện open cv để test kết quả trên thời gian thực

** Bước 1: Import thư viện và Load Model

```python
import cv2
import mediapipe as mp
import numpy as np
import threading
import tensorflow as tf

label = "Warmup...."
n_time_steps = 10
lm_list = []

mpPose = mp.solutions.pose
pose = mpPose.Pose()
mpDraw = mp.solutions.drawing_utils

model = tf.keras.models.load_model("model_19.h5")
```

** Bước 2: Các hàm phụ trợ vẽ điểm nút nối các điểm nút và hàm dự đoán kết quả

```python
def make_landmark_timestep(results):
    c_lm = []
    for id, lm in enumerate(results.pose_landmarks.landmark):
        c_lm.append(lm.x)
        c_lm.append(lm.y)
        c_lm.append(lm.z)
        c_lm.append(lm.visibility)
    return c_lm


def draw_landmark_on_image(mpDraw, results, img):
    mpDraw.draw_landmarks(img, results.pose_landmarks, mpPose.POSE_CONNECTIONS)
    for id, lm in enumerate(results.pose_landmarks.landmark):
        h, w, c = img.shape
        # print(id, lm)
        cx, cy = int(lm.x * w), int(lm.y * h)
        cv2.circle(img, (cx, cy), 5, (255, 0, 0), cv2.FILLED)
    return img


def draw_class_on_image(label, img):
    font = cv2.FONT_HERSHEY_SIMPLEX
    bottomLeftCornerOfText = (10, 30)
    fontScale = 1
    fontColor = (0, 255, 0)
    thickness = 2
    lineType = 2
    cv2.putText(img, label,
                bottomLeftCornerOfText,
                font,
                fontScale,
                fontColor,
                thickness,
                lineType)
    return img


def detect(model, lm_list):
    global label
    lm_list = np.array(lm_list)
    lm_list = np.expand_dims(lm_list, axis=0)
    print(lm_list.shape)

    y_hat = model.predict(lm_list)
    results = np.argmax(y_hat)
    print(y_hat , results)


```

```
    label_predict = ('body','hand','doze','love','clap')
    label = label_predict[results]
    return label
```

** Bước 3: Chạy webcam

```
while True:

    success, img = cap.read()
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = pose.process(imgRGB)
    i = i + 1
    if i > warmup_frames:
        print("Start detect....")

        if results.pose_landmarks:
            c_lm = make_landmark_timestep(results)

            lm_list.append(c_lm)
            if len(lm_list) == n_time_steps:
                # predict
                t1 = threading.Thread(target=detect, args=(model, lm_list))
                t1.start()
                #t1 = detect(model,lm_list)
                lm_list = []


            img = draw_landmark_on_image(mpDraw, results, img)

    img = draw_class_on_image(label, img)
    cv2.imshow("Image", img)
    if cv2.waitKey(1) == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

3.3 Xây dựng mô hình sử dụngConvLSTM2D layer và mô hình LRCN

3.3.1 Tiền xử lý dữ liệu

Thu thập dữ liệu

Trong bài này nhóm xử dụng bộ dự liệu UCF50 - Action Recognition Dataset Bộ dữ liệu Nhận diện hành động, bao gồm các video thực tế được lấy từ YouTube, điều này làm cho bộ dữ liệu này khác biệt so với hầu hết các bộ dữ liệu nhận diện hành động khác vì chúng không phải là thực tế và được đóng bởi diễn viên. Bộ dữ liệu chứa :

- 50 Danh mục hành động
- 133 Số video trung bình mỗi hành động
- 26 Số khung hình trung bình mỗi video

Hình 3.6: Visualize the Data with its Labels .

Tăng cường dữ liệu

Hình 3.7: Visualize the numbers .

Nhận thấy bộ dữ liệu này có sự chênh lệnh về dự liệu cũng như đây là bộ dữ liệu đã có từ 5 năm trước . Hiện nay video đã cải thiện chất lượng hơn . Nhóm đã thêm video tải từ youtube vào để dữ liệu cân bằng cũng như video có độ phân giải tốt hơn để khả năng dự đoán được cải thiện

Sau khi tăng cường dataset mới được lưu trên drive : Dataset

Chuẩn hóa dữ liệu

Đầu tiên, chúng ta sẽ đọc các tệp video từ bộ dữ liệu và thay đổi kích thước các khung hình của video thành chiều rộng và chiều cao cố định, để giảm thiểu tính toán và chuẩn hóa dữ liệu về khoảng [0-1] bằng cách chia giá trị pixel cho 255, điều này làm cho quá trình hội tụ nhanh hơn khi huấn luyện mạng.

Nhưng trước hết, hãy khởi tạo một số hằng số.

```
IMAGE_HEIGHT , IMAGE_WIDTH = 64, 64

SEQUENCE_LENGTH = 10

DATASET_DIR = "UCF50"
```

Tiếp theo tạo một hàm frames_extraction() sẽ tạo một danh sách chứa các khung hình đã thay đổi kích thước và chuẩn hóa của một video, đường dẫn của video sẽ được truyền vào hàm như một đối số. Hàm sẽ đọc tệp video từng khung hình một, mặc dù không phải tất cả các khung hình được thêm vào danh sách vì chúng ta chỉ cần một chuỗi độ dài khung hình phân phối đều.

```
def frames_extraction(video_path):

    frames_list = []
    video_reader = cv2.VideoCapture(video_path)
    video_frames_count =int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
    skip_frames_window = max(int(video_frames_count/SEQUENCE_LENGTH), 1)

    for frame_counter in range(SEQUENCE_LENGTH):

        video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window
            )

        success, frame = video_reader.read()
        if not success:
            break

        resized_frame = cv2.resize(frame, (IMAGE_HEIGHT, IMAGE_WIDTH))
        normalized_frame = resized_frame / 255
        frames_list.append(normalized_frame)
    video_reader.release()

    return frames_list
```

Bây giờ chúng ta sẽ tạo một hàm create_dataset() sẽ lặp qua tất cả các lớp được chỉ định trong hằng số CLASSES_LIST và sẽ gọi hàm frame_extraction() trên mọi tệp video của các lớp được chọn và trả về các khung hình (đặc trưng), chỉ mục lớp (nhãn) và đường dẫn tệp video (video_files_paths).

```
def create_dataset():

    features = []
    labels = []
    video_files_paths = []
    for class_index, class_name in enumerate(CLASSES_LIST):

        print(f'Extracting Data of Class: {class_name}')

        files_list = os.listdir(os.path.join(DATASET_DIR, class_name))

        for file_name in files_list:

            video_file_path = os.path.join(DATASET_DIR, class_name, file_name)

            frames = frames_extraction(video_file_path)

            if len(frames) == SEQUENCE_LENGTH:
                features.append(frames)
                labels.append(class_index)
                video_files_paths.append(video_file_path)
    features = np.asarray(features)
    labels = np.array(labels)

```

```
    return features, labels, video_files_paths
```

Cuối cùng ta lại làm các bước chia dữ liệu thành train và test , one hot encode giống như đối với mô hình LSTM đầu tiên

```
X, y, video_files_paths = create_dataset()

one_hot_encoded_labels = to_categorical(y)

X_train, X_test, y_train, y_test = train_test_split(X, one_hot_encoded_labels,
    test_size = 0.2, shuffle = True,random_state = 42)
```

Sau khi đã chuẩn bị xong dữ liệu , bước tiếp theo là xây dựng model và đưa dữ liệu đã chuẩn bị vào huấn luyện

3.3.2 Xây dựng mô hình sử dụng ConvLSTM2D layer

Hình 3.8: ConvLSTM2D layer.

Trong bước này, nhóm sẽ thực hiện phương pháp thứ 2 bằng cách sử dụng sự kết hợp của các ô ConvLSTM. Một ô ConvLSTM là một biến thể của mạng LSTM chứa các hoạt động tích chập trong mạng. Điều này là một LSTM với tích chập được nhúng trong kiến trúc, làm cho nó có khả năng xác định các đặc trưng không gian của dữ liệu trong khi vẫn giữ quan hệ thời gian.

Đối với phân loại video, phương pháp này hiệu quả trong việc nắm bắt mối quan hệ không gian trong các khung hình cá nhân và mối quan hệ thời gian qua các khung hình

khác nhau. Kết quả của cấu trúc tích chập này, ConvLSTM có khả năng nhận đầu vào 3 chiều (chiều rộng, chiều cao, số kênh) trong khi LSTM đơn giản chỉ nhận đầu vào 1 chiều nên LSTM không tương thích để mô hình dữ liệu không gian-thời gian một cách độc lập. Tương tự như lớp LSTM, nhưng các phép biến đổi đầu vào và các phép biến đổi hồi quy đều là tích chập.Lớp ConvLSTM2D thường được sử dụng để xử lý các chuỗi ảnh (video) 2 chiều .

ConvLSTM khác với LSTM ở chỗ nó kết hợp cả tính chất không gian (qua các bước tích chập) và tính chất thời gian (qua các bước LSTM), cho phép nó xử lý dữ liệu không gian-thời gian một cách hiệu quả. LSTM thông thường không có khả năng này vì nó chỉ xử lý dữ liệu theo một chiều thời gian.

Để hiểu rõ hơn về ConvLSTM2D hãy đọc bài báo Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting để hiểu hơn về kiến trúc này .

Lớp ConvLSTM2D cần 2 tham số chính:

- Số bộ lọc (filters):Xác định số lượng bộ lọc để trích xuất các đặc điểm khác nhau từ ảnh .
- Kích thước kernel (kernel size):Xác định kích thước của bộ lọc hình vuông sẽ được di chuyển trên ảnh để trích xuất các đặc điểm .

Đầu ra của lớp ConvLSTM2D là một tensor 4 chiều (batch_size, timesteps, height, width), với :

- batch_size là số lượng mẫu trong mỗi lần tính toán.
- timesteps là số khung hình trong chuỗi ảnh.
- height và width là chiều cao và rộng của khung hình.

Kiến trúc mô hình

```
model = Sequential()


model.add(ConvLSTM2D(filters = 32, kernel_size = (3, 3), activation = 'tanh',
    data_format = "channels_last",
```

```
                        recurrent_dropout=0.2, return_sequences=True, input_shape =
                            (SEQUENCE_LENGTH,IMAGE_HEIGH,IMAGE_WIDTH,3)))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='
        channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(ConvLSTM2D(filters = 64, kernel_size = (3, 3), activation = 'tanh',
        data_format = "channels_last",
                        recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='
        channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(ConvLSTM2D(filters = 64, kernel_size = (3, 3), activation = 'tanh',
        data_format = "channels_last",
                        recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='
        channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(ConvLSTM2D(filters = 128, kernel_size = (3, 3), activation = 'tanh',
        data_format = "channels_last",
                        recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='
        channels_last'))

    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "softmax"))

    model.summary()
```

Các tham số của convLSTM2D có thể hiểu như sau :

- input_shape = (SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, 3): Kích thước đầu vào, trong đó SEQUENCE_LENGTH là độ dài chuỗi, IMAGE_HEIGHT và IMAGE_WIDTH là kích thước ảnh, và 3 là số kênh màu (RGB).
- filters = 32: Số lượng filter hoặc feature maps được áp dụng.
- kernel_size = (3, 3): Kích thước của filter.
- activation = 'tanh': Hàm kích hoạt là tanh.
- data_format = "channels_last": Định dạng dữ liệu là "channels_last" (chiều cuối cùng là kênh dữ liệu).
- recurrent_dropout = 0.2: Tỷ lệ dropout cho các kết nối tái lặp trong LSTM.
- return_sequences = True: Trả về chuỗi output thay vì chỉ output cuối cùng.

conv_lstm2d_input
InputLayer
input:
output:
[(None, 10, 64, 64, 3)]
[(None, 10, 64, 64, 3)]
conv_lstm2d
ConvLSTM2D
input:
output:
(None, 10, 64, 64, 3)
(None, 10, 62, 62, 32)
max_pooling3d
MaxPooling3D
input:
output:
(None, 10, 62, 62, 32)
(None, 10, 31, 31, 32)
time_distributed(dropout)
TimeDistributed(Dropout)
input:
output:
(None, 10, 31, 31, 32)
(None, 10, 31, 31, 32)
conv_lstm2d_1
ConvLSTM2D
input:
output:
(None, 10, 31, 31, 32)
(None, 10, 29, 29, 64)
max_pooling3d_1
MaxPooling3D
input:
output:
(None, 10, 29, 29, 64)
(None, 10, 15, 15, 64)
time_distributed_1(dropout_1)
TimeDistributed(Dropout)
input:
output:
(None, 10, 15, 15, 64)
(None, 10, 15, 15, 64)
conv_lstm2d_2
ConvLSTM2D
input:
output:
(None, 10, 15, 15, 64)
(None, 10, 13, 13, 64)
max_pooling3d_2
MaxPooling3D
input:
output:
(None, 10, 13, 13, 64)
(None, 10, 7, 7, 64)
time_distributed_2(dropout_2)
TimeDistributed(Dropout)
input:
output:
(None, 10, 7, 7, 64)
(None, 10, 7, 7, 64)
conv_lstm2d_3
ConvLSTM2D
input:
output:
(None, 10, 7, 7, 64)
(None, 10, 5, 5, 128)
max_pooling3d_3
MaxPooling3D
input:
output:
(None, 10, 5, 5, 128)
(None, 10, 3, 3, 128)
flatten
Flatten
input:
output:
(None, 10, 3, 3, 128)
(None, 11520)
dense
Dense
input:
output:
(None, 11520)
(None, 8)

Hình 3.9: Model ConvLSTM2D.

Mô hình nhóm xây dựng để phân biệt 8 hành động , nên nhóm đã để Cuối cùng là 1 lớp kết nối đầy đủ với hàm kích hoạt softmax

3.3.3 Huấn luyện mô hình

Bước tiếp theo nhóm thực hiện là thiết lập và huấn luyện mô hình

```
early_stopping_callback = EarlyStopping(monitor = 'val_loss', patience = 10, mode =
    'min', restore_best_weights = True)
convlstm_model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam',
    metrics = ["accuracy"])

convlstm_model_training_history = convlstm_model.fit(x = X_train, y = y_train,
    epochs = 40,
                                    batch_size=32,shuffle = True, validation_split =
                                        0.2,callbacks,[early_stopping_callback])
```

Giải thích chi tiết :

- Early Stopping Callback:Sử dụng EarlyStopping callback để theo dõi ’val_loss’, chờ đợi 10 epochs trước khi dừng, theo chế độ ’min’ (tối thiểu), và khôi phục trọng số tốt nhất khi kết thúc.
- Compile Model:Compile mô hình sử dụng hàm loss là ’categorical_crossentropy’, trình tối ưu hóa là ’Adam’, và đánh giá theo chỉ số ’accuracy’.
- Training Model:Huấn luyện mô hình sử dụng dữ liệu đầu vào X_train và nhãn y_train, với 40 epochs, batch size là 32, xáo trộn dữ liệu, phân chia validation set là 20%, và sử dụng callbacks bao gồm early_stopping_callback.

Đánh giá mô hình

&

Hình 3.10: Model Accuracy và Model Loss.

Đồ thị ACC, độ chính xác (Accuracy) lên đến 98% và bắt đầu ổn định khi epoch = 30, cho cả tập dữ liệu huấn luyện và 86% tập test, đồ thị Loss của cũng cho thấy sau mức này, Loss của cả tập train và validation. Kết quả cho thấy mô hình có độ phù hợp tốt và không bị quá khớp (over-fitting).

- Đánh giá hiệu suất mô hình:

```
7/7 [==============================] - 2s 89ms/step
              precision    recall  f1-score   support

           0       1.00      0.97      0.98        31
           1       0.87      0.79      0.83        34
           2       0.80      0.80      0.80        30
           3       0.85      0.88      0.86        25
           4       0.89      0.85      0.87        20
           5       0.73      0.89      0.80        18
           6       0.88      0.88      0.88        24
           7       0.91      0.91      0.91        32

    accuracy                           0.87       214
   macro avg       0.87      0.87      0.87       214
weighted avg       0.87      0.87      0.87       214
```

Hình 3.11: Đánh giá hiệu suất mô hình.

- Confusion Matrix:

Hình 3.12: Confusion Matrix.

3.3.4 Xây dựng mô hình LRCN

Hình 3.13: LRCN model.

Trong bước này, nhóm sẽ thực hiện phương pháp thứ 3 xây dựng mô hình LRCN (Long-Term Recurrent Convolutional Network)

Mô hình LRCN (Long-Term Recurrent Convolutional Network) là một mô hình học sâu được sử dụng cho các tác vụ xử lý dữ liệu hình ảnh tuần tự, chẳng hạn như phân loại video, nhận dạng hành động và chú thích hình ảnh.

Mô hình LRCN là một mô hình kết hợp các lớp Convolutional và LSTM trong một mô hình duy nhất.Mô hình LRCN bao gồm hai thành phần chính:

- Các lớp Convolutional trích xuất các đặc trưng không gian từ từng khung hình.

- Các đặc trưng này được đưa trực tiếp vào các lớp LSTM tại từng bước thời gian để mô hình chuỗi thời gian.

Cách tiếp cận này cho phép mô hình học trực tiếp các đặc trưng không gian thời gian trong quá trình huấn luyện end-to-end, dẫn đến một mô hình mạnh mẽ hơn. Cụ thể, mô hình LRCN hoạt động như sau:

- Dữ liệu đầu vào (các sequence hình ảnh) được đưa vào các lớp convolution 2D. Các lớp convolution 2D sẽ trích xuất các đặc trưng không gian từ từng frame trong sequence.
- Các đặc trưng không gian từ các frames được kết hợp lại và đưa vào lớp LSTM. Lớp LSTM sẽ xử lý thông tin thời gian giữa các frames.
- Lớp LSTM sẽ tạo ra một vector biểu diễn cho toàn bộ sequence.
- Vector biểu diễn này được đưa vào một lớp phân loại để dự đoán kết quả của tác vụ.

Mô hình LRCN có một số ưu điểm sau:

- Có thể trích xuất các đặc trưng không gian và thời gian từ dữ liệu hình ảnh tuần tự.
- Có thể nắm bắt các mối quan hệ dài hạn giữa các frames trong một sequence.
- Có thể huấn luyện end-to-end, đơn giản hóa quá trình huấn luyện.

Tuy nhiên, mô hình LRCN cũng có một số nhược điểm sau:

- Có thể gặp khó khăn trong việc nắm bắt các mối quan hệ phức tạp giữa các đặc trưng không gian và thời gian.
- Có thể khó huấn luyện.

Mô hình LRCN đã được áp dụng thành công cho nhiều tác vụ xử lý dữ liệu hình ảnh tuần tự, chẳng hạn như:

- Phân loại video
- Nhận dạng hành động
- Chú thích hình ảnh
- Theo dõi đối tượng

Dưới đây là một số ví dụ về ứng dụng của mô hình LRCN:

- Phân loại video thành các loại khác nhau, chẳng hạn như thể thao, tin tức, giải trí, v.v.
- Nhận dạng các hành động của con người trong video, chẳng hạn như đi bộ, chạy, nhảy, v.v.
- Chú thích hình ảnh bằng văn bản, chẳng hạn như mô tả nội dung của hình ảnh.
- Theo dõi đối tượng trong video, chẳng hạn như xác định vị trí của một người trong video.

Kiến trúc mô hình

```
def create_LRCN_model():


    model = Sequential()


    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu'),
                              input_shape = (SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.25)))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.25)))

    model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.25)))

    model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    #model.add(TimeDistributed(Dropout(0.25)))
```

```
    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(32))

    model.add(Dense(len(CLASSES_LIST), activation = 'softmax'))


    model.summary()


    return model
```

Lớp TimeDistributed(Conv2D) thực hiện các bước sau :

- TimeDistributed: Đảm bảo lớp Conv2D được áp dụng cho từng frame riêng lẻ trong chuỗi hình ảnh.
- Conv2D(16, (3, 3), padding='same', activation='relu'): Tạo một lớp convolutional 2D với các thông số sau:
- input_shape = (SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, 3): Kích thước đầu vào, trong đó SEQUENCE_LENGTH là độ dài chuỗi, IMAGE_HEIGHT và IMAGE_WIDTH là kích thước ảnh, và 3 là số kênh màu (RGB).

Hình 3.14: Kiến trúc model LRCN.

Mô hình nhóm xây dựng để phân biệt 8 hành động , nên nhóm đã để Cuối cùng là 1 lớp kết nối đầy đủ với hàm kích hoạt softmax

Bước tiếp theo nhóm thực hiện là thiết lập và huấn luyện mô hình áp dụng giống như cách làm với mô hình sử dụng convLSTM2D

Đánh giá mô hình

Hình 3.15: Model Accuracy và Model Loss.

Đồ thị ACC, độ chính xác (Accuracy) lên đến 97% và bắt đầu ổn định khi epoch = 20, cho cả tập dữ liệu huấn luyện và và 90% tập test, đồ thị Loss của cũng cho thấy sau mức này, Loss của cả tập train và validation. Kết quả cho thấy mô hình có độ phù hợp tốt và không bị quá khớp (over-fitting).

- Đánh giá hiệu suất mô hình:

```
7/7 [==============================] - 1s 18ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        31
           1       0.97      0.91      0.94        34
           2       0.90      0.90      0.90        30
           3       0.86      0.96      0.91        25
           4       0.89      0.85      0.87        20
           5       0.89      0.94      0.92        18
           6       1.00      1.00      1.00        24
           7       0.94      0.91      0.92        32

    accuracy                           0.93       214
   macro avg       0.93      0.93      0.93       214
weighted avg       0.94      0.93      0.93       214
```

Hình 3.16: Đánh giá hiệu suất mô hình.

- Confusion Matrix:

Hình 3.17: Confusion Matrix.

3.3.5 *Thử nghiệm với video download từ youtube*

Bước 1 : Download video từ youtube : Hàm download_youtube_video có nhiệm vụ download video từ link youtube .

```
def download_youtube_video(url, output_path):
    try:
        youtube = pytube.YouTube(url)
        video = youtube.streams.get_highest_resolution()
        title = video.title
        output_file_path = f"{output_path}/{title}.mp4"
        video.download(output_path)
        return title
    except Exception as e:
        print("Error: ", str(e))
        return None
```

Bước 2 : Viết hàm dự đoán hành động trong video

```
def predict_on_video(video_file_path, output_file_path, SEQUENCE_LENGTH):


    video_reader = cv2.VideoCapture(video_file_path)

    original_video_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
    original_video_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))

    video_writer = cv2.VideoWriter(output_file_path, cv2.VideoWriter_fourcc('M', 'P'
        , '4', 'V'),
                                   video_reader.get(cv2.CAP_PROP_FPS), (
                                       original_video_width, original_video_height))

    frames_queue = deque(maxlen = SEQUENCE_LENGTH)

    predicted_class_name = ''

    while video_reader.isOpened():

        _, frame = video_reader.read()

        if not _:
            break

        resized_frame = cv2.resize(frame, (IMAGE_HEIGHT, IMAGE_WIDTH))

        normalized_frame = resized_frame / 255

        frames_queue.append(normalized_frame)

        if len(frames_queue) == SEQUENCE_LENGTH:

            predicted_labels_probabilities = convlstm_model.predict(np.expand_dims(
                frames_queue, axis = 0))[0]

            predicted_label = np.argmax(predicted_labels_probabilities)

            predicted_class_name = CLASSES_LIST[predicted_label]

        cv2.putText(frame, predicted_class_name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
             1, (0, 255, 0), 2)

        video_writer.write(frame)

    video_reader.release()
```

```
    video_writer.release()
```

Trong đoạn code trên cần chú ý :

- frames_queue = deque(maxlen=SEQUENCE_LENGTH): Tạo một hàng đợi (queue) để lưu trữ các khung hình, giới hạn số lượng khung hình tối đa là SEQUENCE_LENGTH.

Chương 4 Demo sản phẩm và đánh giá các phương pháp

4.1 DEMO SẢN PHẨM

4.1.1 Sử dụng công cụ MediaPipe và xây dựng mạng LSTM

- Swing

Hình 4.1: Swing output.

- Hand Swing

Hình 4.2: Hand Swing output.

- Love

Hình 4.3: Love output.

- Clap

Hình 4.4: Clap output.

- Doze

Hình 4.5: Doze output.

4.1.2 Mô hình ConvLSTM

- Billards

Hình 4.6: Billards output.

- Soccer Juggling

Hình 4.7: Soccer Juggling output.

- Swing

Hình 4.8: Swing output.

- HorseRace

Hình 4.9: HorseRace output.

- PlayingPiano

Hình 4.10: PlayingPiano output.

- PushUps

Hình 4.11: PushUps output.

- Skijet

Hình 4.12: Skijet output.

- Drumming

Hình 4.13: Drumming output.

4.1.3 Mô hình LRCN

- Billards

Hình 4.14: Billards output.

- Soccer Juggling

Hình 4.15: Soccer Juggling output.

- Swing

Hình 4.16: Swing output.

- HorseRace

Hình 4.17: HorseRace output.

- PlayingPiano

Hình 4.18: PlayingPiano output.

- PushUps

Hình 4.19: PushUps output.

- Skijet

Hình 4.20: Skijet output.

- Drumming

Hình 4.21: Drumming output.

4.2 Đánh giá các phương pháp

Trong dự án này ,nhóm đã tiếp cận bài toán theo 3 phương pháp tùy thuộc vào độ phức tạp , độ chính xác cần thiết và khả năng tính toán.Đối với phương pháp sử dụng thư viện mediapipe kết hợp với LSTM phù hợp với các bài toán đơn giản , nhận diện ít hành động.Đối với nhận diện hành động qua video , Nhóm nhận thấy phương pháp xây dựng model LRCN cho ra kết quả nhận diện hành động từ video tốt hơn phương pháp sử dụng ConvLSTM . Tuy nhiên tùy thuộc vào từng bài toán mà chúng ta chọn phương pháp sao cho hiệu quả nhất .

- Model ConvLSTM sử dụng các lớp ConvLSTM2D để học các đặc trưng tĩnh và động từ các khung hình video cùng một lúc.

- Model LRCN sử dụng các lớp TimeDistributed(Conv2D) để trích xuất các đặc trưng tĩnh từ từng khung hình riêng lẻ, sau đó mới đưa các đặc trưng này vào một lớp LSTM để học các mối quan hệ động giữa các khung hình.

4.2.1 Cách tiếp cận sử dụng ConvLSTM2D

- Các lớp ConvLSTM2D trích xuất các đặc trưng không gian thời gian đồng thời duy trì mối quan hệ không gian. - Học các đặc trưng không gian và thời gian đồng thời bằng cách tích hợp các phép toán tích chập trực tiếp vào các lớp LSTM. - Các phép toán tích chập được sử dụng để trích xuất đặc trưng không gian từ các khung hình, trong khi các cổng LSTM được sử dụng để học các mối quan hệ thời gian giữa các khung hình. - Độ phức tạp của video: ConvLSTM có thể phù hợp hơn cho các video có các hành động phức tạp hoặc dài. - Độ chính xác cần thiết: ConvLSTM có thể đạt được độ chính xác cao hơn trong một số trường hợp. - Khả năng tính toán: ConvLSTM thường đòi hỏi nhiều tài nguyên tính toán hơn so với LRCN.

4.2.2 Cách tiếp cận sử dụng LRCN

- LRCN kết hợp các lớp CNN và LSTM trong cùng một mô hình. - Các lớp CNN được sử dụng để trích xuất các đặc trưng không gian từ các frames. - Các đặc trưng không gian được đưa vào các lớp LSTM tại mỗi bước thời gian để mô hình hóa chuỗi thời gian. - Cách tiếp cận này cho phép mô hình học trực tiếp các đặc trưng không gian thời gian trong quá trình huấn luyện end-to-end, dẫn đến một mô hình mạnh mẽ hơn. - Trong LRCN, lớp TimeDistributed được sử dụng để áp dụng cùng một lớp cho mọi khung hình của video một cách độc lập. - Lớp TimeDistributed cho phép model LRCN xử lý toàn bộ video trong một lần, thay vì phải xử lý từng khung hình riêng lẻ.

Ưu điểm của LRCN:

- Có thể học các đặc trưng không gian thời gian trực tiếp từ dữ liệu .
- Có thể nắm bắt các mối quan hệ dài hạn giữa các frames trong một sequence .
- Có thể huấn luyện end-to-end, đơn giản hóa quá trình huấn luyện .

Nhược điểm của LRCN:

- Có thể khó huấn luyện hơn so với cách tiếp cận sử dụng CNN và LSTM riêng biệt.
- Có thể gặp khó khăn trong việc nắm bắt các mối quan hệ phức tạp giữa các đặc trưng không gian và thời gian.

Khi nào nên chọn LRCN:

- Khi cần nắm bắt các mối quan hệ dài hạn giữa các frames trong một sequence.
- Khi cần một mô hình mạnh mẽ có thể học các đặc trưng không gian thời gian trực tiếp từ dữ liệu.

4.3 Hướng phát triển trong tương lai

- Trong tương lai, nhóm muốn tạo ra một app trên điện thoại để có thể đến được gần hơn với nhiều người.
- Muốn phát triển lên thành sản phẩm hoàn chỉnh có tính thực tế cao hơn.
- Cải thiện model tốt hơn , dự đoán chính xác hơn , nhanh hơn .

KẾT LUẬN

Bài viết này mang đến cho bạn đọc một vài giải pháp trong bài toán nhận diện hành động của con người trong video.

Kết quả thực nghiệm cho thấy mô hình nhận dạng sử dụng các thuật toán phân loại phổ biến và tập đặc trưng của chúng tôi cho kết quả rất tốt với các hoạt động thường ngày của con người. Việc đơn giản hoá bộ đặc trưng và lựa chọn kích thước cửa sổ tối ưu từ việc phân tích đặc điểm dữ liệu có thể làm cải thiện tốc độ nhận dạng mà vẫn đảmbảo độ chính xác. Ngoài ra, việc lựa chọn một thuật toán phân loại thích hợp cũng ảnh hưởng rõ rệt tới kết quả nhận dạng. Đối với mô hình thử nghiệm của chúng tôi, thuật toán CNN giúp cải thiện độ chính xác hơn so với DecisionTree, Gradient Boosted Tree, SVM, RF, KNN. Tuy nhiên,hành vi của con người không chỉ là tự nhiên và tự phát,mà con người có thể thực hiện một số hoạt động cùng mộtlúc, hoặc thậm chí thực hiện một số hoạt động không liên quan. Đây cũng là thách thức lớn trong bài toán nhận dạng hành động với thời gian thực. Trong tương lai, chúng tôi tiếp tục phát triển hệ thống của mình để có thể dự đoán và xác định các hoạt động đồng thời, hỗ trợ ứng dụng cho các hoàn cảnh sử dụng phức tạp hơn, hướng tới các ứng dụng không chỉ gắn với con người

Có rất nhiều thứ có thể cải tiến để có được kết quả tốt hơn mà bạn có thể thử nếu áp dụng vào bài toán thực tế:

- Tăng FPS để có thể chạy được realtime: Tối ưu hóa model (pruning, quantization), loại bỏ bớt frame khi nhận diện, sử dụng multi-threading, ...
- Sử dụng các Pose Estimation model khác như AlphaPose, OpenPose, ...

Tài liệu tham khảo

[1] "ConvLSTM2D Layer". In: 58 (). DOI: 10.22144/ctu.jvn.2022.217. URL: https://keras.io/api/layers/recurrent_layers/conv_lstm2d/.

[2] "Long-term Recurrent Convolutional Networks for Visual Recognition and Description". In: (2018). URL: https://arxiv.org/abs/1411.4389source=post_page----------.

[3] "Dataset UCF50". In: (2017). URL: https://www.crcv.ucf.edu/data/UCF50_files/MVAP_UCF50.pdf.

[4] "Convolutional Neural Networks". In: (2021). URL: https://www.ibm.com/topics/convolutional-neural-networks.

[5] "TimeDistributed layer". In: (). URL: https://keras.io/api/layers/recurrent_layers/time_distributed/.

[6] Ekaba Bisong. "Google Colaboratory". In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 59–64. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_7. URL: https://doi.org/10.1007/978-1-4842-4470-8_7.

[1]–[6]

Code

Toàn hộ source code ,model và video demo output project được chia sẻ tại : https://github.com/qua
Human-activity-recognition