

ĐẠI HỌC QUỐC GIA
ĐẠI HỌC BÁCH KHOA TP HỒ CHÍ MINH



Báo cáo bài tập lớn

Nhận diện ngôn ngữ ký hiệu

Giảng viên hướng dẫn: MSc. Nguyễn Khánh Lợi

Danh sách các thành viên

Tên thành viên	MSSV
Nguyễn Thị Thương	2112413
Lý Thị Huỳnh Như	2114336
Trần Vĩ Khang	2111464
Hứa Gia Bảo	2112858

MỤC LỤC

DANH MỤC HÌNH ẢNH.....	4
I.Tổng quan lý thuyết.....	5
1. Khái niệm về Machine Learning	5
1.1 Phân loại (Classification)	6
1.2 Hồi quy (Regression)	7
1.2.1 Mô hình RNN + LSTM	7
1.2.2 Transformer + Attention Mechanism	11
1.3 Học có giám sát và không giám sát	11
1.3.1 Học có giám sát (Supervised Learning)	11
1.3.2 Học không giám sát (Unsupervised Learning)	12
2. Quy trình huấn luyện mô hình	13
2.1 Chia dữ liệu thành tập huấn luyện (Training set) và tập kiểm thử (Test set)	13
2.2 Overfitting và underfitting và cách khắc phục	13
3. Khái niệm về Python và OpenCV	14
3.1 Python	14
3.2 OpenCV	15
4. Detection	15
4.1 Object Detection (Phát hiện đối tượng)	15
4.2 Anomaly Detection (Phát hiện bất thường)	16
4.3 Intrusion Detection (Phát hiện xâm nhập mạng)	16
4.4 Face Detection / Emotion Detection	16
4.5 Keyword Spotting / Voice Activity Detection	16
II. Tổng quan về đề tài	17
1. Phân loại (Classification) trong Sign Language Detection:	18
1.1 Hồi quy (Regression) trong Sign Language Detection	19

1.1.1	Mô hình RNN + LSTM	19
1.1.2	Transformer + Attention Mechanism	20
1.2	Học có giám sát và không giám sát trong Sign Language Detection	20
2.	Quy trình huấn luyện mô hình trong Sign Language Detection	22
2.1	Chia dữ liệu thành tập huấn luyện (Training set) và tập kiểm thử	22
2.2	Overfitting và underfitting trong Sign Language Detection và cách khắc phục	23
III.	Xử lý tiền dữ liệu	25
1.	Thu thập dữ liệu	25
2.	Quá trình xây dựng code	26
3.	Kết quả chạy code	28
IV.	Thực hiện đề tài	29
1.	Thu thập dữ liệu	29
1.1	Dữ liệu mẫu	29
2.	Training Model trên Google Colab	30
2.1	Các bước thực hiện	30
2.2	Biểu đồ Accurary và Loss	39
3.	Đánh giá	40
4.	Kiểm chứng và sử dụng model	41
4.1	Chương trình nhận diện main.py	41
4.2	Kết quả thu được:	43
	TÀI LIỆU THAM KHẢO	45

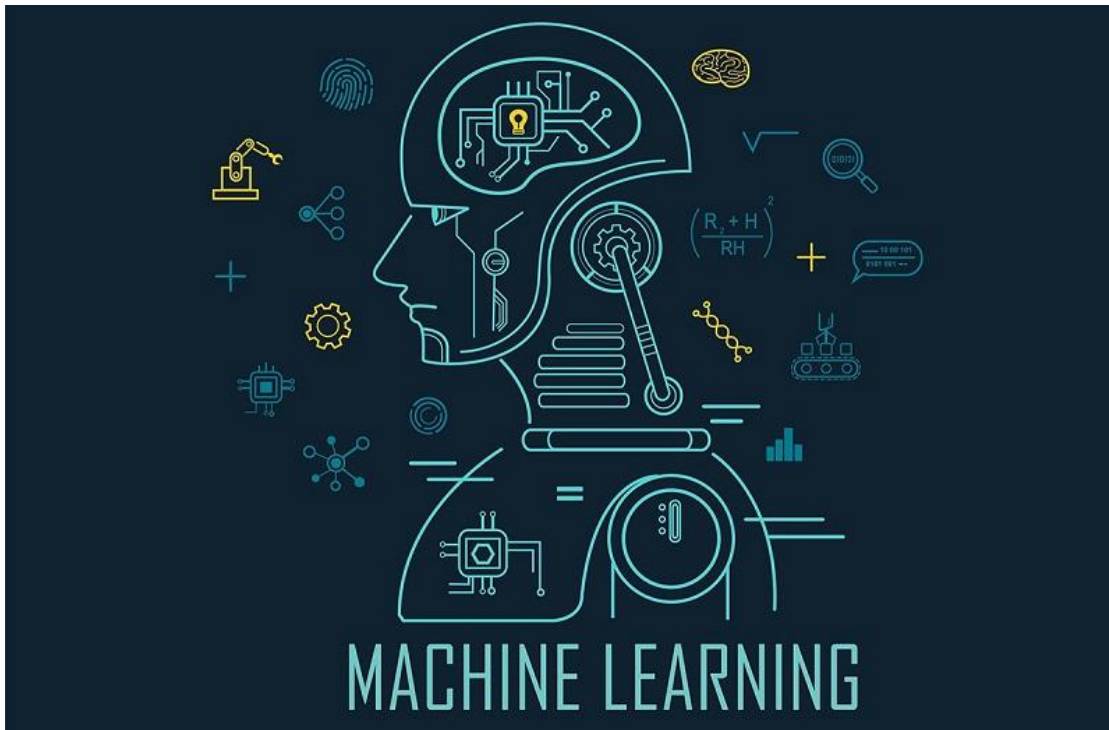
DANH MỤC HÌNH ẢNH

<i>Hình 1: Ảnh minh họa về Machine Learning</i>	5
<i>Hình 2: Machine learning workflow</i>	6
<i>Hình 3: Mô hình RNN</i>	9
<i>Hình 4: Các mô hình khác</i>	9
<i>Hình 5: Mô hình: LSTM (Long Short-Term Memory)</i>	10
<i>Hình 6: Học có giám sát</i>	12
<i>Hình 7: Học không giám sát</i>	13
<i>Hình 8: Overfitting và underfitting</i>	13
<i>Hình 9: Python</i>	14
<i>Hình 10: Nhận diện kí hiệu bằng phân tích hình ảnh hoặc video</i>	18
<i>Hình 11: Nhận diện ngôn ngữ kí hiệu</i>	18
<i>Hình 12: Xoay ảnh ngẫu nhiên (-30 đến 30 độ)</i>	26
<i>Hình 13: Lật ảnh (ngẫu nhiên lật ngang hoặc không)</i>	26
<i>Hình 14: Dịch ảnh (Translation)</i>	26
<i>Hình 15: Điều chỉnh độ sáng và độ tương phản</i>	27
<i>Hình 16: Thêm nhiễu Gaussian</i>	27
<i>Hình 17: Làm mờ ảnh (Blur)</i>	27
<i>Hình 18: Kết quả</i>	28
<i>Hình 19: Ngôn ngữ kí hiệu</i>	29
<i>Hình 20: Biểu đồ Accuracy</i>	39
<i>Hình 21: Biểu đồ Loss</i>	40
<i>Hình 22: Kết quả thu được với chữ T</i>	43
<i>Hình 23: Kết quả thu được với chữ W</i>	43
<i>Hình 24: Kết quả thu được với chữ A</i>	44

I. Tổng quan lý thuyết

1. Khái niệm về Machine Learning

Học máy (Machine Learning – ML) là một lĩnh vực của trí tuệ nhân tạo (AI) tập trung vào việc thiết kế các hệ thống máy tính có khả năng học hỏi từ dữ liệu. Các kỹ thuật học máy phong phú cung cấp khả năng cải thiện hiệu suất của các ứng dụng phần mềm theo thời gian, dựa trên dữ liệu đầu vào.



Hình 1: Ảnh minh họa về Machine Learning

Quy trình làm việc của Machine Learning (Machine learning workflow):

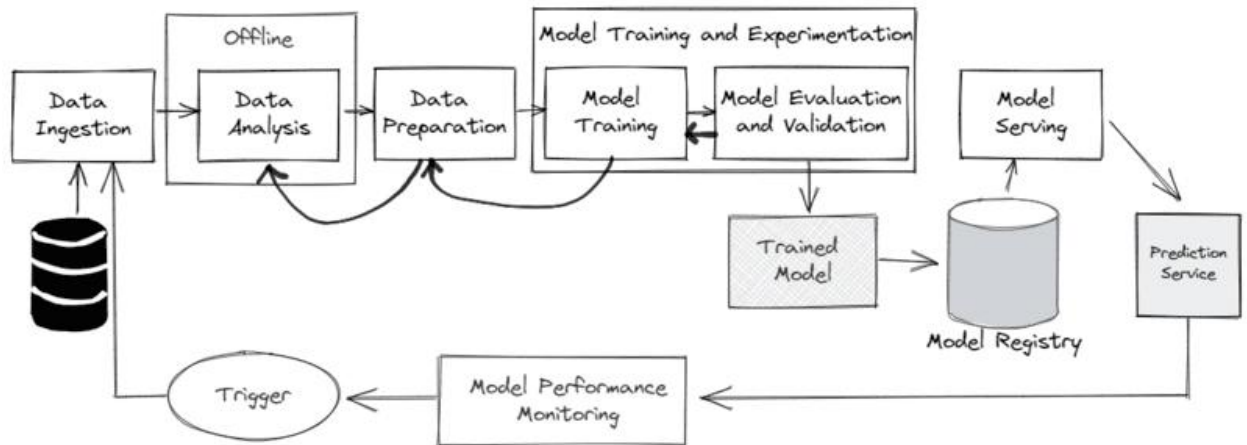
- Thu thập dữ liệu (Data Collection): Đây là bước đầu tiên, cần chuẩn bị một tập dữ liệu (dataset) để máy tính học. Dữ liệu có thể được tự thu thập hoặc sử dụng các tập dữ liệu đã được công bố. Quan trọng là dữ liệu phải đến từ các nguồn đáng tin cậy để đảm bảo tính chính xác, giúp mô hình học hiệu quả hơn.

- Tiền xử lý dữ liệu (Preprocessing): Bước này nhằm chuẩn hóa dữ liệu, loại bỏ các thuộc tính không cần thiết, gán nhãn, mã hóa đặc trưng, trích xuất đặc trưng hoặc giảm kích thước dữ liệu mà vẫn đảm bảo kết quả chính xác. Đây là bước tốn thời gian nhất, thường chiếm hơn 70% tổng thời gian thực hiện quy trình, đặc biệt khi làm việc với lượng dữ liệu lớn.

- Huấn luyện mô hình (Training Model): Ở bước này, mô hình được huấn luyện trên dữ liệu đã qua xử lý. Mục tiêu là giúp mô hình “học” từ dữ liệu để thực hiện các dự đoán hoặc phân loại.

- Đánh giá mô hình (Evaluating Model): Sau khi huấn luyện, mô hình cần được đánh giá bằng cách sử dụng các chỉ số đánh giá (metrics) khác nhau. Độ chính xác trên 80% thường được coi là một mô hình tốt, nhưng các tiêu chí cụ thể có thể thay đổi tùy vào ứng dụng.

- Cải thiện mô hình (Improve): Nếu độ chính xác của mô hình chưa đạt kỳ vọng, bạn cần cải thiện bằng cách điều chỉnh hoặc huấn luyện lại. Quy trình này lặp lại từ bước 3 cho đến khi đạt được kết quả mong muốn. Ba bước cuối thường chiếm khoảng 30% tổng thời gian thực hiện.



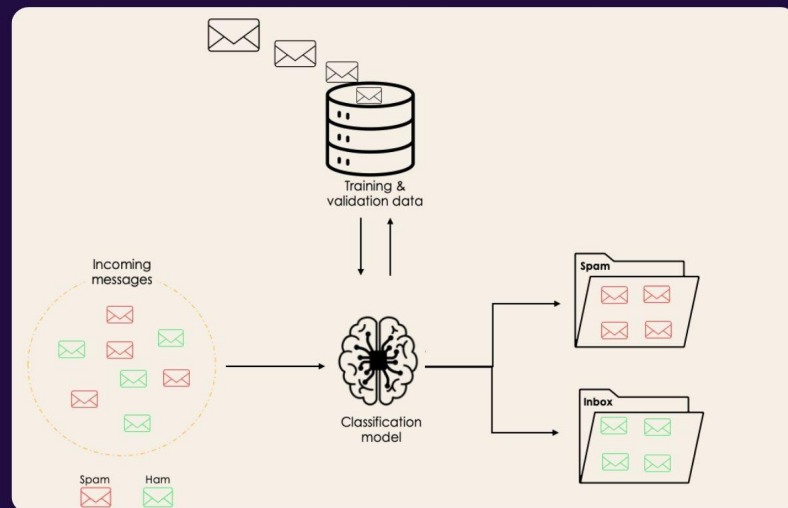
Hình 2: Machine learning workflow

1.1 Phân loại (Classification)

Phân loại trong machine learning là một tác vụ quan trọng, trong đó mô hình học máy được huấn luyện để phân biệt và gán nhãn cho các đối tượng vào một hoặc nhiều lớp dựa trên các đặc điểm của chúng. Mục tiêu của phân loại là dự đoán lớp chính xác của một đối tượng chưa biết dựa trên kiến thức đã học từ một tập dữ liệu đã được gán nhãn. Quá trình này đóng một vai trò thiết yếu trong việc giải quyết nhiều vấn đề thực tế, từ phát hiện gian lận trong giao dịch tài chính, phân loại email là thư rác hay không, đến chẩn đoán y tế.

Tác vụ phân loại có thể được phân loại thành hai loại chính: phân loại nhị phân, nơi chỉ có hai lớp đầu ra có thể (ví dụ, phát hiện email thư rác: thư rác hoặc không thư rác), và phân loại đa lớp, nơi có nhiều hơn hai lớp đầu ra (ví dụ, nhận dạng chữ viết tay: phân loại mỗi bức ảnh thành một trong 10 chữ số từ 0 đến 9).

Classification in Machine Learning



Hình 3: Phân loại lớp

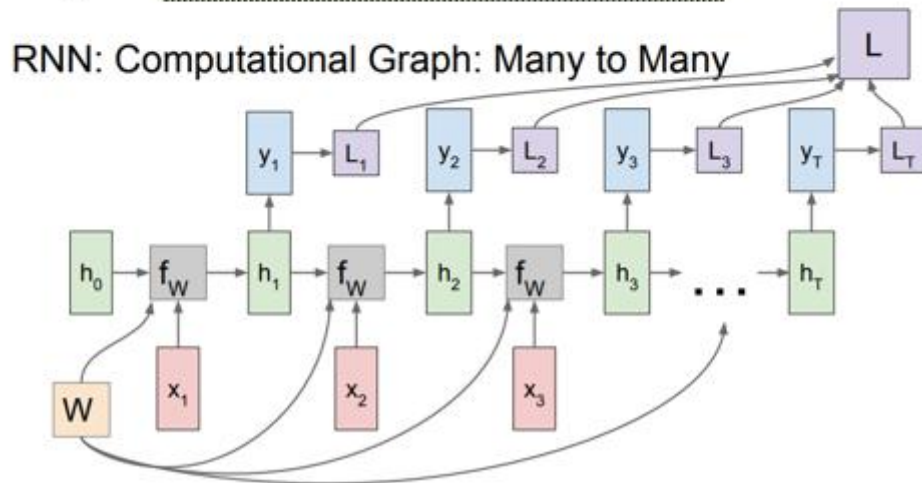
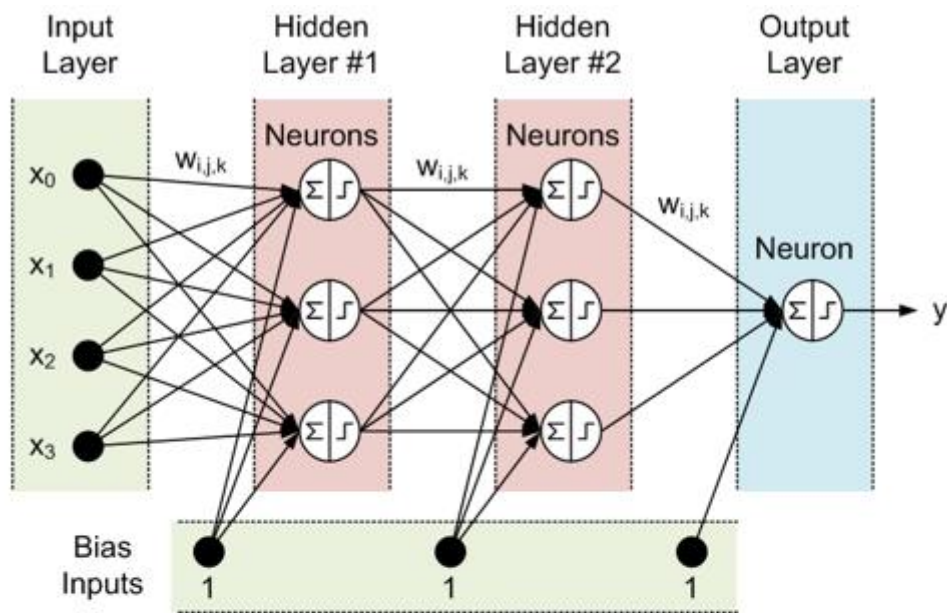
1.2 Hồi quy (Regression)

Hồi quy là kỹ thuật thống kê dùng để ước lượng phương trình phù hợp nhất với các tập hợp kết quả quan sát của biến phụ thuộc và biến độc lập. Nó cho phép đạt được kết quả ước lượng tốt nhất về mối quan hệ chân thực giữa các biến số.

1.2.1 Mô hình RNN + LSTM

- + RNN (Recurrent Neural Network) là mạng nơ-ron hồi quy thích hợp cho dữ liệu tuần tự như văn bản, âm thanh, và cử chỉ tay theo thời gian.

Neural Network bao gồm 3 phần chính là Input layer, Hidden layer và Output layer, ta có thể thấy là đầu vào và đầu ra của mạng neuron này là độc lập với nhau. RNN ra đời với ý tưởng chính là sử dụng một bộ nhớ để lưu lại thông tin từ những bước tính toán xử lý trước để dựa vào nó có thể đưa ra dự đoán chính xác nhất cho bước dự đoán hiện tại.



Hình 3: Mô hình RNN

Nếu như mạng Neural Network chỉ là input layer x đi qua hidden layer h và cho ra output layer y với *full connected* giữa các layer thì trong RNN, các input $x_{t:t}$ sẽ được kết hợp với hidden layer h_{t-1} bằng hàm f_W để tính toán ra hidden layer h_t hiện tại và output y_t sẽ được tính ra từ h_t , W là tập các trọng số và nó được ở tất cả các cụm, các $L1, L2, \dots, L_t$ là các hàm mất mát sẽ được giải thích sau. Như vậy kết quả từ các quá trình tính toán trước đã được "nhớ" bằng cách kết hợp thêm h_{t-1} tính ra h_t để tăng độ chính xác cho những dự đoán ở hiện tại. Cụ thể quá trình tính toán được viết dưới dạng toán như sau:

$$h_t = f_W(h_{t-1}, x_t)$$

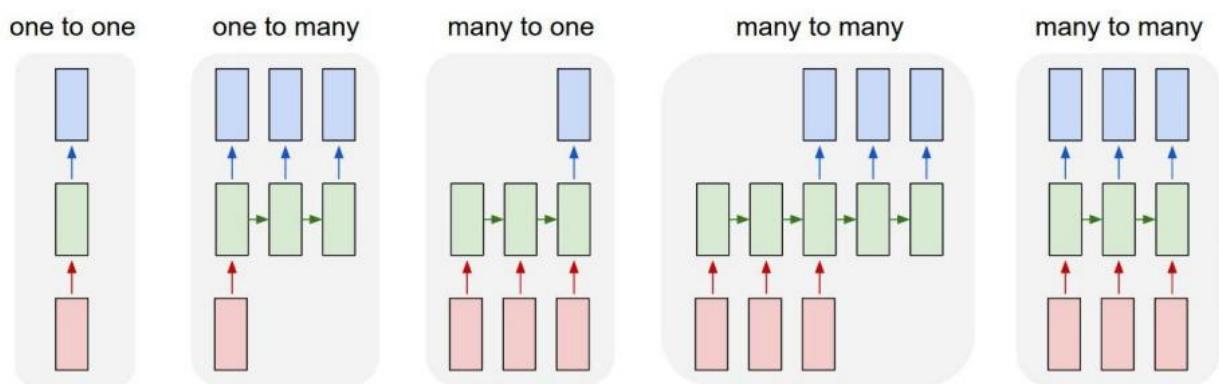
Hàm f_W chúng ta sẽ sử dụng hàm **tanh**, công thức trên sẽ trở thành :

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

Đến đây có 3 thứ mới xuất hiện: W_{xh} , W_{hh} , W_{hy} . Đối với mạng NN chỉ sử dụng một ma trận trọng số W duy nhất thì với RNN nó sử dụng 3 ma trận trọng số cho 2 quá trình tính toán: W_{hh} kết hợp với "bộ nhớ trước" h_{t-1} và W_{xh} kết hợp với x_t để tính ra "bộ nhớ của bước hiện tại" h_t từ đó kết hợp với W_{hy} để tính ra y_t .

Ngoài mô hình Many to Many như ta thấy ở trên thì RNN còn rất nhiều dạng khác như sau:

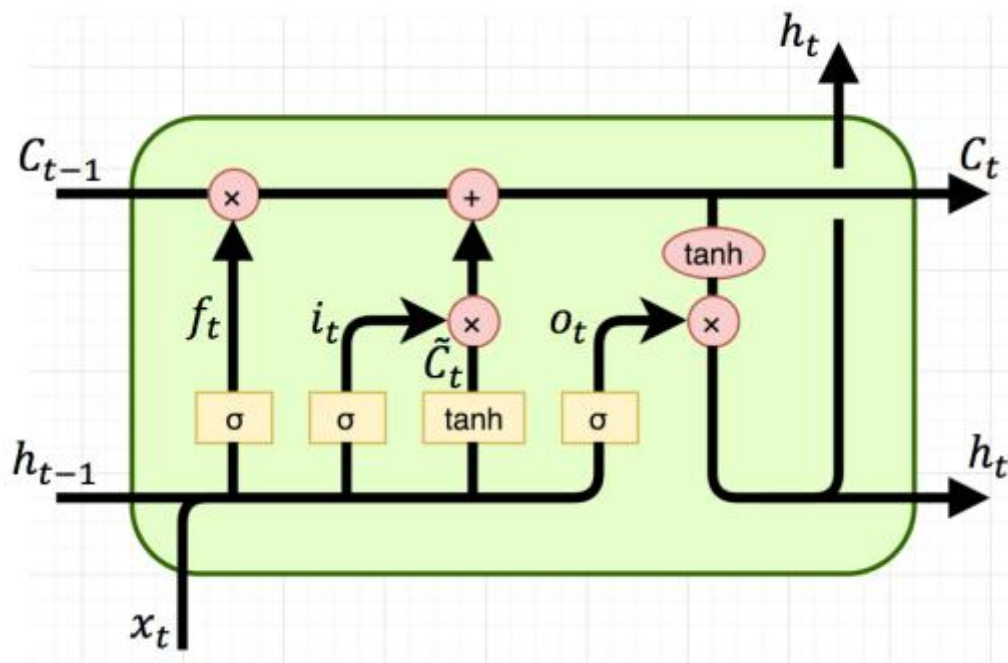


Hình 4: Các mô hình khác

- + LSTM (Long Short-Term Memory) là một loại mạng nơ-ron hồi quy (RNN) được thiết kế để giải quyết vấn đề mất nhớ (vanishing gradient) khi xử lý chuỗi dữ liệu dài.

Ở state thứ t của mô hình LSTM:

- Output: c_t, h_t , ta gọi c là cell state, h là hidden state.
- Input: c_{t-1}, h_{t-1}, x_t . Trong đó x_t là input ở state thứ t của model. c_{t-1}, h_{t-1} là output của layer trước. h đóng vai trò khá giống như s ở RNN, trong khi c là điểm mới của LSTM.



Hình 5: Mô hình: LSTM (Long Short-Term Memory)

Công thức chính:

f_t, i_t, o_t tương ứng với forget gate, input gate và output gate.

- Forget gate: $f_t = \sigma(U_f * x_t + W_f * h_{t-1} + b_f)$

- Input gate: $i_t = \sigma(U_i * x_t + W_i * h_{t-1} + b_i)$

- Output gate: $o_t = \sigma(U_o * x_t + W_o * h_{t-1} + b_o)$

Nhận xét: $0 < f_t, i_t, o_t < 1$; b_f, b_i, b_o là các hệ số bias; hệ số W, U giống như trong bài RNN.

$ct \sim = \tanh(Uc * xt + Wc * ht - 1 + bc)$ bước này giống hệt như tính st trong RNN.

$ct = ft * ct - 1 + it * ct \sim$ forget gate quyết định xem cần lấy bao nhiêu từ cell state trước và input gate sẽ quyết định lấy bao nhiêu từ input của state và hidden layer của layer trước.

$ht = ot * \tanh(ct)$ output gate quyết định xem cần lấy bao nhiêu từ cell state để trở thành output của hidden state. Ngoài ra ht cũng được dùng để tính ra output yt cho state t .

1.2.2 Transformer + Attention Mechanism

- Transformer: Là một kiến trúc mạng nơ-ron dựa hoàn toàn vào cơ chế attention, giúp mô hình tập trung vào các phần quan trọng của chuỗi dữ liệu, bất kể khoảng cách giữa chúng. Điều này đặc biệt hữu ích khi xử lý các chuỗi dài như trong nhận diện ngôn ngữ ký hiệu.

- Cơ chế Attention: Cho phép mô hình đánh giá và tập trung vào các phần quan trọng của dữ liệu đầu vào, giúp cải thiện hiệu suất trong việc nhận diện các mẫu cử chỉ phức tạp.

Quy trình kết hợp Transformer và Attention trong nhận diện ngôn ngữ ký hiệu bao gồm:

Bước 1: Trích xuất đặc trưng: Sử dụng các công cụ như MediaPipe hoặc OpenPose để trích xuất các điểm đặc trưng từ video cử chỉ.

Bước 2: Mã hóa chuỗi cử chỉ: Sử dụng bộ mã hóa (encoder) của Transformer để chuyển đổi chuỗi đặc trưng thành biểu diễn không gian tiềm ẩn.

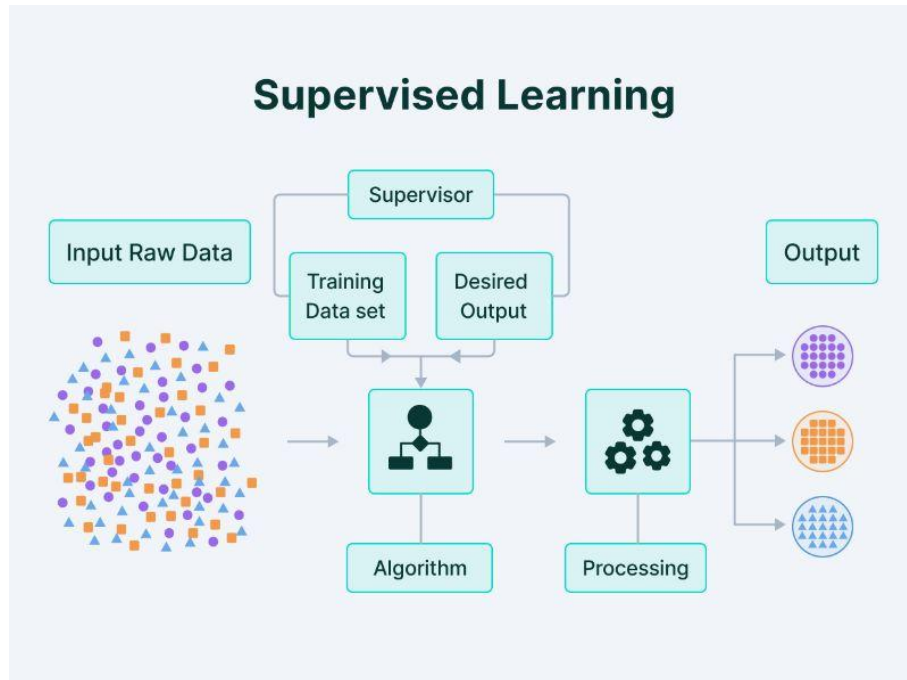
Bước 3: Giải mã và dự đoán: Bộ giải mã (decoder) của Transformer sử dụng thông tin từ encoder để dự đoán ký hiệu tương ứng.

1.3 Học có giám sát và không giám sát

1.3.1 Học có giám sát (Supervised Learning)

Học có giám sát được sử dụng rộng rãi trong việc nhận diện và phân loại các ký hiệu ngôn ngữ dựa trên dữ liệu có nhãn, giúp đạt được độ chính xác cao trong các bài toán phân loại cụ thể.

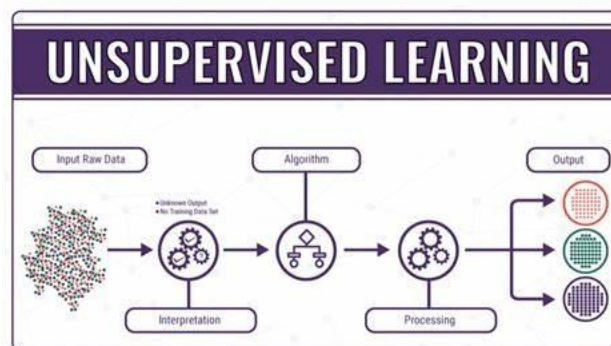
Thường được sử dụng cho các bài toán về phân loại (Classification) và Hồi quy (Regression).



Hình 6: Học có giám sát

1.3.2 Học không giám sát (Unsupervised Learning)

Học không giám sát là một kỹ thuật học máy trong đó mô hình tự học từ dữ liệu không có nhãn. Nó cho phép mô hình khám phá các mẫu và thông tin mà không cần sự giám sát từ người dùng. Các thuật toán trong học không giám sát học từ dữ liệu chưa được gán nhãn và tự động tìm ra cấu trúc hoặc mẫu ẩn trong dữ liệu. Phương pháp này rất hữu ích trong việc xử lý khối lượng dữ liệu lớn mà không cần phải gán nhãn cho từng phần dữ liệu. Từ đó hỗ trợ trong việc phân tích và khai thác



thông tin

Hình 7: Học không giám sát

2. Quy trình huấn luyện mô hình

2.1 Chia dữ liệu thành tập huấn luyện (Training set) và tập kiểm thử (Test set)

Training set: Dùng để huấn luyện mô hình, giúp mô hình học được các quy luật từ dữ liệu.

Test set: Dùng để đánh giá hiệu suất mô hình, kiểm tra xem mô hình có thể tổng quát hóa tốt trên dữ liệu mới hay không.

Thông thường, dữ liệu được chia theo tỷ lệ 80% - 20% hoặc 70% - 30%:

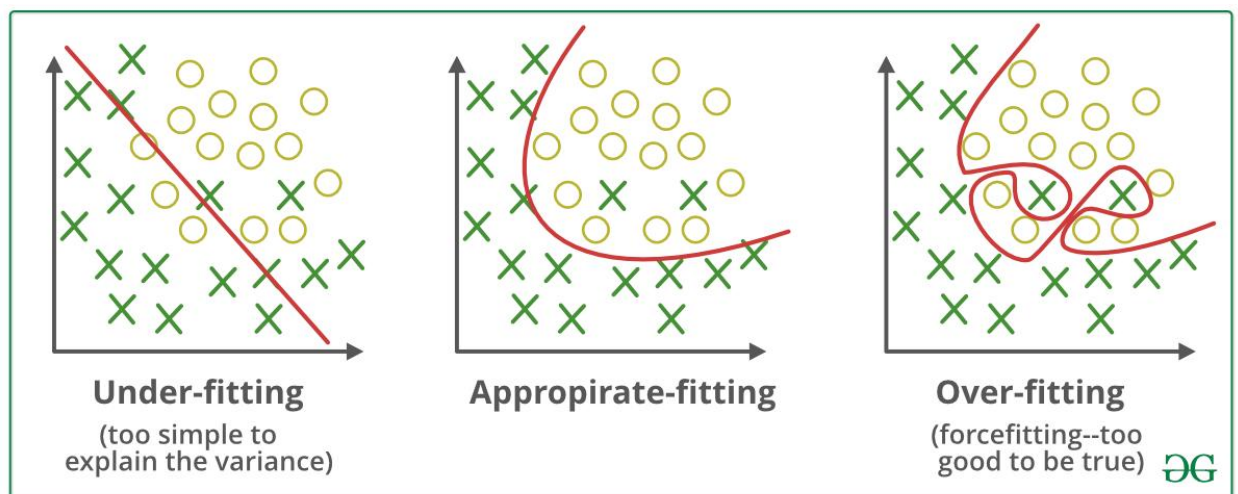
- 80% - 20%: 80% dữ liệu dùng để huấn luyện và 20% còn lại để kiểm tra.

- 70% - 30%: 70% dữ liệu để huấn luyện và 30% để kiểm tra, thường dùng khi dữ liệu có kích thước lớn.

2.2 Overfitting và underfitting và cách khắc phục

Overfitting: Mô hình học quá tốt trên dữ liệu huấn luyện nhưng không tổng quát hóa tốt trên dữ liệu mới. Dấu hiệu: Accuracy trên training set cao nhưng trên test set thấp.

Underfitting: Mô hình quá đơn giản, không học được đặc trưng quan trọng.



Dấu hiệu: Accuracy trên cả training set và test set đều thấp.

Hình 8: Overfitting và underfitting

Cách khắc phục:

- Overfitting:
 - + Data Augmentation: Tạo thêm dữ liệu bằng cách xoay, lật, thay đổi ánh sáng.
 - + Regularization (L1, L2): Giới hạn trọng số để tránh học quá sâu.
 - + Dropout: Tắt ngẫu nhiên một số neuron trong quá trình training.
- Underfitting:
 - + Dùng mô hình phức tạp hơn (ResNet, Transformer).
 - + Thêm nhiều epoch hoặc thay đổi learning rate.

3. Khái niệm về Python và OpenCV

3.1 Python

Python là ngôn ngữ lập trình máy tính bậc cao thường được sử dụng để xây dựng trang web và phần mềm, tự động hóa các tác vụ và tiến hành phân tích dữ liệu. Python là ngôn ngữ có mục đích chung, nghĩa là nó có thể được sử dụng để tạo nhiều chương trình khác nhau và không chuyên biệt cho bất kỳ vấn đề cụ thể nào. Ngôn ngữ phổ biến cho Machine Learning & Computer Vision.



Hình 9: Python

Các thư viện quan trọng:

- TensorFlow, PyTorch: Train mô hình.

- OpenCV: Xử lý ảnh.
- MediaPipe: Nhận diện bàn tay.

3.2 OpenCV

OpenCV là tên viết tắt của open source computer vision library – có thể được hiểu là một thư viện nguồn mở cho máy tính. Cụ thể hơn OpenCV là kho lưu trữ các mã nguồn mở được dùng để xử lý hình ảnh, phát triển các ứng dụng đồ họa trong thời gian thực.

OpenCV cho phép cải thiện tốc độ của CPU khi thực hiện các hoạt động real time. Nó còn cung cấp một số lượng lớn các mã xử lý phục vụ cho quy trình của thị giác máy tính hay các learning machine khác.



4. Detection

4.1 Object Detection (Phát hiện đối tượng)

- Dùng trong: Thị giác máy tính (computer vision)
- Mục tiêu: Tìm ra vị trí và loại của các đối tượng trong ảnh/video.
Ví dụ: Phát hiện xe, người, động vật trong ảnh.

Thuật toán thường dùng:

- CNN + vùng đề xuất: R-CNN, Fast R-CNN, Faster R-CNN
- One-stage detector: YOLO, SSD
- Two-stage detector: như Faster R-CNN
- Transformers: DETR

4.2 Anomaly Detection (Phát hiện bất thường)

- Dùng trong: Phát hiện gian lận, lỗi hệ thống, y tế...
- Mục tiêu: Nhận ra những mẫu không bình thường so với phần còn lại.
- Thuật toán thường dùng:
 - Statistical methods: z-score, Gaussian
 - Machine Learning: Isolation Forest, One-Class SVM
 - Deep Learning: Autoencoder, GAN

4.3 Intrusion Detection (Phát hiện xâm nhập mạng)

- Dùng trong: Bảo mật hệ thống
- Mục tiêu: Phát hiện các hoạt động trái phép, như hacker xâm nhập
- Thuật toán thường dùng: Decision Tree, SVM, kNN, Deep Learning

4.4 Face Detection / Emotion Detection

Phát hiện khuôn mặt / cảm xúc từ ảnh.

Sử dụng Haar Cascade, CNN, MTCNN, hoặc các mô hình mới như RetinaFace, YOLO-face.

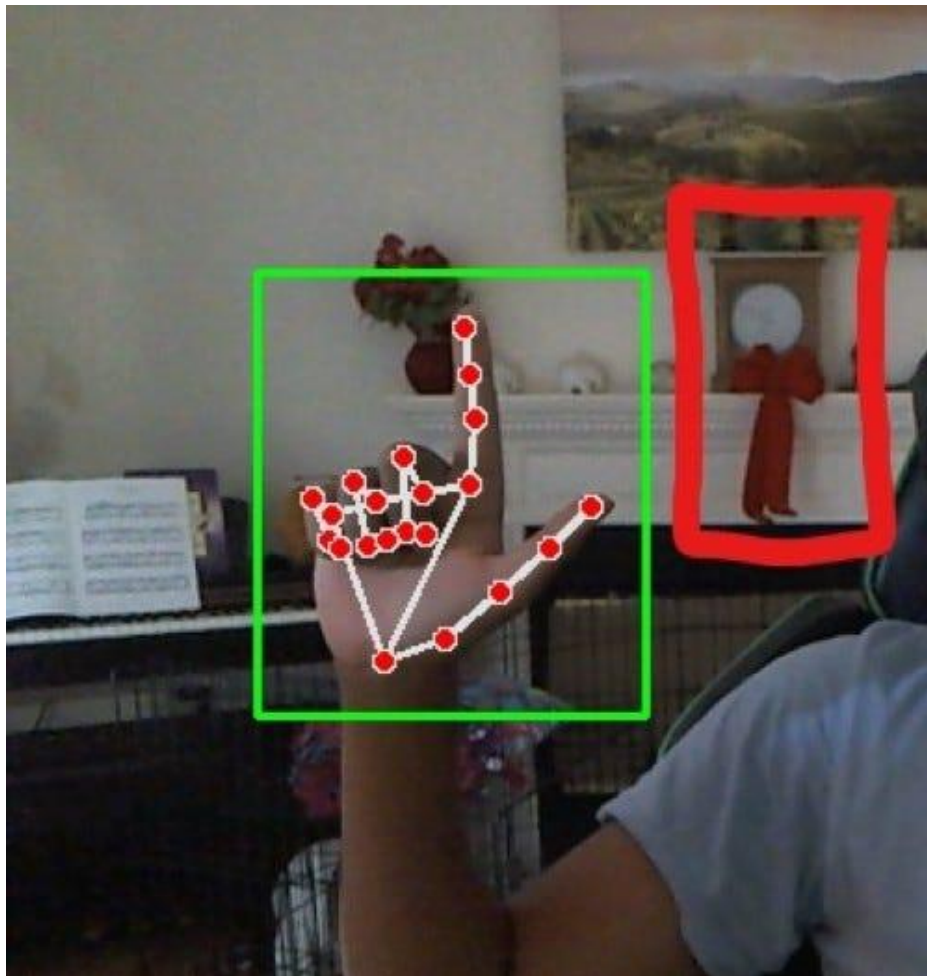
4.5 Keyword Spotting / Voice Activity Detection

Trong xử lý tiếng nói: Phát hiện từ khóa hoặc đoạn có giọng nói trong âm thanh.

II. Tổng quan về đề tài

Ngôn ngữ ký hiệu là phương tiện giao tiếp chính của cộng đồng người khiếm thính hoặc khiếm ngôn. Tuy nhiên, rào cản ngôn ngữ giữa họ và cộng đồng nói chung vẫn là một vấn đề lớn. Việc nghiên cứu và phát triển hệ thống phát hiện (detection) và nhận diện ngôn ngữ ký hiệu có thể tạo ra cầu nối giao tiếp hiệu quả giữa người khiếm thính và người bình thường. Ứng dụng trong các dịch vụ công cộng, giáo dục, chăm sóc y tế, truyền thông...Góp phần thúc đẩy sự hòa nhập và công bằng xã hội.

Trong Sign Language Detection, Machine Learning giúp nhận diện các kí hiệu ngôn ngữ bằng cách phân tích hình ảnh hoặc video.sf



Hình 10: Nhận diện kí hiệu bằng phân tích hình ảnh hoặc video

1. Phân loại (Classification) trong Sign Language Detection:

Đây là một nhiệm vụ phân tích dữ liệu, tức là quá trình tìm kiếm một mô hình mô tả và phân biệt các lớp và khái niệm dữ liệu. Phân loại là vấn đề xác định một tập hợp các danh mục (quần thể con), một dữ liệu quan sát mới thuộc về loại nào, trên cơ sở một tập dữ liệu huấn luyện chứa các quan sát và các loại thành viên đã biết.

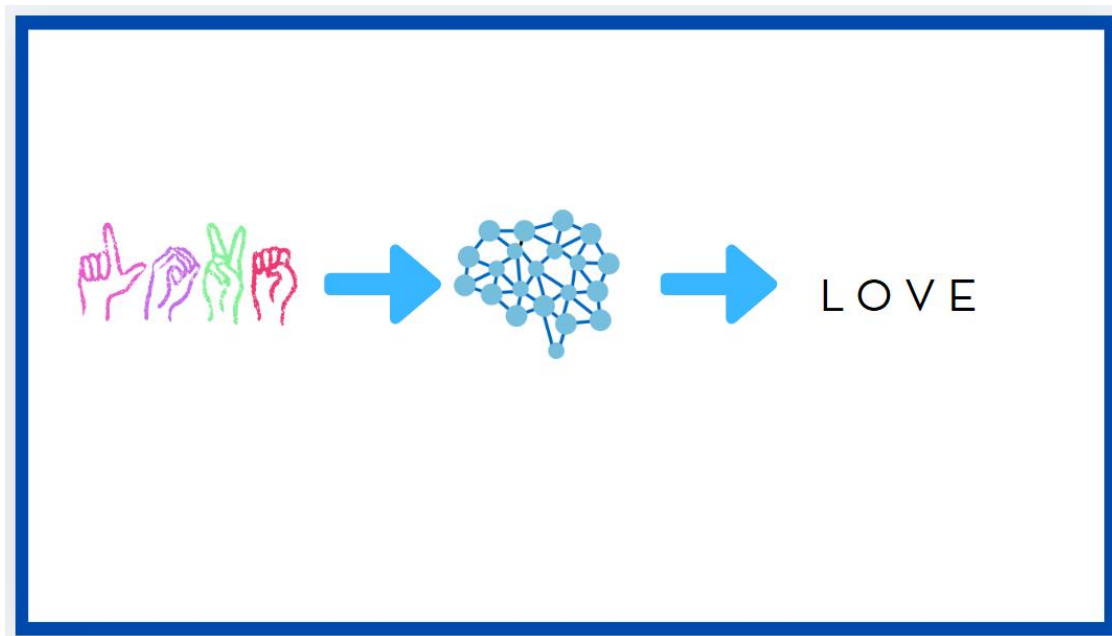
Trong Sign Language Detection:

- Bài toán nhận diện chữ cái trong ngôn ngữ kí hiệu:

Input: ảnh bàn tay đang tạo kí hiệu

Output: một ký tự tương ứng (A,B,C,...Z)

Mô hình phù hợp: CNN (Convolutional Neural Network), MobileNet, ResNet, EfficientNet.



Hình 11: Nhận diện ngôn ngữ kí hiệu

- Bài toán nhận diện từ hoặc câu từ cử chỉ:

Input: Video cử chỉ tay.

Output: Từ hoặc câu trong ngôn ngữ kí hiệu.

Mô hình phù hợp: RNN + LSTM, Transformer – based models.

1.1 Hồi quy (Regression) trong Sign Language Detection

Trong Sign Language Detection: để xác định vị trí tay trong không gian 3D hoặc dự đoán tốc độ chuyển động của tay trong một video cử chỉ ta sử dụng mô hình RNN + LSTM (dự đoán theo thời gian) / Transformer + Attention Mechanism (xử lý chuỗi dài).

1.1.1 Mô hình RNN + LSTM

RNN giúp xử lý dữ liệu chuỗi theo thời gian, nhưng gặp vấn đề vanishing gradient khi chuỗi quá dài.

LSTM là phiên bản cải tiến của RNN, giúp ghi nhớ lâu dài và giảm mất thông tin trong chuỗi.

Giải pháp: Dùng RNN làm bước tiền xử lý, sau đó sử dụng LSTM để xử lý thông tin dài hạn.

Quy trình kết hợp RNN và LSTM trong nhận diện ngôn ngữ kí hiệu:

Bước 1: Thu thập dữ liệu và trích xuất đặc trưng

Sử dụng MediaPipe hoặc OpenPose để trích xuất tọa độ keypoints của bàn tay từ video.

Biểu diễn mỗi cử chỉ thành chuỗi tọa độ theo thời gian (sequence).

Bước 2: Tiền xử lý dữ liệu

Chuỗi dữ liệu trích xuất từ video có dạng:

```
[ [x1, y1, x2, y2, ..., x21, y21], # Frame 1
  [x1, y1, x2, y2, ..., x21, y21], # Frame 2
  ...
]
```

Trong đó:

Mỗi hàng là một khung hình (frame) trong video.

Mỗi cột là một tọa độ keypoint trên bàn tay.

Bước 3: Xây dựng mô hình kết hợp RNN và LSTM

- Dùng RNN để xử lý dữ liệu ngắn hạn (các cử chỉ ngắn, chuyển động nhỏ).
- Dùng LSTM để học các mẫu cử chỉ dài hơn, giúp nhận diện từ hoặc câu đầy đủ.

1.1.2 Transformer + Attention Mechanism

Quy trình kết hợp Transformer và Attention trong nhận diện ngôn ngữ ký hiệu bao gồm:

Bước 1: Trích xuất đặc trưng: Sử dụng các công cụ như MediaPipe hoặc OpenPose để trích xuất các điểm đặc trưng từ video cử chỉ.

Bước 2: Mã hóa chuỗi cử chỉ: Sử dụng bộ mã hóa (encoder) của Transformer để chuyển đổi chuỗi đặc trưng thành biểu diễn không gian tiềm ẩn.

Bước 3: Giải mã và dự đoán: Bộ giải mã (decoder) của Transformer sử dụng thông tin từ encoder để dự đoán ký hiệu tương ứng.

1.2 Học có giám sát và không giám sát trong Sign Language Detection

Nhận diện chữ cái ASL từ hình ảnh: Sử dụng mạng nơ-ron tích chập (CNN) để phân loại các chữ cái trong American Sign Language (ASL) dựa trên hình ảnh.

- Thu thập & tiền xử lý dữ liệu: Chia ảnh thành train/test, chuẩn hóa ảnh.
- Xây dựng CNN: Dùng Convolution, Pooling, Dense layers.
- Huấn luyện mô hình: Dùng Adam Optimizer + Cross Entropy Loss.
- Đánh giá & Dự đoán: Kiểm tra độ chính xác, thử nghiệm với ảnh mới.

❖ Học không giám sát (Unsupervised Learning)

Nhóm các mẫu ký hiệu tay có hình dạng tương tự nhau: Sử dụng các kỹ thuật học không giám sát để nhóm các cử chỉ ký hiệu tay dựa trên sự tương đồng về hình dạng và chuyển động.

- Thu thập & Tiền xử lý dữ liệu: Chuyển đổi ảnh về dạng chuẩn, trích xuất đặc trưng HOG.
 - Trích xuất đặc trưng với Autoencoder: Mã hóa ảnh thành không gian tiềm ẩn (latent space).
 - Phân cụm với K-Means hoặc DBSCAN: Nhóm các mẫu cử chỉ tay có hình dạng tương tự.
 - Đánh giá cụm bằng cách trực quan hóa ảnh trong từng nhóm.
- ❖ Học có giám sát (Supervised Learning)
- Phân cụm cử chỉ tay (Gesture Clustering): Tìm ra các nhóm cử chỉ tay tương tự nhau mà không cần biết nghĩa của chúng. Dùng K-means, DBSCAN, hoặc self-organizing maps (SOM). Giúp xây dựng từ điển ký hiệu tự động từ dữ liệu video lớn. Hỗ trợ các nhà ngôn ngữ học phân tích các biến thể ngôn ngữ ký hiệu trong vùng miền.
 - Học biểu diễn (Representation Learning) bằng Autoencoder / Contrastive Learning: Trích xuất đặc trưng không gian tiềm ẩn từ video cử chỉ tay. Autoencoder / Variational Autoencoder (VAE) để học biểu diễn. SimCLR / MoCo / BYOL để học đặc trưng phân biệt giữa các cử chỉ khác nhau. Làm bước tiền xử lý cho mô hình nhận diện từ/câu ký hiệu. Tăng hiệu quả học khi dữ liệu có nhãn rất ít (dùng kết hợp với học bán giám sát).
 - Segment hành động ký hiệu trong video liên tục (Unsupervised Action Segmentation): Chia video ký hiệu dài thành các đoạn hành động riêng biệt, tương ứng với từng từ/câu. Dùng HMM (Hidden Markov Models) hoặc self-supervised learning dựa trên thời gian và sự thay đổi động tác. Phân tách từng từ ký hiệu trong một câu để huấn luyện mô hình nhận diện từ khóa. Tự động đánh dấu dữ liệu video ký hiệu chưa gán nhãn.
 - Phát hiện các mẫu mới (New Sign Discovery): Phát hiện các cử chỉ mới chưa từng xuất hiện trong tập huấn luyện. Dùng anomaly detection hoặc clustering kết hợp biểu diễn động học. Mở rộng hệ thống để nhận diện thêm các từ mới mà không cần huấn luyện lại từ đầu. Hữu ích cho hệ thống nhận diện ngôn ngữ ký hiệu quy mô lớn.

2. Quy trình huấn luyện mô hình trong Sign Language Detection

2.1 Chia dữ liệu thành tập huấn luyện (Training set) và tập kiểm thử

Bước 1: Thu thập dữ liệu

- Bộ dữ liệu có sẵn:
 - + American Sign Language (ASL) Dataset.
 - + Sign Language MNIST (chữ cái đơn giản).
 - + RWTH-PHOENIX-Weather 2014T (dữ liệu video).
- Tạo bộ dữ liệu riêng bằng OpenCV + MediaPipe.

Bước 2: Tiền xử lý dữ liệu

- Resize ảnh về kích thước chuẩn (224x224 hoặc 128x128).
- Loại bỏ nhiễu (Gaussian Blur, Histogram Equalization).
- Chuyển đổi ảnh sang ảnh xám hoặc chuẩn hóa màu.

Bước 3: Chia dữ liệu thành tập huấn luyện và tập kiểm thử

- Training set (80%): Dùng để huấn luyện mô hình.
- Test set (20%): Đánh giá mô hình sau huấn luyện.

Bước 4: Huấn luyện mô hình

- Với dữ liệu hình ảnh: CNN, MobileNet, ResNet.
- Với dữ liệu video: RNN + LSTM, Transformer.

Bước 5: Kiểm tra và đánh giá mô hình

- Độ chính xác (Accuracy, Precision, Recall).
- Đánh giá bằng F1-score, Confusion Matrix.

2.2 Overfitting và underfitting trong Sign Language Detection và cách khắc phục

Overfitting trong sign language detection:

- Mô hình nhận diện rất tốt các cử chỉ tay đã thấy trong tập huấn luyện.
- Nhưng khi gặp các biến thể (ánh sáng khác, người khác, góc quay khác), mô hình dự đoán sai.

❖ Nguyên nhân thường gặp:

- Dữ liệu huấn luyện không đủ đa dạng
- Mô hình quá phức tạp (nhiều lớp, nhiều tham số)
- Huấn luyện quá lâu (many epochs)

Underfitting trong Sign Language Detection:

- Mô hình không thể phân biệt được rõ các ký hiệu tay → kết quả kém cả trên dữ liệu đã học.

Nguyên nhân thường gặp:

- Mô hình quá đơn giản (ít lớp, thiếu tham số)
- Không huấn luyện đủ lâu
- Thiếu xử lý đặc trưng (feature engineering kém, đầu vào mờ, chưa chuẩn hóa...)

❖ Cách khắc phục:

Overfitting

Phương pháp	Mô tả
Data Augmentation	Biến đổi ảnh (xoay, đổi sáng, nhiễu,...) để mô phỏng đa dạng điều kiện thực tế ⇒ Dùng để tạo nhiều phiên bản cử chỉ tay
Dropout	Bỏ ngẫu nhiên một số neurons khi training để tránh phụ thuộc quá mức ⇒ Dùng trong CNN/RNN
Regularization (L2)	Thêm penalty vào loss function để hạn chế trọng số quá lớn ⇒ Dùng trong training
Early Stopping	Dừng huấn luyện sớm nếu loss không cải thiện ⇒ Tránh học quá sâu gây overfit
Thêm dữ liệu	Thu thập thêm nhiều mẫu từ người dùng khác nhau ⇒ Cực kỳ hiệu quả

Underfitting

Phương pháp	Mô tả
Tăng độ phức tạp mô hình	Thêm layers, filters, sử dụng mô hình mạnh hơn ⇒ CNN → ResNet, MobileNet,...

Huấn luyện lâu hơn	Tăng số epoch, batch size hợp lý ⇒ Kết hợp với early stopping
Chọn đặc trưng đầu vào tốt hơn	Lọc vùng tay chính xác (bounding box, segmentation) ⇒ Dùng MediaPipe, YOLO,...
Tiền xử lý kỹ ảnh	Làm rõ tay, chuẩn hóa ảnh, loại bỏ nhiễu ⇒ Dùng OpenCV hoặc augmentation tốt

- Nếu làm theo ảnh tĩnh:
→ Dùng YOLOv8 + CNN classification + Data Augmentation
- Nếu làm theo video động:
→ Dùng MediaPipe + LSTM hoặc 3D CNN

❖ Ứng dụng của Open CV đối với Sign Language Detection

- Tách bàn tay từ nền bằng Skin Detection.
- Phát hiện contour của bàn tay bằng Thresholding.
- Tracking tay trong video bằng Background Subtraction

❖ Detection trong Sign Language Detection

Detection là quá trình xác định vị trí của bàn tay trong ảnh hoặc video.

Các kỹ thuật ứng dụng:

- Hand Detection (Nhận diện bàn tay)
 - + MediaPipe Hands, OpenPose (Xác định vị trí bàn tay).
 - + OpenCV Haar Cascade, YOLO, SSD (Object Detection).
- Hand Pose Estimation (Ước lượng khớp tay)
 - + Dùng 21 khớp tay từ MediaPipe Hands.
 - + Dự đoán chuyển động ngón tay trong không gian 3D.
- Gesture Recognition (Nhận diện cử chỉ)
 - + Dùng CNN để nhận diện hình dạng tay.
 - + Dùng RNN để nhận diện cử chỉ động.

III. Xử lý tiền dữ liệu

1. Thu thập dữ liệu

Dữ liệu trong đề tài tập trung vào dữ liệu hình ảnh thủ ngữ, cụ thể là những hình ảnh về chữ cái không dấu trong bảng chữ cái thủ ngữ Tiếng anh. Các cử chỉ tay tương ứng với chữ cái được thu thập từ ASL Alphabet

Bộ dữ liệu bao gồm 26 chữ cái không dấu, tương ứng với các ký hiệu trong bảng chữ cái thủ ngữ Tiếng anh, không bao gồm các biến thể ký hiệu khác. Dữ liệu được xử lý và gán nhãn theo từng chữ cái để phục vụ cho quá trình huấn luyện mô hình nhận dạng.

Dữ liệu hình ảnh phục vụ cho việc huấn luyện mô hình nhận diện thủ ngữ được thu thập dưới dạng ảnh tĩnh trực tiếp thông qua webcam của máy tính. Ngôn ngữ lập trình được sử dụng trong quá trình này là Python, cùng với sự hỗ trợ của thư viện OpenCV.

Cv2: Là thư viện OpenCV (Open Source Computer Vision Library), dùng để xử lý ảnh, video, camera, ví dụ như nhận diện khuôn mặt, lọc ảnh, vẽ hình, v.v. Dùng nhiều trong trí tuệ nhân tạo, machine learning, và thị giác máy tính (computer vision).

Numpy (thường được import là np): Thư viện tính toán số học và mảng nhiều chiều (array). Dùng cho xử lý dữ liệu số lượng lớn, nền tảng cho nhiều thư viện như pandas, scikit-learn, opencv, v.v.

Os: Thư viện chuẩn (standard library) trong Python. Dùng để làm việc với hệ điều hành: truy cập thư mục, tập tin, biến môi trường...

Random: Cũng là thư viện chuẩn của Python. Dùng để tạo ra các giá trị ngẫu nhiên, chọn ngẫu nhiên phần tử từ list, sinh số ngẫu nhiên, v.v.

String: Thư viện chuẩn, chứa các hằng số liên quan đến chuỗi, ví dụ: string.ascii_letters, string.digits, v.v. Dùng khi cần xử lý ký tự.

2. Quá trình xây dựng code

Ban đầu, tạo những hàm tiền xử lý dữ liệu bao gồm:

- Xoay ảnh ngẫu nhiên (-30 đến 30 độ)

```
# Xoay ảnh ngẫu nhiên (-30 đến 30 độ)
angle = random.randint(-30, 30)
h, w = image.shape[:2]
M = cv2.getRotationMatrix2D((w // 2, h // 2), angle, 1)
image = cv2.warpAffine(image, M, (w, h))
```

Hình 12: Xoay ảnh ngẫu nhiên (-30 đến 30 độ)

```
# Lật ảnh (ngẫu nhiên lật ngang hoặc không)
if random.random() > 0.5:
    image = cv2.flip(image, 1)
```

- Lật ảnh (ngẫu nhiên lật ngang hoặc không)

Hình 13: Lật ảnh (ngẫu nhiên lật ngang hoặc không)

```
# Dịch ảnh (Translation)
tx, ty = random.randint(-10, 10), random.randint(-10, 10)
M = np.float32([[1, 0, tx], [0, 1, ty]])
image = cv2.warpAffine(image, M, (w, h))
```

- Dịch ảnh (Translation)

Hình 14: Dịch ảnh (Translation)

- Điều chỉnh độ sáng và độ tương phản

```
# Điều chỉnh độ sáng và độ tương phản
alpha = random.uniform(0.8, 1.2)
beta = random.randint(-30, 30)
image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)
```

Hình 15: Điều chỉnh độ sáng và độ tương phản

```
# Thêm nhiễu Gaussian
if random.random() > 0.5:
    noise = np.random.normal(0, 10, image.shape).astype(np.uint8)
    image = cv2.add(image, noise)
```

- Thêm nhiễu Gaussian

Hình 16: Thêm nhiễu Gaussian

- Làm mờ ảnh (Blur)

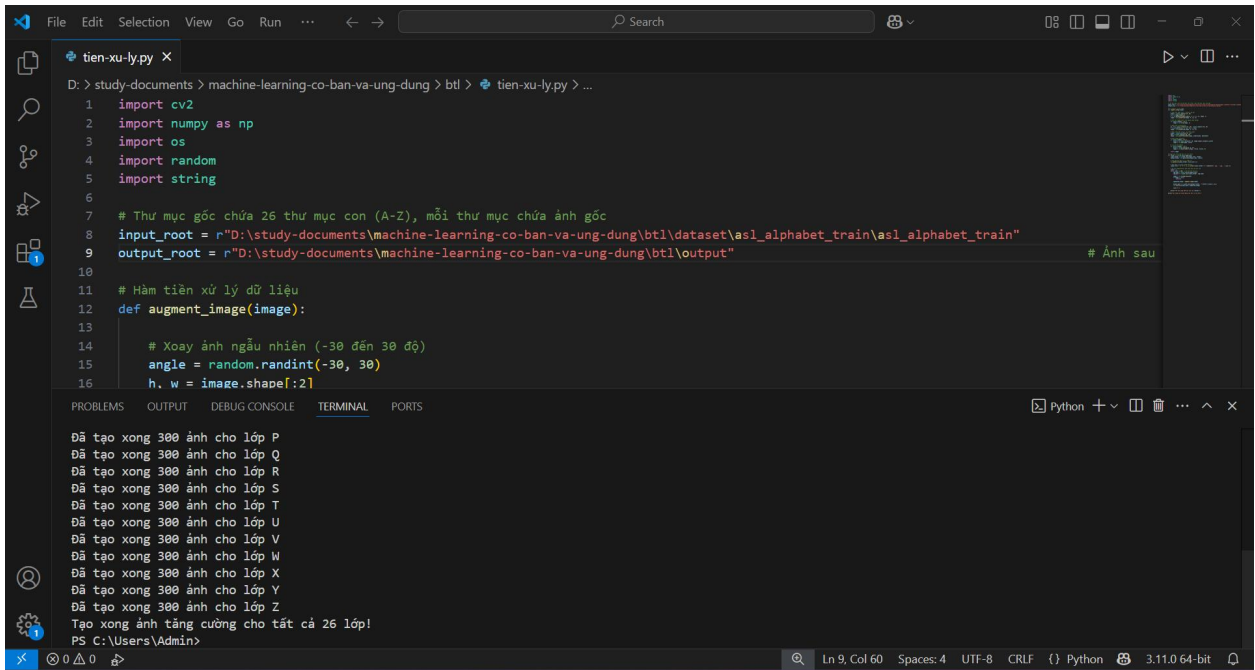
```
# Làm mờ ảnh (Blur)
if random.random() > 0.5:
    ksize = random.choice([3, 5, 7])
    image = cv2.GaussianBlur(image, (ksize, ksize), 0)

return image
```

Hình 17: Làm mờ ảnh (Blur)

Sau đó, duyệt qua từng thư mục chữ cái (A-Z) → Tạo thư mục chứa đầu ra (output) → Đọc danh sách ảnh trong thư mục → Tạo augmentation cho mỗi lớp (300 ảnh) → Quá trình chạy xong thì in ra màn hình dòng chữ “Đã tạo xong 300 ảnh cho lớp ...” và dòng chữ chạy xong tất cả 26 chữ cái “Tạo xong ảnh tăng cường cho tất cả 26 lớp”.

3. Kết quả chạy code



The image shows a Visual Studio Code editor window with a Python file named `tien-xu-ly.py` open. The script is located at `D:\study-documents\machine-learning-co-ban-va-ung-dung> btl > tien-xu-ly.py > ...`. The code imports `cv2`, `numpy`, `os`, `random`, and `string`. It defines `input_root` and `output_root` paths. A function `augment_image(image)` is defined, which rotates a 300x300 image by a random angle between -30 and 30 degrees. The terminal output shows the execution of the script, which creates 300 images for each of the 26 classes (A-Z).

```

1 import cv2
2 import numpy as np
3 import os
4 import random
5 import string
6
7 # Thư mục gốc chứa 26 thư mục con (A-Z), mỗi thư mục chứa ảnh gốc
8 input_root = r"D:\study-documents\machine-learning-co-ban-va-ung-dung\btl\dataset\asl_alphabet_train\asl_alphabet_train"
9 output_root = r"D:\study-documents\machine-learning-co-ban-va-ung-dung\btl\output" # Ảnh sau
10
11 # Hàm tiền xử lý dữ liệu
12 def augment_image(image):
13
14     # Xoay ảnh ngẫu nhiên (-30 đến 30 độ)
15     angle = random.randint(-30, 30)
16     h, w = image.shape[:2]

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Dã tạo xong 300 ảnh cho lớp P
 Đã tạo xong 300 ảnh cho lớp Q
 Đã tạo xong 300 ảnh cho lớp R
 Đã tạo xong 300 ảnh cho lớp S
 Đã tạo xong 300 ảnh cho lớp T
 Đã tạo xong 300 ảnh cho lớp U
 Đã tạo xong 300 ảnh cho lớp V
 Đã tạo xong 300 ảnh cho lớp W
 Đã tạo xong 300 ảnh cho lớp X
 Đã tạo xong 300 ảnh cho lớp Y
 Đã tạo xong 300 ảnh cho lớp Z
 Tạo xong ảnh tăng cường cho tất cả 26 lớp!
 PS C:\Users\Admin>

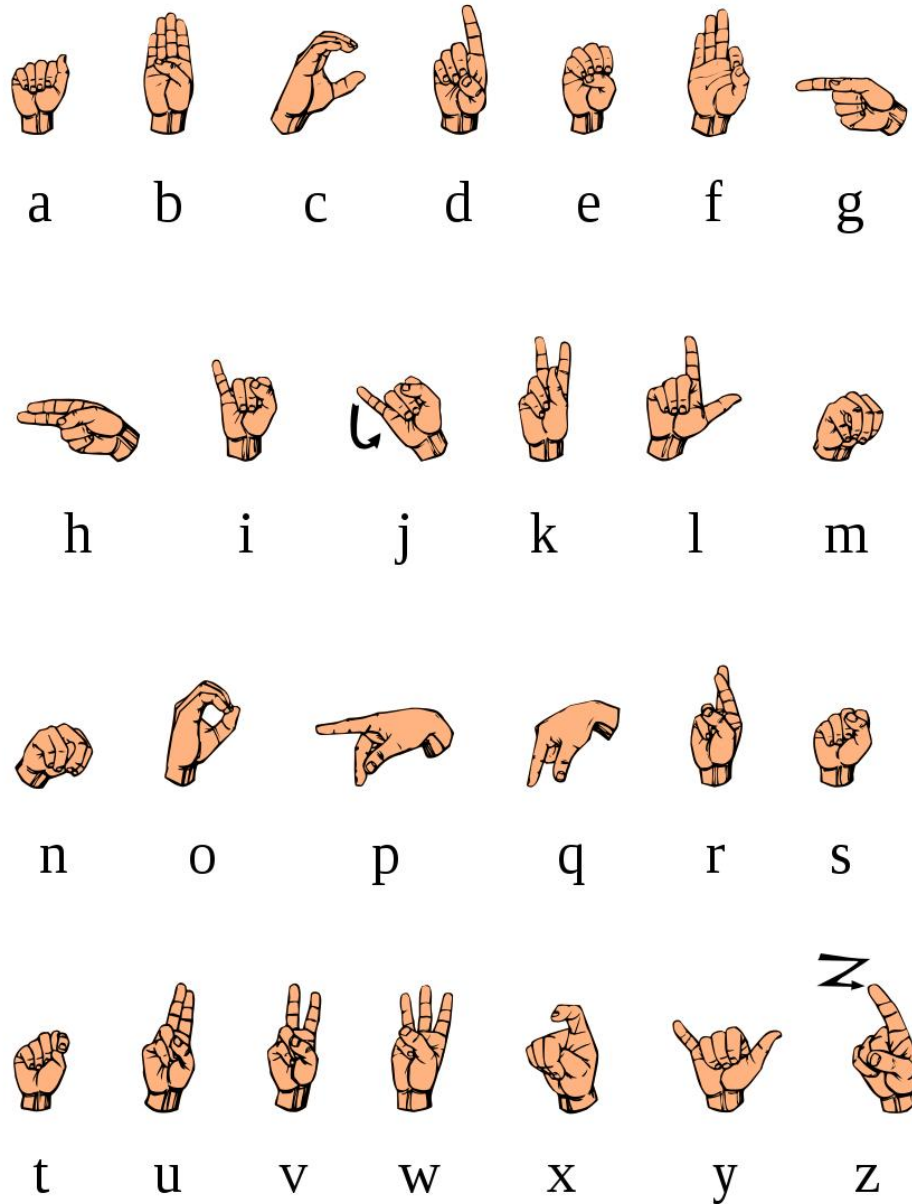
Ln 9, Col 60 Spaces: 4 UTF-8 CRLF Python 3.11.0 64-bit

Hình 18: Kết quả

IV. Thực hiện đề tài

1. Thu thập dữ liệu

1.1 Dữ liệu mẫu



Hình 19: Ngôn ngữ kí hiệu

2. Training Model trên Google Colab

2.1 Các bước thực hiện

❖ Cài đặt thư viện cần thiết:

```
!pip install mediapipe==0.10.9 opencv-python numpy scikit-learn tensorflow
matplotlib
```

❖ Upload file dataset.zip lên GColab:

```
from google.colab import files
uploaded = files.upload()
```

❖ Unzip file dataset:

```
import zipfile
import os
zip_path = '/content/dataset.zip'
extract_folder = '/content/'
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_folder)
```

❖ Import các thư viện đã tải và khởi tạo đối tượng nhận diện bàn tay:

```
import os
import cv2
import numpy as np
import tensorflow as tf
from mediapipe.python.solutions import hands as mp_hands
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
#Khởi tạo đối tượng nhận diện bàn tay
hands = mp_hands.Hands(static_image_mode=True,
max_num_hands=1)
```

❖ Xây dựng hàm nhận diện keypoints trên bàn tay:

```
def extract_hand_keypoints(image_path):
    img = cv2.imread(image_path)
    if img is None:
        print(f"Lỗi đọc ảnh: {image_path}")
        return np.zeros(63)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    result = hands.process(img_rgb)
    if result.multi_hand_landmarks:
        hand_landmarks = result.multi_hand_landmarks[0]
        keypoints = []
        for lm in hand_landmarks.landmark:
            keypoints.extend([lm.x, lm.y, lm.z])
        return np.array(keypoints)
    else:
        return np.zeros(63)
```

❖ Các bước:

- Đọc ảnh từ đường dẫn (image_path) bằng OpenCV.
- Nếu lỗi đọc ảnh, trả về mảng số 0.
- Chuyển ảnh từ BGR sang RGB.
- Dùng MediaPipe để phát hiện bàn tay trong ảnh.
- Nếu phát hiện:
 - Trích xuất x, y, z của 21 điểm landmark.
 - Trả về mảng numpy gồm 63 giá trị.
 - Nếu không phát hiện: Trả về mảng số 0.

❖ Xây dựng Hàm Load dữ liệu từ dataset:

```
def load_dataset_with_subfolders(folder_path):
    X = []
    Y = []
    for label_name in os.listdir(folder_path):
        label_folder = os.path.join(folder_path, label_name)
        if not os.path.isdir(label_folder):
            continue
        for img_file in os.listdir(label_folder):
            img_path = os.path.join(label_folder, img_file)
            if not img_file.lower().endswith(('.jpg', '.jpeg', '.png')):
                continue
            keypoints = extract_hand_keypoints(img_path)
            X.append(keypoints)
            Y.append(label_name)
    return np.array(X), np.array(Y)
```

❖ Các bước thực hiện:

- Khởi tạo hai danh sách rỗng X (dữ liệu) và Y (nhãn).
- Duyệt qua từng thư mục con để xác định nhãn.
- Duyệt qua từng file ảnh trong thư mục con, kiểm tra định dạng file hợp lệ.
- Trích xuất đặc trưng (keypoints) từ ảnh bằng MediaPipe.

- Lưu trữ keypoints và nhãn tương ứng.
- Chuyển danh sách thành mảng numpy để dễ sử dụng cho huấn luyện mô hình.

❖ Chương trình gán nhãn, chuẩn bị dữ liệu train và test:

```
train_dir = '/content/dataset/train'
test_dir = '/content/dataset/test'

X_train, Y_train = load_dataset_with_subfolders(train_dir)
X_test, Y_test = load_dataset_with_subfolders(test_dir)

print("Train shape:", X_train.shape, Y_train.shape)
print("Test shape:", X_test.shape, Y_test.shape)
```

```
In [11]: train_dir = '/content/dataset/train'
         test_dir = '/content/dataset/test'

         X_train, Y_train = load_dataset_with_subfolders(train_dir)
         X_test, Y_test = load_dataset_with_subfolders(test_dir)

         print("Train shape:", X_train.shape, Y_train.shape)
         print("Test shape:", X_test.shape, Y_test.shape)
```

```
Train shape: (1512, 63) (1512,)
Test shape: (72, 63) (72,)
```

Ta thấy :

- X_train chứa 1512 mẫu, mỗi mẫu có 63 đặc trưng (keypoints x, y, z của 21 điểm bàn tay).
- Y_train chứa 1512 nhãn tương ứng với các mẫu trong X_train.
- X_test chứa 72 mẫu dùng để kiểm tra, mỗi mẫu cũng có 63 đặc trưng.
- Y_test chứa 72 nhãn tương ứng với các mẫu trong X_test

❖ Chương trình mã hóa nhãn dán thành chữ số nguyên :

```
encoder = LabelEncoder()

Y_train_encoded = encoder.fit_transform(Y_train)

Y_test_encoded = encoder.transform(Y_test)

print("Các nhãn đã mã hóa:", encoder.classes_)
```

```
In [12]: encoder = LabelEncoder()
Y_train_encoded = encoder.fit_transform(Y_train)
Y_test_encoded = encoder.transform(Y_test)

print("Các nhãn đã mã hóa:", encoder.classes_)

Các nhãn đã mã hóa: ['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R'
'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z']
```

❖ Khởi tạo mô hình Neural Network

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(63,)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(len(encoder.classes_), activation='softmax') # số lượng
    lớp
])

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

model.summary()
```

Layer (type)	Output Shape	Param #	Ý nghĩa
Dense (dense)	(None, 128)	8,192	Lớp fully connected đầu tiên, từ 63 đầu vào → 128 neurons
Dropout (dropout)	(None, 128)	0	Dropout để chống overfitting, không có tham số train
Dense (dense_1)	(None, 64)	8,256	Lớp fully connected thứ 2, từ 128 neurons → 64 neurons
Dropout (dropout_1)	(None, 64)	0	Dropout thứ 2, cũng không có tham số train
Dense (dense_2)	(None, 26)	1,690	Lớp đầu ra, từ 64 neurons → 26 lớp (ứng với 26 chữ cái A-Z)

Kết quả model.summary():

- Mô hình có tổng cộng 18,138 tham số trainable.
- Gồm 3 lớp Dense và 2 lớp Dropout:
- Lớp Dense (128 neuron) nhận đầu vào 63 đặc trưng.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	8,192
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 26)	1,690

Total params: 18,138 (70.85 KB)

Trainable params: 18,138 (70.85 KB)

Non-trainable params: 0 (0.00 B)

- Lớp Dense (64 neuron) tiếp theo.
- Lớp Dense cuối cùng dự đoán 26 lớp tương ứng với 26 ký tự bảng chữ cái.

- Các lớp Dropout giúp giảm overfitting.
- Các tham số chủ yếu nằm ở các lớp Dense (fully connected).

```
history = model.fit(  
X_train, Y_train_encoded,  
epochs=50,  
validation_data=(X_test, Y_test_encoded),  
batch_size=32  
)
```

❖ Huấn luyện Model

- Sử dụng hàm fit() để huấn luyện mô hình trên tập dữ liệu train (X_train, Y_train_encoded).
- Quá trình huấn luyện diễn ra trong 50 epoch.
- Sau mỗi epoch, mô hình sẽ kiểm tra độ chính xác trên tập validation (X_test, Y_test_encoded).
- Batch size được đặt là 32 mẫu để tối ưu tốc độ huấn luyện và bộ nhớ.

- Kết quả huấn luyện (loss, accuracy) được lưu vào biến history để phân tích sau.

```
Epoch 1/50
48/48 — 6s 56ms/step - accuracy: 0.0428 - loss: 3.2728 - val_accuracy: 0.0556 - val_loss: 3.2241
Epoch 2/50
48/48 — 0s 4ms/step - accuracy: 0.0704 - loss: 3.2197 - val_accuracy: 0.0833 - val_loss: 3.1911
Epoch 3/50
48/48 — 0s 4ms/step - accuracy: 0.0710 - loss: 3.2146 - val_accuracy: 0.0694 - val_loss: 3.1289
Epoch 4/50
48/48 — 0s 3ms/step - accuracy: 0.0819 - loss: 3.1598 - val_accuracy: 0.0833 - val_loss: 3.0544
Epoch 5/50
48/48 — 0s 4ms/step - accuracy: 0.1107 - loss: 3.0910 - val_accuracy: 0.0833 - val_loss: 2.9686
Epoch 6/50
48/48 — 0s 3ms/step - accuracy: 0.1238 - loss: 3.0073 - val_accuracy: 0.1250 - val_loss: 2.8387
Epoch 7/50
48/48 — 0s 3ms/step - accuracy: 0.1528 - loss: 2.9368 - val_accuracy: 0.1250 - val_loss: 2.7309
Epoch 8/50
48/48 — 0s 3ms/step - accuracy: 0.1834 - loss: 2.7997 - val_accuracy: 0.2639 - val_loss: 2.6303
Epoch 9/50
48/48 — 0s 4ms/step - accuracy: 0.1992 - loss: 2.6988 - val_accuracy: 0.2500 - val_loss: 2.4805
Epoch 10/50
48/48 — 0s 3ms/step - accuracy: 0.2119 - loss: 2.5832 - val_accuracy: 0.2917 - val_loss: 2.3792
Epoch 11/50
48/48 — 0s 3ms/step - accuracy: 0.2525 - loss: 2.4704 - val_accuracy: 0.2639 - val_loss: 2.3205
Epoch 12/50
48/48 — 0s 3ms/step - accuracy: 0.2615 - loss: 2.4015 - val_accuracy: 0.2778 - val_loss: 2.2174
Epoch 13/50
48/48 — 0s 3ms/step - accuracy: 0.2658 - loss: 2.3447 - val_accuracy: 0.3194 - val_loss: 2.1472
Epoch 14/50
48/48 — 0s 4ms/step - accuracy: 0.2960 - loss: 2.2690 - val_accuracy: 0.3611 - val_loss: 2.0717
Epoch 15/50
48/48 — 0s 3ms/step - accuracy: 0.3016 - loss: 2.2421 - val_accuracy: 0.3333 - val_loss: 2.0351
Epoch 16/50
48/48 — 0s 3ms/step - accuracy: 0.3200 - loss: 2.1701 - val_accuracy: 0.4028 - val_loss: 1.9724
Epoch 17/50
48/48 — 0s 3ms/step - accuracy: 0.3449 - loss: 2.1079 - val_accuracy: 0.4028 - val_loss: 1.9251
Epoch 18/50
48/48 — 0s 4ms/step - accuracy: 0.3097 - loss: 2.1171 - val_accuracy: 0.3889 - val_loss: 1.8970
Epoch 19/50
48/48 — 0s 3ms/step - accuracy: 0.3624 - loss: 2.0502 - val_accuracy: 0.4028 - val_loss: 1.8427
Epoch 20/50
48/48 — 0s 4ms/step - accuracy: 0.3605 - loss: 2.0119 - val_accuracy: 0.3611 - val_loss: 1.8910
```

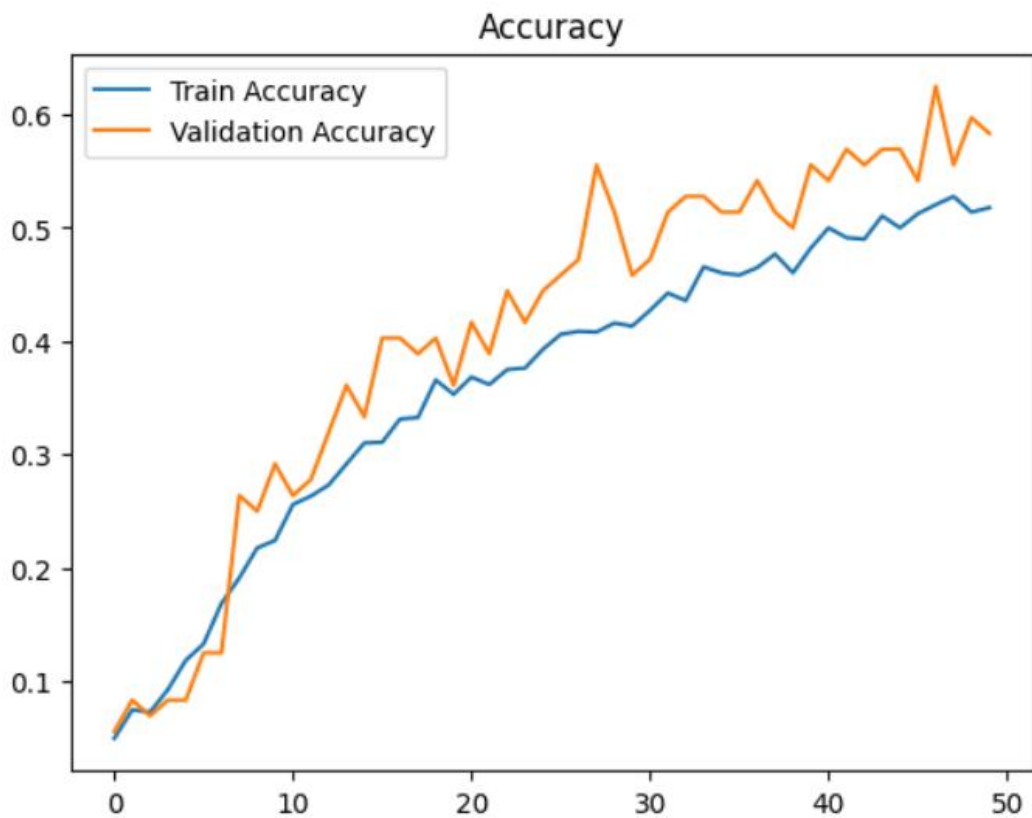
48/48	0s	4ms/step	- accuracy: 0.3605	- loss: 2.0119	- val_accuracy: 0.3611	- val_loss: 1.8910
Epoch 21/50						
48/48	0s	3ms/step	- accuracy: 0.3573	- loss: 2.0536	- val_accuracy: 0.4167	- val_loss: 1.8033
Epoch 22/50						
48/48	0s	3ms/step	- accuracy: 0.3598	- loss: 2.0005	- val_accuracy: 0.3889	- val_loss: 1.8331
Epoch 23/50						
48/48	0s	4ms/step	- accuracy: 0.3795	- loss: 1.9517	- val_accuracy: 0.4444	- val_loss: 1.7564
Epoch 24/50						
48/48	0s	3ms/step	- accuracy: 0.3798	- loss: 1.9669	- val_accuracy: 0.4167	- val_loss: 1.7239
Epoch 25/50						
48/48	0s	3ms/step	- accuracy: 0.3948	- loss: 1.8672	- val_accuracy: 0.4444	- val_loss: 1.6812
Epoch 26/50						
48/48	0s	4ms/step	- accuracy: 0.3993	- loss: 1.8711	- val_accuracy: 0.4583	- val_loss: 1.6832
Epoch 27/50						
48/48	0s	3ms/step	- accuracy: 0.4063	- loss: 1.8658	- val_accuracy: 0.4722	- val_loss: 1.6622
Epoch 28/50						
48/48	0s	4ms/step	- accuracy: 0.4139	- loss: 1.8573	- val_accuracy: 0.5556	- val_loss: 1.6231
Epoch 29/50						
48/48	0s	3ms/step	- accuracy: 0.4115	- loss: 1.7954	- val_accuracy: 0.5139	- val_loss: 1.5959
Epoch 30/50						
48/48	0s	3ms/step	- accuracy: 0.4101	- loss: 1.8137	- val_accuracy: 0.4583	- val_loss: 1.6080
Epoch 31/50						
48/48	0s	3ms/step	- accuracy: 0.4367	- loss: 1.7446	- val_accuracy: 0.4722	- val_loss: 1.5893
Epoch 32/50						
48/48	0s	3ms/step	- accuracy: 0.4539	- loss: 1.7440	- val_accuracy: 0.5139	- val_loss: 1.5697
Epoch 33/50						
48/48	0s	3ms/step	- accuracy: 0.4345	- loss: 1.7404	- val_accuracy: 0.5278	- val_loss: 1.5468
Epoch 34/50						
48/48	0s	3ms/step	- accuracy: 0.4475	- loss: 1.7560	- val_accuracy: 0.5278	- val_loss: 1.5373
Epoch 35/50						
48/48	0s	3ms/step	- accuracy: 0.4656	- loss: 1.6664	- val_accuracy: 0.5139	- val_loss: 1.5491
Epoch 36/50						
48/48	0s	3ms/step	- accuracy: 0.4358	- loss: 1.7240	- val_accuracy: 0.5139	- val_loss: 1.5048
Epoch 37/50						
48/48	0s	5ms/step	- accuracy: 0.4681	- loss: 1.7244	- val_accuracy: 0.5417	- val_loss: 1.4704
Epoch 38/50						
48/48	0s	5ms/step	- accuracy: 0.4660	- loss: 1.6567	- val_accuracy: 0.5139	- val_loss: 1.5216
Epoch 39/50						
48/48	0s	5ms/step	- accuracy: 0.4791	- loss: 1.6252	- val_accuracy: 0.5000	- val_loss: 1.5054
Epoch 40/50						
48/48	0s	5ms/step	- accuracy: 0.4851	- loss: 1.6377	- val_accuracy: 0.5556	- val_loss: 1.4635
Epoch 41/50						
48/48	0s	6ms/step	- accuracy: 0.5253	- loss: 1.5695	- val_accuracy: 0.5417	- val_loss: 1.4378
Epoch 42/50						
48/48	0s	6ms/step	- accuracy: 0.5047	- loss: 1.5558	- val_accuracy: 0.5694	- val_loss: 1.4227
Epoch 43/50						
48/48	0s	4ms/step	- accuracy: 0.4890	- loss: 1.5900	- val_accuracy: 0.5556	- val_loss: 1.4426
Epoch 44/50						
48/48	0s	3ms/step	- accuracy: 0.4963	- loss: 1.5887	- val_accuracy: 0.5694	- val_loss: 1.4066
Epoch 45/50						
48/48	0s	3ms/step	- accuracy: 0.5007	- loss: 1.5730	- val_accuracy: 0.5694	- val_loss: 1.3966
Epoch 46/50						
48/48	0s	3ms/step	- accuracy: 0.5283	- loss: 1.5186	- val_accuracy: 0.5417	- val_loss: 1.4148
Epoch 47/50						
48/48	0s	3ms/step	- accuracy: 0.5218	- loss: 1.5545	- val_accuracy: 0.6250	- val_loss: 1.3795
Epoch 48/50						
48/48	0s	4ms/step	- accuracy: 0.5419	- loss: 1.5426	- val_accuracy: 0.5556	- val_loss: 1.4096
Epoch 49/50						
48/48	0s	3ms/step	- accuracy: 0.5257	- loss: 1.5075	- val_accuracy: 0.5972	- val_loss: 1.3593
Epoch 50/50						
48/48	0s	3ms/step	- accuracy: 0.5170	- loss: 1.5052	- val_accuracy: 0.5833	- val_loss: 1.3506

❖ Lưu và tải xuống Model để sử dụng:

```
model.save('sign_language_model.keras')

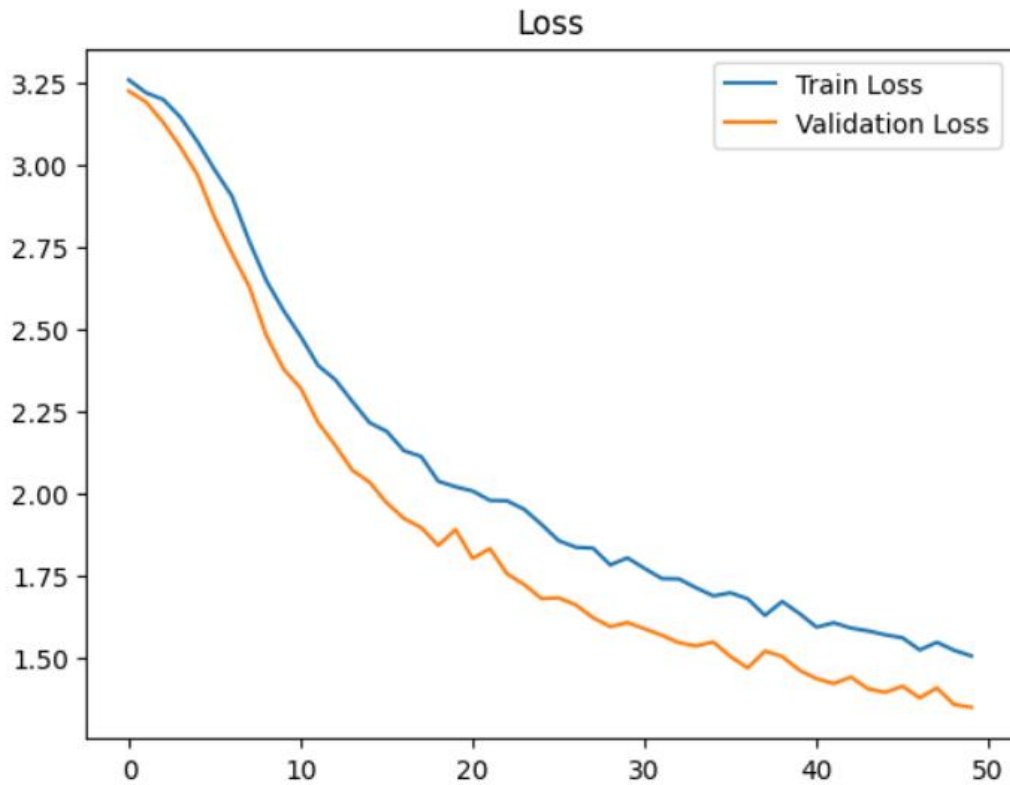
from google.colab import files

files.download('/content/sign_language_model.keras')
```



2.2 Biểu đồ Accuracy và Loss

Hình 20: Biểu đồ Accuracy



Hình 21: Biểu đồ Loss

3. Đánh giá

Quá trình huấn luyện mô hình nhận diện ngôn ngữ ký hiệu bàn tay được tiến hành trong 50 epoch và thu được các kết quả như sau:

- Độ chính xác (accuracy) đầu tiên khá thấp (~4%-7%) do mô hình chưa học được nhiều.
- Trong quá trình huấn luyện, độ chính xác tăng dần đến ~52% trên tập train và ~58% trên tập validation.
- Độ mất mát (loss) ban đầu cao (~3.2) và giảm đều trong quá trình huấn luyện, ổn định quanh ~1.5 khi kết thúc 50 epoch.

- Validation accuracy cao hơn train accuracy, cho thấy mô hình đã học tốt và chưa bị overfitting.

Tóm lại:

- Mô hình có xu hướng loss giảm đều và accuracy tăng đều qua các epoch.
- Đạt được độ chính xác 58% trên tập test sau quá trình huấn luyện.
- Kết quả training cho thấy mô hình học được đặc trưng bàn tay và đáp ứng yêu cầu bài toán nhận diện ngôn ngữ ký hiệu.

4. Kiểm chứng và sử dụng model

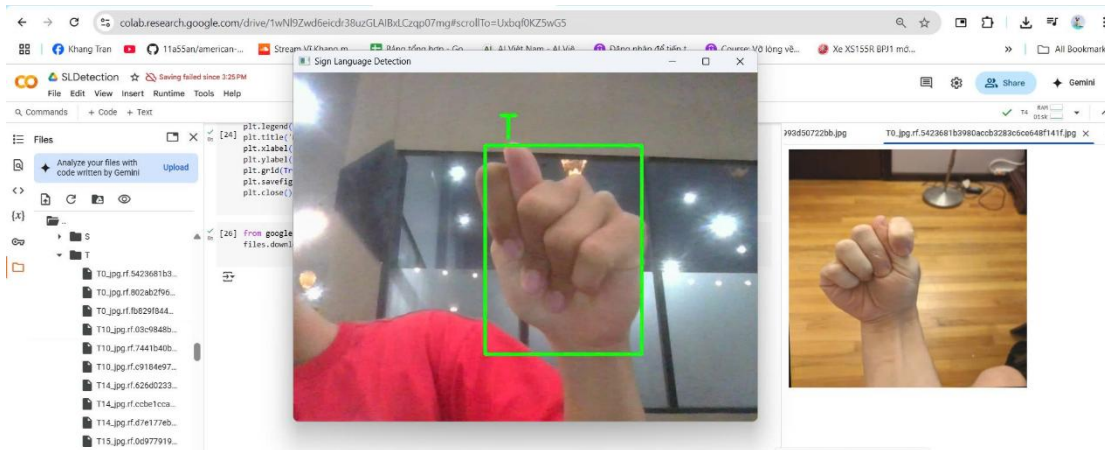
4.1 Chương trình nhận diện main.py

```

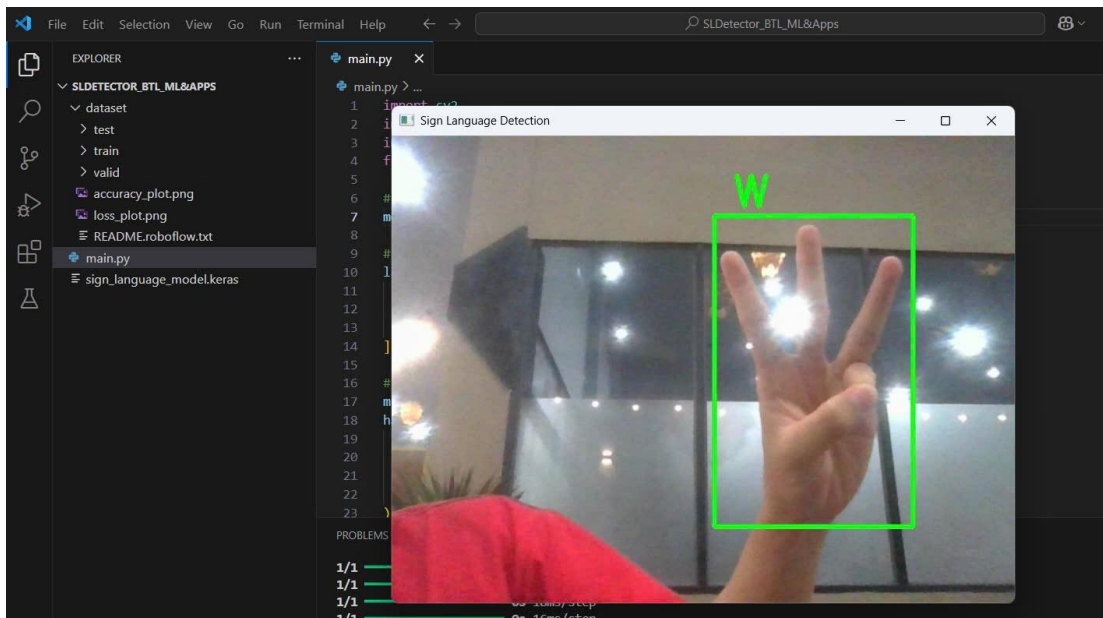
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
# Đọc dữ liệu từ file CSV
file_path = r'kmeans_dataset.csv' # Thay bằng đường dẫn đến file CSV của bạn
data = pd.read_csv(file_path)
# Chọn các cột đặc trưng để áp dụng K-means (ví dụ: chọn tất cả các cột)
X = data.apply(pd.to_numeric, errors='coerce').dropna()
# Kiểm tra xem dữ liệu có bị trống không
print("Kích thước của X sau khi xử lý:", X.shape)
print(X.head()) # In 5 hàng đầu tiên của X để kiểm tra
# Kiểm tra nếu dữ liệu trống
if X.empty:
    print("Dữ liệu trống sau khi xử lý. Kiểm tra lại dữ liệu trong file CSV.")
else:
    # Áp dụng K-means
    kmeans = KMeans(n_clusters=3) # Thay đổi số lượng cụm (n_clusters) tùy ý
    kmeans.fit(X)
    y_kmeans = kmeans.predict(X)
    # Sử dụng PCA để giảm số chiều xuống 2 chiều để vẽ biểu đồ
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X)
    # Vẽ biểu đồ với dữ liệu đã giảm chiều
    plt.figure(figsize=(10, 6))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_kmeans, cmap='viridis', s=50)
    centers_pca = pca.transform(kmeans.cluster_centers_)
    plt.scatter(centers_pca[:, 0], centers_pca[:, 1], s=200, c='red', marker='X', label='Centroids')
    plt.xlabel("Thành phần chính 1")
    plt.ylabel("Thành phần chính 2")
    plt.title("K-means Clustering với nhiều đặc trưng (sử dụng PCA)")
    plt.legend()
    plt.show()

```

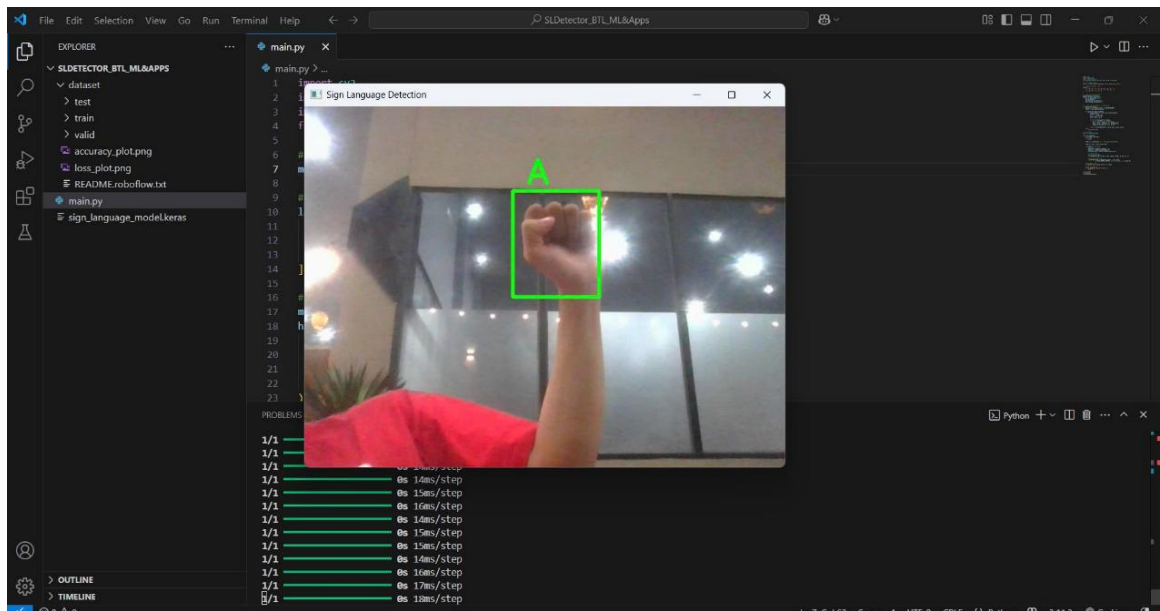
4.2 Kết quả thu được:



Hình 22: Kết quả thu được với chữ T



Hình 23: Kết quả thu được với chữ W



Hình 24: Kết quả thu được với chữ A

TÀI LIỆU THAM KHẢO

<https://github.com/AvishakeAdhikary/Realtime-Sign-Language-Detection-Using-LSTM-Model>

<https://nttuan8.com/bai-14-long-short-term-memory-lstm/>

<https://viblo.asia/p/recurrent-neural-networkphan-1-tong-quan-va-ung-dung-jvElaB4m5kw>

Bài báo khoa học: "Sign Language Recognition with Transformer Networks"
- Nghiên cứu này trình bày việc áp dụng mô hình Transformer trong nhận diện ngôn ngữ ký hiệu.

<https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/>