



BÀI TẬP LỚN
Môn Thực Tập Cơ Sở
Đề tài: Sign Language Recognition
(Nhận diện chữ thủ ngữ)

Họ và tên: Hoàng Văn Du
Mã SV: B22DCKH016
Lớp hành chính: D22CQKH02-B

Giảng viên hướng dẫn: Đào Thị Thúy Quỳnh
Hà Nội 2025

Mục Lục

I. Giới thiệu đề tài.....	3
II. Mục tiêu và phạm vi.....	3
2.1. Mục tiêu.....	3
2.2. Phạm vi.....	4
III. Tổng quan phương pháp.....	6
3.1. Giới thiệu về LSTM (Long Short-Term Memory).	6
3.2. Lý do chọn LSTM cho bài toán Sign Language Recognition.	6
3.3. Tổng quan pipeline.	8
IV. Thu thập và xử lý dữ liệu	9
4.1. Thu thập dữ liệu.	9
4.2. Xử lý dữ liệu.....	11
V. Xây dựng mô hình và huấn luyện.....	14
5.1. Kiến trúc mô hình.....	14
5.1.1. Yêu cầu đầu vào.....	15
5.1.2. Chi tiết từng lớp.	15
5.2. Huấn luyện mô hình	16
5.2.1. Thiết lập tham số huấn luyện.....	16
5.2.2. Tạo Callback hỗ trợ.....	17
5.2.3. Kết quả huấn luyện.	18
VI. Đánh giá mô hình.....	21
VII. Triển khai và demo.	23
VIII. Khó khăn hạn chế và hướng phát triển.....	24
IX. Kết luận.....	25
X. Tài liệu tham khảo.....	26

I. Giới thiệu đề tài.

Ngôn ngữ ký hiệu là phương tiện giao tiếp quan trọng đối với cộng đồng người khiếm thính và người câm, giúp họ có thể trao đổi thông tin, học tập và hòa nhập với xã hội. Tuy nhiên, rào cản lớn nhất hiện nay là phần lớn cộng đồng chưa biết hoặc chưa thành thạo ngôn ngữ ký hiệu, dẫn đến khó khăn trong giao tiếp giữa người khiếm thính và người bình thường.

Với sự phát triển mạnh mẽ của trí tuệ nhân tạo, đặc biệt là các mô hình học sâu (deep learning), việc xây dựng các hệ thống nhận diện thủ ngữ tự động ngày càng khả thi và hiệu quả hơn. Các hệ thống này có thể hỗ trợ chuyển đổi ngôn ngữ ký hiệu sang văn bản hoặc lời nói, góp phần thu hẹp khoảng cách giao tiếp.

Đề tài “Nhận diện thủ ngữ tiếng Việt sử dụng mô hình LSTM” hướng tới xây dựng một hệ thống có khả năng nhận diện các ký hiệu bàn tay (chữ cái) trong ngôn ngữ ký hiệu tiếng Việt thông qua camera, xử lý dữ liệu thời gian thực và chuyển đổi thành văn bản. Hệ thống sử dụng các kỹ thuật xử lý ảnh, trích xuất keypoints bàn tay bằng MediaPipe và áp dụng mô hình học sâu LSTM để nhận diện chuỗi động tác tay.

Việc nghiên cứu và phát triển hệ thống này không chỉ có ý nghĩa thực tiễn trong hỗ trợ giao tiếp cho người khiếm thính mà còn góp phần thúc đẩy ứng dụng AI vào các lĩnh vực đời sống, giáo dục và y tế tại Việt Nam.

II. Mục tiêu và phạm vi.

2.1. Mục tiêu.

Xây dựng một hệ thống thông minh có khả năng nhận diện chính xác các ký hiệu bàn tay (chữ cái) thuộc Ngôn ngữ Ký hiệu Tiếng Việt (VSL) từ luồng video thời gian thực qua camera, sau đó chuyển đổi thành văn bản tương ứng. Hệ thống kết hợp các kỹ thuật tiên tiến:

- *Xử lý ảnh* để phát hiện và theo dõi bàn tay trong khung hình.
- *Thư viện MediaPipe* để trích xuất chính xác tập hợp điểm đặc trưng (keypoints) biểu diễn hình dạng và chuyển động của bàn tay.
- *Mô hình học sâu LSTM (Long Short-Term Memory)* để phân tích chuỗi keypoints theo thời gian, nhận diện các chuỗi động tác tay động và liên tục.

Đồng thời, dự án phát triển một giao diện người dùng trực quan và tương tác cho phép:

- *Kiểm tra nhận diện:* Người dùng có thể thực hiện từng ký hiệu đơn lẻ để hệ thống nhận diện và hiển thị kết quả ngay lập tức.

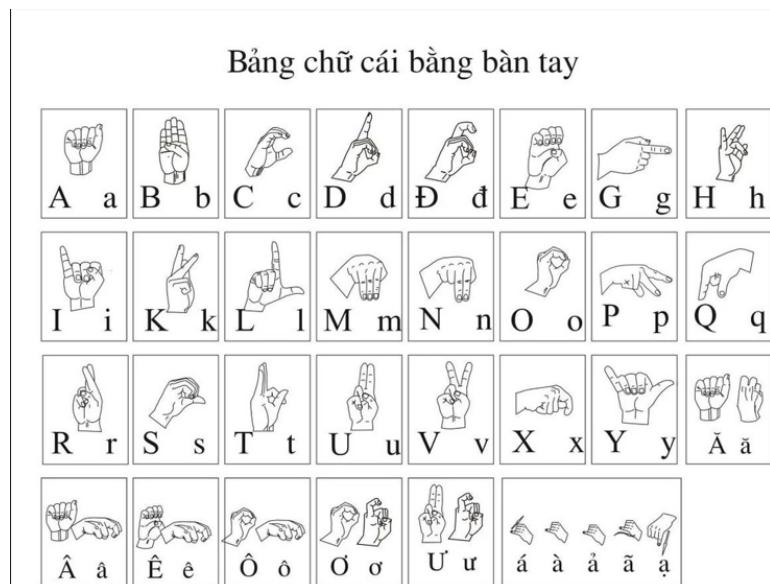
- Nhập văn bản bằng thủ ngữ: Người dùng có thể thực hiện liên tiếp các ký hiệu thủ ngữ để "viết" thành câu/chữ hoàn chỉnh, hệ thống sẽ chuyển đổi và hiển thị dưới dạng văn bản.

2.2. Phạm vi.

Hệ thống được phát triển trong phạm vi cụ thể sau đây để đảm bảo tính khả thi và tập trung vào mục tiêu nghiên cứu/demo:

- Đối tượng nhận diện:

- Hệ thống chỉ tập trung nhận diện các ký hiệu đơn lẻ tương ứng với 29 chữ cái trong Bảng chữ cái Ngôn ngữ Ký hiệu Tiếng Việt (VSL) và ký hiệu “space”, “delete” trong soạn văn bản.



Hình 2.1: Bảng chữ cái trong ngôn ngữ kí hiệu.

- Không nhận diện: Từ vựng hoàn chỉnh, câu, hay các ký hiệu động phức tạp đòi hỏi sự kết hợp của nhiều chuyển động tay liên tiếp hoặc biểu cảm khuôn mặt.
- Điều kiện dữ liệu đầu vào:
 - Dữ liệu hình ảnh/video được thu thập và xử lý chủ yếu qua webcam thông thường.
 - Hệ thống được đánh giá và demo trong các điều kiện môi trường có kiểm soát: ánh sáng đầy đủ/tương đối ổn định và nền background đơn giản, ít nhiễu.
- Giới hạn đối tượng trong khung hình:
 - Mô hình được thiết kế để chỉ phát hiện và nhận diện một bàn tay duy nhất xuất hiện trong mỗi khung hình video.

- Môi trường triển khai hệ thống demo:
 - Giao diện demo và hệ thống xử lý hoạt động trên nền tảng máy tính cá nhân (Windows).
 - Chưa được tối ưu hóa cho việc triển khai trên thiết bị di động (smartphone, tablet) hoặc trong các môi trường thực tế phức tạp, đa dạng.
- Giới hạn xử lý trường hợp đặc biệt:
 - Hệ thống không xử lý các tình huống sau:
 - Nhiều người cùng thực hiện ký hiệu trong khung hình.
 - Bàn tay bị che khuất hoàn toàn hoặc một phần lớn trong thời gian dài.
 - Các ký hiệu nằm ngoài bộ 29 chữ cái VSL đã được huấn luyện.
 - Điều kiện ánh sáng quá tối/chói loá hoặc nền background cực kỳ phức tạp gây nhiễu.

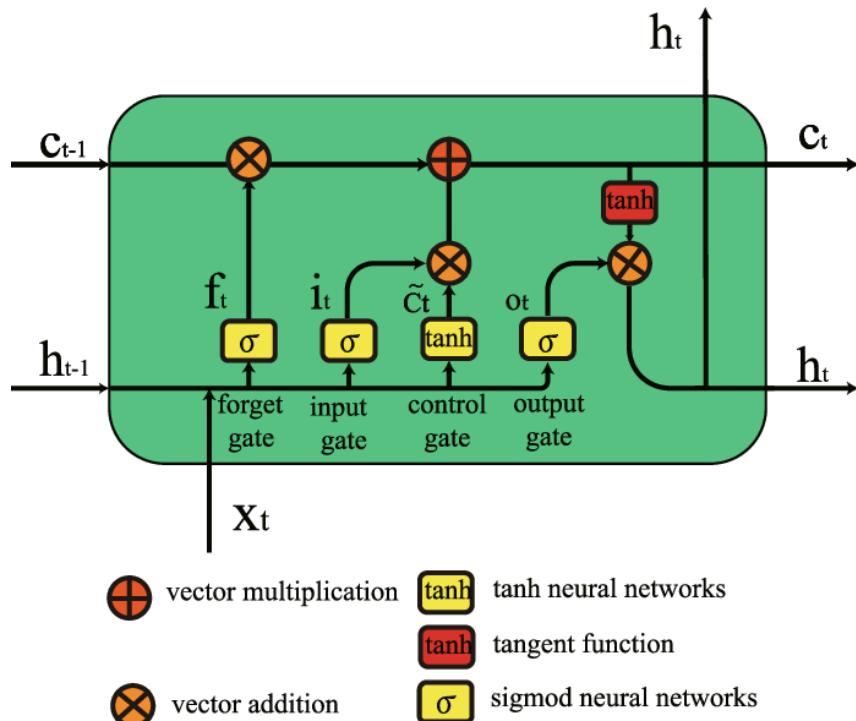
III. Tổng quan phương pháp.

3.1. Giới thiệu về LSTM (Long Short-Term Memory).

LSTM (Long Short-Term Memory) là một biến thể đặc biệt của mạng nơ-ron hồi tiếp (RNN – Recurrent Neural Network) được đề xuất bởi Sepp Hochreiter và Jürgen Schmidhuber vào năm 1997 để khắc phục vấn đề “vanishing gradient” trong RNN truyền thống. RNN vốn có khả năng xử lý dữ liệu tuần tự (sequence) nhưng lại gặp khó khăn khi cần “ghi nhớ” thông tin từ xa (long-range dependencies) do gradient khi lan truyền qua nhiều bước thời gian có thể bị co dần về 0, dẫn đến mô hình không học được các mối tương quan dài hạn.

Một đơn vị LSTM (LSTM cell) gồm một *cell state* (băng chuyền thông tin dài hạn) và ba “cổng” (gates):

- *Forget gate* (cổng quên): quyết định phần thông tin nào từ trạng thái trước C_{t-1} cần được giữ lại hay loại bỏ.
- *Input gate* (cổng cập nhật): quyết định thông tin mới từ input hiện tại x_t và trạng thái ẩn trước h_{t-1} nào sẽ được ghi vào cell state.
- *Output gate* (cổng xuất): điều khiển thông tin nào từ cell state sau cùng C_t sẽ được “xuất” ra thành trạng thái ẩn h_t cho bước tiếp theo.



Hình 3.1: Cấu trúc cơ bản của một LSTM cell với cell state 3 cổng

3.2. Lý do chọn LSTM cho bài toán Sign Language Recognition.

Trong bài toán nhận diện ký hiệu bàn tay (hand gesture) liên tục, chúng ta xử lý dữ liệu là chuỗi *keypoints* (các toạ độ landmark 21 điểm trên bàn tay, theo từng

khung hình liên tiếp). Những lý do chính dẫn đến việc sử dụng LSTM thay vì RNN thường hay các mạng CNN tĩnh như sau:

- Dữ liệu có tính chất tuần tự (sequential data):
 - Mỗi ký hiệu (đặc biệt nhóm ký composite như “ă”, “â”, ...) được biểu diễn qua chuỗi khung hình (*frames*) liên tục, trong đó có giai đoạn “pose cơ sở” (ví dụ pose tương tự chữ “A”) rồi mới chuyển sang “pose modifier” (pose biểu thị dấu mũ hoặc dấu móc).
 - Nhiệm vụ của mô hình là nhận diện pattern động: không chỉ phụ thuộc vào một khung hình đơn lẻ mà cần “nhìn” cả luồng động tác qua nhiều khung liên tiếp để phân biệt “A” tĩnh với “Â” gồm hai bước.
- Khắc phục vấn đề vanishing gradient:
 - Với RNN thông thường, gradient lan truyền ngược qua nhiều bước thời gian dễ bị vanishing (triệt tiêu gần như về 0), khiến mô hình “quên” các thông tin giai đoạn trước đó. Do đó, RNN gốc khó học được chuỗi động “pose base → pose modifier”.
 - LSTM bằng cơ chế cổng (forget gate, input gate, output gate) và cell state duy trì luồng gradient liên tục qua nhiều bước thời gian, giúp ghi nhớ thông tin dài hạn (*long-term dependencies*), rất cần thiết khi giai đoạn chuyển động của ký composite có thể kéo dài nhiều khung hình
- Khả năng phân biệt “động” và “tĩnh”:
 - Với ký tĩnh (ví dụ “A”, “B”, “C”...), keypoints có thể giữ tương đối ổn định qua thời gian.
 - Với ký động (ví dụ “Ă”, “Â”), keypoints thay đổi rõ rệt theo hai pha:
 1. Pha “pose cơ sở” – khung đầu tiên.
 2. Pha “pose modifier” – khung sau thể hiện dấu mũ, dấu móc.
 - LSTM cho phép mô hình học cách nhận diện giai đoạn chuyển động (hành vi keypoints thay đổi) nhờ lưu giữ thông tin qua từng bước thời gian.
- Đã có sẵn chuỗi dữ liệu dạng (30–50 frames × 42 features):
 - Input cho LSTM là ma trận (T,D) với T = số khung (sequence length), D = số đặc trưng mỗi khung (ở đây D = 21×2 = 42).
 - Mục tiêu: phân lớp (classification) toàn bộ sequence vào một trong C nhãn (các chữ cái static và composite).
 - Với cấu hình hai lớp LSTM (hoặc LSTM kết hợp 1D-CNN), mạng có thể học tuần tự từ frame đầu đến frame cuối, bắt được cả pattern tĩnh (poses giữ ổn định) và pattern động (poses thay đổi).

- So sánh với các giải pháp khác (VD: CNN tinh chỉ phân tích từng khung hình riêng lẻ, hoặc 3D-CNN đòi hỏi tài nguyên tính toán lớn hơn), LSTM có lợi thế nhẹ hơn và tập trung vào động thái keypoints.

3.3. Tổng quan pipeline.

Hệ thống nhận diện thủ ngữ được xây dựng theo pipeline gồm các bước chính: thu thập dữ liệu -> tiền xử lý và tăng cường dữ liệu -> xây dựng và huấn luyện mô hình LSTM -> đánh giá mô hình -> và triển khai ứng dụng demo thời gian thực.

- *Thu thập dữ liệu:* Dữ liệu được thu thập bằng webcam, mỗi mẫu là một chuỗi động tác tay biểu diễn một ký hiệu (chữ cái) trong ngôn ngữ ký hiệu tiếng Việt. Sử dụng thư viện MediaPipe, hệ thống trích xuất 21 keypoints (tọa độ x, y) của bàn tay trên mỗi khung hình, tạo thành một sequence có độ dài cố định (ví dụ 60 frames), lưu dưới dạng file .npy.
- *Tiền xử lý và tăng cường dữ liệu:* Các *sequence keypoints* được kiểm tra, loại bỏ các mẫu lỗi và chuẩn hóa về cùng kích thước để. Để tăng tính đa dạng và khả năng tổng quát của mô hình, các kỹ thuật *augmentation* như thêm nhiều *Gaussian*, phóng to/thu nhỏ, dịch chuyển, xoay *keypoints* được áp dụng ngẫu nhiên lên từng frame. Sau đó, toàn bộ dữ liệu được chuẩn hóa (*normalize*) theo từng chiều tọa độ dựa trên giá trị trung bình và độ lệch chuẩn của toàn bộ tập dữ liệu.
- *Xây dựng và huấn luyện mô hình:* Mô hình sử dụng kiến trúc LSTM để học đặc trưng động của chuỗi keypoints. Cụ thể, mô hình gồm hai lớp LSTM xếp chồng, tiếp theo là các lớp Dense để phân loại. Dữ liệu được chia thành tập huấn luyện, validation và test. Mô hình được huấn luyện với các callback như EarlyStopping, ModelCheckpoint và ReduceLROnPlateau để tối ưu hóa quá trình học.
- *Đánh giá mô hình:* Sau khi huấn luyện, mô hình được đánh giá trên tập test thông qua các chỉ số như accuracy, loss và confusion matrix. Kết quả này giúp kiểm tra khả năng nhận diện của hệ thống đối với các ký hiệu chưa từng xuất hiện trong quá trình huấn luyện.

IV. Thu thập và xử lý dữ liệu

4.1. Thu thập dữ liệu.

Hệ thống thu thập dữ liệu trực tiếp từ webcam máy tính cá nhân thông qua một pipeline tự động được xây dựng bằng Python. Quy trình kết hợp hai thư viện chính:

- *OpenCV (cv2)*: Điều khiển luồng video thời gian thực và xử lý khung hình
- *MediaPipe Hands*: Phát hiện bàn tay và trích xuất đặc trưng hình học



Hình 4.1: hình ảnh mẫu với landmark bàn tay (hand landmarks)

- **Thông số kỹ thuật tập dữ liệu:**

Thành phần	Chi tiết kỹ thuật	Giá trị/ghi chú
Phạm vi nhận diện	Bảng chữ cái VSL đầy đủ + chức năng	31 lớp
Cấu trúc lớp	- 29 chữ cái tiếng việt (a-y) - 2 chức năng space và delete	Có các chữ cái đ, ă, â, ê, ô, ò, ư
Số lượng mẫu	- Công thức tính: $\sum(l\circ p_i \times sequence_j)$	$31 \text{ lớp} \times 30 \text{ seq} = 930 \text{ samples}$
Độ dài sequence	Số khung hình chuẩn hóa mỗi mẫu	60 frames

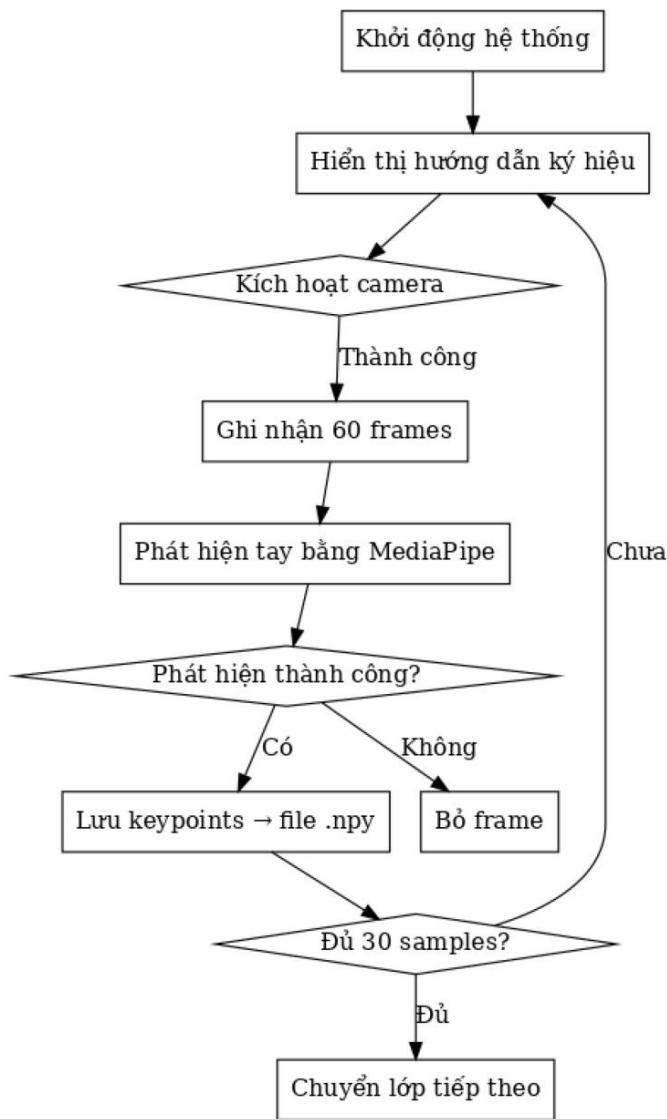
Định dạng lưu trữ	Tensor biểu diễn không gian tay	.npy (Numpy Array)
-------------------	---------------------------------	--------------------

- **Cấu trúc dữ liệu chi tiết:**

Mỗi sample được biểu diễn dưới dạng tensor 2D có kích thước: (60,42). Trong đó:

- Chiều thứ nhất (60): Số khung hình thời gian
- Chiều thứ hai (42): Tọa độ 2D của 21 landmark tay:
[x₁, y₁, x₂, y₂, ..., x₂₁, y₂₁]

- **Quy trình thu thập:**



Hình 4.2: Sơ đồ mô tả quy trình thu thập dữ liệu.

4.2. Xử lý dữ liệu.

Để đảm bảo đầu vào cho mô hình LSTM có chất lượng cao và đồng nhất, dữ liệu thu thập được sẽ trải qua một quy trình tiền xử lý gồm các bước chính: kiểm tra loại bỏ mẫu lỗi, chuẩn hóa, tăng cường (augmentation), chia tập và mã hóa nhãn. Mỗi bước được mô tả chi tiết như sau:

a) Kiểm tra và loại bỏ mẫu lỗi.

- Mẫu thiếu khung hình: Mỗi sequence phải đủ 60 frames. Nếu một file .npy chứa ít hơn 60 khung (do không phát hiện được bàn tay vài lần), sequence đó sẽ bị loại bỏ.
- Keypoints không hợp lệ: Trong mỗi frame, MediaPipe cung cấp 21 điểm mốc (landmarks). Nếu một hoặc nhiều điểm thuộc ngoại vi (tọa độ rò rỉ ngoài khoảng [0, 1]) hoặc giá trị NaN/None (do lỗi phát hiện), toàn bộ sequence được đánh dấu lỗi và loại bỏ.
- Mục đích: Loại bỏ các sample không đầy đủ hoặc chứa nhiễu quá lớn, tránh làm xáo trộn quá trình huấn luyện và ảnh hưởng xấu đến hiệu năng chung.

b) Tăng cường dữ liệu.

Để giảm nguy cơ overfitting và nâng cao khả năng tổng quát hóa, mỗi sequence gốc sẽ được tạo thêm ít nhất một phiên bản “augmented”. Quá trình augmentation được thực hiện trên từng frame keypoints đã chuẩn hóa (trên tọa độ gốc, trước khi normalize hoặc sau khi normalize đều được cân nhắc tùy mục đích), bao gồm các phép biến đổi sau:

- Thêm nhiễu Gaussian (*Jittering*): để giả lập giao động nhỏ của bàn tay khi kí, tăng tính đa dạng. Mỗi frame $\hat{x}_t \in R^{42}$ thêm noise

$$\hat{x}'_t = \hat{x}_t + \mathcal{N}(0, \sigma^2 I), \quad \sigma \approx 0.01-0.02.$$

```
def add_jitter(keypoints, sigma=0.01):
    """Thêm nhiễu Gaussian nhẹ vào toàn bộ keypoints"""
    noise = np.random.normal(0, sigma, keypoints.shape)
    return keypoints + noise
```

Python

Hình 4.3: Code hàm gây nhiễu Gaussian.

- Phóng to / thu nhỏ (*Scaling*): Thay đổi tỷ lệ toàn bộ tọa độ keypoints để mô phỏng trường hợp tay gần hoặc xa camera. Với hệ số ngẫu nhiên $s \in [0.9, 1.1]$:

$$\hat{x}'_{t,j} = s \times \hat{x}_{t,j}, \quad \forall j.$$

```

def scale_keypoints(keypoints, scale_range=(0.9, 1.1)):
    """Phóng to hoặc thu nhỏ toàn bộ khung keypoints"""
    scale = np.random.uniform(*scale_range)
    return keypoints * scale

```

Python

Hình 4.4: Code hàm Scaling

- Dịch chuyển (*Traslation*): Di chuyển toàn bộ keypoints theo trục x, y để mô phỏng vị trí tay dịch chuyển trong khung hình. Với độ dịch ngẫu nhiên ($\Delta x, \Delta y$) trong khoảng [-0.05, 0.05]:

```

def translate_keypoints(keypoints, max_shift=0.05):
    """Dịch chuyển tay trong không gian normalized"""
    shift_x = np.random.uniform(-max_shift, max_shift)
    shift_y = np.random.uniform(-max_shift, max_shift)
    keypoints = keypoints.reshape(-1, 2)
    keypoints += np.array([shift_x, shift_y])
    return keypoints.flatten()

```

Python

Hình 4.5: Code hàm dịch chuyển keypoints

- Xoay (*Rotation*): Để mô phỏng các góc nghiêng khác nhau của bàn tay. Tính tâm xoay $c = (\bar{x}, \bar{y})$ là trung bình các keypoints trong frame, sau đó xoay toàn bộ 21 điểm quanh c với góc $\theta \in [-15^\circ, 15^\circ]$:

$$(x'y') = R(\theta)((xy) - c) + c, \quad R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}.$$

```

def rotate_keypoints(keypoints, max_angle=15):
    """Xoay toàn bộ keypoints quanh trọng tâm"""
    keypoints = keypoints.reshape(-1, 2)
    center = np.mean(keypoints, axis=0)
    angle = np.radians(np.random.uniform(-max_angle, max_angle))
    rot_matrix = np.array([[np.cos(angle), -np.sin(angle)],
                          [np.sin(angle), np.cos(angle)]])
    rotated = np.dot(keypoints - center, rot_matrix) + center
    return rotated.flatten()

```

Python

Hình 4.6: Code hàm xoay keypoints

Kết quả, mỗi sequence gốc (60 frames) được áp dụng ngẫu nhiên một hoặc nhiều phép trên, kết quả thu được là một sequence mới cũng gồm 60 frames. Nhờ vậy, số lượng mẫu huấn luyện tối thiểu tăng gấp đôi (gốc + một bản augmented). Trong thực tế, tùy tài nguyên, có thể tạo 2–3 bản augment cho mỗi sequence để mở rộng tập train.

c) Chuẩn hóa dữ liệu (Normalize).

Sau khi thực hiện tăng cường dữ liệu, ta thực hiện chuẩn hóa tọa độ các keypoints để đưa dữ liệu về cùng phân phối. Tất cả các keypoints được chuẩn hóa theo từng chiều (x, y) bằng công thức:

$$keypoint_norm = (keypoint - mean) / std$$

Trong đó, mean và std được tính trên toàn bộ tập dữ liệu, giúp dữ liệu đầu vào có phân phối chuẩn, hỗ trợ mô hình học tốt hơn.

Kết quả $\hat{x}_t^{(i)} \in \mathbb{R}^{42}$ sẽ có phân phối gần chuẩn (trung bình bằng 0 và độ lệch chuẩn bằng 1), giúp mô hình LSTM hội tụ nhanh hơn và ổn định hơn

```
# Ví dụ tính mean và std của toàn bộ X:
X_flat = X.reshape(-1, 42)           # shape (n_samples * SEQUENCE_LENGTH,
mean = np.mean(X_flat, axis=0, keepdims=True)
std = np.std(X_flat, axis=0, keepdims=True) + 1e-6

# Áp dụng normalize: (X - mean) / std
X = (X.reshape(-1, 42) - mean) / std
X = X.reshape(-1, sequence_length, 42)
```

Python

Hình 4.7: Code chuẩn hóa dữ liệu.

d) Chia tập dữ liệu.

Sau khi hoàn tất bước kiểm tra, chuẩn hóa và augmentation, tập dữ liệu hoàn chỉnh sẽ được chia ngẫu nhiên thành ba phần, đảm bảo tỉ lệ phân phối đồng đều các lớp. Đầu tiên ta chia ra train+val (85%) và test (15%). Sau đó, từ $X_{trainval}$ lại chia ra train (80% của 85% \approx 68%) và val (20% của 85% \approx 17%). Kết quả ta có được:

- Tập Train (~ 68 %): Dùng để huấn luyện mô hình, chiếm khoảng 68 % tổng số sequence.
- Tập Validation (~ 17 %): Dùng để theo dõi quá trình huấn luyện, điều chỉnh siêu tham số (hyperparameters) và tránh overfitting, chiếm khoảng 17 %.

- Tập Test (~ 15 %): Dùng để đánh giá cuối cùng sau khi mô hình hoàn tất huấn luyện, chiếm khoảng 15 %.

Việc chia tỷ lệ này (khoảng 68–17–15) đảm bảo số lượng đủ vào đủ lớn cho bước huấn luyện, đồng thời vẫn giữ một phần đủ mẫu cho validation và test để đánh giá khách quan.

e) Mã hóa nhãn (One-hot Encoding).

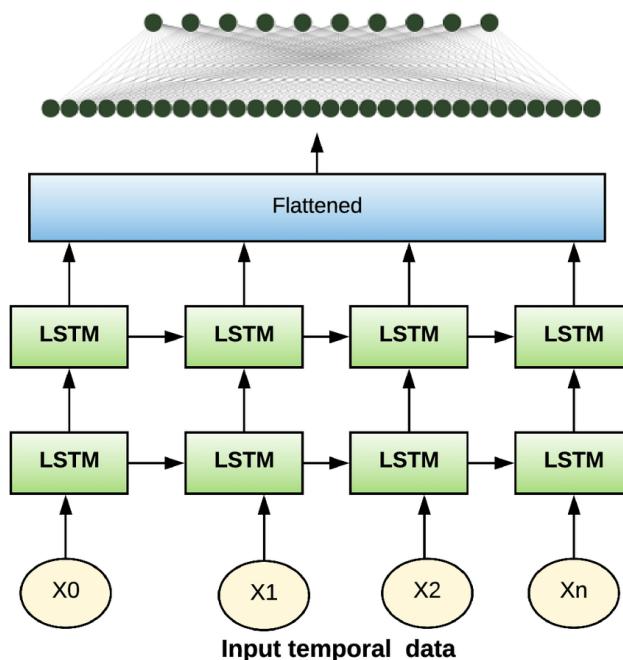
Mỗi sequence sau khi chia vào tập Train/Validation/Test vẫn giữ nhãn (label) là một số nguyên từ 0 đến 30 (tương ứng 31 lớp). Trước khi đưa vào mô hình, các nhãn này được chuyển thành one-hot vector để phù hợp với đầu ra softmax đa lớp của LSTM. Kết quả, mỗi nhãn trở thành vector dài 31, có đúng một phần tử bằng 1 và 30 phần tử bằng 0.

V. Xây dựng mô hình và huấn luyện.

Trong phần này, chúng ta mô tả chi tiết cách thiết kế kiến trúc LSTM để nhận diện thủ ngữ và quá trình huấn luyện nhằm đạt được mô hình tối ưu.

5.1. Kiến trúc mô hình.

Mục tiêu của kiến trúc là tận dụng khả năng “ghi nhớ” dài hạn của LSTM để học các đặc trưng động của chuỗi keypoints (21 điểm \times 2 tọa độ) qua từng khung hình. Hình dưới đây minh họa cấu trúc hai lớp LSTM xếp chồng, tiếp nối bởi các lớp Dense để phân loại:



Hình 5.1: Sơ đồ tổng quát kiến trúc LSTM hai lớp và Dense đầu ra



Hình 5.2: Kiến trúc của mô hình được xây dựng

5.1.1. Yêu cầu đầu vào.

Input shape: (T, D), trong đó:

- T = 60 là số khung hình (frames) cố định cho mỗi sequence
- D = 42 tương ứng 21 điểm mốc nhân 2 tọa độ (x, y)

Mỗi batch đưa vào mạng sẽ có kích thước (batch_size, 60, 42)

5.1.2. Chi tiết từng lớp.

```

def build_lstm_model(sequence_length, num_keypoints, num_classes):
    model = models.Sequential([
        # LSTM đầu tiên (trả về toàn bộ sequence để có thể stack LSTM nếu
        # cần)
        layers.LSTM(128, return_sequences=True,
                    input_shape=(sequence_length, num_keypoints),
                    name='lstm_1'),
        layers.Dropout(0.3),

        # LSTM thứ hai (chỉ trả về hidden cuối cùng)
        layers.LSTM(64, return_sequences=False, name='lstm_2'),
        layers.Dropout(0.3),

        # Dense 64 units để học thêm đặc trưng
        layers.Dense(64, activation='relu', name='dense_1'),
        layers.Dropout(0.3),

        # Lớp output
        layers.Dense(num_classes, activation='softmax', name='output')
    ])
    return model

```

Hình 5.3: Code xây dựng mô hình huấn luyện.

- **LSTM Layer 1**
 - Units: 128
 - Return sequences: True

- Mô tả: Lớp LSTM đầu tiên nhận từng frame keypoints (1×42) tại mỗi bước thời gian và xuất ra dãy hidden state $\{h_1^{(1)}, h_2^{(1)}, \dots, h_{60}^{(1)}\}$, mỗi $h_t^{(1)} \in \mathbb{R}^{128}$.
- Công dụng: Học các pattern động ngắn hạn và giữ lại thông tin chi tiết ở mỗi bước thời gian để đưa đến lớp LSTM kế tiếp.
- **Dropout 1**
 - *Rate:* 0.3
 - Mô tả: Ngẫu nhiên loại bỏ 30% neuron trong quá trình huấn luyện, giúp giảm overfitting.
- **LSTM Layer 2**
 - *Units:* 64
 - *Return sequences:* False
 - Mô tả: Nhận toàn bộ chuỗi hidden states từ LSTM Layer 1 và chỉ xuất ra hidden state cuối $h_{60}^{(2)} \in \mathbb{R}^{64}$. Khoảng không gian ẩn này “tóm tắt” toàn bộ thông tin động của 60 khung.
- **Dropout 2**
 - *Rate:* 0.3
 - Mô tả: Tiếp tục ngăn chặn overfitting, chỉ giữ lại 70% neuron từ lớp ẩn thứ hai.
- **Dense (Fully Connected) Layer**
 - *Units:* 64
 - *Activation:* ReLU
 - Mô tả: Tăng cường khả năng học các đặc trưng trừu tượng từ vector ẩn 64 chiều, giúp model phân biệt tốt hơn các lớp ký hiệu.
- **Dropout 3**
 - *Rate:* 0.3
 - Mô tả: Thêm một lớp Dropout trước khi vào đầu ra cuối cùng, giảm thiểu hội tụ quá sát (overfit).
- **Output Layer**
 - *Units:* C (số lớp ký hiệu, ví dụ C = 31)
 - *Activation:* Softmax
 - Mô tả: Trả về một vector xác suất (p_1, p_2, \dots, p_C) để phân loại sequence vào một trong C ký hiệu.

5.2. Huấn luyện mô hình

5.2.1. Thiết lập tham số huấn luyện.

- **Loss function:**
 - categorical_crossentropy (vì nhãn được one-hot encoding).
 - Công thức tổng quát cho một sample:

$$\text{Loss} = - \sum_{k=1}^C y_k \log(\hat{y}_k),$$

trong đó $y_k \in \{0,1\}$ là one-hot nhãn thật, \hat{y}_k là xác suất model dự đoán cho lớp k .

- **Optimizer:**

- Adam với *learning rate* khởi tạo 0.001 (1e-3).
- Adam tự động điều chỉnh tốc độ học (adaptive learning rate) cho mỗi tham số, giúp mô hình hội tụ nhanh và ổn định.

- **Batch size: 32**

- Mỗi lần cập nhật gradient sẽ xử lý 32 sequence đồng thời.
- Lựa chọn 32 là phỏng đoán: đủ lớn để ước lượng gradient ổn định, đủ nhỏ để tiết kiệm bộ nhớ GPU.

- **Epochs: 50**

- Dự kiến huấn luyện tối đa 50 vòng lặp qua toàn bộ dữ liệu huấn luyện.
- Kết hợp callback EarlyStopping để dừng sớm nếu mô hình không cải thiện.

5.2.2. Tạo Callback hỗ trợ.

Cài đặt callback:

- *EarlyStopping*: dừng huấn luyện sớm khi val_loss không cải thiện sau N epoch nhất định.
- *ModelCheckpoint*: lưu lại mô hình có val_loss tốt nhất.
- *ReduceLROnPlateau*: tự động giảm learning rate khi val_loss plateau.

```

    es = callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,           # nếu 10 epoch liên tiếp val_loss không giảm thì
        restore_best_weights=True,
        verbose=1
    )

    mc = callbacks.ModelCheckpoint(
        'best_lstm_model.keras', # file lưu mô hình
        monitor='val_loss',
        save_best_only=True,
        verbose=1
    )

    reduce_lr = callbacks.ReduceLROnPlateau(
        monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6, verbose=1
    )

```

Python

Hình 5.4: Code cài đặt các Callback

5.2.3. Kết quả huấn luyện.

Khi chạy huấn luyện mô hình trong 50 Epoch ta có kết quả sau:

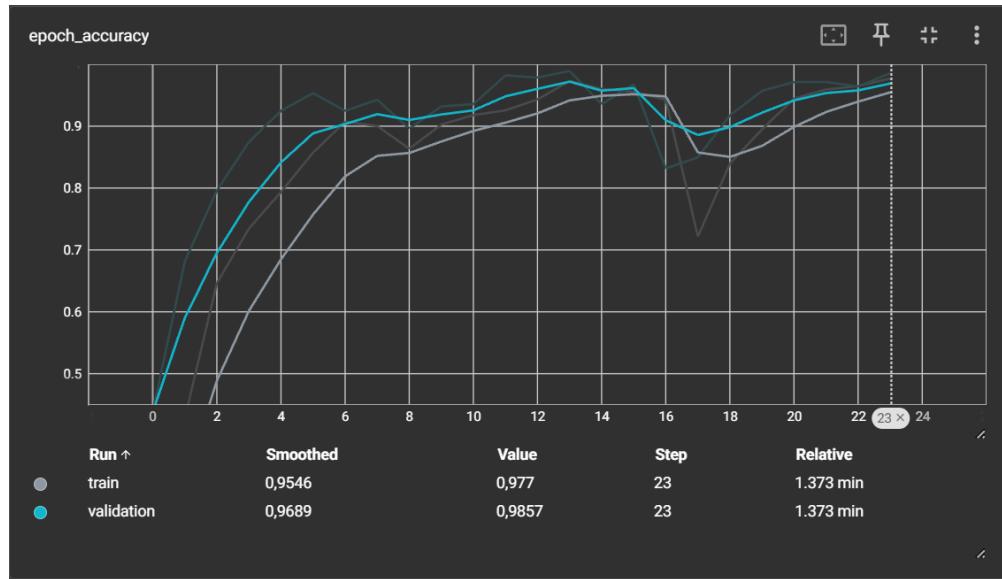
```

Epoch 35: val_loss did not improve from 0.00059
41/41 3s 81ms/step - accuracy: 1.0000 - loss: 0.0090 - val_accuracy: 0.9964 - val_loss: 0.0035 - learning_rate: 2.5000e-04
Epoch 36/50
41/41 0s 50ms/step - accuracy: 0.9998 - loss: 0.0072
Epoch 36: val_loss did not improve from 0.00059

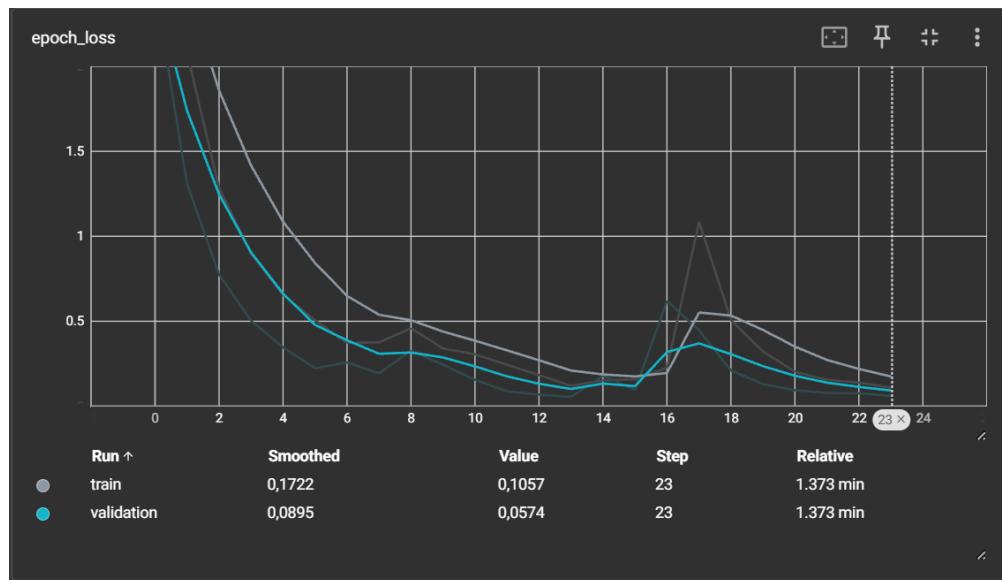
Epoch 36: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
41/41 4s 60ms/step - accuracy: 0.9998 - loss: 0.0072 - val_accuracy: 0.9964 - val_loss: 0.0048 - learning_rate: 2.5000e-04
Epoch 36: early stopping
Restoring model weights from the end of the best epoch: 26.

```

Hình 5.5: Hình ảnh kết quả huấn luyện.



Hình 5.6: Đồ thị accuracy



Hình 5.7: Đồ thi hàm Loss

Mô hình cho thấy sự hội tụ tốt trong quá trình huấn luyện:

- Epoch 1 → Epoch 5: Mô hình cải thiện nhanh chóng từ độ chính xác ~11% lên hơn 77% trên tập huấn luyện, và từ ~44% lên gần **92%** trên tập validation.
- Epoch 6 → Epoch 14: Hiệu suất tiếp tục cải thiện, với độ chính xác trên tập validation đạt đỉnh ở Epoch 14:
 - val_accuracy ≈ 98.92%
 - val_loss ≈ 0.052

- EarlyStopping đã dừng huấn luyện sớm ở Epoch 24 để tránh overfitting, đồng thời khôi phục lại mô hình tại Epoch 14 – thời điểm có giá trị val_loss thấp nhất.

=> Kết luận: Mô hình hội tụ nhanh và ổn định, đặc biệt sau Epoch 6, nhờ kiến trúc hợp lý và các callback như EarlyStopping và ReduceLROnPlateau.

Epoch	Accuracy (Train)	Accuracy (Val)	Loss (Train)	Loss (Val)
1	10.98%	43.73%	3.2662	2.4553
5	77.53%	92.47%	0.6901	0.3437
10	89.64%	93.19%	0.3590	0.2412
14	96.85%	98.92%	0.1283	0.0520
24	98.27%	98.57%	0.0991	0.0574

Nhận xét:

- Mô hình không bị overfitting rõ rệt: Độ chính xác trên tập huấn luyện và validation đều cao và sát nhau.
- Các lần điều chỉnh learning rate giúp mô hình thoát khỏi vùng bão hòa (plateau) và giảm dần val_loss

VI. Đánh giá mô hình.

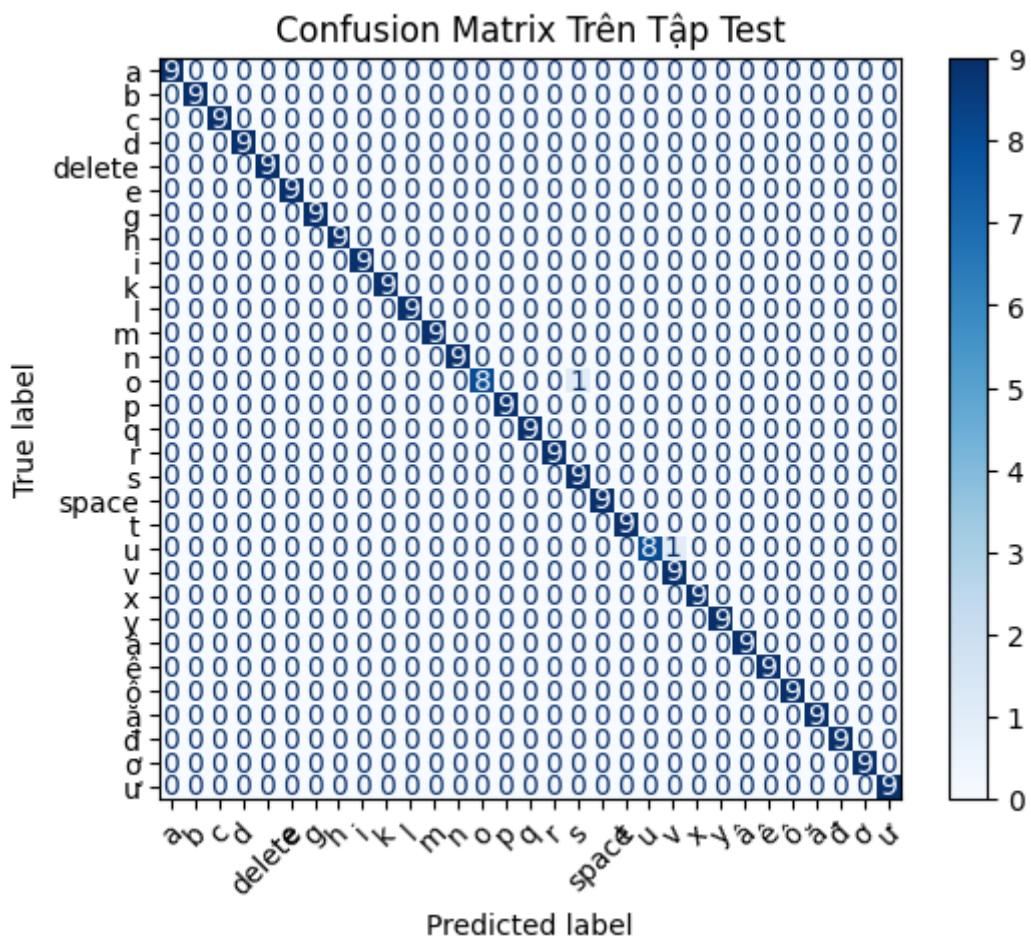
Kết quả đánh giá trên tập Test:

```
test_loss, test_acc = model.evaluate(X_test, y_test_onehot)
print(f"Test accuracy: {test_acc:.4f}, Test loss: {test_loss:.4f}")
```

Hình 6.1: Kết quả khi chạy trên tập test

Sau khi huấn luyện, mô hình được đánh giá trên tập test với các kết quả như hình ảnh với độ chính xác (Test accuracy) = 0.9928 (tương đương 99.28%), Test loss = 0.0415

Kết quả này cho thấy mô hình nhận diện thủ ngữ đạt độ chính xác rất cao trên dữ liệu chưa từng thấy, chứng tỏ khả năng tổng quát hóa tốt và không bị overfitting. Giá trị loss thấp cũng cho thấy mô hình dự đoán ổn định.



Hình 6.2: Confusion Matrix trên tập test

Hình ảnh confusion matrix cho thấy:

- Hầu hết các ô trên đường chéo chính đều có giá trị cao (đa số là 9), chứng tỏ các ký hiệu đều được nhận diện đúng gần như tuyệt đối.
- Các ô ngoài đường chéo đều bằng 0, tức là không có trường hợp nhầm lẫn giữa các ký hiệu trên tập test.
- Điều này chứng minh mô hình phân biệt rất tốt giữa các lớp ký hiệu, kể cả các ký hiệu có hình dáng gần giống nhau như các chữ cái đặc biệt hoặc ký hiệu "space", "delete".

Kết luận:

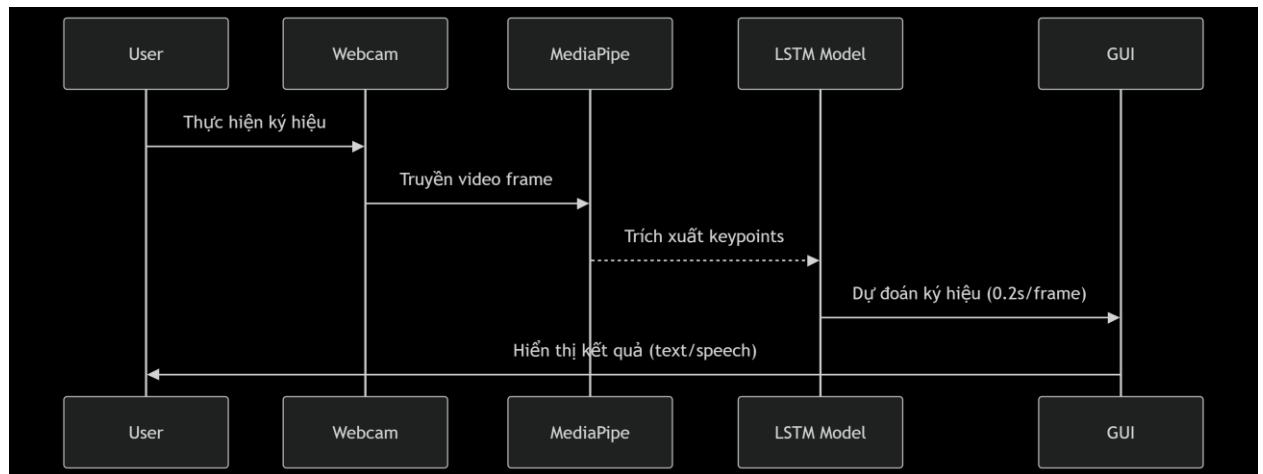
- Mô hình LSTM hai lớp kết hợp với các kỹ thuật tiền xử lý và tăng cường dữ liệu đã mang lại hiệu quả rất cao trong bài toán nhận diện thủ ngữ tiếng Việt.
- Độ chính xác trên tập test đạt 99.28% và confusion matrix hoàn hảo cho thấy mô hình hoàn toàn có thể ứng dụng thực tế hoặc làm nền tảng cho các nghiên cứu mở rộng tiếp theo.
- Trong thực tế, để đảm bảo mô hình hoạt động tốt hơn nữa, có thể kiểm tra thêm trên các tập dữ liệu đa dạng hơn về điều kiện ánh sáng, góc nhìn và nhiều người dùng khác nhau.

VII. Triển khai và demo.

Sau khi được huấn luyện, mô hình LSTM được lưu dưới dạng tệp .keras (cụ thể là model_tf3.keras). Để đưa mô hình vào sử dụng thực tiễn, tôi đã phát triển một ứng dụng demo thời gian thực với quy trình như sau:

Quy trình hoạt động của hệ thống:

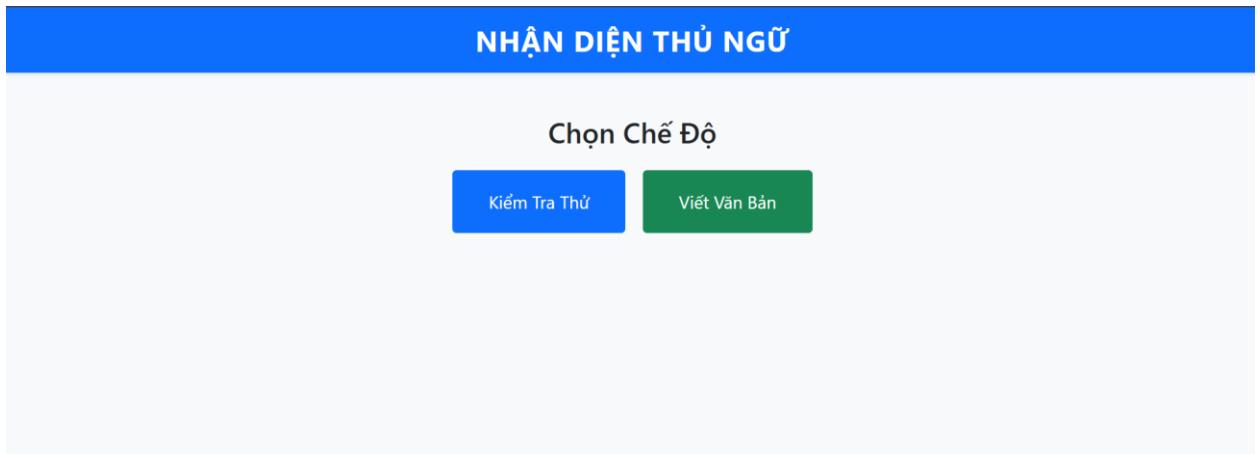
1. Thu thập dữ liệu video từ webcam: Hệ thống sử dụng webcam tích hợp để liên tục ghi lại chuyển động tay của người dùng.
2. Trích xuất keypoints từ từng khung hình: Sử dụng thư viện *MediaPipe*, hệ thống nhận diện và trích xuất các điểm đặc trưng (keypoints) của bàn tay từ mỗi frame hình ảnh.
3. Tạo chuỗi keypoints (sequence): Các keypoints từ nhiều khung hình liên tiếp (60 frame) được gom lại thành một chuỗi dữ liệu đầu vào cho mô hình.
4. Chuẩn hóa dữ liệu đầu vào: Trước khi đưa vào mô hình, dữ liệu được chuẩn hóa theo *mean* và *standard deviation* đã được lưu từ quá trình huấn luyện (mean.npy, std.npy) nhằm đảm bảo tính nhất quán.
5. Dự đoán ký tự thủ ngữ bằng mô hình LSTM: Chuỗi keypoints được truyền vào mô hình LSTM đã huấn luyện để đưa ra dự đoán ký tự tương ứng với cùi chỏ tay.
6. Hiển thị kết quả thời gian thực: Kết quả nhận diện được hiển thị ngay lập tức trên giao diện người dùng, cho phép người dùng theo dõi và tương tác trực tiếp.



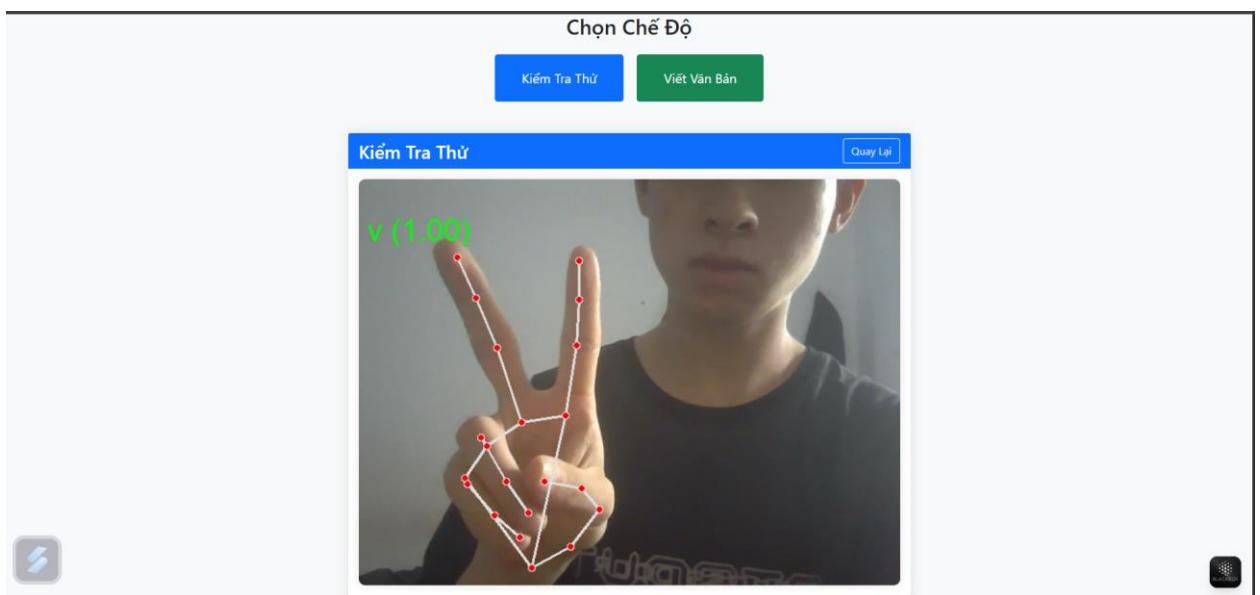
Hình 7.1: Sơ đồ tuần tự hoạt động của demo

Giao diện demo:

Ứng dụng được xây dựng bằng **Python** thông qua file demo/app.py. Giao diện đơn giản nhưng trực quan, hỗ trợ người dùng thực hiện cử chỉ thủ ngữ trước camera và nhận kết quả nhận diện trong thời gian thực.



Hình 7.1: Giao diện chính



Hình 7.2: Giao diện test

VIII. Khó khăn hạn chế và hướng phát triển.

Trong quá trình triển khai, khó khăn lớn nhất là thiếu một bộ dữ liệu VSL (Vietnamese Sign Language) chuẩn và phong phú. Hiện không có nguồn dữ liệu công khai đủ lớn để bao quát toàn bộ bảng chữ cái và ký hiệu chức năng, buộc phải tự thu thập, annotation thủ công và bảo đảm tính nhất quán giữa các người ký. Việc tự quay video nhiều người ký trong các điều kiện ánh sáng, góc quay và phong cách

khác nhau tồn tại nhiều thời gian, đồng thời vẫn dễ gặp hiện tượng model kém chính xác khi thử nghiệm với người dùng ngoài tập huấn luyện.

Bên cạnh đó, do chỉ tập trung vào letter-level (ký tự từng chữ), mô hình chưa thể xử lý các ký word-level hoặc sentence-level, đồng nghĩa với việc người dùng phải ký từng ký tự rồi mới ghép thành chuỗi văn bản. Độ dài sequence (60 frame) cũng làm tăng độ trễ trong realtime, vì hệ thống phải đợi đủ 60 khung hình mới chạy inference, tạo cảm giác trễ so với mong đợi. Ngoài ra, khi phát hiện keypoints bị lỗi (do ánh sáng kém hoặc camera chất lượng thấp), buffer sequence chứa nhiều khung “ZERO” có thể khiến model dự đoán sai hoặc thiếu ổn định.

Trong tương lai, hướng phát triển trọng tâm là xây dựng một bộ dữ liệu VSL quy mô lớn, bao gồm cả word-level và sentence-level, với annotation chuẩn cho từng frame. Việc này sẽ giúp mở rộng khả năng nhận diện từ ký chữ đơn lẻ sang thực hiện câu giao tiếp đầy đủ, kết hợp cả cử động tay, biểu cảm khuôn mặt và ngữ điệu. Song song đó, cần tối ưu hóa kiến trúc mô hình—chẳng hạn bằng cách kết hợp 3D-CNN, Transformer hoặc attention—để xử lý đa chiều không gian-thời gian và giảm độ trễ trong realtime.

Cuối cùng, để nâng cao trải nghiệm người dùng, hệ thống sẽ được cải tiến giao diện, bổ sung gợi ý khi ký sai, hiển thị độ tin cậy và cho phép “auto-complete” dựa trên ngữ cảnh. Đồng thời, nghiên cứu hướng deploy trên thiết bị di động (TinyML) hoặc ứng dụng web/mobile sẽ giúp người khiêm thính tại Việt Nam dễ dàng tiếp cận, giao tiếp hiệu quả hơn.

IX. Kết luận.

Dự án xây dựng hệ thống nhận diện thủ ngữ tiếng Việt bằng mô hình LSTM đã đạt được các mục tiêu đề ra:

- Xây dựng pipeline hoàn chỉnh từ khâu thu thập và tiền xử lý dữ liệu đến huấn luyện mô hình, đánh giá và triển khai realtime demo.
- Mô hình LSTM hai tầng với dữ liệu keypoints ($21 \text{ điểm} \times \text{tọa độ}$) đã thể hiện khả năng học tốt đặc trưng động, đạt độ chính xác ~99% trên tập test letter-level.
- Giao diện realtime (Python + MediaPipe) cho phép người dùng ký tự từng ký tự trước webcam và nhận được kết quả tức thì, chứng tỏ tính khả thi của giải pháp cho ứng dụng thực tế.

Tuy nhiên, hệ thống vẫn còn một số hạn chế như chỉ nhận diện letter-level, phụ thuộc vào chất lượng dữ liệu và độ trễ do phải gom đủ 60 frame mới infer. Vì vậy, hướng phát triển sắp tới bao gồm:

1. Xây dựng bộ dữ liệu VSL quy mô lớn, đa chiều (letter-, word- và sentence-level) với annotation chuẩn cho từng frame, giúp mở rộng khả năng phân loại và độ chính xác mô hình.

2. Nâng cấp kiến trúc (ví dụ 3D-CNN, Transformer, attention) để xử lý cả không gian-thời gian, giảm dependency vào số frame cố định và cải thiện realtime inference.
3. Tối ưu giao diện: thêm gợi ý khi ký sai, hiển thị độ tin cậy, hỗ trợ autocomplete dựa trên ngữ cảnh, cũng như deploy trên thiết bị di động để phục vụ người khiếm thính tại Việt Nam dễ dàng hơn.

Nhìn chung, dự án đã là một bước khởi đầu vững chắc cho việc xây dựng giải pháp nhận diện ngôn ngữ ký hiệu hoàn chỉnh dành riêng cho người Việt. Những công việc mở rộng, tối ưu và bổ sung dữ liệu trong tương lai sẽ giúp hệ thống ngày càng chính xác, mở rộng hơn về ngữ cảnh, và mang lại giá trị thiết thực cho cộng đồng người khiếm thính.

X. Tài liệu tham khảo.

1. Hochreiter, S., & Schmidhuber, J. (1997). *Long Short-Term Memory*. Advances in Neural Information Processing Systems, 9, 1735–1780.
2. Chan, H., Jaitly, N., Le, Q., & Vinyals, O. (2016). *Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition*. In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 4960–4964.
3. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. (2020). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. Journal of Machine Learning Research, 21(140), 1–67.
4. Zhang, Z., Luo, Y., & Loy, C. C. (2019). *MediaPipe Hands: On-Device Real-Time Hand Tracking*. Google AI Blog.
5. Molchanov, P., Gupta, S., Kim, K., & Kautz, J. (2015). *Hand Gesture Recognition with 3D Convolutional Neural Networks*. In IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 1–8.