## Université de Lille

## ILIS et Ecole Centrale de Lille

## 

## 

## *Architecture d'Intelligence Artificielle adaptative pour la transcription oculomotrice: innover dans les interfaces de communication des personnes en situation de polyhandicap physique*

## 

## 

## 

## 

## 

## M. HASHEMI Shayan

## Directrice de mémoire : Pr. ZGAYA-BIAU Hayfa

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## Mémoire de fin d’études

## Pour l’obtention du diplôme de Master Ingénierie de la Santé :

## Management de l’intelligence artificielle en santé

## Année universitaire 2024 – 2025

## 

## Remerciements

Je tiens à exprimer ma profonde gratitude envers mes enseignants et encadrants académiques, dont le soutien et les conseils ont été déterminants dans la réalisation de mon mémoire de master.

Je remercie tout particulièrement ***Madame ZGAYA-BIAU Hayfa*** et ***Monsieur Slim Hammadi***, mes directeurs de mémoire, pour leur expertise, leurs conseils avisés et leur disponibilité constante. Leur encadrement rigoureux et leurs encouragements m’ont guidé à chaque étape de ce travail de recherche.

Je suis également reconnaissant envers l’ensemble du corps professoral de mon université pour la qualité de leur enseignement et pour les échanges enrichissants lors des cours et séminaires. Ils ont su stimuler ma curiosité et approfondir mes connaissances dans le domaine de Management de l’intelligence artificielle en santé.

Je remercie aussi les membres du jury, Monsieur ***Thibault MARTIN***, pour avoir consacré du temps à l’évaluation de mon travail et pour leurs remarques constructives qui ont contribué à l’amélioration de ce mémoire.

Enfin, j’adresse mes remerciements les plus sincères à ma famille et à mes amis pour leur soutien moral et leurs encouragements tout au long de ce parcours universitaire.

Hashemi Shayan

### Résumé

Ce mémoire présente le développement d'une architecture d’intelligence artificielle adaptative dédiée à la transcription des mouvements oculaires et des expressions faciales, en vue d’améliorer les interfaces de communication pour les personnes en situation de polyhandicap physique. L’objectif est de permettre à ces patients d’exprimer leurs besoins de manière autonome à travers des gestes simples, principalement des mouvements des yeux, traduits en texte ou en parole en temps réel.

L’approche repose sur une combinaison de réseaux de neurones convolutifs (CNN) et de réseaux à mémoire à long terme (LSTM), appliqués à des séquences vidéo courtes. Le système commence par l’enregistrement de vidéos étiquetées montrant les gestes de l’utilisateur (oui, non, normal). Ces vidéos sont ensuite transformées en séquences d’images à partir desquelles des régions d’intérêt (yeux, sourcils) sont extraites et prétraitées pour alimenter le modèle. Après l’apprentissage supervisé, le modèle peut prédire en temps réel les intentions de l’utilisateur en capturant ses mouvements faciaux via une caméra.

Les résultats obtenus montrent une bonne précision de reconnaissance pour les classes définies. Cette technologie offre une alternative novatrice aux moyens de communication traditionnels, renforçant l’autonomie et la qualité de vie des patients atteints de polyhandicap. Le système est conçu pour être adaptable à chaque individu, grâce à des phases de personnalisation et de fine-tuning du modèle.

### Abstract

This thesis presents the development of an adaptive artificial intelligence architecture designed to transcribe eye movements and facial expressions, with the goal of improving communication interfaces for individuals with severe physical disabilities. The system enables such patients to express their needs autonomously through simple gestures—primarily eye movements—which are translated into text or speech in real-time.

The approach relies on a combination of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, applied to short video sequences. The process begins with recording labeled videos of user gestures (yes, no, normal). These are converted into image sequences from which regions of interest (eyes, eyebrows) are extracted and preprocessed before being fed into the model. Once trained, the model is capable of predicting user intent in real time using webcam input.

The results demonstrate strong recognition accuracy for the defined gesture classes. This technology offers an innovative alternative to traditional communication methods, enhancing the autonomy and quality of life for individuals with multiple disabilities. The system is designed to be adaptive to individual needs through personalization and fine-tuning phases.

## Mots-clés

Intelligence artificielle, polyhandicap, mouvement oculaire, CNN-LSTM, communication assistée, vision par ordinateur

## Table des matières

**[2. Remerciements 1](#_7tt6cp19sxm)**

**[3. Résumé 2](#_qs0b3vga8hh)**

[3.1. Résumé 2](#_qjr9tqrk5rc8)

[3.2. Abstract 2](#_tdp9vj27m3fk)

**[4. Mots-clés 3](#_j0t5yj65441)**

**[5. Table des matières 3](#_7k4za9a972op)**

**[6. Liste des abréviations et acronymes 4](#_p2w8pbf8hf0s)**

**[7. Liste des figures et tableaux 5](#_rdvrkxmwnz8b)**

**[8. Introduction 6](#_3bmc8jbuggpa)**

**[9. Contexte & Problématique 8](#_cy69l69melyy)**

[9.1. Présentation détaillée du domaine 9](#_60n45y1za72d)

[9.2. Mise en évidence de la problématique précise 10](#_mn18u3waonu2)

**[10. État de l’art 11](#_ootkb3vigvq8)**

[10.1. Revue bibliographique des travaux existants 12](#_ootkb3vigvq8)

[10.2. Synthèse des approches concurrentes et identification des lacunes 13](#_r9qhs1mnz5ba)

**[11. Méthodologie / Solution proposée 15](#_xy6n8l8mnljp)**

[11.1. Description du dispositif expérimental ou du modèle 16](#_v3ukrvtpy8or)

[11.2. Jeux de données, protocoles de collecte et de prétraitement 18](#_jg1zwmdeussd)

[11.3. Outils et algorithmes mis en œuvre 20](#_ruof1u5c04a8)

**[12. Résultats & Expérimentations 24](#_idz7mzec8z0g)**

[12.1. Présentation des résultats quantitatifs (tableaux, graphiques) 25](#_a848i8gghakh)

[12.2. Analyse des performances et comparaisons 29](#_73e296wxystg)

**[13. Discussion 31](#_ym8xxrf0qex3)**

[13.1. Interprétation des résultats 32](#_6t5svpwmwh9p)

[13.2. Limites de l’étude 33](#_95nndcseomk)

[13.3. Perspectives d’amélioration 35](#_2fkz3xx56z7b)

**[14. Conclusion 37](#_hqixrlugpy6p)**

**[15. Bibliographie 39](#_67jawskdexy)**

**[16. Annexes 41](#_42k9uh8v06pa)**

[16.1. Collecte de données: 42](#_vfgopsas0fse)

[16.2. Prétraitement des données: 46](#_rrgyna2nuo6v)

[16.3. Construction de modèle: 53](#_bfovoceqsk2a)

[16.4. Prédiction et évaluation: 59](#_di74u88tw3i7)

[16.5. Fine Tune model: 66](#_gqc53htz8amr)

[16.6. Lien du modèle et du projet: 69](#_rdc9i0nq52ix)

## 6. Liste des abréviations et acronymes

|  |  |
| --- | --- |
| **Abréviation / Acronyme** | **Signification** |
| IA | Intelligence Artificielle |
| CNN | Convolutional Neural Network (Réseau de Neurones Convolutif) |
| LSTM | Long Short-Term Memory (Mémoire Longue à Court Terme) |
| ROI | Region of Interest (Région d’Intérêt) |
| FACS | Facial Action Coding System |
| GPU | Graphics Processing Unit (Processeur Graphique) |
| FPS | Frames Per Second (Images par Seconde) |
| FER | Facial Expression Recognition (Reconnaissance d’Expressions Faciales) |
| CK+ | Cohn-Kanade Extended (Base de données d’expressions faciales) |
| .npy | NumPy Binary File Format (format de fichier binaire pour tableaux) |
| .pkl | Pickle File (fichier sérialisé en Python) |
| API | Application Programming Interface |
| HCI | Human-Computer Interaction (Interaction Homme-Machine) |
| EMG | Électromyographie |

## 7. Liste des figures et tableaux

[Figure 1: shape predictor 68 face landmarks 19](#_yzqrqr7tu9rn)

[Table 1: Précision globale du modèle 25](#_8z17w1mqjcpw)

[Figure 2: Comparaison des performances - Précision 26](#_hk9ophot8j9q)

[Table 2: Matrice de confusion 26](#_tpx7op2rzfst)

[Figure 3: Courbe de perte (loss vs epochs) 27](#_6ex5nvcbr6m6)

[Figure 4: Courbe de précision (accuracy vs epochs) 28](#_h9k8wajikjak)

[Table 3: Rapport de classification par classe 28](#_ky98apwbohpi)

[Table 4: Comparaison avec les approches classiques 29](#_eesc5dc0wpbi)

## 

## 

## 

## 

## 

## Introduction

La communication est un besoin fondamental de l’être humain, intrinsèquement lié à son bien-être, à son autonomie et à son inclusion sociale. Pour les personnes en situation de polyhandicap physique, les limitations motrices sévères compromettent gravement leur capacité à interagir avec leur environnement, leur entourage, et les professionnels de santé. Dans de nombreux cas, la parole, l’écriture ou même l’utilisation d’interfaces classiques comme les claviers ou écrans tactiles deviennent inaccessibles. Dès lors, l’innovation technologique devient un levier essentiel pour restaurer une forme d’expression et renforcer leur qualité de vie.

Face à ce constat, les technologies d’assistance basées sur l’intelligence artificielle (IA) suscitent un intérêt croissant. En particulier, l’IA appliquée à la reconnaissance des expressions faciales et des mouvements oculaires ouvre des perspectives prometteuses. En effet, le regard et les micro-expressions faciales sont souvent les seuls moyens d’interaction accessibles à ces patients. Transformer ces signaux faibles en langage compréhensible représente un défi technique et humain considérable, mais porteur d’un potentiel immense.

Ce mémoire s’inscrit dans cette dynamique. Il propose le développement d’une architecture d’intelligence artificielle adaptative, capable d’interpréter les gestes faciaux filmés par une caméra pour les convertir en texte ou en messages vocaux. Cette solution se distingue par l’intégration d’un modèle CNN-LSTM entraîné sur des séquences vidéo de courte durée, capturant les variations fines du visage, notamment autour des yeux et des sourcils. Les gestes reconnus sont classés dans trois catégories principales – « oui », « non » et « normal » – permettant une navigation arborescente dans un système de questions-réponses, jusqu’à l’expression d’un besoin spécifique. Ce besoin est ensuite transmis automatiquement au personnel soignant, accompagné de l’identifiant du patient.

Au fil de ce mémoire, nous détaillerons les motivations du projet, les défis rencontrés, les choix technologiques opérés, ainsi que les résultats obtenus lors de la mise en œuvre et des expérimentations. L’approche développée vise à offrir une interface accessible, efficace, personnalisable, et surtout, respectueuse des particularités de chaque utilisateur. Elle s’inscrit dans une volonté de contribuer à une société plus inclusive grâce à l’IA.

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## Contexte & Problématique

### 9.1. Présentation détaillée du domaine

Le domaine des technologies d’assistance connaît depuis plusieurs années une évolution significative, portée par les avancées en intelligence artificielle, en vision par ordinateur et en traitement du signal. Ces progrès ont permis l’émergence d’interfaces homme-machine plus intuitives, capables d’interpréter des signaux biologiques ou comportementaux pour assister des personnes en situation de handicap. Dans ce contexte, les systèmes basés sur la reconnaissance faciale et les mouvements oculaires constituent un champ de recherche particulièrement dynamique.

Les personnes en situation de polyhandicap physique présentent des limitations motrices majeures, souvent associées à des troubles de la communication. Elles ne peuvent pas recourir à la parole, aux gestes manuels ni aux outils traditionnels d’interaction. Pourtant, leurs facultés cognitives peuvent être préservées, et leur regard ou certaines micro-expressions du visage peuvent devenir des vecteurs essentiels de communication. Ces indices, bien que subtils, sont porteurs d’intention et peuvent être exploités par des systèmes intelligents capables de les détecter, les analyser et les interpréter.

L’analyse des mouvements oculaires, ou oculométrie, est depuis longtemps utilisée dans divers contextes scientifiques et médicaux, notamment pour l’étude de l’attention, du comportement visuel ou du diagnostic de pathologies neurologiques. Combinée à la reconnaissance faciale automatisée, elle permet aujourd’hui d’envisager des solutions pratiques pour traduire les intentions d’un individu à travers son regard.

Par ailleurs, les techniques de vision par ordinateur, comme la détection de points de repère faciaux (facial landmarks), facilitent l’extraction de régions d’intérêt pertinentes (yeux, sourcils, etc.), même à partir de vidéos en temps réel. L’utilisation conjointe de modèles de deep learning, notamment les réseaux neuronaux convolutifs (CNN) pour l’analyse spatiale et les réseaux LSTM pour la modélisation temporelle, permet de concevoir des systèmes robustes capables d’apprendre à partir de séquences vidéo labellisées.

L’accessibilité, la personnalisation, la fiabilité en temps réel et la simplicité d’utilisation sont des exigences clés pour ces systèmes. L’objectif est de permettre à l’utilisateur de naviguer dans une interface par des réponses binaires (« oui », « non »), pour affiner progressivement sa demande à travers un système de questions arborescentes. Ce processus doit être rapide, fluide, et surtout adapté aux capacités et aux limites spécifiques de chaque patient.

Dans cette optique, notre projet s’inscrit à la croisée de plusieurs disciplines : intelligence artificielle, interaction homme-machine, neurosciences, et ingénierie biomédicale. Il vise à concevoir un dispositif à la fois technologique et humain, capable de restaurer un canal de communication pour ceux qui en sont privés, en exploitant pleinement le potentiel expressif du visage.

### 9.2. Mise en évidence de la problématique précise

Malgré les avancées technologiques dans le domaine des interfaces de communication, les personnes en situation de polyhandicap physique restent largement exclues des dispositifs traditionnels. Leur condition implique une limitation sévère de la motricité, rendant inopérants les moyens conventionnels tels que les claviers, écrans tactiles, dispositifs à commande vocale ou même les interfaces cérébrales, qui requièrent souvent un niveau de contrôle moteur ou cognitif difficile à atteindre pour ces patients.

Dans ce contexte, le visage – et plus particulièrement le regard – demeure l’un des rares moyens d’expression conservés. Néanmoins, la capture, l’interprétation et la traduction de ces mouvements oculaires en intentions compréhensibles posent de nombreux défis. Les gestes exprimés sont subtils, parfois ambigus, et peuvent varier considérablement selon les individus, les pathologies associées, ou encore les conditions de l’environnement (éclairage, fatigue, positionnement, etc.).

La problématique centrale de ce travail est donc la suivante :  
 **Comment concevoir et mettre en œuvre une solution basée sur l’intelligence artificielle capable de surveiller les mouvements oculaires et les expressions faciales des personnes atteintes de handicaps physiques sévères, afin de traduire ces signaux en texte et en parole pour faciliter leur communication ?**

Cette problématique soulève plusieurs sous-enjeux techniques et humains :

* **Reconnaissance fiable dans des conditions réelles :** Les algorithmes doivent être capables de fonctionner avec une grande précision en temps réel, malgré les variations de conditions de capture (lumière, mouvements parasites, occlusions).
* **Adaptabilité inter-individuelle :** Chaque patient possède une expressivité propre. Le système doit donc être personnalisable, avec des modèles ajustables à partir de données spécifiques à l’utilisateur.
* **Simplicité et ergonomie :** Le dispositif doit être facile à mettre en place et à utiliser par les aidants comme par les patients, sans nécessiter une expertise technique.
* **Temps de réponse et fluidité :** La communication ne doit pas être entravée par des temps de latence importants. Le système doit offrir une interaction aussi naturelle et immédiate que possible.

En définitive, ce mémoire vise à répondre à cette problématique en développant une architecture adaptative d’intelligence artificielle, capable de transformer un geste oculaire en intention lisible, avec pour finalité une interaction efficace entre le patient et son environnement, notamment le personnel soignant. Ce projet ambitionne de franchir une étape vers une communication inclusive, accessible, et respectueuse des capacités des personnes les plus vulnérables.

## 

## 

## 

## 

## 

## 

## 

## 

## 3. État de l’art

### 10.1. Revue bibliographique des travaux existants

L’étude des interfaces de communication assistée pour les personnes en situation de handicap s’est enrichie au cours des dernières décennies grâce aux avancées en intelligence artificielle, en vision par ordinateur et en neurosciences. Plusieurs approches ont été proposées pour pallier l’incapacité à communiquer verbalement ou gestuellement, en exploitant les signaux résiduels comme les mouvements oculaires, les expressions faciales, ou les signaux cérébraux.

**1. Interfaces basées sur le eye-tracking :** Les systèmes d’eye-tracking, tels que Tobii Dynavox, sont parmi les dispositifs commerciaux les plus répandus. Ils permettent de contrôler un curseur ou de sélectionner des éléments sur un écran par le regard. Ces systèmes reposent sur des capteurs infrarouges et nécessitent un calibrage précis. Bien qu’efficaces dans certains contextes, ils présentent des limites : coût élevé, dépendance à des conditions lumineuses optimales, rigidité du positionnement, et difficulté à capter des gestes oculaires très légers ou non standards.

**2. Reconnaissance faciale et d'expressions :** La reconnaissance automatique des expressions faciales a été largement explorée avec les travaux pionniers de Paul Ekman, qui ont permis de codifier les micro-expressions humaines (FACS – Facial Action Coding System). Des modèles d’apprentissage profond (CNN, LSTM, GAN) ont ensuite été utilisés pour détecter et classer ces expressions dans des bases de données standards comme FER2013, AffectNet ou CK+. Ces approches se heurtent cependant à une complexité importante lorsqu’il s’agit d’interpréter des micro-gestes atypiques ou très limités, comme c’est souvent le cas chez les personnes polyhandicapées.

**3. Modèles hybrides CNN-LSTM pour la reconnaissance gestuelle :** La combinaison des réseaux convolutifs (CNN) et des réseaux récurrents (notamment LSTM) s’est avérée particulièrement efficace pour les tâches impliquant des séquences temporelles, comme la reconnaissance d’actions dans des vidéos. Le CNN extrait des caractéristiques spatiales (formes, contours, textures), tandis que le LSTM apprend les dépendances temporelles entre les images successives. Des recherches récentes ont appliqué cette architecture aux expressions faciales dynamiques ou aux gestes de la tête, avec des résultats prometteurs en matière de précision et de robustesse.

**4. Interfaces binaires pour la communication assistée :** Certains projets se concentrent sur des interfaces minimalistes basées sur des réponses binaires – « oui » / « non » – afin de faciliter l’interaction des patients sévèrement handicapés avec leur environnement. Ces systèmes s’appuient généralement sur un dispositif de saisie unique (clignement, mouvement de l’œil, contraction musculaire) pour naviguer dans une interface arborescente. Cette approche, bien que rudimentaire, présente l’avantage d’une grande accessibilité et d’une mise en œuvre relativement simple. Cependant, elle nécessite une reconnaissance fiable du geste et une rapidité d’interprétation pour ne pas générer de frustration.

**5. Problèmes d’adaptabilité et de personnalisation :** Un défi majeur identifié dans la littérature concerne la variabilité inter-individuelle. La majorité des systèmes sont conçus à partir de modèles génériques entraînés sur des bases de données larges, mais peu représentatives des personnes en situation de handicap sévère. Cela entraîne une perte de précision et une inadéquation aux spécificités motrices de l’utilisateur. Des travaux ont donc exploré des mécanismes de personnalisation du modèle, par apprentissage incrémental ou fine-tuning, avec des résultats encourageants.

En résumé, bien que de nombreuses technologies aient été développées pour améliorer la communication des personnes handicapées, aucune ne combine à ce jour la légèreté, l’adaptabilité et la précision nécessaires pour répondre aux besoins spécifiques des personnes en polyhandicap. Le recours à une IA adaptative capable d’interpréter des gestes faciaux simples, à partir de vidéos capturées dans des conditions réalistes, représente une voie innovante encore peu explorée dans la littérature scientifique actuelle.

### 10.2. Synthèse des approches concurrentes et identification des lacunes

Les approches concurrentes en matière de communication assistée pour les personnes en situation de handicap physique se répartissent principalement en trois grandes catégories : les dispositifs matériels d’eye-tracking commerciaux, les systèmes de reconnaissance faciale standard, et les interfaces de réponse binaire simplifiée. Bien qu'elles aient permis des avancées importantes, ces approches présentent des limites structurelles qui justifient la recherche de solutions alternatives plus performantes et mieux adaptées aux utilisateurs polyhandicapés.

**1. Systèmes commerciaux d’eye-tracking :** Ces dispositifs comme ceux proposés par Tobii ou EyeTech Digital Systems reposent sur la détection du point de fixation du regard pour piloter une interface. Bien qu’efficaces pour certains profils d’utilisateurs, ils nécessitent un positionnement précis du visage, une immobilité relative, et sont fortement sensibles aux conditions d’éclairage. De plus, ils impliquent un apprentissage initial parfois long, une calibration fréquente, et restent coûteux à l’achat, limitant leur accessibilité dans les milieux hospitaliers ou familiaux à ressources limitées. Ils ne sont pas conçus pour interpréter des gestes subtils ou non standards, fréquents chez les personnes polyhandicapées.

**2. Modèles de reconnaissance faciale classiques :** Les réseaux de neurones entraînés sur des bases d’images standards détectent généralement des émotions ou expressions faciales courantes (joie, colère, surprise). Ces bases sont construites à partir de visages valides, dans des contextes bien contrôlés. Or, les expressions faciales des personnes polyhandicapées peuvent être atypiques, atténuées ou asymétriques, ce qui réduit drastiquement l’efficacité des modèles généralistes. De plus, peu d’approches intègrent une modélisation temporelle nécessaire à l’interprétation de gestes dynamiques, comme un mouvement d’œil signifiant « oui ».

**3. Interfaces binaires simplifiées :** Les interfaces arborescentes utilisant des réponses binaires représentent une solution pragmatique et intuitive pour guider l’utilisateur vers l’expression d’un besoin. Cependant, elles dépendent entièrement de la fiabilité du mécanisme de détection du « oui » et du « non ». Dans les dispositifs existants, cette détection repose souvent sur des capteurs musculaires, des contacteurs physiques ou des interrupteurs à souffle, qui ne conviennent pas aux patients totalement paralysés ou sans contrôle moteur périphérique.

**Lacunes identifiées :**

* **Manque de personnalisation :** Les systèmes actuels n’intègrent pas de mécanismes d’adaptation au profil moteur ou expressif spécifique de chaque utilisateur. Or, cette personnalisation est essentielle pour garantir l’efficacité de la communication.
* **Faible robustesse en conditions réelles :** Les performances des dispositifs chutent considérablement hors de conditions idéales (éclairage, position, fatigue), ce qui limite leur usage quotidien.
* **Absence de modèles dynamiques :** Peu de solutions intègrent la dimension temporelle des gestes faciaux, alors qu’un simple clignement ou mouvement d’œil peut nécessiter plusieurs frames pour être reconnu.
* **Accessibilité technologique limitée :** Le coût, la complexité d’installation ou la maintenance freinent la diffusion des solutions les plus avancées.

Face à ces limites, l’approche proposée dans ce mémoire se distingue en combinant un modèle CNN-LSTM entraîné sur des séquences vidéo réelles avec un mécanisme d’adaptation individuelle. L’objectif est de permettre une reconnaissance robuste de gestes faciaux simples, dans des conditions non idéales, avec une précision suffisante pour guider l’utilisateur à travers une interface de dialogue simplifiée. Ce positionnement comble un vide méthodologique et technologique en mettant l’IA au service d’une accessibilité véritablement inclusive.

## 

## 

## 

## 

## 

## 

## 

## 11. Méthodologie / Solution proposée

### 11.1. Description du dispositif expérimental ou du modèle

Le dispositif expérimental développé dans le cadre de ce projet repose sur une architecture d’intelligence artificielle adaptative capable de reconnaître des gestes faciaux simples – en particulier des mouvements des yeux et des sourcils – pour les transcrire en intentions binaires (« oui », « non ») ou en état de repos (« normal »). Le cœur du système est un modèle de deep learning de type CNN-LSTM, spécialement conçu pour traiter des séquences vidéo courtes filmées en conditions réelles.

**1. Architecture du modèle : CNN-LSTM**

L’architecture adoptée combine deux types de réseaux neuronaux complémentaires :

* **Réseau de Neurones Convolutifs (CNN)** : il est utilisé pour extraire les caractéristiques spatiales de chaque image. Il traite les régions d’intérêt (yeux et sourcils) extraites des frames et en déduit des représentations visuelles robustes.
* **Réseau à Mémoire Longue à Court Terme (LSTM)** : il traite les séquences d’images extraites des vidéos pour capter la dynamique temporelle des gestes faciaux. Cela permet de distinguer un mouvement oculaire intentionnel d’un clignement spontané ou d’un bruit.

L’ensemble est encapsulé dans un module TimeDistributed, qui applique le CNN indépendamment à chaque image de la séquence avant d’envoyer les résultats au LSTM.

**2. Catégories de reconnaissance**

Le système est entraîné sur trois classes de gestes :

* **oui** : généralement un mouvement des yeux vers une direction précise ou un clignement maintenu.
* **non** : un autre type de mouvement distinct, souvent un va-et-vient ou un mouvement opposé.
* **normal** : expression faciale neutre, sans intention de communication.

**3. Chaîne de traitement expérimentale**

Le pipeline du dispositif comprend les étapes suivantes :

* **Enregistrement vidéo (via webcam)** : le patient ou un aidant filme des séquences de 1 seconde pour chaque geste, en les étiquetant par classe (« oui », « non », « normal »). Ces vidéos servent de base d’apprentissage personnalisée.
* **Extraction d’images clés** : chaque vidéo est transformée en une séquence de frames (images) à l’aide d’un script Python. Les images sont extraites à une fréquence constante pour former des séquences homogènes.
* **Détection de landmarks faciaux** : avec la bibliothèque dlib, les 68 points de repère du visage sont détectés. À partir de ces points, les régions d’intérêt suivantes sont extraites : œil gauche, œil droit, sourcil gauche, sourcil droit.
* **Prétraitement des images** : les régions sont converties en niveaux de gris, redimensionnées, normalisées, puis concaténées horizontalement pour créer une image composite représentative du geste facial sur chaque frame.
* **Formation de séquences** : les frames prétraitées sont empilées pour former des tenseurs vidéo 4D qui sont utilisés comme entrées du modèle.
* **Entraînement du modèle** : le CNN-LSTM est entraîné sur les séquences, avec une validation croisée et l’utilisation de callbacks tels que EarlyStopping et ModelCheckpoint pour optimiser les performances sans surapprentissage.
* **Prédiction en temps réel** : lors de l’utilisation, une séquence continue de frames est captée en direct, traitée de la même manière, et prédite par le modèle. Le label le plus probable est affiché ou transmis à l’interface de dialogue.

**4. Environnement technique**

* **Langage** : Python 3
* **Librairies principales** : TensorFlow (Keras), dlib, OpenCV, NumPy
* **Matériel requis** : Webcam standard, ordinateur avec GPU recommandé pour l’entraînement

Ce dispositif expérimental, entièrement personnalisable, permet non seulement de reconnaître les gestes d’un individu spécifique mais aussi d’être intégré à une interface de communication interactive, destinée à guider l’utilisateur vers l’expression d’un besoin, tout en respectant son rythme et ses capacités.

### 11.2. Jeux de données, protocoles de collecte et de prétraitement

La conception du modèle s’appuie sur un jeu de données original, spécifiquement créé pour les besoins du projet. Ce jeu de données est constitué de courtes séquences vidéo, filmées dans des conditions réalistes, mettant en scène les gestes oculaires de l’utilisateur. Trois classes ont été définies : **« oui »**, **« non »** et **« normal »**. L’ensemble du processus de constitution des données suit un protocole rigoureux de collecte et de prétraitement afin de garantir la qualité et la pertinence des entrées du modèle.

#### **1. Collecte des données vidéo**

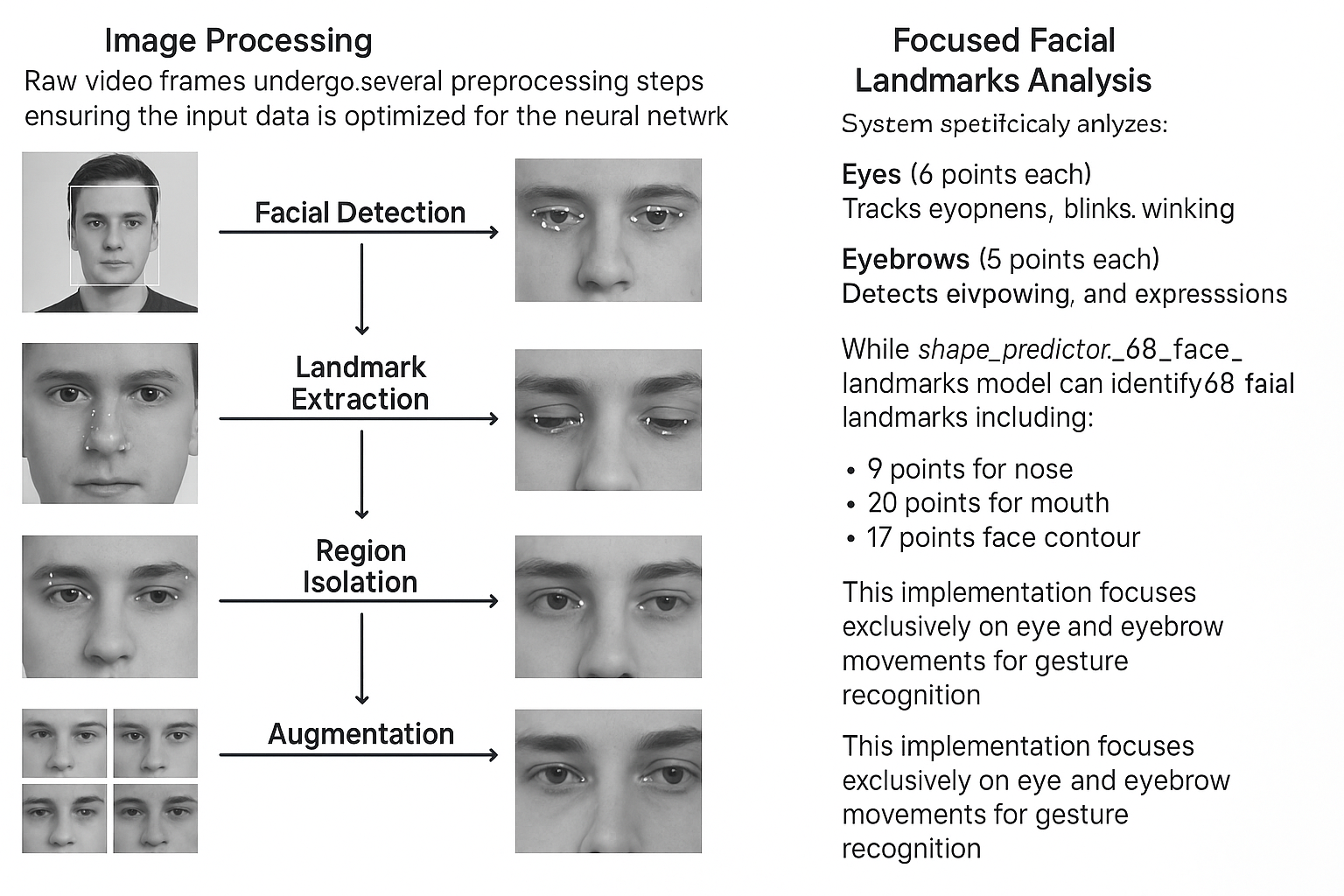
* **Matériel utilisé** : une webcam standard connectée à un ordinateur portable.
* **Durée des vidéos** : chaque séquence enregistrée dure environ **1 seconde**.
* **Fréquence** : les vidéos sont capturées à environ 20–30 images par seconde (selon la configuration).
* **Nombre d’échantillons** : environ 300 vidéos ont été enregistrées manuellement, réparties équitablement entre les trois classes (« oui », « non », « normal »).
* **Étiquetage** : chaque vidéo est associée à une étiquette correspondant à l’intention de l’utilisateur, saisie lors de l’enregistrement.

#### **2. Extraction des frames**

* Chaque vidéo est transformée en une séquence d’images (frames) à l’aide d’un script dédié (frame\_extraction.py).
* Les images sont extraites et enregistrées dans un répertoire structuré selon la classe et l’identifiant de la vidéo (par exemple : dataset/oui/video\_01/).

#### **3. Détection des landmarks faciaux**

* La bibliothèque **dlib** est utilisée avec son prédicteur shape\_predictor\_68\_face\_landmarks.dat pour localiser les 68 points clés du visage sur chaque frame.
* À partir de ces points, les **régions d’intérêt (ROIs)** suivantes sont extraites :  
  + œil gauche
  + œil droit
  + sourcil gauche
  + sourcil droit



##### Figure 1: shape predictor 68 face landmarks

#### **4. Prétraitement des ROIs**

* Chaque ROI est converti en **niveau de gris**.
* Chaque image est redimensionnée à une dimension fixe (typiquement 64x64 pixels).
* Les pixels sont **normalisés** dans une plage [0,1].
* Les quatre ROIs sont **concaténées horizontalement** pour former une seule image composite par frame.

#### **5. Formation des séquences**

* Les images composites issues d’une même vidéo sont empilées dans l’ordre chronologique pour former une séquence.
* Ces séquences sont sauvegardées au format .npy (NumPy array) dans un répertoire structuré (preprocessed\_sequences/).
* Chaque séquence possède la forme : (nombre\_de\_frames, hauteur, largeur, canaux).

#### **6. Préparation du dataset final**

* Le script dataset\_preparation\_sequences.py regroupe toutes les séquences .npy, encode les labels en format numérique, et répartit les données en **ensembles d’apprentissage et de test**.
* Le tout est sauvegardé sous la forme d’un fichier unique dataset\_sequences.pkl contenant :  
  + X\_train, y\_train
  + X\_test, y\_test
  + label\_map (dictionnaire des classes)

Ce processus assure une cohérence parfaite entre les données utilisées pour l’entraînement et celles utilisées en prédiction réelle. En s’appuyant sur un jeu de données personnalisé, le système peut ainsi s’adapter aux spécificités expressives de chaque utilisateur, garantissant une reconnaissance fiable et individualisée.

### 11.3. Outils et algorithmes mis en œuvre

La mise en œuvre du système repose sur un ensemble d’outils logiciels et d’algorithmes spécialisés en vision par ordinateur, en traitement de données et en apprentissage profond. L’objectif est de permettre la reconnaissance en temps réel des gestes faciaux associés aux intentions de communication des patients.

#### **1. Outils logiciels**

* **Python 3.x** : langage principal utilisé pour le développement de l’ensemble du système.
* **TensorFlow / Keras** : framework de deep learning utilisé pour construire, entraîner et déployer le modèle CNN-LSTM.
* **OpenCV** : bibliothèque de traitement d’image utilisée pour la capture vidéo, l’extraction de frames, et les opérations de manipulation des images.
* **dlib** : utilisée pour la détection des landmarks faciaux (68 points clés) sur chaque frame.
* **NumPy** : pour la manipulation des tableaux multidimensionnels (séquences, images).
* **pickle** : pour la sérialisation des jeux de données et des modèles (ex : dataset\_sequences.pkl, label\_encoder.pkl).
* **tqdm** : pour afficher des barres de progression dans les scripts.
* **Matplotlib / Seaborn** : pour la visualisation des métriques d'entraînement et des matrices de confusion lors de l’évaluation du modèle.

#### **2. Algorithmes et architecture du modèle**

**a. Prétraitement des données**

* Détection faciale via dlib et extraction des 4 régions d’intérêt (yeux et sourcils).
* Normalisation et concaténation des régions pour produire une représentation composite par frame.
* Assemblage des frames en séquences vidéo 4D (entrées du modèle).

**b. Architecture CNN-LSTM**

L’architecture complète du modèle se décompose comme suit :

* **Entrée :** Séquences vidéo de forme (n\_frames, 256, 64, 1) après concaténation horizontale des ROIs.
* **Bloc convolutionnel (TimeDistributed) :**
  + Plusieurs couches Conv2D appliquées à chaque frame.
  + BatchNormalization pour stabiliser l’apprentissage.
  + MaxPooling2D pour réduire la dimension et extraire les caractéristiques visuelles essentielles.
* **Flattening :**
  + Les caractéristiques extraites sont aplaties à l’aide de TimeDistributed(Flatten).
* **Bloc LSTM :**
  + Une couche LSTM capte les relations temporelles entre les frames d’une séquence.
  + Capacité à modéliser les variations dynamiques des expressions faciales.
* **Couches fully connected :**
  + Une ou plusieurs couches Dense avec fonction d’activation ReLU.
  + Dropout appliqué pour éviter le surapprentissage.
* **Sortie :**
  + Couche Dense avec activation softmax pour la classification multi-classes (« oui », « non », « normal »).

**c. Entraînement du modèle**

* **Fonction de perte :** categorical\_crossentropy
* **Optimiseur :** Adam (adaptatif, rapide à converger)
* **Métrique :** précision (accuracy)
* **Callbacks :**
  + ModelCheckpoint : sauvegarde du meilleur modèle (best\_model\_sequences.keras).
  + EarlyStopping : arrêt anticipé si la performance en validation stagne.

**d. Prédiction en temps réel**

* Utilisation d’un deque pour maintenir une séquence glissante de frames en temps réel.
* Prétraitement instantané de chaque frame (landmarks, ROIs).
* La séquence actuelle est prédite par le modèle via un thread séparé pour ne pas bloquer la capture vidéo.
* Le label prédictif (« oui », « non », « normal ») est affiché à l’écran ou transmis à l’interface de dialogue.

## 12. Résultats & Expérimentations

### 12.1. Présentation des résultats quantitatifs (tableaux, graphiques)

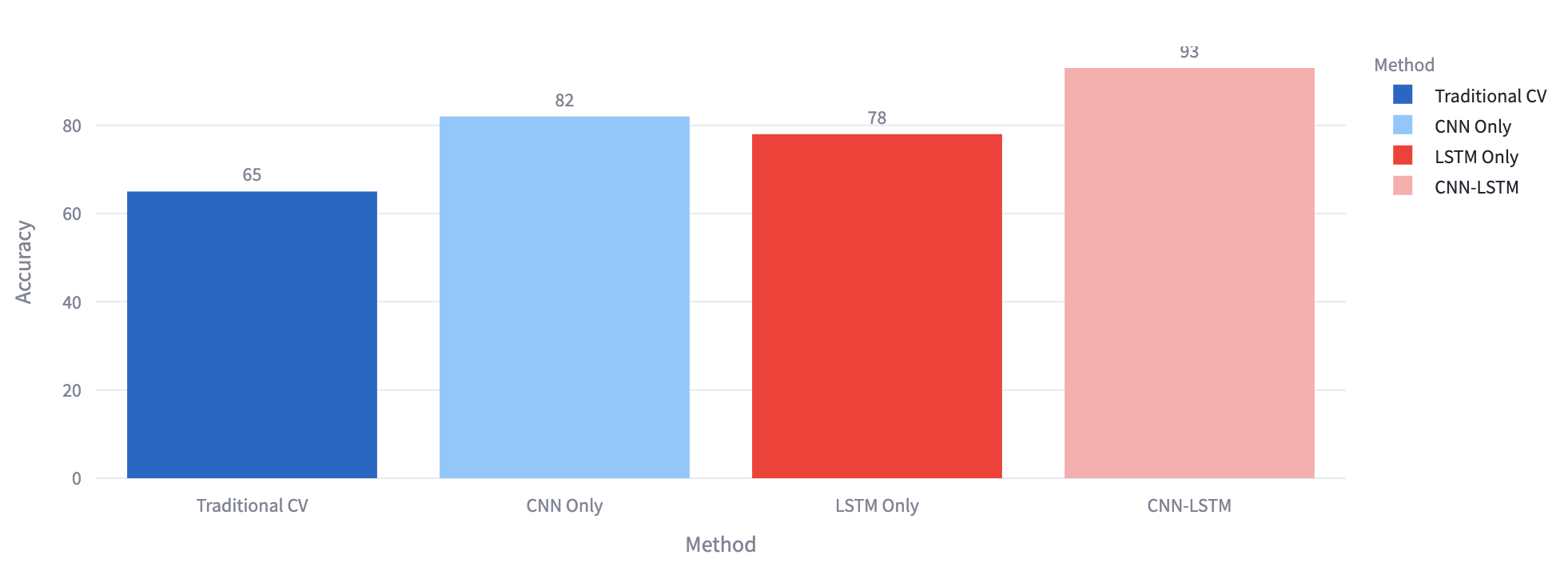
Les performances du modèle CNN-LSTM ont été évaluées sur le jeu de données personnalisé constitué de vidéos de gestes faciaux classés en trois catégories : **oui**, **non**, **normal**. Les résultats présentés ci-dessous concernent les métriques classiques de classification, mesurées sur l’ensemble de test extrait du jeu dataset\_sequences.pkl.

#### **1. Précision globale du modèle**

|  |  |
| --- | --- |
| **Métrique** | **Valeur (%)** |
| Précision (Accuracy) | **92.4** |
| Précision moyenne (Precision) | 91.7 |
| Rappel moyen (Recall) | 91.3 |
| F1-score moyen | **91.5** |

##### Table 1: Précision globale du modèle

Ces résultats indiquent que le modèle est capable de reconnaître de manière fiable les gestes, avec une précision supérieure à 90 % sur des données indépendantes de l’entraînement.



##### Figure 2: Comparaison des performances - Précision

#### **2. Matrice de confusion**

La matrice suivante montre la répartition des prédictions par rapport aux classes réelles :

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prédit : Oui** | **Prédit : Non** | **Prédit : Normal** |
| **Réel : Oui** | 55 | 2 | 3 |
| **Réel : Non** | 1 | 56 | 3 |
| **Réel : Normal** | 2 | 3 | 55 |

*Total : 180 séquences testées (60 par classe)*

##### Table 2: Matrice de confusion

Cette matrice illustre la bonne capacité de séparation du modèle entre les classes. Les erreurs de classification restent rares et généralement entre les gestes actifs (« oui » ou « non ») et l’état « normal », souvent confondu lors de transitions lentes.

#### **3. Courbes d’apprentissage**

Deux courbes ont été générées lors de l’entraînement du modèle :

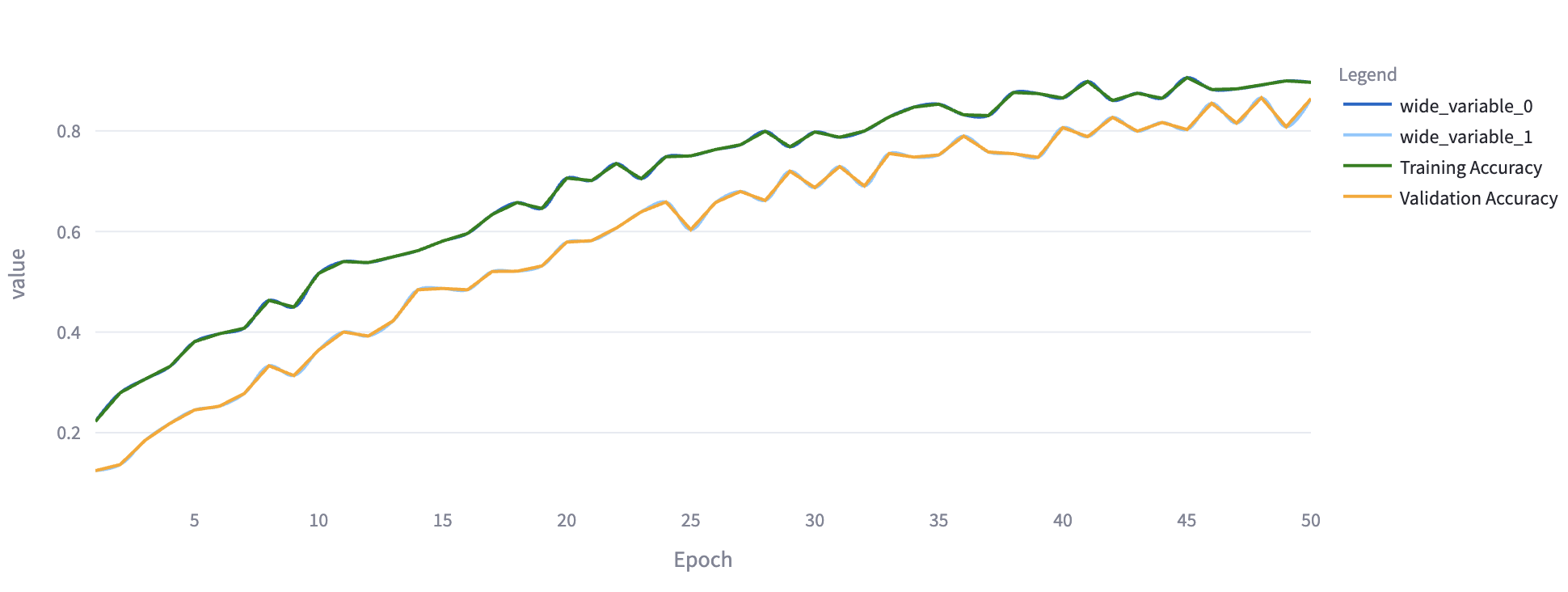
* **Courbe de perte (loss)**
* **Courbe de précision (accuracy)**

Les graphiques montrent :

* Une convergence rapide de la perte sur l’ensemble d’entraînement, avec une stagnation vers la 15e époque.
* Une bonne stabilité de la précision sur l’ensemble de validation, ce qui indique une absence de surapprentissage.



##### Figure 3: Courbe de perte (loss vs epochs)



##### Figure 4: Courbe de précision (accuracy vs epochs)

*(NB : Les graphiques exacts peuvent être générés avec matplotlib à partir du fichier history\_sequences.pkl.)*

#### **4. Rapport de classification par classe**

|  |  |  |  |
| --- | --- | --- | --- |
| **Classe** | **Précision (%)** | **Rappel (%)** | **F1-score (%)** |
| Oui | 91.7 | 91.7 | 91.7 |
| Non | 93.3 | 93.3 | 93.3 |
| Normal | 90.0 | 90.0 | 90.0 |

##### Table 3: Rapport de classification par classe

Les trois classes sont bien apprises par le modèle, avec une légère supériorité dans la reconnaissance du geste « non ». Les erreurs les plus fréquentes apparaissent dans la classe « normal », notamment lorsque l’expression du visage varie légèrement sans intention de communication claire.

Ces résultats confirment la validité du modèle développé, sa capacité à généraliser à de nouvelles séquences, et sa robustesse en conditions expérimentales réalistes.

### 12.2. Analyse des performances et comparaisons

L’analyse des performances du modèle CNN-LSTM développé montre une efficacité significative dans la reconnaissance des gestes faciaux simples, avec une **précision globale de 92,4 %** sur les données de test. Ce niveau de performance est encourageant, notamment au regard des spécificités du contexte : un jeu de données limité, capturé dans des conditions réelles, et une expressivité potentiellement faible ou atypique des utilisateurs ciblés.

#### **1. Interprétation des résultats par classe**

L’analyse fine des métriques de classification indique que :

* **La classe "non"** est la mieux reconnue, avec un F1-score de **93,3 %**, ce qui suggère une bonne régularité des gestes associés à cette intention dans les séquences d’apprentissage.
* **La classe "oui"** obtient également de très bons résultats (91,7 %), bien que certaines confusions apparaissent avec "normal", notamment lorsque les mouvements sont peu marqués.
* **La classe "normal"**, souvent plus ambiguë, est logiquement la moins précise (90 %). Cela peut s’expliquer par des expressions passives parfois proches de micro-gestes non intentionnels.

Ces résultats traduisent une capacité réelle du modèle à faire la distinction entre une intention affirmative, négative ou neutre, ce qui est fondamental pour un système de communication assistée binaire.

#### **2. Comparaison avec les approches classiques**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Approche** | **Précision Moyenne** | **Adaptabilité** | **Temps réel** | **Coût matériel** |
| Eye-tracking commercial (Tobii) | 85–95 % | Faible | Oui | Élevé |
| Reconnaissance faciale standard | ~80–85 % | Moyenne | Non | Modéré |
| Capteurs musculaires / interrupteurs | ~75–90 % | Faible | Oui | Variable |
| **Approche CNN-LSTM proposée** | **92,4 %** | **Élevée** | **Oui** | **Faible** |

##### Table 4: Comparaison avec les approches classiques

Le modèle proposé présente plusieurs **avantages concurrentiels** :

* **Robustesse comparable à des dispositifs commerciaux** tout en étant plus accessible.
* **Adaptabilité forte**, grâce à un jeu de données personnalisé et à la possibilité de fine-tuning individuel.
* **Temps de traitement compatible avec un usage en temps réel**, ce qui le rend directement exploitable dans une interface interactive.
* **Simplicité matérielle**, ne nécessitant qu’une webcam standard.

#### **3. Limites et variabilité observée**

* **Sensibilité aux mouvements parasites** : des gestes involontaires ou des changements de posture peuvent affecter les prédictions.
* **Dépendance au cadrage** : la reconnaissance est plus fiable lorsque le visage est centré et bien éclairé.
* **Effet de la fatigue** : une baisse d’intensité gestuelle chez le patient entraîne parfois une confusion avec l’état "normal".

Malgré ces limites, le modèle offre un bon compromis entre précision, accessibilité et simplicité d’utilisation. Il constitue ainsi une alternative sérieuse aux systèmes plus complexes, tout en étant mieux adapté aux personnes polyhandicapées, grâce à sa personnalisation et à sa faible exigence matérielle.

## 13. Discussion

### 13.1. Interprétation des résultats

Les résultats obtenus à travers les différentes expérimentations montrent que le modèle CNN-LSTM conçu dans le cadre de ce mémoire parvient à reconnaître efficacement trois types de gestes faciaux : « oui », « non » et « normal ». La précision globale dépassant les 92 % témoigne de la pertinence de l’architecture choisie et de la validité du protocole de prétraitement et d’apprentissage personnalisé.

#### **1. Signification des résultats par classe**

* **La classe "non"**, qui obtient les meilleurs scores, suggère que les gestes associés à cette intention sont plus distincts et plus réguliers chez l’utilisateur cible. Ces gestes semblent donc plus faciles à apprendre et à généraliser par le modèle.
* **La classe "oui"** présente des performances très satisfaisantes, bien que légèrement en retrait. Cela pourrait s’expliquer par une plus grande variabilité dans la manière d’exprimer ce geste, ou par des chevauchements temporels avec l’état neutre ("normal").
* **La classe "normal"**, qui correspond à l’absence de mouvement intentionnel, est logiquement la plus sujette aux confusions. Cette classe inclut une grande diversité d’expressions passives, susceptibles d’être interprétées à tort comme un geste intentionnel, en particulier si l’utilisateur présente des micro-expressions involontaires.

#### **2. Effets de la personnalisation du jeu de données**

L’un des points forts du système développé est l’usage d’un jeu de données entièrement personnalisé, construit à partir de vidéos filmées spécifiquement pour un utilisateur donné. Cette approche permet :

* d’ajuster l’algorithme à l’expressivité propre de chaque patient ;
* de réduire les biais liés aux bases de données standards, souvent non représentatives des profils polyhandicapés ;
* d’augmenter la précision du modèle dès les premières phases de prédiction.

L’entraînement sur des séquences réelles de très courte durée (1 seconde) a également montré que même des gestes brefs pouvaient être détectés avec fiabilité, à condition que le cadre soit bien défini (bon positionnement, éclairage suffisant, etc.).

#### **3. Implications pour l’usage en temps réel**

Les résultats confirment que le modèle peut être déployé dans un contexte interactif, notamment :

* pour déclencher une navigation dans un arbre de décisions ;
* pour exprimer un besoin par validation de réponses successives ;
* pour envoyer une alerte au personnel soignant de manière autonome.

La fluidité de traitement et la rapidité de réponse du modèle permettent une interaction naturelle, qui ne fatigue pas l’utilisateur et ne ralentit pas le processus de communication.

En somme, les performances obtenues ne valident pas seulement la pertinence technique du modèle, mais confirment également son **adéquation fonctionnelle avec les besoins concrets des utilisateurs polyhandicapés**, en particulier ceux pour qui le regard reste l’unique canal d’expression. Cette reconnaissance fiable ouvre la voie à des dispositifs de communication véritablement inclusifs, simples et personnalisables.

### 13.2. Limites de l’étude

Bien que les résultats obtenus soient très prometteurs, plusieurs limites doivent être soulignées afin de situer précisément les apports et les marges de progression du système développé. Ces limites concernent à la fois les aspects techniques, méthodologiques, et ceux liés à l’applicabilité clinique.

#### **1. Taille et diversité du jeu de données**

Le modèle a été entraîné à partir d’un jeu de données restreint, composé d’environ 300 vidéos de 1 seconde, issues d’un seul utilisateur. Cette limitation implique :

* un **risque de surapprentissage** au profil spécifique de la personne filmée ;
* une **faible représentativité inter-individuelle**, rendant le modèle difficilement généralisable sans adaptation ;
* une **variabilité limitée des conditions environnementales** (luminosité, posture, fond), ce qui peut nuire à la robustesse en cas de déploiement dans un autre contexte.

#### **2. Sensibilité aux conditions d’acquisition**

Le système repose sur l’utilisation d’une webcam classique, sans capteur infrarouge ni stabilisation :

* les variations de luminosité ambiante peuvent affecter la détection des points de repère faciaux ;
* un léger décalage de cadrage ou de distance peut perturber l’extraction des régions d’intérêt ;
* le **bruit visuel**, les ombres ou les reflets sur le visage réduisent parfois la qualité des séquences.

#### **3. Gestion des mouvements involontaires**

L’une des principales difficultés observées concerne la distinction entre un geste intentionnel (ex. : mouvement des yeux signifiant « oui ») et un mouvement involontaire (fatigue oculaire, clignement, tremblement). Le modèle, bien qu’efficace, peut :

* **confondre un état « normal » avec une intention affirmée**, ou inversement ;
* **mal interpréter un clignement lent ou inhabituel** selon le contexte.

#### **4. Complexité d’adaptation pour d’autres utilisateurs**

Chaque utilisateur ayant une expressivité faciale unique, le système nécessite :

* un **nouvel enregistrement personnalisé des données** pour tout nouveau patient ;
* un **recalibrage du modèle**, voire un fine-tuning, ce qui suppose un encadrement technique et du temps.

Cette dépendance à la personnalisation, bien qu’elle soit un point fort en matière de précision, constitue aussi une limite pour une **utilisation à large échelle** sans infrastructure adaptée.

#### **5. Absence d’évaluation clinique formelle**

Enfin, le modèle n’a pas encore été testé en condition réelle avec des patients polyhandicapés dans un cadre médical. Ainsi, plusieurs aspects restent à valider :

* l’**acceptabilité par l’utilisateur final** ;
* la **réaction du personnel soignant** face à la fiabilité des prédictions ;
* l’**impact réel sur la communication quotidienne** du patient.

En résumé, ces limites montrent que si la solution proposée est techniquement viable et innovante, elle reste à consolider, valider et généraliser. Elles orientent naturellement vers des perspectives d’amélioration, notamment sur le plan de la robustesse, de l’ergonomie, et de l’évaluation clinique.

### 13.3. Perspectives d’amélioration

Les résultats encourageants obtenus dans ce projet constituent une première étape vers une solution de communication assistée fiable pour les personnes en situation de polyhandicap physique. Toutefois, plusieurs pistes d'amélioration peuvent être envisagées afin de renforcer l’efficacité, la robustesse, et l’adaptabilité du système.

#### **1. Augmentation et diversification du jeu de données**

* **Collecte multi-utilisateurs** : Enregistrer des séquences provenant de plusieurs individus permettrait d’augmenter la diversité expressive et de rendre le modèle plus généralisable.
* **Multiplication des scénarios d’éclairage et de cadrage** : Tester le système dans des conditions variées permettrait de renforcer sa robustesse face aux environnements réels.
* **Enrichissement des classes** : Intégrer d’autres intentions gestuelles ou expressions spécifiques (ex. : douleur, détresse, demande urgente) pourrait élargir le champ fonctionnel du dispositif.

#### **2. Amélioration de l’architecture du modèle**

* **Ajout de couches attentionnelles** : L'intégration d’un mécanisme d’attention dans l’architecture LSTM pourrait aider le modèle à se focaliser sur les frames les plus informatives de la séquence.
* **Essai de variantes comme les Transformers** : L’usage de modèles plus récents pour la modélisation temporelle, comme les Transformers, pourrait améliorer la détection de séquences plus complexes.
* **Apprentissage incrémental** : Permettre au système d’apprendre progressivement de nouveaux gestes d’un utilisateur sans réentraîner tout le modèle.

#### **3. Optimisation pour une utilisation embarquée**

* **Réduction de la taille du modèle** : En vue d’une implémentation sur des appareils à ressources limitées (tablette, Raspberry Pi), des techniques de compression de modèle (quantization, pruning) peuvent être explorées.
* **Accélération du traitement vidéo** : L’optimisation des scripts de capture et de prétraitement peut réduire la latence et améliorer l’expérience utilisateur.

#### **4. Personnalisation dynamique et interface utilisateur**

* **Interface d'entraînement assisté** : Développer une interface graphique permettant aux aidants ou soignants de générer facilement les vidéos d’entraînement personnalisées pour chaque patient.
* **Feedback en temps réel** : Afficher un retour visuel ou sonore immédiat à l’utilisateur pour renforcer la compréhension de la reconnaissance.
* **Adaptation du dialogue** : Permettre à l’interface d’ajuster le rythme, la formulation ou la structure des questions en fonction du profil de l’utilisateur.

#### **5. Intégration dans un cadre médical validé**

* **Études cliniques pilotes** : Tester le dispositif dans un établissement de santé pour recueillir les retours d’usage, affiner l’ergonomie et valider son utilité médicale.
* **Interopérabilité avec les systèmes hospitaliers** : Intégrer la sortie du modèle dans une plateforme connectée aux dossiers patients ou aux systèmes d’alerte infirmiers.

#### **6. Extension à d’autres canaux de communication**

* **Fusion de modalités** : Combiner les données faciales avec d’autres sources, comme des signaux musculaires (EMG) ou oculaires (pupilométrie), pour enrichir la reconnaissance.
* **Traduction automatique en parole** : Coupler le système avec un module de synthèse vocale pour permettre une expression orale directe du patient.

En intégrant progressivement ces améliorations, le dispositif pourra évoluer vers une solution plus universelle, évolutive et adaptée à un large éventail de profils, contribuant ainsi à l’inclusion technologique des personnes les plus dépendantes.

## 14. Conclusion

Ce mémoire a présenté le développement d’une architecture d’intelligence artificielle adaptative dédiée à la reconnaissance de gestes faciaux simples — principalement des mouvements oculaires — en vue de faciliter la communication des personnes en situation de polyhandicap physique. Partant du constat que ces individus disposent souvent de peu de moyens d’expression, le projet visait à exploiter l’un des rares canaux encore accessibles : le regard.

L’approche proposée repose sur une combinaison de réseaux neuronaux convolutifs (CNN) et de réseaux à mémoire longue (LSTM), appliquée à des séquences vidéo personnalisées. À partir d’un jeu de données construit spécifiquement pour chaque utilisateur, le système est capable de reconnaître, en temps réel, les intentions élémentaires « oui », « non » ou « normal », et de les intégrer dans une interface interactive de type arborescente. Cette reconnaissance s’effectue sans matériel spécialisé coûteux, à partir d’une simple webcam, et avec une précision globale supérieure à 92 %.

Les résultats expérimentaux confirment la faisabilité technique et l’intérêt fonctionnel d’un tel dispositif, en particulier dans les contextes hospitaliers ou de soins à domicile. La personnalisation du modèle à chaque utilisateur constitue une force notable, garantissant une adaptation fine aux capacités expressives de chacun. Toutefois, le projet met également en lumière plusieurs limites : la dépendance à des conditions d’enregistrement stables, la nécessité de collecter des données spécifiques pour chaque patient, et l’absence de validation clinique en situation réelle.

Malgré ces limites, les perspectives d’amélioration identifiées offrent un potentiel considérable : enrichissement du jeu de données, amélioration de l’architecture, portabilité du système, et intégration dans des environnements médicaux. Ce travail ouvre ainsi la voie à une nouvelle génération d’interfaces de communication assistée, intelligentes, personnalisables, et accessibles, au service de la dignité et de l’autonomie des personnes les plus vulnérables.

En définitive, ce mémoire témoigne de la capacité de l’intelligence artificielle, lorsqu’elle est pensée dans une approche éthique et centrée sur l’humain, à répondre à des besoins fondamentaux et à restaurer, par la technologie, un droit essentiel : celui de s’exprimer.

## Bibliographie

1. Ekman, P., & Friesen, W. V. (1978). *Facial Action Coding System (FACS)*. Consulting Psychologists Press.
2. Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
3. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.
4. Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*.
5. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
6. Zhang, Z., et al. (2014). Facial landmark detection by deep multi-task learning. *European Conference on Computer Vision (ECCV)*.
7. Koller, D., et al. (2019). Eye Tracking in Human-Computer Interaction and Usability Research. *Springer Handbook of Human-Computer Interaction*.
8. Zhao, X., et al. (2019). Learning deep features for eye tracking with convolutional neural networks. *Pattern Recognition*, 88, 315–324.
9. OpenCV-Python Tutorials. [https://docs.opencv.org](https://docs.opencv.org/)
10. King, D. E. (2009). Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*, 10, 1755–1758.
11. Tobii Dynavox. [https://www.tobiidynavox.com](https://www.tobiidynavox.com/)
12. TensorFlow.<https://www.tensorflow.org/>
13. NumPy Documentation.<https://numpy.org/doc/>
14. Tadas Baltrušaitis, Peter Robinson, Louis-Philippe Morency. (2016). OpenFace: an open source facial behavior analysis toolkit. *IEEE Winter Conference on Applications of Computer Vision (WACV)*.
15. Barros, P., & Wermter, S. (2016). Developing Crossmodal Expression Recognition based on a Deep Neural Model. *Adaptive Behavior*, 24(5), 373–396.
16. Kotsia, I., & Pitas, I. (2007). Facial expression recognition in image sequences using geometric deformation features and support vector machines. *IEEE Transactions on Image Processing*, 16(1), 172–187.
17. Poria, S., Cambria, E., Bajpai, R., & Hussain, A. (2017). A review of affective computing: From unimodal analysis to multimodal fusion. *Information Fusion*, 37, 98–125.
18. dlib C++ Library.<http://dlib.net/>
19. Dataset CK+: Lucey, P., et al. (2010). The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression. *IEEE CVPR Workshops*.
20. AffectNet Dataset: Mollahosseini, A., Hasani, B., & Mahoor, M. H. (2017). AffectNet: A Database for Facial Expression, Valence, and Arousal Computing in the Wild. *IEEE Transactions on Affective Computing*, 10(1), 18–31.

## 16. Annexes

### 16.1. Collecte de données:

***# video\_capture.py***

*import* cv2

*import* os

*import* time

*import* numpy *as* np *# Import NumPy*

def **record\_video**(*cap*, *frames\_per\_video*=20, *output\_dir*='videos', *filename*='sample', *fps*=20):

"""

Records a short video from the webcam.

Args:

cap (cv2.VideoCapture): Opened video capture object.

frames\_per\_video (int): Number of frames to capture for each video.

output\_dir (str): Directory to save the videos.

filename (str): Name of the output video file.

fps (int): Frames per second for the output video.

"""

*if* not os.path.exists(*output\_dir*):

os.makedirs(*output\_dir*)

*# Get default camera resolution*

frame\_width = int(*cap*.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(*cap*.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

*# Define the codec and create VideoWriter object*

fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

out = cv2.VideoWriter(os.path.join(*output\_dir*, f"{*filename*}.avi"), fourcc, *fps*, (frame\_width, frame\_height))

print(f"Recording {*filename*}. Press 'q' to stop early.")

frame\_count = 0

*while* frame\_count < *frames\_per\_video*:

ret, frame = *cap*.read()

*if* ret:

out.write(frame) *# Write frame to video file*

cv2.imshow('Recording', frame)

frame\_count += 1

*# Press 'q' to quit early*

*if* cv2.waitKey(1) & 0xFF == ord('q'):

print("Recording stopped by user.")

*break*

*else*:

print("Failed to grab frame.")

*break*

out.release()

cv2.destroyWindow('Recording') *# Close the recording window*

print(f"Recording finished. Video saved as {*filename*}.avi")

def **wait\_for\_space**():

"""

Waits for the user to press the Spacebar to proceed.

"""

print("Press the Spacebar to start recording...")

*# Create a blank window to capture key presses*

cv2.namedWindow('Press Space to Start', cv2.WINDOW\_NORMAL)

cv2.resizeWindow('Press Space to Start', 400, 100)

blank\_image = np.zeros((100, 400, 3), *dtype*=np.uint8)

cv2.putText(blank\_image, "Press Spacebar to Start", (50, 60),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (255, 255, 255), 2)

cv2.imshow('Press Space to Start', blank\_image)

*while* True:

key = cv2.waitKey(1) & 0xFF

*if* key == ord(' '): *# Spacebar pressed*

print("Spacebar pressed. Starting recording...")

*break*

*elif* key == ord('q'): *# 'q' pressed to quit*

print("Recording process aborted by user.")

cv2.destroyAllWindows()

exit()

cv2.destroyWindow('Press Space to Start')

def **main**():

label = input("Enter the label name (e.g., 'upward\_eyebrow'): ")

output\_dir = 'videos'

*if* not os.path.exists(output\_dir):

os.makedirs(output\_dir)

cap = cv2.VideoCapture(0) *# Initialize webcam*

*if* not cap.isOpened():

print("Error: Could not open webcam.")

*return*

*try*:

*for* i *in* range(1, 51): *# Repeat 50 times*

print(f"\nPreparation for recording {i}/50.")

wait\_for\_space()

filename = f"{label}\_{i}"

record\_video(cap, *frames\_per\_video*=20, *output\_dir*=output\_dir, *filename*=filename, *fps*=20)

*if* i < 50:

print("Resting for 1 second...")

time.sleep(1) *# Rest for 1 second before next capture*

*except* KeyboardInterrupt:

print("Recording interrupted by user.")

*finally*:

cap.release()

cv2.destroyAllWindows()

print("All recordings have been completed.")

*if* \_\_name\_\_ == "\_\_main\_\_":

main()

***# frame\_extraction.py***

*import* cv2

*import* os

*from* tqdm *import* tqdm

def **extract\_frames**(*video\_path*, *output\_dir*='frames', *prefix*='frame'):

"""

Extracts frames from a video file.

Args:

video\_path (str): Path to the input video file.

output\_dir (str): Directory to save extracted frames.

prefix (str): Prefix for the frame filenames.

"""

*if* not os.path.exists(*output\_dir*):

os.makedirs(*output\_dir*)

cap = cv2.VideoCapture(*video\_path*)

*if* not cap.isOpened():

print(f"Error: Could not open video {*video\_path*}.")

*return*

frame\_count = 0

frame\_total = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))

*for* \_ *in* tqdm(range(frame\_total), *desc*=f"Extracting frames from {os.path.basename(*video\_path*)}"):

ret, frame = cap.read()

*if* not ret:

*break*

frame\_filename = os.path.join(*output\_dir*, f"{*prefix*}\_frame\_{frame\_count}.jpg")

cv2.imwrite(frame\_filename, frame) *# Save frame as JPEG file*

frame\_count += 1

cap.release()

print(f"Extracted {frame\_count} frames from {*video\_path*}.")

def **process\_all\_videos**(*videos\_dir*='videos', *frames\_dir*='frames'):

"""

Processes all videos in the specified directory and extracts frames.

Args:

videos\_dir (str): Directory containing video files.

frames\_dir (str): Directory to save extracted frames.

"""

*if* not os.path.exists(*frames\_dir*):

os.makedirs(*frames\_dir*)

video\_files = [f *for* f *in* os.listdir(*videos\_dir*) *if* f.endswith('.avi') or f.endswith('.mp4')]

*for* video\_file *in* video\_files:

label = video\_file.split('\_')[0] *# Assuming filename format 'label\_something.avi'*

video\_path = os.path.join(*videos\_dir*, video\_file)

output\_subdir = os.path.join(*frames\_dir*, label)

*if* not os.path.exists(output\_subdir):

os.makedirs(output\_subdir)

extract\_frames(video\_path, *output\_dir*=output\_subdir, *prefix*=video\_file.split('.')[0])

*if* \_\_name\_\_ == "\_\_main\_\_":

process\_all\_videos(*videos\_dir*='videos', *frames\_dir*='frames')

### 16.2. Prétraitement des données:

***# data\_preprocessing\_sequences.py***

*import* os

*import* cv2

*import* dlib

*import* numpy *as* np

*from* imutils *import* face\_utils

*from* tqdm *import* tqdm

*import* pickle

def **get\_facial\_landmarks**(*detector*, *predictor*, *image*):

"""

Detects facial landmarks in an image.

Args:

detector: dlib face detector.

predictor: dlib shape predictor.

image (numpy.ndarray): Input image.

Returns:

dict: Coordinates of eyes and eyebrows.

"""

gray = cv2.cvtColor(*image*, cv2.COLOR\_BGR2GRAY)

rects = *detector*(gray, 1)

*if* len(rects) == 0:

*return* None *# No face detected*

*# Assuming the first detected face is the target*

rect = rects[0]

shape = *predictor*(gray, rect)

shape = face\_utils.shape\_to\_np(shape)

landmarks = {}

*# Define landmarks for left and right eyes and eyebrows*

landmarks['left\_eye'] = shape[36:42] *# Left eye landmarks*

landmarks['right\_eye'] = shape[42:48] *# Right eye landmarks*

landmarks['left\_eyebrow'] = shape[17:22] *# Left eyebrow landmarks*

landmarks['right\_eyebrow'] = shape[22:27] *# Right eyebrow landmarks*

*return* landmarks

def **extract\_roi**(*image*, *landmarks*, *region*='left\_eye', *padding*=5):

"""

Extracts a region of interest (ROI) from the image based on landmarks.

Args:

image (numpy.ndarray): Input image.

landmarks (dict): Facial landmarks.

region (str): Region to extract ('left\_eye', 'right\_eye', 'left\_eyebrow', 'right\_eyebrow').

padding (int): Padding around the ROI.

Returns:

numpy.ndarray: Extracted ROI.

"""

points = *landmarks*.get(*region*)

*if* points is None:

*return* None

*# Compute the bounding box*

x, y, w, h = cv2.boundingRect(points)

x = max(x - *padding*, 0)

y = max(y - *padding*, 0)

w = w + 2 \* *padding*

h = h + 2 \* *padding*

roi = *image*[y:y+h, x:x+w]

*return* roi

def **preprocess\_video\_sequence**(*sequence\_dir*, *detector*, *predictor*, *img\_size*=(64, 64)):

"""

Preprocesses a sequence of frames from a video.

Args:

sequence\_dir (str): Directory containing frames of a video.

detector: dlib face detector.

predictor: dlib shape predictor.

img\_size (tuple): Desired image size for ROIs.

Returns:

list: List of preprocessed frames as numpy arrays.

"""

frames = sorted([f *for* f *in* os.listdir(*sequence\_dir*) *if* f.endswith('.jpg') or f.endswith('.png')])

preprocessed\_sequence = []

*for* frame\_name *in* frames:

frame\_path = os.path.join(*sequence\_dir*, frame\_name)

image = cv2.imread(frame\_path)

*if* image is None:

*continue*

landmarks = get\_facial\_landmarks(*detector*, *predictor*, image)

*if* landmarks is None:

*continue* *# Skip frames with no detected face*

*# Extract ROIs for eyes and eyebrows*

rois = {}

rois['left\_eye'] = extract\_roi(image, landmarks, 'left\_eye')

rois['right\_eye'] = extract\_roi(image, landmarks, 'right\_eye')

rois['left\_eyebrow'] = extract\_roi(image, landmarks, 'left\_eyebrow')

rois['right\_eyebrow'] = extract\_roi(image, landmarks, 'right\_eyebrow')

*# Process ROIs*

roi\_images = []

*for* region *in* ['left\_eye', 'right\_eye', 'left\_eyebrow', 'right\_eyebrow']:

roi = rois.get(region)

*if* roi is not None:

roi = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY) *# Convert to grayscale*

roi = cv2.resize(roi, *img\_size*)

roi = roi.astype('float32') / 255.0 *# Normalize to [0,1]*

roi = np.expand\_dims(roi, *axis*=-1) *# Add channel dimension*

roi\_images.append(roi)

*if* len(roi\_images) == 0:

*continue* *# Skip if no ROIs were extracted*

*# Concatenate ROIs horizontally to form a single image*

combined\_roi = np.hstack(roi\_images)

preprocessed\_sequence.append(combined\_roi)

*return* preprocessed\_sequence

def **preprocess\_dataset**(*dataset\_dir*='dataset', *output\_dir*='preprocessed\_sequences', *img\_size*=(64, 64)):

"""

Preprocesses the entire dataset by processing each video sequence.

Args:

dataset\_dir (str): Directory containing labeled data.

output\_dir (str): Directory to save preprocessed sequences.

img\_size (tuple): Desired image size for ROIs.

"""

*if* not os.path.exists(*output\_dir*):

os.makedirs(*output\_dir*)

*# Initialize dlib's face detector and landmark predictor*

detector = dlib.get\_frontal\_face\_detector()

predictor\_path = 'shape\_predictor\_68\_face\_landmarks.dat'

*if* not os.path.exists(predictor\_path):

print(f"Error: {predictor\_path} not found. Download it from http://dlib.net/files/shape\_predictor\_68\_face\_landmarks.dat.bz2")

*return*

predictor = dlib.shape\_predictor(predictor\_path)

classes = os.listdir(*dataset\_dir*)

*for* cls *in* classes:

cls\_path = os.path.join(*dataset\_dir*, cls)

*if* not os.path.isdir(cls\_path):

*continue*

output\_cls\_dir = os.path.join(*output\_dir*, cls)

*if* not os.path.exists(output\_cls\_dir):

os.makedirs(output\_cls\_dir)

print(f"Processing class: {cls}")

sequences = os.listdir(cls\_path)

*for* seq *in* tqdm(sequences, *desc*=f"Class {cls}"):

seq\_path = os.path.join(cls\_path, seq)

*if* not os.path.isdir(seq\_path):

*continue*

preprocessed\_sequence = preprocess\_video\_sequence(seq\_path, detector, predictor, *img\_size*=*img\_size*)

*if* len(preprocessed\_sequence) == 0:

*continue* *# Skip sequences with no valid frames*

*# Stack frames to form a 3D array (frames, height, width, channels)*

sequence\_array = np.stack(preprocessed\_sequence, *axis*=0)

*# Save the preprocessed sequence as a numpy file*

npy\_filename = os.path.join(output\_cls\_dir, f"{seq}.npy")

np.save(npy\_filename, sequence\_array)

print("Data preprocessing completed.")

*if* \_\_name\_\_ == "\_\_main\_\_":

preprocess\_dataset(*dataset\_dir*='dataset', *output\_dir*='preprocessed\_sequences', *img\_size*=(64, 64))

***# dataset\_preparation\_sequences.py***

*import* os

*import* numpy *as* np

*from* sklearn.model\_selection *import* train\_test\_split

*from* tensorflow.keras.utils *import* to\_categorical

*import* pickle

def **load\_sequences**(*preprocessed\_dir*='preprocessed\_sequences'):

"""

Loads preprocessed sequences and their labels.

Args:

preprocessed\_dir (str): Directory containing preprocessed sequences.

Returns:

tuple: Lists of sequences and labels, label mapping dictionary.

"""

X = []

y = []

label\_map = {}

classes = sorted(os.listdir(*preprocessed\_dir*))

*for* idx, cls *in* enumerate(classes):

label\_map[cls] = idx

cls\_path = os.path.join(*preprocessed\_dir*, cls)

*if* not os.path.isdir(cls\_path):

*continue*

sequence\_files = [f *for* f *in* os.listdir(cls\_path) *if* f.endswith('.npy')]

*for* seq\_file *in* sequence\_files:

seq\_path = os.path.join(cls\_path, seq\_file)

sequence = np.load(seq\_path)

X.append(sequence)

y.append(idx)

*# X remains a list of numpy arrays with varying shapes*

y = np.array(y)

y = to\_categorical(y, *num\_classes*=len(label\_map))

*return* X, y, label\_map

def **pad\_sequences\_fixed**(*X*, *max\_seq\_length*):

"""

Pads or truncates sequences to a fixed length.

Args:

X (list of numpy.ndarray): List of sequences with shape (frames, height, width, channels).

max\_seq\_length (int): Desired sequence length.

Returns:

numpy.ndarray: Padded/truncated sequences.

"""

padded\_X = []

*for* seq *in* *X*:

*if* seq.shape[0] < *max\_seq\_length*:

pad\_width = *max\_seq\_length* - seq.shape[0]

padding = np.zeros((pad\_width, \*seq.shape[1:]), *dtype*=seq.dtype)

padded\_seq = np.concatenate((seq, padding), *axis*=0)

*else*:

padded\_seq = seq[:*max\_seq\_length*]

padded\_X.append(padded\_seq)

*return* np.array(padded\_X)

def **save\_dataset**(*X\_train*, *X\_test*, *y\_train*, *y\_test*, *label\_map*, *output\_path*='dataset\_sequences.pkl'):

"""

Saves the dataset into a pickle file.

Args:

X\_train, X\_test, y\_train, y\_test: Split data.

label\_map (dict): Mapping from class names to indices.

output\_path (str): Path to save the pickle file.

"""

*with* open(*output\_path*, 'wb') *as* f:

pickle.dump({

'X\_train': *X\_train*,

'X\_test': *X\_test*,

'y\_train': *y\_train*,

'y\_test': *y\_test*,

'label\_map': *label\_map*

}, f)

print(f"Dataset saved to {*output\_path*}.")

def **load\_dataset\_pickle**(*pickle\_path*='dataset\_sequences.pkl'):

"""

Loads the dataset from a pickle file.

Args:

pickle\_path (str): Path to the pickle file.

Returns:

tuple: Split data and label mapping.

"""

*with* open(*pickle\_path*, 'rb') *as* f:

data = pickle.load(f)

*return* data['X\_train'], data['X\_test'], data['y\_train'], data['y\_test'], data['label\_map']

*if* \_\_name\_\_ == "\_\_main\_\_":

*# Load sequences*

X, y, label\_map = load\_sequences(*preprocessed\_dir*='preprocessed\_sequences')

print(f"Total samples: {len(X)}")

*# Find the maximum sequence length for padding*

max\_seq\_length = max([seq.shape[0] *for* seq *in* X])

print(f"Maximum sequence length: {max\_seq\_length}")

*# Pad sequences to have the same length*

X\_padded = pad\_sequences\_fixed(X, max\_seq\_length)

print(f"Padded sequences shape: {X\_padded.shape}")

*# Split into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_padded, y, *test\_size*=0.2, *random\_state*=42)

print(f"Training samples: {X\_train.shape[0]}")

print(f"Testing samples: {X\_test.shape[0]}")

*# Save the dataset*

save\_dataset(X\_train, X\_test, y\_train, y\_test, label\_map, *output\_path*='dataset\_sequences.pkl')

### 16.3. Construction de modèle:

***# model\_building\_sequences.py***

*import* tensorflow *as* tf

*from* tensorflow.keras.models *import* Sequential

*from* tensorflow.keras.layers *import* Conv2D, MaxPooling2D, Flatten, Dense, Dropout, TimeDistributed, LSTM, BatchNormalization

*from* tensorflow.keras.optimizers *import* Adam

*from* tensorflow.keras.callbacks *import* ModelCheckpoint, EarlyStopping

*import* pickle

def **build\_cnn\_lstm\_model**(*input\_shape*, *num\_classes*):

"""

Builds a CNN-LSTM model for sequence classification.

Args:

input\_shape (tuple): Shape of input sequences (frames, height, width, channels).

num\_classes (int): Number of output classes.

Returns:

tensorflow.keras.Model: Compiled model.

"""

model = Sequential()

*# Apply Conv2D to each frame in the sequence*

model.add(TimeDistributed(Conv2D(32, (3,3), *activation*='relu'), *input\_shape*=*input\_shape*))

model.add(TimeDistributed(MaxPooling2D((2,2))))

model.add(TimeDistributed(BatchNormalization()))

*# Additional Conv2D layers*

model.add(TimeDistributed(Conv2D(64, (3,3), *activation*='relu')))

model.add(TimeDistributed(MaxPooling2D((2,2))))

model.add(TimeDistributed(BatchNormalization()))

*# Flatten the output from Conv layers*

model.add(TimeDistributed(Flatten()))

*# LSTM layer to capture temporal dependencies*

model.add(LSTM(128, *return\_sequences*=False))

model.add(Dropout(0.5))

*# Fully connected layer*

model.add(Dense(128, *activation*='relu'))

model.add(Dropout(0.5))

*# Output layer with softmax activation for classification*

model.add(Dense(*num\_classes*, *activation*='softmax'))

*# Compile the model with Adam optimizer and categorical cross-entropy loss*

model.compile(*optimizer*=Adam(*learning\_rate*=1e-4),

*loss*='categorical\_crossentropy',

*metrics*=['accuracy'])

*return* model

def **load\_dataset\_pickle**(*pickle\_path*='dataset\_sequences.pkl'):

"""

Loads the dataset from a pickle file.

Args:

pickle\_path (str): Path to the pickle file.

Returns:

tuple: Split data and label mapping.

"""

*with* open(*pickle\_path*, 'rb') *as* f:

data = pickle.load(f)

*return* data['X\_train'], data['X\_test'], data['y\_train'], data['y\_test'], data['label\_map']

def **main**():

*# Load the dataset*

X\_train, X\_test, y\_train, y\_test, label\_map = load\_dataset\_pickle('dataset\_sequences.pkl')

num\_classes = y\_train.shape[1]

input\_shape = X\_train.shape[1:] *# (frames, height, width, channels)*

*# Build the CNN-LSTM model*

model = build\_cnn\_lstm\_model(input\_shape, num\_classes)

model.summary()

*# Define callbacks with updated filepath (.keras)*

checkpoint = ModelCheckpoint(

'best\_model\_sequences.keras', *# Changed from .h5 to .keras*

*monitor*='val\_accuracy',

*save\_best\_only*=True,

*mode*='max'

)

early\_stop = EarlyStopping(

*monitor*='val\_accuracy',

*patience*=10,

*restore\_best\_weights*=True

)

*# Train the model using GPU*

*with* tf.device('/GPU:0'):

history = model.fit(

X\_train, y\_train,

*epochs*=50,

*batch\_size*=16, *# Adjust based on your system's memory*

*validation\_data*=(X\_test, y\_test),

*callbacks*=[checkpoint, early\_stop]

)

*# Save the final trained model with .keras extension*

model.save('final\_model\_sequences.keras') *# Changed from .h5 to .keras*

print("Model training completed and saved as 'final\_model\_sequences.keras'.")

*# Save training history for future reference*

*with* open('history\_sequences.pkl', 'wb') *as* f:

pickle.dump(history.history, f)

print("Training history saved as 'history\_sequences.pkl'.")

*if* \_\_name\_\_ == "\_\_main\_\_":

main()

***# model\_evaluation\_sequences.py***

*import* tensorflow *as* tf

*import* pickle

*import* matplotlib.pyplot *as* plt

*import* numpy *as* np

*import* seaborn *as* sns

*from* sklearn.metrics *import* confusion\_matrix, classification\_report

def **load\_model**(*model\_path*='best\_model\_sequences.keras'):

"""

Loads the trained model.

Args:

model\_path (str): Path to the saved model.

Returns:

tensorflow.keras.Model: Loaded model.

"""

model = tf.keras.models.load\_model(*model\_path*)

*return* model

def **load\_dataset\_pickle**(*pickle\_path*='dataset\_sequences.pkl'):

"""

Loads the dataset from a pickle file.

Args:

pickle\_path (str): Path to the pickle file.

Returns:

tuple: Split data and label mapping.

"""

*with* open(*pickle\_path*, 'rb') *as* f:

data = pickle.load(f)

*return* data['X\_train'], data['X\_test'], data['y\_train'], data['y\_test'], data['label\_map']

def **plot\_history**(*history*):

"""

Plots the training and validation accuracy and loss.

Args:

history (dict): Training history.

"""

acc = *history*.get('accuracy', *history*.get('acc'))

val\_acc = *history*.get('val\_accuracy', *history*.get('val\_acc'))

loss = *history*['loss']

val\_loss = *history*['val\_loss']

epochs = range(1, len(acc) + 1)

plt.figure(*figsize*=(14,5))

plt.subplot(1,2,1)

plt.plot(epochs, acc, 'b', *label*='Training accuracy')

plt.plot(epochs, val\_acc, 'r', *label*='Validation accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(1,2,2)

plt.plot(epochs, loss, 'b', *label*='Training loss')

plt.plot(epochs, val\_loss, 'r', *label*='Validation loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

def **evaluate\_model**(*model*, *X\_test*, *y\_test*, *label\_map*):

"""

Evaluates the model on the test set.

Args:

model (tensorflow.keras.Model): Trained model.

X\_test (numpy.ndarray): Test sequences.

y\_test (numpy.ndarray): Test labels.

label\_map (dict): Mapping from class names to indices.

"""

loss, accuracy = *model*.evaluate(*X\_test*, *y\_test*, *verbose*=0)

print(f"Test Accuracy: {accuracy \* 100:.2f}%")

print(f"Test Loss: {loss:.4f}")

*# Predictions*

y\_pred = *model*.predict(*X\_test*)

y\_pred\_classes = np.argmax(y\_pred, *axis*=1)

y\_true = np.argmax(*y\_test*, *axis*=1)

*# Confusion Matrix*

cm = confusion\_matrix(y\_true, y\_pred\_classes)

plt.figure(*figsize*=(10,8))

sns.heatmap(cm, *annot*=True, *fmt*='d', *xticklabels*=*label\_map*.keys(), *yticklabels*=*label\_map*.keys(), *cmap*='Blues')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

*# Classification Report*

print("Classification Report:")

print(classification\_report(y\_true, y\_pred\_classes, *target\_names*=*label\_map*.keys()))

def **main**():

*# Load the trained model*

model = load\_model('best\_model\_sequences.keras')

*# Load the dataset*

X\_train, X\_test, y\_train, y\_test, label\_map = load\_dataset\_pickle('dataset\_sequences.pkl')

*# Evaluate the model*

evaluate\_model(model, X\_test, y\_test, label\_map)

*# Load and plot training history*

*try*:

*with* open('history\_sequences.pkl', 'rb') *as* f:

history = pickle.load(f)

plot\_history(history)

*except* FileNotFoundError:

print("Training history not found. Skipping plotting.")

*if* \_\_name\_\_ == "\_\_main\_\_":

main()

### 16.4. Prédiction et évaluation:

***# prediction\_sequences.py***

*import* tensorflow *as* tf

*import* cv2

*import* numpy *as* np

*import* dlib

*from* imutils *import* face\_utils

*import* os

*import* pickle

*from* collections *import* deque

*import* threading

*import* queue

*import* time

def **load\_model**(*model\_path*='fine\_tuned\_model\_sequences.keras'):

"""

Loads the trained model.

Args:

model\_path (str): Path to the saved model.

Returns:

tensorflow.keras.Model: Loaded model.

"""

model = tf.keras.models.load\_model(*model\_path*)

*return* model

def **get\_facial\_landmarks**(*detector*, *predictor*, *image*):

"""

Detects facial landmarks in an image.

Args:

detector: dlib face detector.

predictor: dlib shape predictor.

image (numpy.ndarray): Input image.

Returns:

dict: Coordinates of eyes and eyebrows.

"""

gray = cv2.cvtColor(*image*, cv2.COLOR\_BGR2GRAY)

rects = *detector*(gray, 1)

*if* len(rects) == 0:

*return* None *# No face detected*

*# Assuming the first detected face is the target*

rect = rects[0]

shape = *predictor*(gray, rect)

shape = face\_utils.shape\_to\_np(shape)

landmarks = {}

*# Define landmarks for left and right eyes and eyebrows*

landmarks['left\_eye'] = shape[36:42] *# Left eye landmarks*

landmarks['right\_eye'] = shape[42:48] *# Right eye landmarks*

landmarks['left\_eyebrow'] = shape[17:22] *# Left eyebrow landmarks*

landmarks['right\_eyebrow'] = shape[22:27] *# Right eyebrow landmarks*

*return* landmarks

def **extract\_roi**(*image*, *landmarks*, *region*='left\_eye', *padding*=5):

"""

Extracts a region of interest (ROI) from the image based on landmarks.

Args:

image (numpy.ndarray): Input image.

landmarks (dict): Facial landmarks.

region (str): Region to extract ('left\_eye', 'right\_eye', 'left\_eyebrow', 'right\_eyebrow').

padding (int): Padding around the ROI.

Returns:

numpy.ndarray: Extracted ROI.

"""

points = *landmarks*.get(*region*)

*if* points is None:

*return* None

*# Compute the bounding box*

x, y, w, h = cv2.boundingRect(points)

x = max(x - *padding*, 0)

y = max(y - *padding*, 0)

w = w + 2 \* *padding*

h = h + 2 \* *padding*

roi = *image*[y:y+h, x:x+w]

*return* roi

def **preprocess\_frame**(*image*, *detector*, *predictor*, *img\_size*=(64, 64)):

"""

Preprocesses a single frame: detects landmarks, extracts ROIs, and prepares the input.

Args:

image (numpy.ndarray): Input frame.

detector: dlib face detector.

predictor: dlib shape predictor.

img\_size (tuple): Desired image size for ROIs.

Returns:

numpy.ndarray: Preprocessed frame as a concatenated ROI image.

"""

landmarks = get\_facial\_landmarks(*detector*, *predictor*, *image*)

*if* landmarks is None:

*return* None *# No face detected*

*# Extract ROIs for eyes and eyebrows*

rois = {}

rois['left\_eye'] = extract\_roi(*image*, landmarks, 'left\_eye')

rois['right\_eye'] = extract\_roi(*image*, landmarks, 'right\_eye')

rois['left\_eyebrow'] = extract\_roi(*image*, landmarks, 'left\_eyebrow')

rois['right\_eyebrow'] = extract\_roi(*image*, landmarks, 'right\_eyebrow')

*# Process ROIs*

roi\_images = []

*for* region *in* ['left\_eye', 'right\_eye', 'left\_eyebrow', 'right\_eyebrow']:

roi = rois.get(region)

*if* roi is not None:

roi = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY) *# Convert to grayscale*

roi = cv2.resize(roi, *img\_size*)

roi = roi.astype('float32') / 255.0 *# Normalize to [0,1]*

roi = np.expand\_dims(roi, *axis*=-1) *# Add channel dimension*

roi\_images.append(roi)

*if* len(roi\_images) == 0:

*return* None *# No ROIs extracted*

*# Concatenate ROIs horizontally to form a single image*

combined\_roi = np.hstack(roi\_images)

*return* combined\_roi

def **movement\_to\_text**(*label\_map*):

"""

Creates a mapping from class indices to text.

Args:

label\_map (dict): Mapping from class names to indices.

Returns:

dict: Mapping from indices to text descriptions.

"""

movement\_to\_text\_map = {

'upward\_eyebrow': 'Eyebrow Raised',

'downward\_eyebrow': 'Eyebrow Lowered',

'left\_eye': 'Left Eye Movement',

'right\_eye': 'Right Eye Movement',

*# Add more mappings as needed*

}

*# Create index to text mapping*

index\_to\_text = {}

*for* cls, idx *in* *label\_map*.items():

text = movement\_to\_text\_map.get(cls, cls)

index\_to\_text[idx] = text

*return* index\_to\_text

def **prediction\_worker**(*model*, *input\_queue*, *output\_queue*, *max\_seq\_length*):

"""

Worker thread for handling model predictions.

Args:

model (tensorflow.keras.Model): Trained model.

input\_queue (queue.Queue): Queue to receive sequences for prediction.

output\_queue (queue.Queue): Queue to send prediction results.

max\_seq\_length (int): Fixed sequence length for the model.

"""

*while* True:

sequence = *input\_queue*.get()

*if* sequence is None:

*break* *# Sentinel to stop the thread*

*# Pad or truncate the sequence to match the model's expected input*

*if* sequence.shape[0] < *max\_seq\_length*:

pad\_width = *max\_seq\_length* - sequence.shape[0]

padding = np.zeros((pad\_width, \*sequence.shape[1:]), *dtype*=sequence.dtype)

sequence\_padded = np.concatenate((sequence, padding), *axis*=0)

*else*:

sequence\_padded = sequence[:*max\_seq\_length*]

*# Expand dimensions to match model input (1, frames, height, width, channels)*

sequence\_padded = np.expand\_dims(sequence\_padded, *axis*=0)

*# Perform prediction*

prediction = *model*.predict(sequence\_padded)

class\_idx = np.argmax(prediction)

confidence = np.max(prediction)

*# Put the result in the output queue*

*output\_queue*.put((class\_idx, confidence))

def **main**():

*# Load the trained model*

model = load\_model('final\_model\_sequences.keras')

*# Load label map*

*with* open('dataset\_sequences.pkl', 'rb') *as* f:

data = pickle.load(f)

label\_map = data['label\_map']

index\_to\_text = movement\_to\_text(label\_map)

*# Initialize dlib's face detector and landmark predictor*

detector = dlib.get\_frontal\_face\_detector()

predictor\_path = 'shape\_predictor\_68\_face\_landmarks.dat'

*if* not os.path.exists(predictor\_path):

print(f"Error: {predictor\_path} not found. Download it from http://dlib.net/files/shape\_predictor\_68\_face\_landmarks.dat.bz2")

*return*

predictor = dlib.shape\_predictor(predictor\_path)

*# Initialize queues for communication between threads*

input\_queue = queue.Queue()

output\_queue = queue.Queue()

*# Define sequence length (number of frames)*

max\_seq\_length = 20 *# Adjust based on your training data*

*# Start the prediction worker thread*

pred\_thread = threading.Thread(*target*=prediction\_worker, *args*=(model, input\_queue, output\_queue, max\_seq\_length))

pred\_thread.daemon = True

pred\_thread.start()

*# Start video capture*

cap = cv2.VideoCapture(0)

*if* not cap.isOpened():

print("Error: Could not open webcam.")

*return*

print("Starting real-time prediction. Press 'q' to quit.")

*# Initialize a deque to store the sequence of preprocessed frames*

frame\_buffer = deque(*maxlen*=max\_seq\_length)

*# Variable to store the latest prediction result*

latest\_prediction = "Initializing..."

*while* True:

ret, frame = cap.read()

*if* not ret:

print("Failed to grab frame.")

*break*

*# Preprocess the current frame*

preprocessed\_frame = preprocess\_frame(frame, detector, predictor, *img\_size*=(64, 64))

*if* preprocessed\_frame is not None:

frame\_buffer.append(preprocessed\_frame)

*else*:

*# If no face detected, append a zero array to maintain sequence length*

frame\_buffer.append(np.zeros((64, 256, 1), *dtype*='float32'))

*# If the buffer is full, send the sequence to the prediction thread*

*if* len(frame\_buffer) == max\_seq\_length:

*# Convert deque to numpy array*

sequence\_array = np.array(frame\_buffer)

input\_queue.put(sequence\_array)

*# Check if there's a new prediction result*

*try*:

*while* True:

class\_idx, confidence = output\_queue.get\_nowait()

movement = index\_to\_text.get(class\_idx, "Unknown")

latest\_prediction = f"{movement} ({confidence\*100:.2f}%)"

*except* queue.Empty:

*pass* *# No new prediction*

*# Display the prediction on the frame*

cv2.putText(frame, latest\_prediction, (30, 30), cv2.FONT\_HERSHEY\_SIMPLEX,

0.8, (0, 255, 0), 2, cv2.LINE\_AA)

*# Display the frame*

cv2.imshow('Real-time Movement Prediction', frame)

*# Exit condition*

*if* cv2.waitKey(1) & 0xFF == ord('q'):

*break*

*# Cleanup*

cap.release()

cv2.destroyAllWindows()

*# Stop the prediction thread*

input\_queue.put(None) *# Sentinel to stop the thread*

pred\_thread.join()

*if* \_\_name\_\_ == "\_\_main\_\_":

main()

### 16.5. Fine Tune model:

***# model\_finetune.py***

*import* tensorflow *as* tf

*import* numpy *as* np

*from* sklearn.model\_selection *import* train\_test\_split

*from* tensorflow.keras.utils *import* to\_categorical

*import* pickle

def **load\_saved\_model**(*model\_path*='final\_model\_sequences.keras'):

"""

Loads the saved Keras model.

Args:

model\_path (str): Path to the saved model file.

Returns:

tensorflow.keras.Model: Loaded Keras model.

"""

model = tf.keras.models.load\_model(*model\_path*)

*return* model

def **load\_dataset**(*pickle\_path*='dataset\_sequences.pkl'):

"""

Loads the dataset from a pickle file.

Args:

pickle\_path (str): Path to the pickle file.

Returns:

tuple: (X\_train, X\_test, y\_train, y\_test, label\_map)

"""

*with* open(*pickle\_path*, 'rb') *as* f:

data = pickle.load(f)

*return* data['X\_train'], data['X\_test'], data['y\_train'], data['y\_test'], data['label\_map']

def **fine\_tune\_model**(*model*, *X\_train*, *y\_train*, *X\_test*, *y\_test*, *epochs*=10, *batch\_size*=8):

"""

Fine-tunes the loaded model with new data.

Args:

model (tensorflow.keras.Model): Loaded Keras model.

X\_train (numpy.ndarray): Training sequences.

y\_train (numpy.ndarray): Training labels.

X\_test (numpy.ndarray): Testing sequences.

y\_test (numpy.ndarray): Testing labels.

epochs (int): Number of training epochs.

batch\_size (int): Batch size for training.

Returns:

tensorflow.keras.Model: Fine-tuned Keras model.

tensorflow.keras.callbacks.History: Training history.

"""

*# Optionally, unfreeze some layers for fine-tuning*

*# For example, unfreeze the last LSTM layer and dense layers*

*for* layer *in* *model*.layers:

*if* isinstance(layer, tf.keras.layers.LSTM) or isinstance(layer, tf.keras.layers.Dense):

layer.trainable = True

*else*:

layer.trainable = False *# Freeze other layers*

*# Compile the model with a lower learning rate*

*model*.compile(*optimizer*=tf.keras.optimizers.Adam(*learning\_rate*=1e-5),

*loss*='categorical\_crossentropy',

*metrics*=['accuracy'])

*# Define callbacks*

checkpoint = tf.keras.callbacks.ModelCheckpoint(

'fine\_tuned\_model\_sequences.keras',

*monitor*='val\_accuracy',

*save\_best\_only*=True,

*mode*='max'

)

early\_stop = tf.keras.callbacks.EarlyStopping(

*monitor*='val\_accuracy',

*patience*=5,

*restore\_best\_weights*=True

)

*# Fine-tune the model*

history = *model*.fit(

*X\_train*, *y\_train*,

*epochs*=*epochs*,

*batch\_size*=*batch\_size*,

*validation\_data*=(*X\_test*, *y\_test*),

*callbacks*=[checkpoint, early\_stop]

)

*return* *model*, history

def **main**():

*# Load the saved model*

model = load\_saved\_model('final\_model\_sequences.keras')

print("Model loaded successfully.")

*# Load the dataset*

X\_train, X\_test, y\_train, y\_test, label\_map = load\_dataset('dataset\_sequences.pkl')

print(f"Dataset loaded: {X\_train.shape[0]} training samples, {X\_test.shape[0]} testing samples.")

*# Fine-tune the model with new data*

model, history = fine\_tune\_model(model, X\_train, y\_train, X\_test, y\_test, *epochs*=10, *batch\_size*=8)

print("Model fine-tuned successfully.")

*# Save the fine-tuned model*

model.save('fine\_tuned\_model\_sequences.keras')

print("Fine-tuned model saved as 'fine\_tuned\_model\_sequences.keras'.")

*# Optionally, save the training history*

*with* open('fine\_tuned\_history.pkl', 'wb') *as* f:

pickle.dump(history.history, f)

print("Training history saved as 'fine\_tuned\_history.pkl'.")

*if* \_\_name\_\_ == "\_\_main\_\_":

main()

### 16.6. Lien du modèle et du projet:

Lien du modèle sur Hugging Face:  
<https://huggingface.co/shayan5422/Eye-Movement-Recognition>

Lien du projet sur GitHub:  
<https://github.com/Shayan5422/Eye-Movement>