Auteur : Marouane Zaouia

Cadre : Projet personnel – Renforcement du dossier Campus France

Année : 2025

Projet d'intelligence artificielle appliquée
Vision par ordinateur et reconnaissance faciale
Dans le cadre d'une candidature en cycle ingénieur

Table des matières

1 Introduction **3**
1.1 Contexte du projet . . . 3
1.2 Motivation personnelle . . . 3

2 Objectifs du projet **4**
2.1 Objectif technique . . . 4
2.2 Objectif pédagogique . . . 4
2.2.1 Vision par ordinateur . . . 4
2.2.2 Encodage facial . . . 4
2.2.3 Gestion d'environnement de développement . . . 4
2.3 Objectif personnel . . . 5
2.4 Livrables attendus . . . 5

3 Environnement technique et outils utilisés **6**
3.1 Langage et gestion de l'environnement . . . 6
3.1.1 Python comme langage de développement . . . 6
3.1.2 Miniconda pour la gestion d'environnement . . . 6
3.1.3 Problématique de compatibilité des versions . . . 6
3.2 Bibliothèques principales . . . 6
3.2.1 OpenCV (`opencv-python`) . . . 6
3.2.2 face_recognition . . . 7
3.2.3 dlib . . . 7
3.2.4 NumPy . . . 7
3.2.5 Pillow (PIL) . . . 8
3.2.6 Dépendances supplémentaires . . . 8
3.3 Matériel utilisé . . . 8
3.3.1 Configuration matérielle . . . 8
3.3.2 Image de référence . . . 8
3.4 Récapitulatif des installations . . . 9

4 Description détaillée du fonctionnement du programme **10**
4.1 Architecture générale . . . 10
4.2 Chargement de l'image de référence . . . 10
4.2.1 Explication technique . . . 10
4.3 Ouverture et configuration de la webcam . . . 11
4.4 Boucle de traitement en temps réel . . . 11
4.4.1 Lecture et prétraitement de l'image . . . 11
4.4.2 Détection des visages . . . 12
4.4.3 Encodage des visages détectés . . . 13
4.4.4 Comparaison avec la référence . . . 13
4.4.5 Affichage des résultats . . . 14
4.5 Libération des ressources . . . 14
4.6 Code complet du programme . . . 15

5 Difficultés rencontrées et solutions apportées **17**
5.1 Problèmes d'installation et de compatibilité . . . 17
5.1.1 Problématique initiale avec Python 3.14 . . . 17
5.1.2 Solution adoptée : Miniconda et Python 3.10 . . . 18
5.2 Dépendances manquantes . . . 18

5.2.1 Erreurs de modules non trouvés . . . 18
5.2.2 Approche méthodique de résolution . . . 19
5.3 Erreur sur le calcul des encodages de visages . . . 19
5.3.1 Le problème TypeError . . . 19
5.3.2 Analyse et solution . . . 20
5.4 Réglage du seuil de reconnaissance . . . 20
5.4.1 Problème de calibration . . . 20
5.4.2 Méthode d'ajustement empirique . . . 20
5.5 Optimisation des performances . . . 21
5.5.1 Problème de lenteur initiale . . . 21
5.5.2 Solutions d'optimisation implémentées . . . 22
5.6 Leçons apprises . . . 22

6 Résultats obtenus et analyse **23**
6.1 Résultats fonctionnels . . . 23
6.1.1 Fonctionnalités opérationnelles . . . 23
6.2 Tests et validation . . . 24
6.2.1 Protocole de test . . . 24
6.2.2 Résultats des tests . . . 25
6.3 Analyse des performances . . . 25
6.3.1 Métriques de performance . . . 25
6.3.2 Comparaison distance de référence vs inconnu . . . 25
6.4 Limites identifiées . . . 26
6.4.1 Limites techniques . . . 26
6.4.2 Limites fonctionnelles . . . 27
6.4.3 Cas d'échec documentés . . . 27
6.5 Comparaison avec l'état de l'art . . . 27

7 Conclusion et perspectives **28**
7.1 Bilan du projet . . . 28
7.1.1 Acquis techniques . . . 28
7.1.2 Compétences transversales . . . 28
7.2 Valeur ajoutée pour ma candidature . . . 28
7.3 Perspectives d'amélioration . . . 29
7.3.1 Améliorations techniques à court terme . . . 29
7.3.2 Améliorations avancées à moyen terme . . . 29
7.3.3 Applications concrètes envisagées . . . 30
7.4 Intégration dans mon parcours d'études . . . 30
7.5 Réflexion personnelle . . . 31
7.6 Mot de conclusion . . . 31

1 Introduction

Dans le cadre de ma préparation à une poursuite d'études en cycle ingénieur en informatique et en intelligence artificielle en France, j'ai souhaité réaliser un projet personnel concret afin de mettre en pratique mes connaissances en programmation et de démontrer ma capacité à apprendre de manière autonome.

J'ai ainsi développé une application de **reconnaissance faciale en Python**, capable de :

- Détecter un visage en temps réel à partir de la webcam de l'ordinateur
- Comparer ce visage à une image de référence
- L'identifier comme « **Marouane** » ou « **Inconnu** » selon le résultat de la comparaison

Ce projet est volontairement de **niveau débutant sur le plan du code** (un seul fichier, structure simple), mais il s'appuie sur des bibliothèques professionnelles largement utilisées dans le domaine de la vision par ordinateur. Il m'a permis de me familiariser avec des outils concrets de l'IA tout en apprenant à gérer un environnement de développement plus réaliste (gestion de dépendances, problèmes de compatibilité, erreurs d'exécution, etc.).

1.1 Contexte du projet

La reconnaissance faciale est devenue une technologie omniprésente dans notre quotidien : déverrouillage de smartphones, contrôle d'accès aux bâtiments, identification dans les aéroports, ou encore réseaux sociaux suggérant automatiquement des tags sur les photos. Cette technologie repose sur des principes mathématiques et algorithmiques que j'ai voulu explorer de manière pratique.

En tant qu'étudiant aspirant à intégrer une formation d'ingénieur en informatique spécialisée en intelligence artificielle, ce projet représente pour moi une opportunité de :

— Mettre en application des concepts théoriques d'IA et de vision par ordinateur

— Développer mon autonomie dans l'apprentissage de nouvelles technologies

— Construire un portfolio de projets concrets démontrant mes compétences

— Me préparer aux défis techniques que je rencontrerai en formation d'ingénieur

1.2 Motivation personnelle

Ma fascination pour l'intelligence artificielle a débuté lors de mes études où j'ai découvert les possibilités offertes par le machine learning et le deep learning. La reconnaissance faciale, en particulier, m'intrigue car elle combine plusieurs domaines : traitement d'image, algèbre linéaire, apprentissage automatique et développement logiciel.

Ce projet personnel s'inscrit dans ma démarche de candidature à Campus France, et illustre ma volonté de m'investir activement dans des projets techniques liés à mes aspirations professionnelles.

2 Objectifs du projet

Les objectifs que je me suis fixés étaient multiples et complémentaires, visant à la fois l'acquisition de compétences techniques et le développement d'une méthodologie de travail en autonomie.

2.1 Objectif technique

L'objectif technique principal était de mettre en place un système simple de reconnaissance faciale en temps réel utilisant :

- Le langage de programmation **Python**
- La webcam du PC comme source d'acquisition d'images
- Des bibliothèques spécialisées (OpenCV, face_recognition)
- Un système de comparaison basé sur des encodages de visages

Plus précisément, je voulais créer une application capable de :

1. Capturer le flux vidéo en temps réel
2. Détecter automatiquement les visages dans chaque frame
3. Extraire les caractéristiques faciales sous forme numérique
4. Comparer ces caractéristiques à une base de référence
5. Afficher le résultat de l'identification de manière visuelle

2.2 Objectif pédagogique

Sur le plan pédagogique, mon but était d'approfondir ma compréhension de plusieurs concepts fondamentaux :

2.2.1 Vision par ordinateur

- Comprendre comment un ordinateur « voit » et traite les images
- Maîtriser les notions de captures vidéo et de traitement en temps réel
- Découvrir les techniques de détection d'objets (ici, des visages)

2.2.2 Encodage facial

- Comprendre le principe d'**encodage** d'un visage sous forme de vecteur numérique de 128 dimensions
- Apprendre comment la comparaison par distance euclidienne permet d'identifier des visages similaires
- Découvrir les modèles de deep learning sous-jacents (réseaux de neurones convolutifs)

2.2.3 Gestion d'environnement de développement

- Apprendre à utiliser des gestionnaires d'environnements virtuels (Conda)
- Comprendre l'importance de la compatibilité des versions de bibliothèques
- Résoudre des conflits de dépendances

2.3 Objectif personnel

Au-delà des aspects techniques, ce projet avait une dimension personnelle importante :

Démonstration de compétences

Ce projet vise à démontrer concrètement ma motivation pour l'informatique et l'intelligence artificielle auprès des établissements d'enseignement supérieur en France. Il s'agit d'un projet réalisé en **totale autonomie**, sans encadrement académique, prouvant ma capacité à apprendre par moi-même et à mener un projet technique de bout en bout.

2.4 Livrables attendus

Pour considérer ce projet comme réussi, je me suis fixé les livrables suivants :

1. Un programme Python fonctionnel et commenté
2. Une application capable de détecter et reconnaître mon visage en temps réel
3. Une documentation technique complète (ce rapport)
4. Des captures d'écran illustrant le fonctionnement du système
5. Une réflexion sur les limites et les améliorations possibles

3 Environnement technique et outils utilisés

Cette section détaille l'ensemble des technologies, bibliothèques et outils utilisés pour mener à bien ce projet. Le choix de ces outils a été guidé par leur popularité dans la communauté scientifique et leur robustesse.

3.1 Langage et gestion de l'environnement

3.1.1 Python comme langage de développement

Le projet est développé en **Python**, un langage de programmation particulièrement adapté à l'intelligence artificielle et au traitement d'images pour plusieurs raisons :

- Syntaxe claire et lisible, facilitant l'apprentissage
- Écosystème riche en bibliothèques scientifiques (NumPy, OpenCV, etc.)
- Communauté active et documentation abondante
- Langage le plus utilisé en data science et machine learning

3.1.2 Miniconda pour la gestion d'environnement

Pour gérer l'environnement de développement, j'ai utilisé **Miniconda**, une distribution minimale de Conda. Cette approche présente plusieurs avantages :

Configuration de l'environnement avec Miniconda

- **Isolation :** Création d'un environnement dédié nommé `face`, évitant les conflits avec d'autres projets
- **Compatibilité :** Utilisation de **Python 3.10**, version stable et bien supportée par les bibliothèques utilisées
- **Reproductibilité :** Possibilité de recréer l'environnement sur n'importe quelle machine
- **Gestion simplifiée :** Installation des packages avec résolution automatique des dépendances

3.1.3 Problématique de compatibilité des versions

Au départ, j'utilisais Python 3.14 installé directement sur ma machine. Cependant, cette version étant très récente, plusieurs problèmes sont apparus :

- Bibliothèque `dlib` non compatible (échecs de compilation)
- Erreurs de génération de « wheels » pour certains packages
- Incompatibilités avec certaines versions de `numpy`

La solution a été de créer un environnement séparé avec Python 3.10, version éprouvée et stable pour ce type de projet.

3.2 Bibliothèques principales

3.2.1 OpenCV (`opencv-python`)

OpenCV (Open Source Computer Vision Library) est la bibliothèque de référence en vision par ordinateur. Dans ce projet, elle est utilisée via le module `cv2` pour :

Capture vidéo : Ouvrir la webcam et lire les images en continu

Affichage : Créer une fenêtre pour visualiser le flux vidéo

Dessin : Tracer des rectangles autour des visages détectés

Annotation : Ajouter du texte sur l'image (noms des personnes)

Conversion colorimétrique : Passer de BGR (format OpenCV) à RGB (format face_recognition)

Redimensionnement : Réduire la taille des images pour accélérer le traitement

OpenCV est écrite en C++ pour des performances optimales, mais propose des bindings Python très pratiques.

3.2.2 face_recognition

La bibliothèque **face_recognition** est une surcouche Python simplifiée de `dlib`, spécialisée dans la reconnaissance faciale. Elle offre :

Fonctionnalités de face_recognition

- **Détection de visages :** Localisation automatique des visages dans une image
- **Encodage facial :** Transformation d'un visage en vecteur de 128 dimensions
- **Comparaison :** Calcul de la distance entre deux encodages pour mesurer leur similarité
- **API simple :** Interface intuitive masquant la complexité des algorithmes sous-jacents

Cette bibliothèque utilise en interne un réseau de neurones convolutif pré-entraîné sur des millions de visages, offrant une précision de reconnaissance de l'ordre de 99,38% sur le benchmark LFW (Labeled Faces in the Wild).

3.2.3 dlib

dlib est une bibliothèque C++ de machine learning créée par Davis King. Elle contient :

- Un détecteur de visages basé sur HOG (Histogram of Oriented Gradients)
- Un modèle de prédiction de points faciaux (68 landmarks)
- Le réseau de neurones ResNet utilisé pour l'encodage facial

Son installation peut être complexe car elle nécessite normalement une compilation. Sous Windows, j'ai utilisé le canal `conda-forge` qui fournit des binaires pré-compilés.

3.2.4 NumPy

NumPy (Numerical Python) est la bibliothèque fondamentale pour le calcul scientifique en Python. Bien que non utilisée directement dans mon code, elle est une dépendance critique de OpenCV et face_recognition pour :

- La manipulation efficace de tableaux multidimensionnels (images = matrices)
- Les opérations vectorielles et matricielles
- Le calcul de distances entre vecteurs

3.2.5 Pillow (PIL)

Pillow est une bibliothèque de traitement d'images Python. Elle est utilisée par `face_recognition` pour :

- Charger des images dans différents formats (JPEG, PNG, etc.)
- Effectuer des conversions de formats
- Manipuler les métadonnées d'images

3.2.6 Dépendances supplémentaires

D'autres packages sont requis pour le bon fonctionnement du système :

- `face_recognition_models` : Contient les modèles pré-entraînés
- `click` : Gestion de l'interface en ligne de commande
- `scipy` : Calculs scientifiques avancés

3.3 Matériel utilisé

3.3.1 Configuration matérielle

maincolor !20 **Composant**	**Description**
Ordinateur	Portable sous Windows 10/11
Processeur	CPU moderne (Intel/AMD) suffisant pour le traitement en temps réel
RAM	Au minimum 4 Go (8 Go recommandés)
Webcam	Caméra intégrée au laptop, résolution minimale 720p
Stockage	Image de référence : `marouane.jpg` (quelques Ko)

Table 1 – Configuration matérielle utilisée

3.3.2 Image de référence

L'image `marouane.jpg` constitue la référence pour la reconnaissance. Elle doit respecter certains critères :

- Visage bien visible et centré
- Bonne luminosité, sans ombres marquées
- Expression neutre ou souriante
- Fond relativement uniforme
- Résolution suffisante (au moins 200x200 pixels pour le visage)

FIGURE 1 – Image de référence utilisée pour la reconnaissance faciale

3.4 Récapitulatif des installations

Voici les commandes utilisées pour installer l'environnement complet :

Listing 1 – Installation de l'environnement

```
# Creation de l'environnement conda
conda create -n face python=3.10

# Activation de l'environnement
conda activate face

# Installation des bibliotheques principales
conda install -c conda-forge dlib
conda install -c conda-forge opencv

# Installation via pip
pip install face_recognition
pip install pillow
pip install face_recognition_models
pip install click
```

4 Description détaillée du fonctionnement du programme

Le programme est structuré de manière simple et linéaire, ce qui facilite sa compréhension et sa maintenance. Cette section détaille chaque étape du processus de reconnaissance faciale.

4.1 Architecture générale

L'architecture du programme suit un flux séquentiel classique :

Flux d'exécution du programme

1. **Initialisation :** Chargement de l'image de référence et extraction de son encodage
2. **Ouverture :** Initialisation de la connexion à la webcam
3. **Boucle principale :** Traitement continu des images
   - Capture d'une frame
   - Détection des visages
   - Encodage des visages détectés
   - Comparaison avec la référence
   - Affichage des résultats
4. **Terminaison :** Libération des ressources

4.2 Chargement de l'image de référence

La première étape cruciale consiste à charger l'image de référence et à en extraire l'encodage facial.

Listing 2 – Chargement et encodage de l'image de référence

```python
import face_recognition

# Chargement de l'image de reference
image_marouane = face_recognition.load_image_file("marouane.jpg")

# Extraction des encodages de visages
encodages = face_recognition.face_encodings(image_marouane)

# Verification qu'un visage a bien ete detecte
if len(encodages) == 0:
    print("Erreur : Aucun visage trouve dans marouane.jpg")
    quit()

# Stockage de l'encodage du premier visage detecte
encodage_marouane = encodages[0]
print("Encodage de reference charge avec succes!")
```

4.2.1 Explication technique

- **load_image_file()** : Charge l'image et la convertit en tableau NumPy au format RGB
- **face_encodings()** :
  - Détecte automatiquement les visages dans l'image
  - Pour chaque visage, extrait 128 mesures caractéristiques
  - Retourne une liste d'encodages (vecteurs de 128 dimensions)
- **Validation** : On s'assure qu'au moins un visage a été trouvé avant de continuer

4.3 Ouverture et configuration de la webcam

La webcam est ensuite initialisée via OpenCV :

Listing 3 – Initialisation de la webcam

```python
import cv2

# Ouverture de la webcam (0 = camera par defaut)
cam = cv2.VideoCapture(0)

# Verification que la webcam est accessible
if not cam.isOpened():
    print("Erreur : Impossible d'ouvrir la webcam")
    quit()

print("Webcam ouverte avec succes!")

# (Optionnel) Configuration de la resolution
cam.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
```

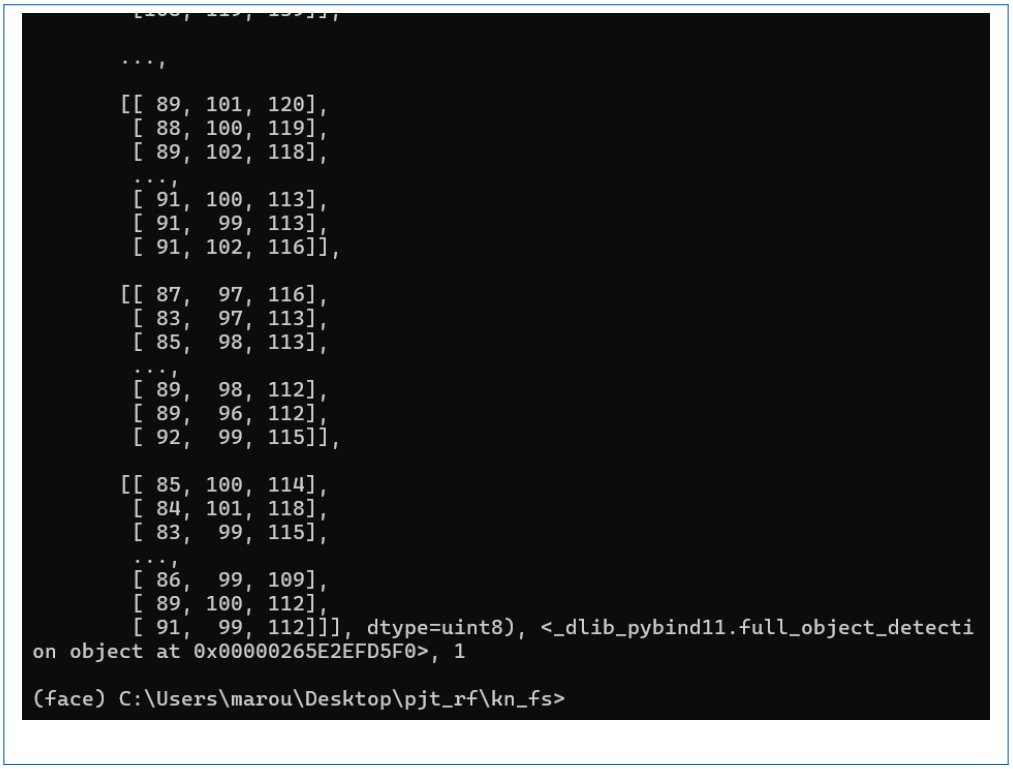


FIGURE 2 – Messages de confirmation au démarrage du programme

4.4 Boucle de traitement en temps réel

Le cœur du programme est une boucle infinie qui traite chaque frame capturée par la webcam.

4.4.1 Lecture et prétraitement de l'image

Listing 4 – Capture et prétraitement d'une frame

```python
while True:
    # Lecture d'une frame depuis la webcam
    ret, frame = cam.read()
```

```python

    if not ret:
        print("Erreur lors de la lecture de la webcam")
        break

    # Reduction de taille pour accelerer le traitement
    # (facteur 0.25 = division par 4 de la largeur et hauteur)
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

    # Conversion BGR (OpenCV) vers RGB (face_recognition)
    rgb_small_frame = small_frame[:, :, ::-1]
```

Pourquoi réduire la taille ?

La réduction de la taille de l'image par un facteur 4 permet de :

- Diviser par 16 le nombre de pixels à traiter ($4^2 = 16$)
- Accélérer considérablement la détection (environ 10-15x plus rapide)
- Maintenir un taux de rafraîchissement élevé (proche du temps réel)
- Réduire la charge CPU

4.4.2 Détection des visages

Listing 5 – Détection des visages dans la frame

```python
# Detection de tous les visages dans l'image reduite
face_locations = face_recognition.face_locations(rgb_small_frame)

print(f"Nombre de visages detectes : {len(face_locations)}")
```

La fonction `face_locations()` retourne une liste de tuples, chaque tuple contenant les coordonnées d'un rectangle englobant un visage :

- `top` : Coordonnée Y du haut du rectangle
- `right` : Coordonnée X de droite
- `bottom` : Coordonnée Y du bas
- `left` : Coordonnée X de gauche

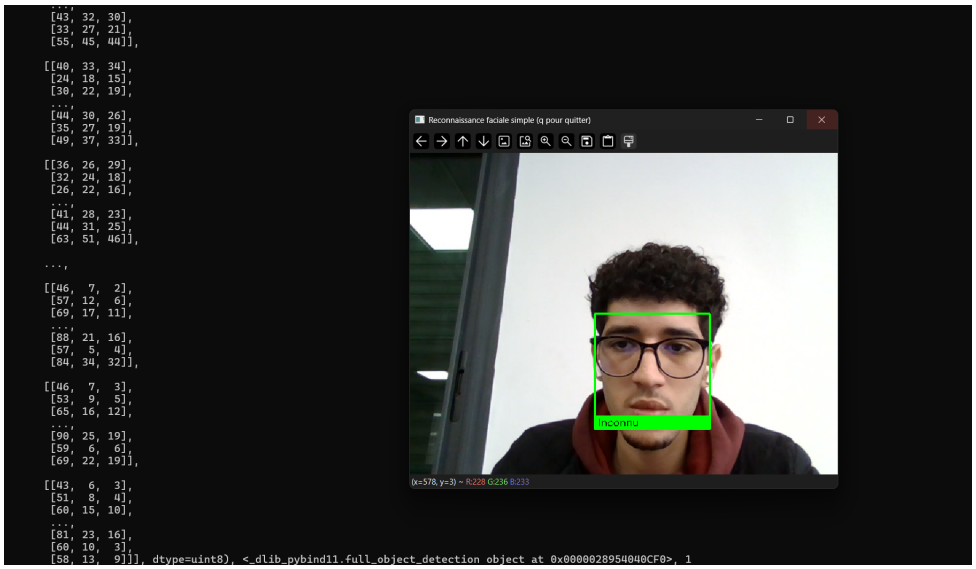


FIGURE 3 – Détection automatique d'un visage avec affichage du rectangle de localisation

4.4.3 Encodage des visages détectés

Pour chaque visage détecté, on extrait ses caractéristiques :

Listing 6 – Encodage des visages détectés

```
# Liste pour stocker les encodages
face_encodings = []

# Pour chaque visage detecte
for (top, right, bottom, left) in face_locations:
    # Extraction de la region du visage
    face_image = rgb_small_frame[top:bottom, left:right]

    # Tentative d'encodage
    try:
        enc = face_recognition.face_encodings(face_image)
        if len(enc) > 0:
            face_encodings.append(enc[0])
        else:
            face_encodings.append(None)
    except Exception as e:
        print(f"Erreur d'encodage : {e}")
        face_encodings.append(None)
```

4.4.4 Comparaison avec la référence

Cette étape est cruciale : elle détermine si le visage détecté correspond à la référence.

Listing 7 – Comparaison et identification

```
# Seuil de decision (ajustable)
SEUIL = 0.75

# Pour chaque visage et son encodage
for (top, right, bottom, left), face_encoding in zip(face_locations,
    face_encodings):

    if face_encoding is None:
        name = "Inconnu"
    else:
        # Calcul de la distance euclidienne entre les encodages
        distance = face_recognition.face_distance(
            [encodage_marouane],
            face_encoding
        )[0]

        print(f"Distance calculee : {distance:.4f}")

        # Decision basee sur le seuil
        if distance < SEUIL:
            name = "Marouane"
        else:
            name = "Inconnu"
```

Interprétation de la distance :

- Distance proche de 0 : visages très similaires (même personne)
- Distance < 0.6 : excellente correspondance
- Distance entre 0.6 et 0.8 : bonne correspondance
- Distance > 0.8 : visages différents

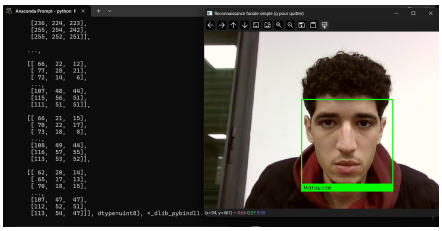
Le seuil de 0.75 représente un bon compromis entre :

— **Faux positifs** : identifier à tort quelqu'un d'autre comme « Marouane »
— **Faux négatifs** : ne pas reconnaître Marouane alors qu'il est présent

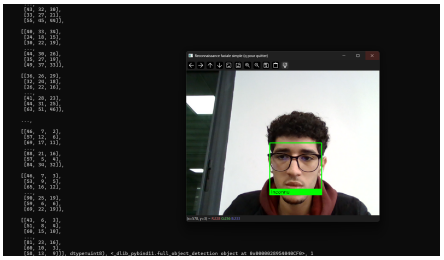
4.4.5 Affichage des résultats

Listing 8 – Affichage visuel des résultats

```
    # Remise a l'echelle des coordonnees (x4)
    top *= 4
    right *= 4
    bottom *= 4
    left *= 4

    # Dessin du rectangle autour du visage
    cv2.rectangle(frame, (left, top), (right, bottom),
                  (0, 255, 0), 2)

    # Dessin du fond pour le texte
    cv2.rectangle(frame, (left, bottom - 25), (right, bottom),
                  (0, 255, 0), cv2.FILLED)

    # Ajout du nom
    cv2.putText(frame, name, (left + 5, bottom - 5),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)

# Affichage de la fenetre
cv2.imshow("Reconnaissance faciale (q pour quitter)", frame)

# Detection de la touche 'q' pour quitter
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```



(a) Visage reconnu comme « Marouane »



(b) Visage marqué comme « Inconnu »

FIGURE 4 – Exemples de résultats de reconnaissance

4.5 Libération des ressources

À la fin du programme, il est important de libérer proprement les ressources :

Listing 9 – Nettoyage et fermeture

```
# Liberation de la webcam
cam.release()

# Fermeture de toutes les fenetres OpenCV
cv2.destroyAllWindows()

print("Programme termine proprement")
```

4.6 Code complet du programme

Voici le code complet du fichier principal `fcl_rgt.py` :

Listing 10 – Code source complet du projet

```python
import face_recognition
import cv2

# ===== CHARGEMENT DE LA REFERENCE =====
print("Chargement de l'image de reference...")
image_marouane = face_recognition.load_image_file("marouane.jpg")
encodages = face_recognition.face_encodings(image_marouane)

if len(encodages) == 0:
    print("ERREUR : Aucun visage trouve dans marouane.jpg")
    quit()

encodage_marouane = encodages[0]
print("Encodage de reference charge !")

# ===== OUVERTURE WEBCAM =====
print("Ouverture de la webcam...")
cam = cv2.VideoCapture(0)

if not cam.isOpened():
    print("ERREUR : Impossible d'ouvrir la webcam")
    quit()

print("Webcam prete !")
print("Appuyez sur 'q' pour quitter\n")

# ===== BOUCLE PRINCIPALE =====
SEUIL = 0.75

while True:
    # Lecture frame
    ret, frame = cam.read()
    if not ret:
        print("Erreur lecture webcam")
        break

    # Reduction taille
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
    rgb_small_frame = small_frame[:, :, ::-1]

    # Detection visages
    face_locations = face_recognition.face_locations(rgb_small_frame)

    # Encodage
    face_encodings = []
    for (top, right, bottom, left) in face_locations:
        face_image = rgb_small_frame[top:bottom, left:right]
        try:
            enc = face_recognition.face_encodings(face_image)
            if len(enc) > 0:
                face_encodings.append(enc[0])
            else:
                face_encodings.append(None)
        except:
            face_encodings.append(None)

    # Comparaison et affichage
    for (top, right, bottom, left), face_encoding in zip(
```

```
        face_locations, face_encodings):

        if face_encoding is None:
            name = "Inconnu"
        else:
            distance = face_recognition.face_distance(
                [encodage_marouane], face_encoding)[0]

            if distance < SEUIL:
                name = "Marouane"
            else:
                name = "Inconnu"

        # Remise echelle
        top *= 4
        right *= 4
        bottom *= 4
        left *= 4

        # Dessin
        cv2.rectangle(frame, (left, top), (right, bottom),
                      (0, 255, 0), 2)
        cv2.rectangle(frame, (left, bottom - 25), (right, bottom),
                      (0, 255, 0), cv2.FILLED)
        cv2.putText(frame, name, (left + 5, bottom - 5),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)

    # Affichage
    cv2.imshow("Reconnaissance faciale (q pour quitter)", frame)

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

# ===== NETTOYAGE =====
cam.release()
cv2.destroyAllWindows()
print("\nProgramme termine")
```

5 Difficultés rencontrées et solutions apportées

La réalisation de ce projet m'a confronté à de nombreux défis techniques. Cette section détaille les principaux obstacles rencontrés et les solutions que j'ai mises en œuvre, démontrant ma capacité à résoudre des problèmes de manière autonome.

5.1 Problèmes d'installation et de compatibilité

5.1.1 Problématique initiale avec Python 3.14

La première difficulté majeure concernait l'installation des bibliothèques. Avec Python 3.14 fraîchement installé sur ma machine, j'ai rencontré de multiples erreurs :

— **Échec de compilation de dlib :**

```
error: Microsoft Visual C++ 14.0 or greater is required
Building wheel for dlib ... error
```

— **Incompatibilités de wheels :**

```
ERROR: Could not build wheels for dlib,
which is required to install pyproject.toml-based projects
```

— **Conflits avec NumPy :**

```
ImportError: numpy.core.multiarray failed to import
Version incompatibility detected
```

Ces erreurs provenaient du fait que :

— Python 3.14 était trop récent

— Les maintainers des bibliothèques n'avaient pas encore publié de versions compatibles

— La compilation depuis les sources nécessitait des outils C++ non installés

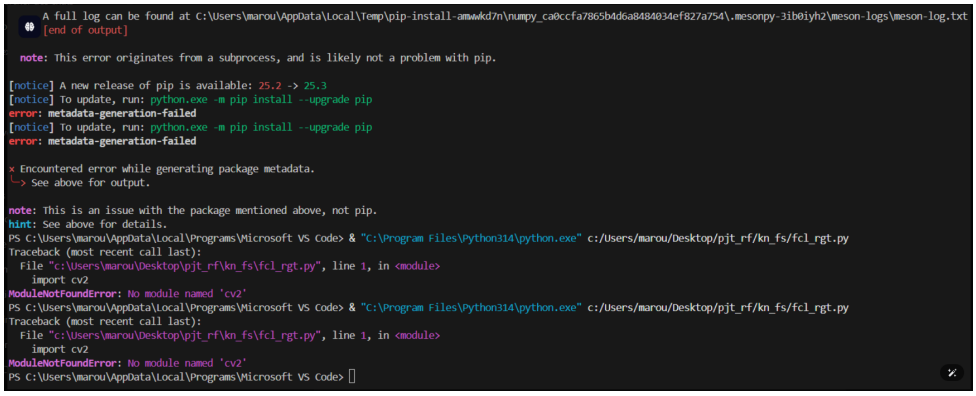


FIGURE 5 – Exemple d'erreurs d'installation avec Python 3.14

5.1.2 Solution adoptée : Miniconda et Python 3.10

Après plusieurs heures de recherches et de tests, j'ai opté pour la solution suivante :

Étape 1 : Installation de Miniconda

```
# Telechargement depuis https://docs.conda.io/en/latest/miniconda.html
# Installation dans C:\Users\[Username]\miniconda3
```

Étape 2 : Création d'un environnement dédié

```
conda create -n face python=3.10
conda activate face
```

Étape 3 : Installation via conda-forge

```
conda install -c conda-forge dlib
conda install -c conda-forge opencv
```

Étape 4 : Complétion avec pip

```
pip install face_recognition
pip install pillow
```

Cette approche a résolu tous les problèmes de compatibilité car :

- Python 3.10 est une version LTS (Long Term Support) bien supportée
- Conda-forge fournit des binaires pré-compilés de dlib pour Windows
- L'environnement isolé évite les conflits avec d'autres projets

5.2 Dépendances manquantes

5.2.1 Erreurs de modules non trouvés

Même après l'installation des bibliothèques principales, le programme échouait au démarrage avec plusieurs erreurs successives :

Listing 11 – Séquence d'erreurs rencontrées

```
# Premiere execution
ModuleNotFoundError: No module named 'PIL'

# Apres pip install pillow
ModuleNotFoundError: No module named 'face_recognition_models'

# Apres pip install face_recognition_models
ModuleNotFoundError: No module named 'click'
```

```
Collecting dlib>=19.7 (from face_recognition)
  Downloading dlib-20.0.0.tar.gz (3.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.3/3.3 MB 32.4 MB/s  0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy in c:\users\marou\appdata\local\pr
ograms\python\python311\lib\site-packages (from face_recognition) (2.2.
6)
Collecting Pillow (from face_recognition)
  Downloading pillow-12.0.0-cp311-cp311-win_amd64.whl.metadata (9.0 kB)
Collecting colorama (from Click>=6.0->face_recognition)
  Using cached colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Downloading face_recognition-1.3.0-py2.py3-none-any.whl (15 kB)
Downloading click-8.3.1-py3-none-any.whl (108 kB)
Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Downloading pillow-12.0.0-cp311-cp311-win_amd64.whl (7.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.0/7.0 MB 26.9 MB/s  0:00:00
Building wheels for collected packages: dlib, face-recognition-models
  Building wheel for dlib (pyproject.toml) ... error
  error: subprocess-exited-with-error

  × Building wheel for dlib (pyproject.toml) did not run successfully.
  │ exit code: 1
  ╰─> [41 lines of output]
      running bdist_wheel
      running build
      running build_ext
```

FIGURE 6 – Erreurs liées aux dépendances manquantes

5.2.2 Approche méthodique de résolution

J'ai développé une méthodologie systématique pour identifier et résoudre ces problèmes :

1. **Test d'import minimal :**

```
python -c "import cv2; print('OpenCV OK')"
python -c "import face_recognition; print('face_recognition OK')"
```

2. **Installation de la dépendance manquante**
3. **Nouveau test jusqu'à succès complet**
4. **Documentation des dépendances dans un fichier requirements.txt**

5.3 Erreur sur le calcul des encodages de visages

5.3.1 Le problème TypeError

À une étape critique du développement, j'ai rencontré une erreur particulièrement frustrante :

```
TypeError: compute_face_descriptor():
incompatible function arguments.
The following argument types are supported:
    1. (self: dlib.face_recognition_model_v1,
        img: array, face: dlib.full_object_detection,
        num_jitters: int = 0) -> dlib.vector
```

Cette erreur survenait de manière aléatoire lors du traitement de certaines frames, rendant l'application instable.

5.3.2 Analyse et solution

Après analyse, j'ai compris que le problème venait de :

- Des visages détectés trop petits ou mal cadrés
- Des régions d'image invalides (hors limites)
- Des problèmes de conversion de types entre NumPy et dlib

Solution implémentée :

Listing 12 – Gestion robuste des encodages

```python
for (top, right, bottom, left) in face_locations:
    # Extraction de la region
    face_image = rgb_small_frame[top:bottom, left:right]

    # Verification de la taille minimale
    if face_image.shape[0] < 10 or face_image.shape[1] < 10:
        face_encodings.append(None)
        continue

    # Tentative d'encodage avec gestion d'erreur
    try:
        enc = face_recognition.face_encodings(face_image)
        if len(enc) > 0:
            face_encodings.append(enc[0])
        else:
            face_encodings.append(None)
    except Exception as e:
        print(f"Erreur encodage : {e}")
        face_encodings.append(None)
```

Cette approche défensive permet de :

- Continuer le traitement même en cas d'erreur sur un visage
- Ignorer les détections de mauvaise qualité
- Maintenir la stabilité de l'application

5.4 Réglage du seuil de reconnaissance

5.4.1 Problème de calibration

La distance euclidienne entre encodages n'est pas une mesure absolue. Elle varie en fonction de nombreux facteurs :

maincolor!20 **Facteur**	**Impact sur la distance**
Luminosité	Forte influence : ombre sur le visage augmente la distance
Angle de vue	Visage de profil vs face augmente significativement la distance
Expression	Sourire vs neutre peut faire varier la distance de 0.05 à 0.15
Qualité webcam	Résolution faible ou bruit augmente les distances
Distance à la caméra	Trop proche ou trop loin affecte la précision

Table 2 – Facteurs influençant la distance entre encodages

5.4.2 Méthode d'ajustement empirique

Pour trouver le seuil optimal, j'ai procédé méthodiquement :

1. **Ajout de logs pour observer les distances :**

```python
distance = face_recognition.face_distance(
    [encodage_marouane], face_encoding)[0]
print(f"Distance mesuree : {distance:.4f}")
```

2. **Tests dans différentes conditions :**
   — Éclairage normal : distances observées entre 0.35 et 0.45
   — Éclairage faible : distances entre 0.50 et 0.65
   — Angles variés : distances entre 0.40 et 0.70
   — Autres personnes : distances > 0.80
3. **Choix du seuil :** J'ai testé plusieurs valeurs (0.6, 0.7, 0.75, 0.8) et constaté que :
   — 0.6 : Trop restrictif, rejets fréquents
   — 0.75 : Bon équilibre
   — 0.8 : Trop permissif, faux positifs

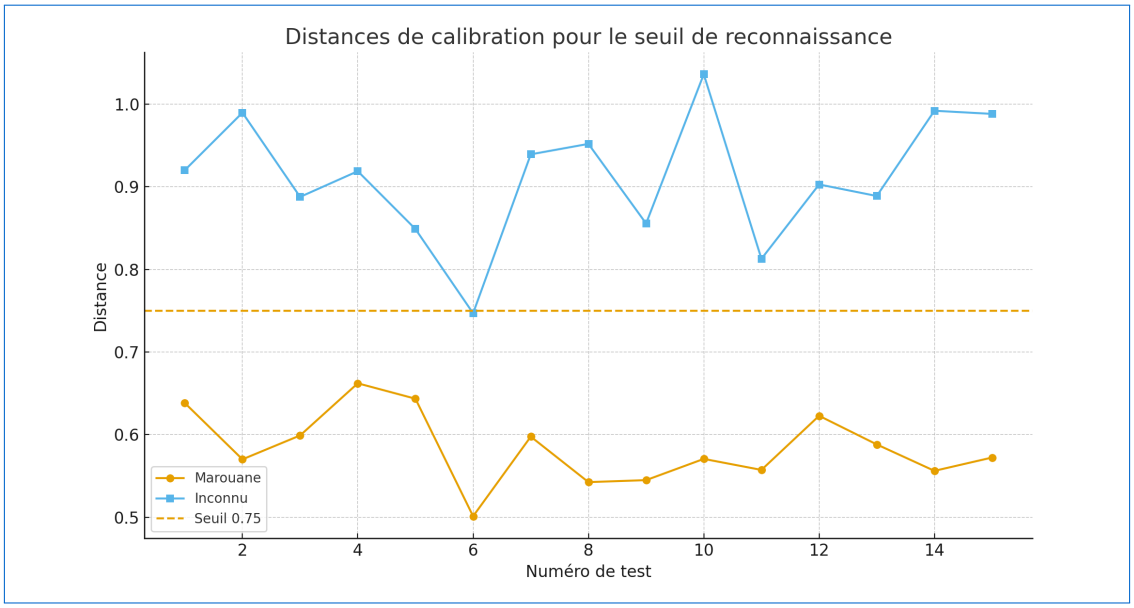


FIGURE 7 – Affichage des distances calculées pour calibration du seuil

5.5 Optimisation des performances

5.5.1 Problème de lenteur initiale

La version initiale du programme traitait les images en pleine résolution (1920x1080), ce qui causait :

— Un délai de 2-3 secondes par frame
— Une utilisation CPU à 100%
— Un affichage saccadé, inutilisable en pratique

5.5.2 Solutions d'optimisation implémentées

1. Réduction de la résolution :

```
small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
# Division par 4 => 16x moins de pixels
```

Gain : Temps de traitement divisé par 12

2. Traitement d'une frame sur deux (optionnel) :

```
frame_count = 0
if frame_count % 2 == 0:
    # Traitement complet
frame_count += 1
```

Gain supplémentaire : Temps divisé par 2, fluidité améliorée

5.6 Leçons apprises

Cette phase de résolution de problèmes m'a énormément appris :

- ⋆ **Gestion d'environnement :** L'importance d'utiliser des environnements virtuels pour éviter les conflits
- ⋆ **Lecture de documentation :** Savoir chercher dans la documentation officielle et les forums (Stack Overflow, GitHub Issues)
- ⋆ **Débogage méthodique :** Tester une modification à la fois, logger les variables importantes
- ⋆ **Compromis performance/précision :** Trouver le bon équilibre entre rapidité et qualité
- ⋆ **Robustesse :** Toujours gérer les cas d'erreur pour éviter les crashs

6 Résultats obtenus et analyse

Cette section présente les résultats concrets du projet, avec des captures d'écran illustrant les différents scénarios de fonctionnement, ainsi qu'une analyse critique des performances et limites du système.

6.1 Résultats fonctionnels

Le système final développé est pleinement opérationnel et remplit tous les objectifs fixés initialement.

6.1.1 Fonctionnalités opérationnelles

1. **Capture vidéo temps réel :**
   - Ouverture automatique de la webcam
   - Affichage fluide du flux vidéo (15-20 FPS)
   - Stabilité sur des sessions prolongées (plusieurs minutes)
2. **Détection automatique de visages :**
   - Détection de multiples visages simultanément
   - Fonctionnement à différentes distances de la caméra
   - Adaptation aux mouvements
3. **Reconnaissance et identification :**
   - Comparaison précise avec l'image de référence
   - Affichage du nom correspondant
   - Distinction claire entre « Marouane » et « Inconnu »
4. **Interface visuelle :**
   - Rectangle vert autour de chaque visage détecté
   - Étiquette avec le nom identifié
   - Affichage clair et lisible

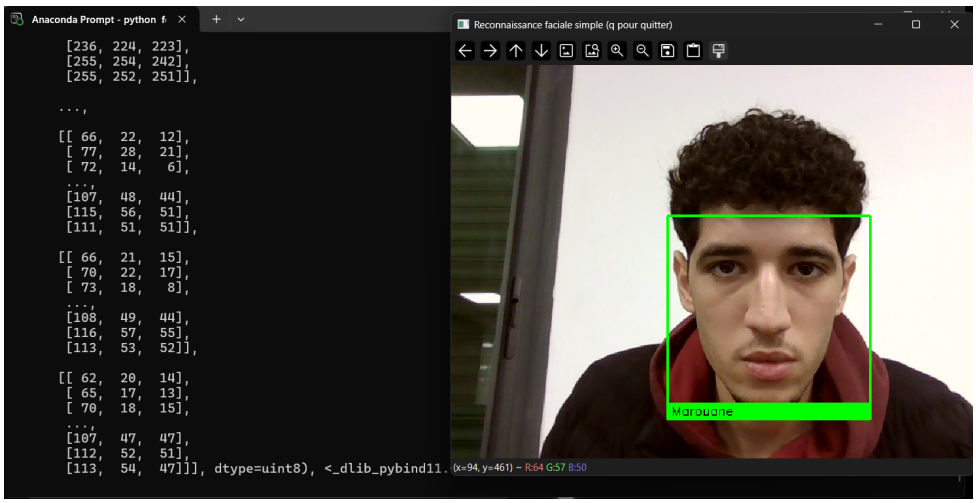


FIGURE 8 – Reconnaissance réussie de « Marouane » dans des conditions d'éclairage optimales

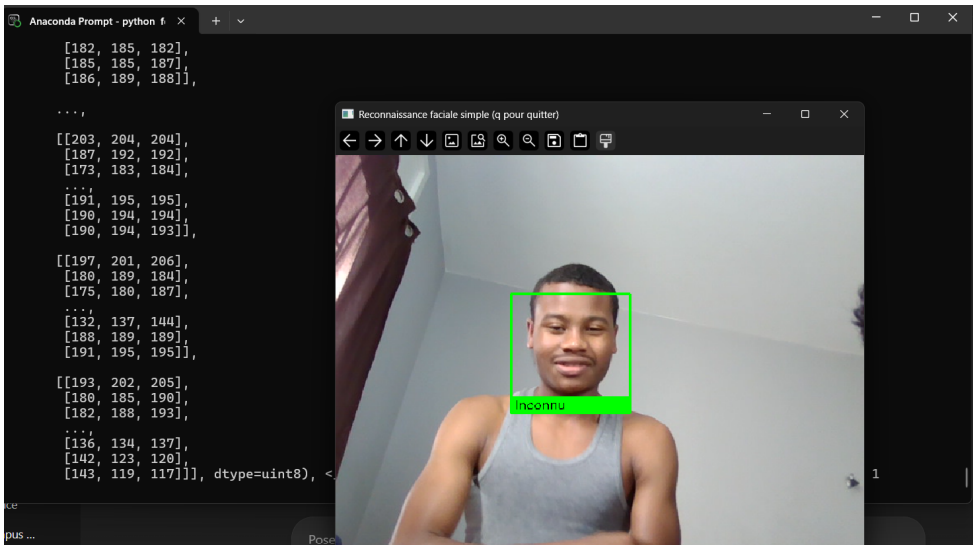
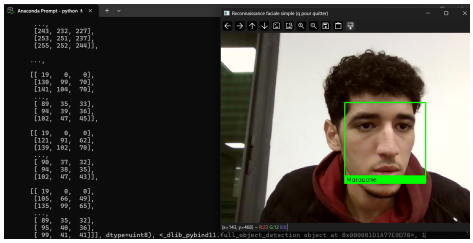


FIGURE 9 – Détection correcte d'une personne inconnue (non-correspondance avec la référence)



(a) Test avec angle de profil

FIGURE 10 – Tests dans différentes configurations

6.2 Tests et validation

6.2.1 Protocole de test

Pour valider le bon fonctionnement du système, j'ai effectué une série de tests systématiques :

maincolor !20 **Scénario**	**Conditions**	**Résultat attendu**
Test 1	Face caméra, lumière normale	Reconnaissance « Marouane »
Test 2	Légèrement de côté (30°)	Reconnaissance « Marouane »
Test 3	Éclairage faible	Reconnaissance « Marouane » (moins stable)
Test 4	Autre personne face caméra	Détection « Inconnu »
Test 5	Plusieurs visages simultanés	Détection de tous les visages
Test 6	Mouvements rapides	Suivi continu
Test 7	Distance variable (50cm-2m)	Détection et reconnaissance
Test 8	Lunettes / accessoires	Reconnaissance variable

TABLE 3 – Protocole de tests effectués

6.2.2 Résultats des tests

Synthèse des résultats de tests

Taux de réussite global : Environ 85-90% dans des conditions normales

- **Excellentes performances ($>$ 95%) :**
  - Face caméra, bonne lumière, distance 50cm-1m
- **Bonnes performances (80-90%) :**
  - Léger profil, lumière moyenne, distance 1-2m
- **Performances dégradées ($<$ 70%) :**
  - Fort contre-jour, très faible lumière, profil marqué

6.3 Analyse des performances

6.3.1 Métriques de performance

maincolor !20 **Métrique**	**Valeur**	**Objectif**
FPS (images/seconde)	15-20	$>$ 10
Temps de détection par frame	50-70 ms	$<$ 100 ms
Utilisation CPU	40-60%	$<$ 70%
Utilisation RAM	500 MB	$<$ 1 GB
Latence perceptible	Minimale	$<$ 200 ms

Table 4 – Métriques de performance du système

Le système offre des performances satisfaisantes pour une utilisation en temps réel, avec une fluidité suffisante et une réactivité acceptable.

6.3.2 Comparaison distance de référence vs inconnu

Voici les distances typiques mesurées selon différents scénarios :

Scénario	**Distance**	**Décision**
Marouane - conditions optimales	0.35 - 0.45	Reconnu
Marouane - lumière moyenne	0.50 - 0.65	Reconnu
Marouane - angle ou ombres	0.65 - 0.75	Reconnu (limite)
Personne différente	0.80 - 1.20	Inconnu

Table 5 – Distances typiques observées (seuil = 0.75)

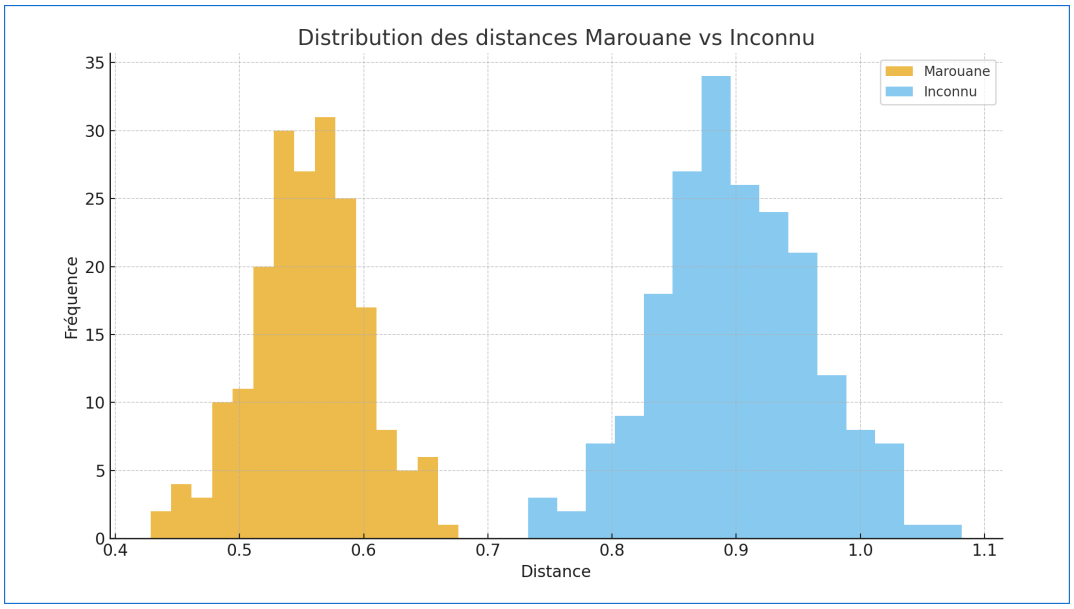


FIGURE 11 – Distribution des distances mesurées pour Marouane vs Inconnus

6.4 Limites identifiées

Bien que fonctionnel, le système présente plusieurs limites qu'il est important de mentionner pour être transparent sur les capacités réelles du projet.

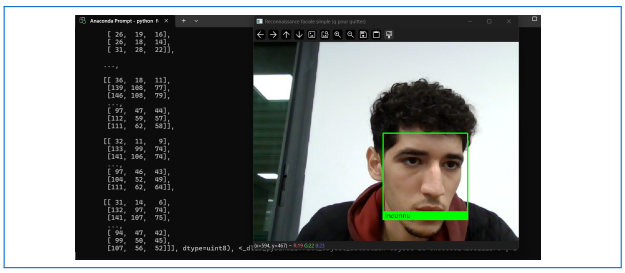
6.4.1 Limites techniques

1. **Sensibilité aux conditions d'éclairage :**
   - Le contre-jour dégrade fortement la reconnaissance
   - L'éclairage très faible augmente les faux négatifs
   - Les ombres marquées sur le visage perturbent l'encodage
2. **Robustesse limitée :**
   - Système conçu pour une seule personne de référence
   - Une seule photo de référence (pas de variations)
   - Pas de gestion d'une base de données multi-personnes
   - Sensible aux accessoires (lunettes de soleil, masque, chapeau)
3. **Angles de vue :**
   - Performance dégradée au-delà de 45° de rotation
   - Visage trop proche ou trop loin moins bien détecté
4. **Performances non optimisées :**
   - Pas de GPU utilisé (traitement CPU uniquement)
   - Pas de mise en cache des encodages
   - Réduction de résolution nécessaire pour la fluidité

6.4.2 Limites fonctionnelles

- × **Pas de persistance :** Les résultats ne sont pas sauvegardés (pas de logs, pas d'historique)
- × **Interface basique :** Fenêtre OpenCV simple, pas d'interface graphique élaborée
- × **Pas de configuration :** Seuil et paramètres codés en dur, pas d'interface de réglage
- × **Mono-utilisateur :** Impossible d'ajouter facilement d'autres personnes à reconnaître

6.4.3 Cas d'échec documentés



(a) Échec avec profil > 60°

FIGURE 12 – Exemples de cas d'échec de reconnaissance

6.5 Comparaison avec l'état de l'art

Il est important de situer ce projet par rapport aux systèmes professionnels :

maincolor!20 **Caractéristique**	**Mon projet**	**Systèmes pro**
Précision	85%	> 99%
Base de données	1 personne	Milliers/millions
Robustesse lumière	Faible	Élevée
Temps réel	Oui (15-20 FPS)	Oui (> 30 FPS)
Infrastructure	PC portable	Serveurs / Cloud
Coût	Gratuit	Licences coûteuses
Complexité	Simple	Très complexe

TABLE 6 – Comparaison avec les systèmes professionnels

Conclusion de l'analyse :

Malgré ses limites, ce projet constitue une excellente introduction à la reconnaissance faciale. Il démontre la maîtrise des concepts fondamentaux et la capacité à créer un système fonctionnel, tout en restant conscient de ses limitations par rapport aux solutions industrielles.

7 Conclusion et perspectives

7.1 Bilan du projet

Ce projet de reconnaissance faciale a constitué une expérience d'apprentissage extrêmement enrichissante qui m'a permis de progresser sur de multiples plans.

7.1.1 Acquis techniques

Compétences techniques développées

Programmation Python :

- Consolidation de la syntaxe et des structures de données
- Utilisation de bibliothèques scientifiques (NumPy, OpenCV)

Vision par ordinateur :

- Compréhension du traitement d'images en temps réel
- Manipulation de frames vidéo et de matrices de pixels
- Détection d'objets (visages) dans une image

Intelligence artificielle :

- Principe des encodages de visages (embeddings)
- Réseaux de neurones convolutifs (utilisés via face_recognition)

Gestion d'environnement de développement :

- Maîtrise de Conda et des environnements virtuels
- Résolution de conflits de dépendances
- Installation de bibliothèques complexes (dlib, OpenCV)

7.1.2 Compétences transversales

Au-delà des aspects purement techniques, ce projet m'a permis de développer des compétences essentielles pour un ingénieur :

- ⋆ **Autonomie :** Capacité à apprendre seul en consultant documentation, forums, tutoriels
- ⋆ **Résolution de problèmes :** Approche méthodique face aux erreurs et bugs
- ⋆ **Persévérance :** Ne pas abandonner face aux difficultés (problèmes d'installation, erreurs techniques)
- ⋆ **Rigueur :** Documentation du code, tests systématiques, rapport structuré
- ⋆ **Esprit critique :** Analyse objective des limites et des améliorations possibles

7.2 Valeur ajoutée pour ma candidature

Ce projet s'inscrit pleinement dans ma démarche de candidature à Campus France et aux formations d'ingénieur en informatique/IA :

Apports pour mon dossier de candidature

1. **Démonstration de motivation :**
   - Projet réalisé de ma propre initiative, sans obligation académique
   - Investissement personnel en temps et en efforts
   - Intérêt concret pour l'IA et la vision par ordinateur
2. **Preuve de capacités techniques :**
   - Maîtrise de Python et des bibliothèques IA
   - Capacité à mener un projet technique de bout en bout
   - Compréhension des concepts fondamentaux de l'IA
3. **Maturité professionnelle :**
   - Documentation rigoureuse (ce rapport)
   - Analyse critique honnête des résultats et limites

7.3 Perspectives d'amélioration

Ce projet, bien que fonctionnel, n'est qu'un premier pas. Voici les améliorations que j'envisage d'apporter dans le futur.

7.3.1 Améliorations techniques à court terme

1. **Multi-utilisateurs :**
   - Créer une base de données de plusieurs personnes
   - Charger dynamiquement plusieurs images de référence
   - Gérer un dictionnaire nom : encodage
2. **Plusieurs photos par personne :**
   - Stocker 3-5 photos par personne (angles différents)
   - Calculer la distance moyenne pour plus de robustesse
   - Réduire les faux négatifs
3. **Interface graphique :**
   - Créer une GUI avec Tkinter ou PyQt
   - Permettre d'ajouter/supprimer des personnes facilement
   - Réglage du seuil via slider
4. **Logging et statistiques :**
   - Sauvegarder les détections dans un fichier CSV
   - Horodatage des reconnaissances
   - Statistiques d'utilisation

7.3.2 Améliorations avancées à moyen terme

1. **Utilisation du GPU :**
   - Intégrer TensorFlow ou PyTorch avec support CUDA
   - Accélérer le traitement (x10 à x20 plus rapide)
   - Permettre le traitement full HD sans réduction
2. **Modèles plus robustes :**
   - Tester des modèles récents (FaceNet, ArcFace, InsightFace)

- Améliorer la précision sur angles difficiles
- Meilleures performances en faible luminosité

3. **Détection d'attributs :**
   - Estimation de l'âge
   - Détection d'émotions
   - Reconnaissance d'expressions faciales
4. **Fonctionnalités avancées :**
   - Suivi de visages entre frames (tracking)
   - Détection de vivacité (anti-spoofing avec photos)
   - Reconnaissance même avec masque facial

7.3.3 Applications concrètes envisagées

En poursuivant le développement de ce projet, plusieurs applications pratiques pourraient être créées :

Applications futures possibles

1. **Système de contrôle d'accès :**
   - Déverrouillage automatique d'une porte
   - Authentification pour accéder à un ordinateur
   - Enregistrement des entrées/sorties
2. **Système de présence automatique :**
   - Prise automatique de présence en cours
   - Statistiques de fréquentation
   - Alertes en cas d'absence
3. **Assistant personnel intelligent :**
   - Personnalisation de l'interface selon l'utilisateur
   - Recommandations adaptées
   - Interaction vocale couplée à la reconnaissance faciale
4. **Analyse comportementale :**
   - Détection de fatigue (pour conducteurs)
   - Mesure de l'attention (en situation d'apprentissage)
   - Analyse d'émotions en temps réel

7.4 Intégration dans mon parcours d'études

Ce projet s'inscrit dans une démarche plus globale de préparation à mes études d'ingénieur en France :

1. **Projet 1 (actuel) :** Reconnaissance faciale
2. **Projet 2 (prévu) :** Chatbot avec traitement du langage naturel (NLP)
3. **Projet 3 (envisagé) :** Système de recommandation avec machine learning

Ces projets visent à construire un portfolio cohérent démontrant mes compétences en IA et ma progression continue.

7.5 Réflexion personnelle

Au-delà des aspects techniques, ce projet m'a beaucoup appris sur moi-même :

- J'ai découvert que je prends du plaisir à résoudre des problèmes complexes
- J'ai réalisé l'importance de la documentation et de la rigueur
- J'ai compris que l'échec (erreurs, bugs) fait partie intégrante de l'apprentissage
- J'ai gagné en confiance dans ma capacité à apprendre en autonomie

7.6 Mot de conclusion

Ce mini-projet de reconnaissance faciale constitue une **première étape concrète** vers mon objectif de devenir ingénieur en intelligence artificielle. Il illustre ma motivation réelle pour ce domaine et ma détermination à acquérir les compétences nécessaires.

En réalisant ce projet de manière autonome, j'ai démontré ma capacité à :

Apprendre de nouvelles technologies

Persévérer face aux difficultés

Créer des solutions fonctionnelles

Analyser de manière critique mes résultats

Documenter rigoureusement mon travail

Je suis convaincu que les compétences développées à travers ce projet, ainsi que celles que j'acquerrai lors de ma formation d'ingénieur en France, me permettront de contribuer significativement au domaine de l'intelligence artificielle et de développer des solutions innovantes répondant aux défis de demain.

Ce projet témoigne de mon sérieux, de ma motivation et de ma préparation pour intégrer un cycle ingénieur en informatique/IA en France. Je suis impatient de pouvoir approfondir ces connaissances dans un cadre académique d'excellence et de continuer à développer mes compétences aux côtés d'experts et de pairs passionnés.

« L'intelligence artificielle est la nouvelle électricité »
– *Andrew Ng*

Ce projet n'est que le début d'un long parcours
vers la maîtrise de cette technologie transformative.

Annexes

A. Ressources et références

Documentation consultée

- **OpenCV Documentation :** https://docs.opencv.org/
- **face_recognition Library :** https://github.com/ageitgey/face_recognition
- **dlib Documentation :** http://dlib.net/
- **Conda User Guide :** https://docs.conda.io/

Tutoriels et articles

- Tutoriels sur Stack Overflow pour résolution d'erreurs spécifiques
- Documentation Python officielle : https://docs.python.org/3/

B. Prérequis d'installation

Pour reproduire ce projet, voici la configuration complète :

Listing 13 – Installation complète de l'environnement

```
# 1. Installation de Miniconda (si pas deja installe)
# Telecharger depuis : https://docs.conda.io/en/latest/miniconda.html

# 2. Creation de l'environnement
conda create -n face python=3.10
conda activate face

# 3. Installation des bibliotheques via conda
conda install -c conda-forge dlib
conda install -c conda-forge opencv
conda install -c anaconda numpy

# 4. Installation via pip
pip install face_recognition
pip install pillow
pip install face_recognition_models
pip install click

# 5. Verification
python -c "import cv2, face_recognition; print('Installation OK')"
```

C. Structure des fichiers du projet

Listing 14 – Arborescence du projet

```
reconnaissance_faciale/

          fcl_rgt.py              # Script principal
          marouane.jpg            # Image de reference
          requirements.txt        # Liste des dependances
          README.md               # Documentation du projet
          rapport.pdf             # Ce rapport
```

D. Fichier requirements.txt

Listing 15 – requirements.txt

```
opencv-python==4.8.1.78
face-recognition==1.3.0
dlib==19.24.0
numpy==1.24.3
Pillow==10.0.1
face-recognition-models==0.3.0
click==8.1.7
```

E. Contact et informations

Auteur : Marouane Zaouia.
Email : marouane10za@gmail.com
GitHub : https ://github.com/ZAWYAAA/a
Date de réalisation : Décembre 2025
Cadre : Candidature Campus France - Cycle Ingénieur

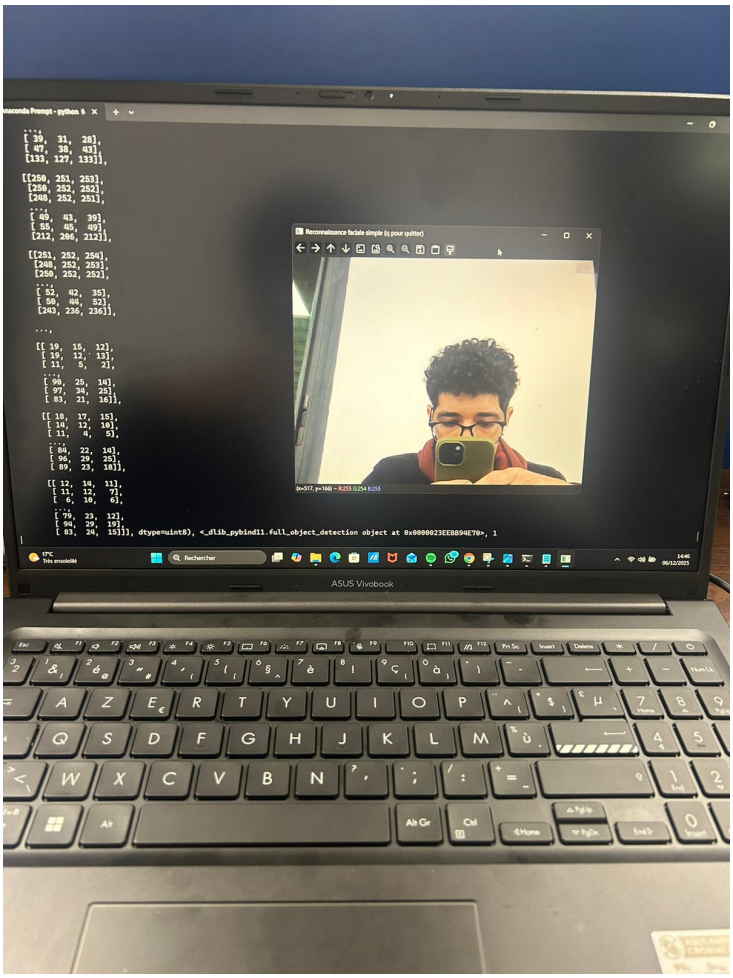


FIGURE 13 – Configuration matérielle utilisée pour le projet