Vision Artificielle

Traitement et analyse des images numériques

FST Tanger

Année Scolaire 2023/2024

La Détection de l'Etat Emotionnel des Personnes

Devoir 2 : Mini Projet

Préparé par :

ABARGHACHE Najlae
AOULEK Ibtissam

Supervisé par :

Pr. M'hamed AIT KBIR

Remerciement !

Nous tenons à exprimer nos sincères remerciements à tous les enseignants qui nous ont enseigné et qui, grâce à leurs compétences, nous ont soutenus dans la poursuite de nos études. Nous remercions également l'administration pour ses efforts. Nous sommes profondément reconnaissantes pour votre assistance.

Sommaire

I. Introduction

L'intégration synergique de la vision par ordinateur, du traitement d'image et des technologies de reconnaissance faciale a ouvert la voie à des avancées significatives dans la compréhension des expressions émotionnelles humaines. Ce projet se penche sur l'exploitation des repères du visage en utilisant Python, DLib, et OpenCV pour la détection précise de l'état émotionnel des individus à travers les six expressions faciales universellement reconnues par les psychologues : la joie, la tristesse, la surprise, la colère, la peur, et le dégoût.

Les repères du visage, ou bien Facial Landmarks, représentant les points caractéristiques du visage tels que les yeux, le nez, et la bouche, constituent une empreinte digitale visuelle qui capture les subtilités des mouvements musculaires faciaux. Avec l'efficacité de DLib et OpenCV, deux puissantes bibliothèques Python, nous sommes en mesure d'explorer en détail ces caractéristiques, permettant ainsi une analyse fine des expressions émotionnelles.

Dans ce rapport, nous détaillerons notre approche méthodologique, mettant en lumière les mécanismes sous-jacents des Facial Landmarks, leur importance dans la reconnaissance émotionnelle, ainsi que la manière dont Python, avec DLib et OpenCV, devient le cadre idéal pour traduire ces concepts complexes en une application pratique. Notre objectif ultime est d'offrir une contribution significative à la compréhension automatisée des émotions humaines, avec des implications potentielles dans des domaines allant de la psychologie à l'intelligence artificielle émotionnelle.

II. Problématique

Bien que l'utilisation de la vision par ordinateur, des repères du visage (Facial Landmarks), et des technologies telles que DLib et OpenCV ait considérablement amélioré la capacité à déchiffrer les expressions faciales, plusieurs défis demeurent dans la détection précise de l'état émotionnel des individus. La problématique centrale de ce projet réside dans la nécessité de surpasser les limites actuelles de la technologie pour une compréhension plus fine et contextuelle des émotions humaines à travers les six expressions universellement reconnues : la joie, la tristesse, la surprise, la colère, la peur, et le dégoût.

- Variabilité individuelle : Chaque individu exprime les émotions de manière unique, introduisant une variabilité considérable dans la détection des repères du visage.
- Expressivité subtile : Les émotions humaines ne se limitent pas toujours à des expressions faciales évidentes, mais peuvent également impliquer des variations subtiles dans les mouvements des muscles du visage
- Défis liés à l'environnement : Les conditions d'éclairage, les angles de vue, et les obstacles potentiels dans l'environnement peuvent affecter la précision de la détection des repères du visage.
- Temps réel et efficacité : Dans des applications pratiques telles que les interfaces homme-machine, la détection émotionnelle doit souvent s'effectuer en temps réel.

III. Outils Utilisés

2. Python :

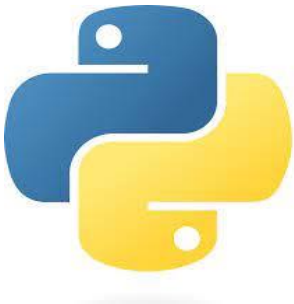


Figure 1 : Langage de Programmation Python

Python est un langage de programmation interprété, généraliste et de haut niveau. Il est apprécié pour sa syntaxe simple et lisible, ce qui en fait un choix populaire pour le développement d'applications, y compris dans le domaine de l'apprentissage automatique et de la vision par ordinateur.

3. OpenCV:

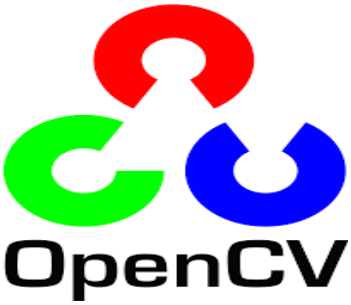


Figure 2 : Bibliothèque de Traitement d'images OpenCV

OpenCV (Open Source Computer Vision) est une bibliothèque open source spécialisée dans la vision par ordinateur et le traitement d'images. Elle propose des fonctions pour la capture, le traitement et l'analyse d'images et de vidéos, ainsi que des algorithmes avancés de vision par ordinateur.

4. Dlib :

Figure 3 : Bibliothèque de Développement de Logiciels Dlib

Dlib est une bibliothèque C++ open source utilisée principalement pour le développement de logiciels liés à la vision par ordinateur. Elle propose des outils pour la détection de visages, l'extraction de repères du visage, et d'autres tâches liées à l'analyse d'images.

5. Scikit-learn (sklearn) :

Figure 4 : Bibliothèque Scikit-learn pour l'Apprentissage Automatique

Scikit-learn est une bibliothèque open source en Python dédiée à l'apprentissage automatique. Elle offre des outils pour la classification, la régression, le clustering, la réduction de dimension, et bien d'autres, facilitant la mise en œuvre d'algorithmes d'apprentissage automatique.

6. Jupyter Notebook :

Figure 5 : Application Web Jupiter NoteBook Pour la création et le partage de Documents/Code

Jupyter Notebook est une application web interactive permettant de créer et de partager des documents qui intègrent du code en direct, des équations, des visualisations, et du texte narratif. C'est un environnement de développement interactif largement utilisé dans le domaine de la data science et de l'apprentissage automatique.

Chaque outil mentionné ci-dessus joue un rôle important dans la réalisation du projet :

Python est le langage de programmation principal qui orchestre l'ensemble du projet.
OpenCV est utilisé pour la manipulation et le traitement d'images, y compris la capture d'images, la conversion en niveaux de gris, etc.
Dlib contribue à la détection de visages et à l'extraction de Facial Landmarks.
scikit-learn fournit le modèle de machine learning SVM utilisé pour la classification émotionnelle.
Jupyter Notebook utilisé pour le développement interactif et la visualisation des résultats.

En combinant ces outils, le projet bénéficie d'un ensemble puissant d'outils pour la manipulation d'images, l'extraction de caractéristiques, l'apprentissage automatique, et la présentation interactive des résultats.

IV. Entrainement du Modèle et Algorithme

1. Composants Essentiels de l'Algorithme

1.1. Modèle d'Apprentissage SVM

Le modèle d'apprentissage adopté dans ce projet est un Support Vector Machine (SVM), un algorithme de classification couramment utilisé en apprentissage automatique. Le choix d'un SVM repose sur sa capacité à séparer efficacement des données non linéaires en utilisant des hyperplans, ce qui le rend approprié pour la classification d'émotions à partir des repères du visage (Facial Landmarks).

Le SVM est paramétré avec un noyau linéaire, ce qui implique que l'algorithme cherche à trouver un hyperplan linéaire optimal pour séparer les données en fonction des classes d'émotions. Cette décision découle de la nature des données de Facial Landmarks et de l'hypothèse que des relations linéaires existent entre ces caractéristiques et les expressions émotionnelles.

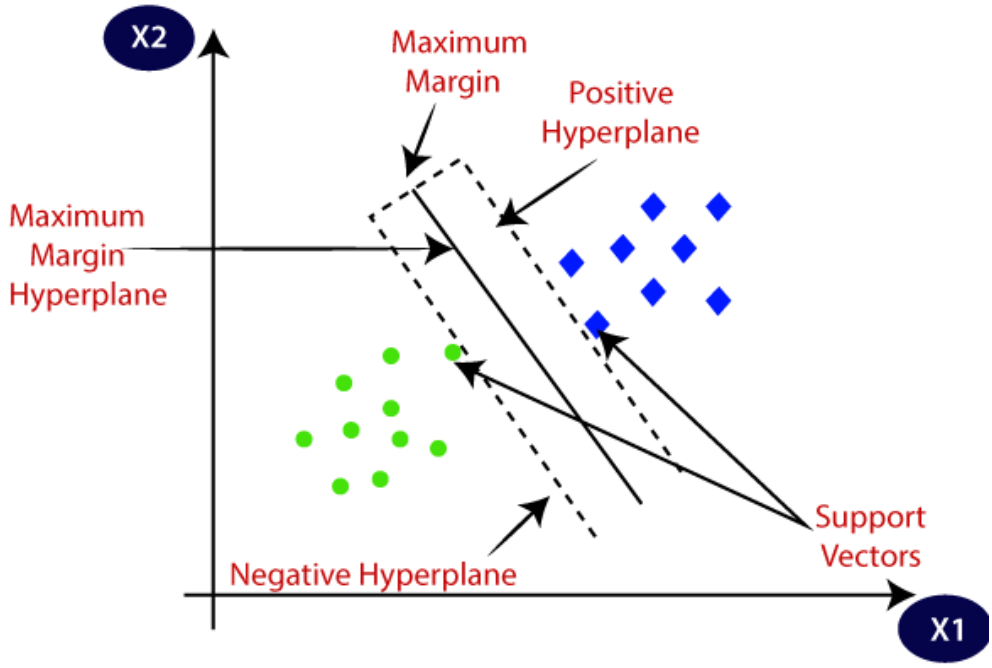


Figure 6 : Modèle d'Apprentissage SVM

1.2. Bibliothèque Dlib pour la détection des Repères Faciaux

Qu'est-ce que sont les repères faciaux ?

Les repères faciaux sont des points sur un visage qui représentent des caractéristiques faciales telles que les yeux, le nez, la bouche, etc. Ces repères faciaux peuvent être utilisés pour

différentes applications telles que l'estimation de la pose de la tête, le remplacement de visage, l'alignement de visage, et plus encore.

Les 68 caractéristiques faciales de Dlib

Dlib fournit un détecteur de repères faciaux pré-entraîné qui peut détecter 68 points sur un visage.

L'image ci-dessous montre l'emplacement de ces 68 points :

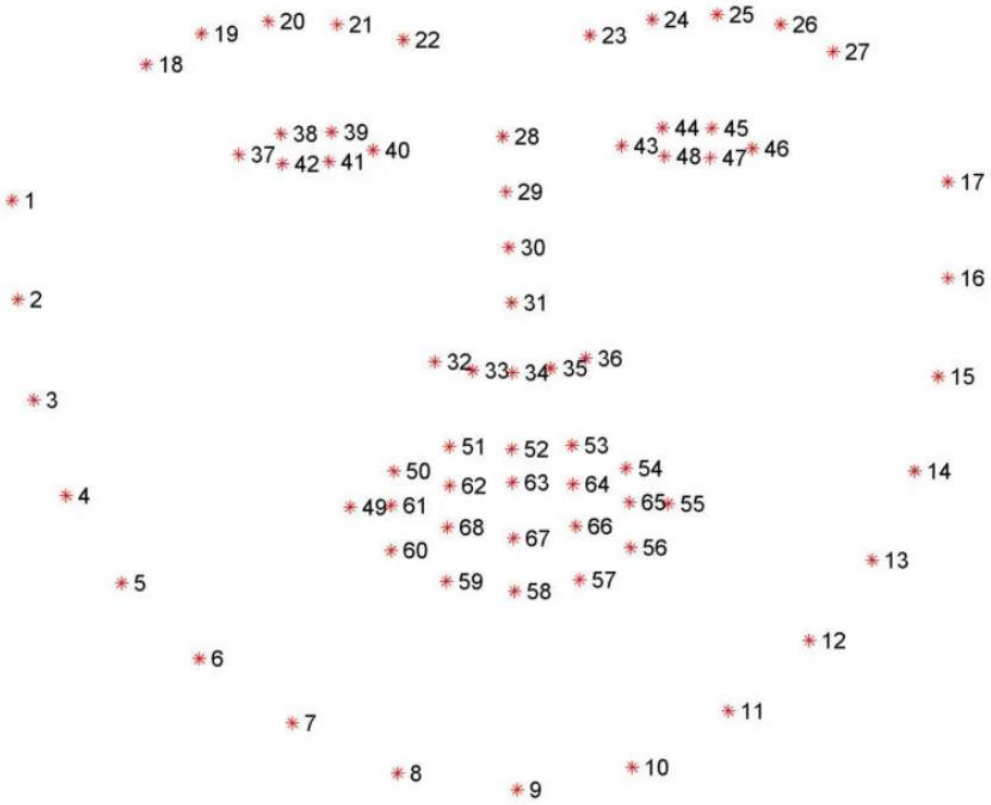


Figure 7: Visualisation des 68 coordonnées de repères du visage

Comme on peut le voir dans l'image ci-dessus, chaque caractéristique faciale est cartographiée avec un ensemble de points. Si on veut, par exemple, localiser une bouche sur le visage, on peut utiliser les points de 49 à 68.

Étant donné qu'on va utiliser un détecteur de repères faciaux pré-entraîné, on doit télécharger le fichier shape_predictor_68_face_landmarks.dat à cet effet.

2. Flux d'Algorithme d'apprentissage

Le processus de l'algorithme se déploie comme suit pour l'analyse des expressions faciales:

- **Prétraitement des images :**
  - Conversion des images en niveaux de gris.

- Application d'une égalisation adaptative d'histogramme pour améliorer le contraste.

- **Détection des Facial Landmarks :**

  - Utilisation du détecteur de visages de dlib pour localiser les visages dans les images.
  - Exploitation du prédicteur de landmarks de dlib pour extraire les coordonnées des 68 points caractéristiques du visage.

- **Extraction des caractéristiques :**

  - Calcul des vecteurs de caractéristiques en utilisant les coordonnées des Facial Landmarks.
  - Chaque vecteur représente une combinaison de positions, distances, et angles entre les landmarks, fournissant ainsi une représentation riche des expressions faciales.

- **Entraînement du SVM :**

  - Utilisation des vecteurs de caractéristiques pour entraîner le SVM avec un noyau linéaire.
  - Chaque vecteur est associé à une étiquette correspondant à l'émotion réelle associée à la personne dans l'image.

- **Évaluation de la performance :**

  - Division des données en ensembles d'entraînement et de prédiction pour évaluer la capacité du modèle à généraliser.
  - Calcul de la précision du modèle sur l'ensemble de prédiction pour évaluer sa performance.

- **Sauvegarde du modèle :**

  - Sauvegarde du modèle SVM entraîné dans un fichier pickle.
  - Cette étape permet d'utiliser le modèle formé ultérieurement sans avoir à reprendre le processus d'entraînement.

3. Calculs Mathématiques pour les Vecteurs de Repères Faciaux

Dans cette partie du code, les coordonnées X et Y des 68 repères faciaux sont extraits pour chaque image et des statistiques sont calculées pour les valeurs obtenues. Voici les formules associées :

3.1. Extraction des Coordonnées X et Y des Repères Faciaux

$$x_i = shape.part(i).x$$

$$y_i = shape.part(i).y$$

Où i varie de 1 à 68, représentant les 68 repères faciaux et $shape$ représente l'ensemble des repères faciaux détectés pour le visage spécifique dans l'image donnée.

3.2. Calcule de la Moyenne des coordonnées X et Y

$$x_{mean} = \frac{1}{68}\sum_{i=1}^{68} x_i$$

$$y_{mean} = \frac{1}{68}\sum_{i=1}^{68} y_i$$

Cela calcule la moyenne des coordonnées X et Y de tous les repères faciaux.

3.3. Déviation centrale des coordonnées X et Y

$$x_{central} = x_i - xmean$$

$$y_{central} = y_i - ymean$$

Cela calcule la déviation centrale de chaque coordonnée X et Y par rapport à leur moyenne respective.

Le calcul de la déviation centrale par rapport à la moyenne respective permet de normaliser les coordonnées des repères faciaux. Et c'est essentiel pour rendre les données indépendantes de la position absolue du visage dans l'image, en se concentrant plutôt sur la structure relative des repères faciaux ou bien landmarks.

3.4. Calcul de la Distance euclidienne entre un Repère Facial et la Moyenne

$$dist = \sqrt{(x - x_{mean})^2 + (y - y_{mean})^2}$$

Où (x_{mean}, y_{mean}) est le centre de gravité (moyenne des coordonnées des landmarks).

3.5. Calcul de L'angle de Chaque Repère Facial par rapport à la Déviation Centrale Correspondante

$$angle = \frac{\arctan\left(\frac{y_{central}}{x_{central}}\right) \times 360}{2 \times \pi}$$

Où $x_{central}$ et $y_{central}$ sont les déviations centrales de chaque coordonnée X et Y par rapport à leurs moyennes respectives.

Ces formules sont utilisées pour caractériser la position relative des repères faciaux par rapport au centre de gravité du visage. La distance euclidienne mesure la distance entre chaque repère et le centre de gravité, et l'angle mesure l'orientation de chaque repère par rapport au centre de gravité.

4. Implémentation Algorithmique : Analyse Détaillée et Explication du Code

4.1. Importation des Bibliothèques et Initialisation des Composants Essentiels :

```
import cv2
import glob
import random
import math
import numpy as np
import dlib
import itertools
from sklearn.svm import SVC as SVMClassifier
import pickle
```

```
emotions = ["colere", "degout", "peur", "joie", "tristesse", "surprise", "neutre"]
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8)) #Création d'un égaliseur
detector = dlib.get_frontal_face_detector() #Initialisation du détecteur de visages
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat") #Initiali
clf = SVMClassifier(kernel='linear', probability=True, tol=1e-3) #Initialisation du

data = {} #Dictionnaire pour toutes les valeurs
```

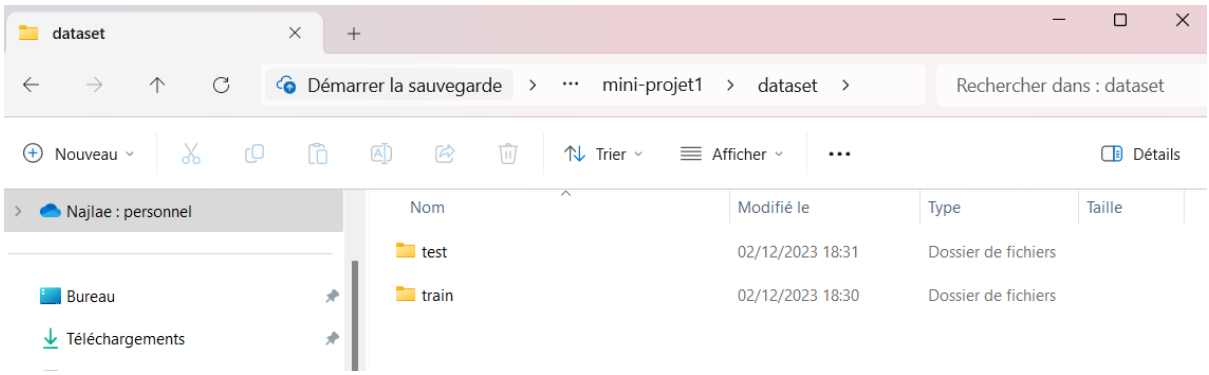
Ce segment de code initie les composants essentiels pour la détection d'émotions à partir d'images faciales. Il établit une liste d'émotions, configure un égaliseur d'histogramme, initialise un détecteur de visages, un prédicteur de repères faciaux (landmarks), et un classificateur SVM avec un noyau linéaire. De plus, il crée un dictionnaire pour stocker les données requises. Ces préparatifs revêtent une importance cruciale pour les étapes suivantes, telles que l'extraction des caractéristiques faciales et l'entraînement du modèle.

4.2. Préparation des Chemins d'Images

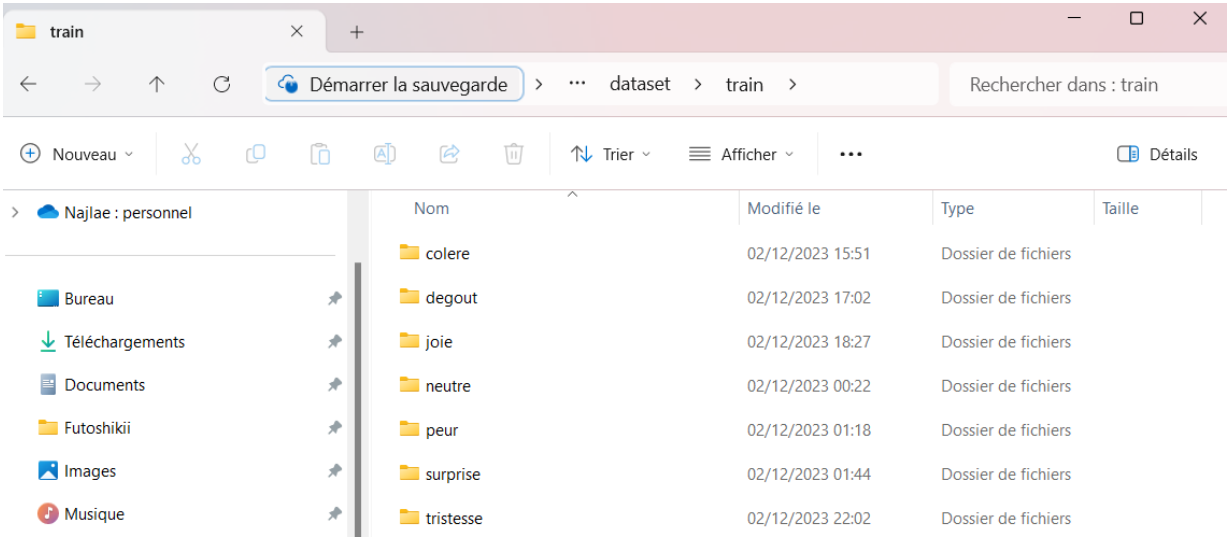
```python
'''Récupère les chemins de fichiers des images associées
    à une émotion spécifique à partir des ensembles
    d'entraînement et de test.'''
def get_files(emotion):
    train_files = glob.glob("dataset\\train\\%s\\*" % emotion)
    random.shuffle(train_files)

    test_files = glob.glob("dataset\\test\\%s\\*" % emotion)
    random.shuffle(test_files)

    return train_files, test_files
```

Cette fonction récupère de manière aléatoire les chemins des images associées à une émotion spécifique, tant dans l'ensemble d'entraînement que dans l'ensemble de test, facilitant ainsi la préparation des données pour l'apprentissage du modèle.

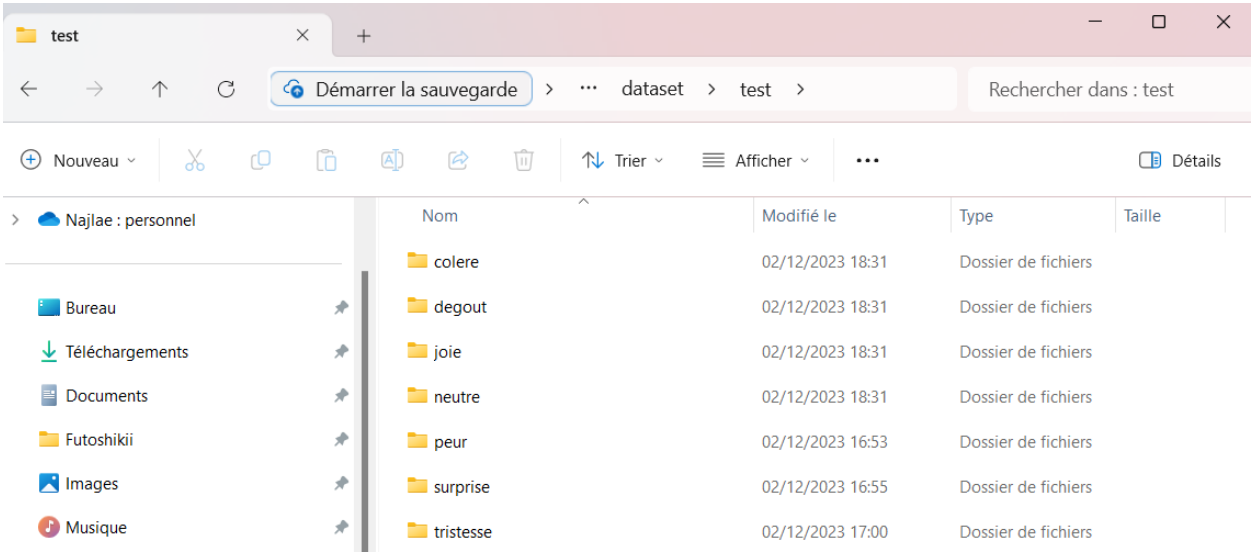
❖ Structure du dossier `dataset` contenant l'ensemble de données (images) pour l'entrainement et le test :



Sous-dossier `train` :



Sous-dossier `test` :



4.3. Visualisation des Repères Faciaux

```python
def get_points(image):
    # Vérifier si l'image a été correctement lue
    if image is None:
        print("Erreur: Impossible de lire l'image.")
        return

    # Convertir l'image OpenCV en niveaux de gris
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Détecter les visages avec le détecteur dlib
    detections = detector(gray_image, 1)

    for k, d in enumerate(detections):
        shape = predictor(gray_image, d)

        # Dessiner les points sur l'image couleur
        for i in range(68):
            cv2.circle(image, (shape.part(i).x, shape.part(i).y), 1, (0, 0, 255), -1)

        # Dessiner un rectangle autour du visage
        cv2.rectangle(image, (d.left(), d.top()), (d.right(), d.bottom()), (255, 0, 0), 2)

    # Afficher l'image avec les landmarks
    cv2.imshow("Image avec landmarks", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Exemple d'utilisation
image_path = "image1.png"
image = cv2.imread(image_path)
get_points(image)
```

La fonction **`get_points`** prend une image en entrée, détecte les visages, extrait et dessine les repères faciaux (68 points spécifiques) sur l'image, ainsi qu'un rectangle autour du visage. L'exemple d'utilisation démontre son application pour visualiser ces repères sur une image spécifique.

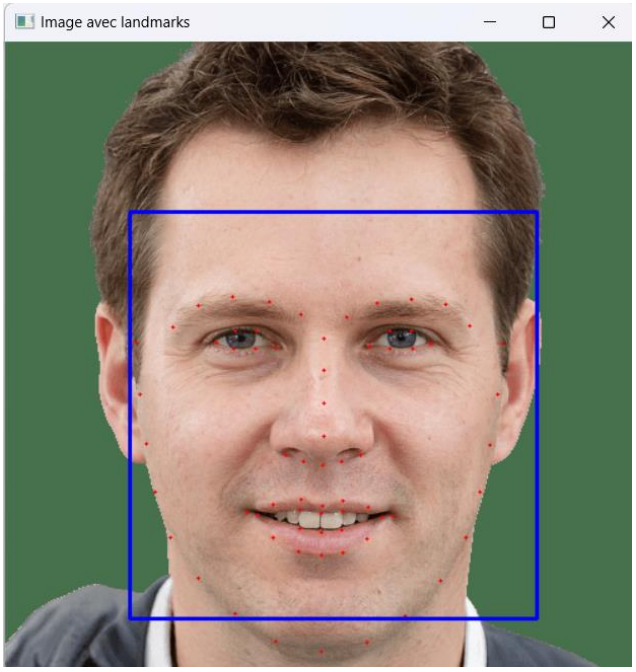


Figure 8 : Résultats de la Visualisation des repères faciaux sur l'image "image1.png"

4.4. Extraction et Normalisation des Repères Faciaux

```
def get_landmarks(image):
    detections = detector(image, 1) # Utilise le détecteur de visages de dlib pour trouver toutes
    for k,d in enumerate(detections): #Pour chaque instance de visage détectée individuellement
        shape = predictor(image, d) #Dessine les landmarks faciaux avec la classe predictor
        xlist = []
        ylist = []
        for i in range(1,68): #Itère sur les 68 landmarks faciaux et Stocke les coordonnées X et Y
            xlist.append(float(shape.part(i).x))
            ylist.append(float(shape.part(i).y))
        # Enregistre les valeurs moyennes des coordonnées X et Y
        xmean = np.mean(xlist)
        ymean = np.mean(ylist)
        # Stocke la déviation centrale
        xcentral = [(x-xmean) for x in xlist]
        ycentral = [(y-ymean) for y in ylist]

        landmarks_vectorised = []
        for x, y, w, z in zip(xcentral, ycentral, xlist, ylist):# Analyse la présence de landmarks
            landmarks_vectorised.append(w)
            landmarks_vectorised.append(z)
            # Extrait le centre de gravité avec la moyenne des axes
            meannp = np.asarray((ymean,xmean))
            coornp = np.asarray((z,w))
            # Mesure de la distance et de l'angle de chaque landmark par rapport au centre de grav
            dist = np.linalg.norm(coornp-meannp)
            landmarks_vectorised.append(dist)
            landmarks_vectorised.append((math.atan2(y, x)*360)/(2*math.pi))

        data['landmarks_vectorised'] = landmarks_vectorised #Stocke les landmarks dans le dictionn
    if len(detections) < 1: # Si aucun visage n'est détecté, stocke une erreur dans le dictionnair
        data['landmarks_vestorised'] = "error"
```

La fonction **`get_landmarks`** utilise le détecteur de visages de dlib pour localiser toutes les instances de visages dans une image. Pour chaque visage détecté, elle extrait les landmarks faciaux, puis normalise ces landmarks en calculant les coordonnées moyennes et les déviations centrales. Les coordonnées normalisées, ainsi que la distance et l'angle de chaque landmark par rapport au centre de gravité, sont stockés dans un vecteur. Ce vecteur est ensuite ajouté au dictionnaire global data. En cas de non-détection de visage, une erreur est signalée dans le dictionnaire. Cette fonction prépare les données des landmarks pour l'analyse des expressions faciales et l'entraînement du modèle.

4.5. Création des Ensembles d'Entraînement et de Prédiction

```python
def make_sets():
    training_data = []
    training_labels = []
    prediction_data = []
    prediction_labels = []
    for emotion in emotions: # Itère sur chaque émotion dans la liste emotions
        print(" working on %s" %emotion)
        training, prediction = get_files(emotion) # obtenir les chemins des fichiers d'entraînement et de prédiction

        #Ajoute les données à la liste d'entraînement et de prédiction, et génère les étiquettes 0-7
        for item in training:
            image = cv2.imread(item) # Ouvre l'image
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # Convertit en niveaux de gris
            clahe_image = clahe.apply(gray)  # Applique une égalisation d'histogramme locale
            get_landmarks(clahe_image) # Extrait les landmarks
            if data['landmarks_vectorised'] == "error": # Si aucun visage n'est détecté, affiche un message d'erreur
                print("no face detected on this one")
            else: # Sinon, ajoute le vecteur de landmarks à la liste training_data et ajoute l'indice de l'émotion à
                training_data.append(data['landmarks_vectorised'])
                training_labels.append(emotions.index(emotion))

        # Fait la même chose pour chaque image dans l'ensemble de prédiction
        for item in prediction:
            image = cv2.imread(item)
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            clahe_image = clahe.apply(gray)
            get_landmarks(clahe_image)
            if data['landmarks_vectorised'] == "error":
                print("no face detected on this one")
            else:
                prediction_data.append(data['landmarks_vectorised'])
                prediction_labels.append(emotions.index(emotion))

    return training_data, training_labels, prediction_data, prediction_labels
```

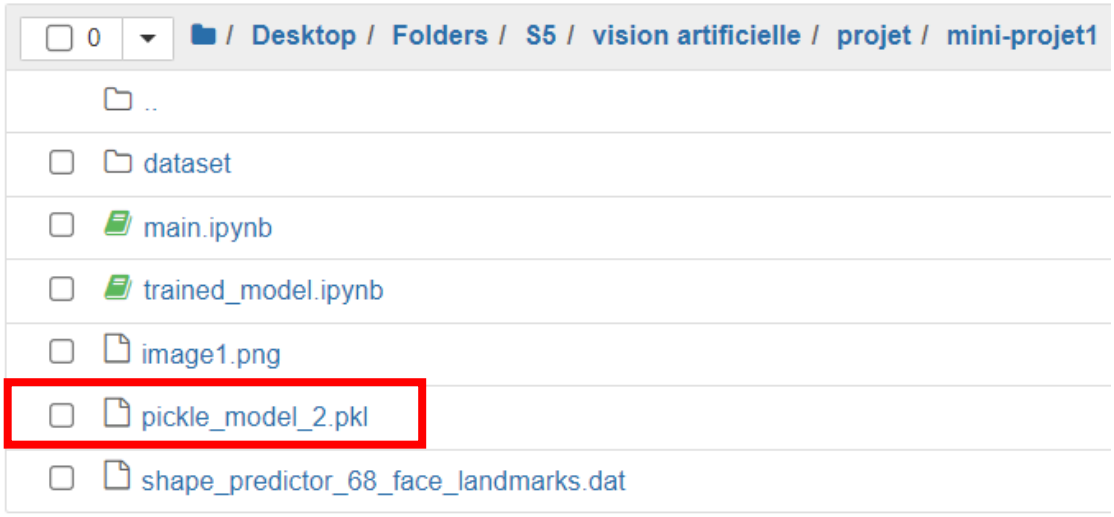
Cette fonction, **`make_sets`**, itère sur chaque émotion, récupère les chemins des fichiers d'entraînement et de prédiction, puis extrait les landmarks normalisés pour chaque image. Les vecteurs de landmarks sont ajoutés aux listes d'entraînement et de prédiction, accompagnés des indices correspondants des émotions. Cette fonction prépare les données pour l'apprentissage du modèle de reconnaissance émotionnelle.

4.6. Entraînement et Évaluation du Modèle SVM Linéaire

```
accur_lin = []
for i in range(0,7): # Itère sur 7'itérations d'entraînement
    print("Making sets %s" %i)
    training_data, training_labels, prediction_data, prediction_labels = make_sets()
    npar_train = np.array(training_data) # Convertir l'ensemble d'entraînement en un
    print(training_labels)
    npar_trainlabs = np.array(training_labels) # Convertir les étiquettes d'entraînen
    print("training SVM linear %s" %i) # Entraîne le classificateur SVM sur l'ensembl
    clf.fit(npar_train, training_labels)

    print("getting accuracies %s" %i)
    npar_pred = np.array(prediction_data) # Convertir l'ensemble de prédiction en un
    pred_lin = clf.score(npar_pred, prediction_labels) # Utiliser la fonction score(,
    print("linear: ", pred_lin)
    accur_lin.append(pred_lin) # Stocker la précision dans la liste accur_lin

# Sauvegarder le modèle SVM entraîné dans un fichier pickle
pkl_filename = "pickle_model_2.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(clf, file) # Écrire le modèle dans le fichier pickle

print("Mean value lin svm: %s" %np.mean(accur_lin)) # Affiche la valeur moyenne de la
```

```
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
training SVM linear 5

getting accuracies 5
linear:  0.1897018970189702
Making sets 6
 working on colere
 working on degout
 working on peur
 working on joie
 working on tristesse
 working on surprise
 working on neutre
```

Ce bloc de code itère sur 7 cycles d'entraînement, utilisant la fonction **`make_sets`** pour obtenir les ensembles d'entraînement et de prédiction. Les données d'entraînement sont converties en tableaux NumPy, et un classificateur SVM linéaire (clf) est entraîné sur ces données. La précision du modèle est évaluée sur l'ensemble de prédiction, et la valeur de précision est stockée dans la liste `accur_lin`. À la fin des itérations, le modèle SVM entraîné est sauvegardé dans un fichier pickle, et la valeur moyenne de la précision sur les sept itérations est affichée. Ce processus permet de quantifier la performance du modèle dans la reconnaissance des émotions faciales.



V. Programme de Test du Modèle Entrainé

1. Code du Programme de Test

1.1. Importation des Bibliothèques et Initialisation des Composants Essentiels

```
import cv2
import dlib
import pickle
import numpy as np
import math
from sklearn.svm import SVC
```

```
data = {}
data['landmarks_vectorised'] = []
emotions = ["colere", "degout", "peur", "joie", "tristesse", "surprise", "neutre"]
```

Ce segment de code importe les bibliothèques et initie les composants essentiels incluant un dictionnaire `data` pour stocker les valeurs, et `data['landmarks_vectorised']` qui est initialisé pour enregistrer les landmarks. Ainsi, il établit la liste d'émotions à détecter `emotions`.

1.2. Extraction des Landmarks

```python
def get_landmarks(image):
    # Détecter les visages dans l'image
    detections = detector(image, 1)

    # Pour chaque instance de visage détectée individuellement
    for k, d in enumerate(detections):
        # Dessiner les landmarks faciaux avec la classe predictor
        shape = predictor(image, d)

        # Initialiser les listes pour stocker les coordonnées X et Y
        xlist = []
        ylist = []

        # Stocker les coordonnées X et Y dans les listes respectives
        for i in range(1, 68):
            xlist.append(float(shape.part(i).x))
            ylist.append(float(shape.part(i).y))

        # Calculer les valeurs moyennes des coordonnées X et Y
        xmean = np.mean(xlist)
        ymean = np.mean(ylist)

        # Calculer les déviations centrales par rapport aux moyennes
        xcentral = [(x - xmean) for x in xlist]
        ycentral = [(y - ymean) for y in ylist]

        # Initialiser la liste pour stocker les landmarks normalisés
        landmarks_vectorised = []

        # Analyser la présence de landmarks faciaux
        for x, y, w, z in zip(xcentral, ycentral, xlist, ylist):
            landmarks_vectorised.append(w)
            landmarks_vectorised.append(z)
```

```python
            # Extraire le centre de gravité avec la moyenne des axes
            meannp = np.asarray((ymean, xmean))
            coornp = np.asarray((z, w))

            # Mesurer la distance et l'angle de chaque landmark par rapport au ce
            dist = np.linalg.norm(coornp - meannp)
            landmarks_vectorised.append(dist)
            landmarks_vectorised.append((math.atan2(y, x) * 360) / (2 * math.pi))

        # Stocker les landmarks dans le dictionnaire global
        data['landmarks_vectorised'] = landmarks_vectorised

    # Si aucun visage n'est détecté, stocker une erreur dans le dictionnaire
    if len(detections) < 1:
        data['landmarks_vestorised'] = "error"
```

La fonction `get_landmarks` prend une image en entrée, utilise le détecteur de visages de `dlib` pour trouver les visages, puis extrait les landmarks normalisés pour chaque visage détecté.

1.3. Chargement du Modèle Pré-Entrainé

```
# Configuration des objets requis
pkl_filename = 'pickle_model_2.pkl'  # Fichier du modèle entraîné
with open(pkl_filename, 'rb') as file:  # Charger tous les poids du modèle
    pickle_model = pickle.load(file)
```

`pkl_filename` est le nom du fichier où le modèle pré-entraîné est enregistré. Le modèle est chargé à partir de ce fichier à l'aide de pickle.

1.4. Initialisation de la Webcam et des Détecteurs

```
# Capture de la webcam
video_capture = cv2.VideoCapture(0)

# Initialisation du détecteur de visages
detector = dlib.get_frontal_face_detector()

# Initialisation du prédicteur de landmarks
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

# Taille du cadre de capture (ajustable)
FACE_SHAPE = (200, 200)
```

`video_capture` est l'objet pour la capture de la webcam, et `detector` et `predictor` sont utilisés pour détecter les visages et extraire les landmarks.

1.5. Boucle d'Inférence sur la Webcam

```python
while True:
    ret, frame = video_capture.read()
    # Redimensionner le cadre à une taille plus petite pour un traitement plus rapide
    frame = cv2.resize(frame, FACE_SHAPE)
    # Convertir l'image en niveaux de gris (notre ensemble de données est en niveaux de gris)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Appliquer une égalisation adaptative de l'histogramme local (CLAHE) pour améliorer le contraste
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    clahe_image = clahe.apply(gray)
    # Obtenir les landmarks à partir de l'image traitée
    get_landmarks(clahe_image)

    # Si des landmarks sont détectés
    if data['landmarks_vectorised'] != "error":
        # Convertir les landmarks en tableau numpy
        prediction_data = np.array(data['landmarks_vectorised'])

        # Prédire les étiquettes émotionnelles
        predicted_labels = pickle_model.predict(prediction_data.reshape(1, -1))

        # Afficher l'émotion prédite sur la vidéo en rouge
        emotion_text = emotions[predicted_labels[0]]
        if emotion_text != "neutre":
            cv2.putText(frame, f"Emotion: {emotion_text}", (10, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
    else:
        # Afficher un message si aucun visage n'est détecté
        cv2.putText(frame, "Aucun visage détecté.", (10, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

    # Afficher la sortie de la webcam
    cv2.imshow("image", frame)

    # Sortir de la boucle si l'utilisateur appuie sur 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        video_capture.release()
        cv2.destroyAllWindows()
        break
```

Ce code constitue une boucle infinie qui capture en continu des images à partir de la webcam. Pour chaque image capturée, elle subit un processus de prétraitement, incluant le redimensionnement, la conversion en niveaux de gris, et l'application de l'égalisation adaptative de l'histogramme local (CLAHE) pour améliorer le contraste. Ensuite, les landmarks faciaux sont extraits de l'image traitée à l'aide de la fonction `get_landmarks`. Si des landmarks sont détectés, ils sont convertis en tableau numpy et utilisés pour prédire les étiquettes émotionnelles à l'aide du modèle SVM pré-entraîné. Si une émotion non neutre est prédite, elle est affichée en rouge sur la vidéo. Si aucun visage n'est détecté, un message est affiché. La sortie de la webcam est ensuite affichée dans une fenêtre, et la boucle continue jusqu'à ce que l'utilisateur appuie sur la touche 'q', déclenchant ainsi la libération des ressources de la webcam et la fermeture de la fenêtre vidéo.

2. Résulats Programme de Test

Le programme de test a été exécuté avec succès, fournissant des résultats significatifs dans la détection de certaines émotions faciales à partir de captures vidéo en temps réel. Les captures d'écran obtenues représentent les expressions émotionnelles suivantes : la colère, le dégoût, la peur, la joie, la tristesse, et la surprise. Les émotions détectées sont annotées en rouge dans la partie supérieure gauche de chaque capture d'écran pour une identification claire. Les performances du modèle formé sont visuellement évidentes, démontrant la capacité du système à interpréter et à réagir aux expressions faciales avec précision. Ces résultats sont présentés ci-dessous, reflétant l'efficacité du modèle dans la reconnaissance et la classification des émotions humaines.

Dans cette section, nous présentons les résultats obtenus lors du test du modèle sur des images statiques. Le code ci-dessous a été utilisé pour prédire les émotions à partir des images individuelles :

```python
# Chemin de l'image à prédire
image_path = "surprise.jpg"

# Lire l'image à partir du chemin spécifié
frame = cv2.imread(image_path)

# Redimensionner le cadre à la taille spécifiée
frame = cv2.resize(frame, FACE_SHAPE)

# Convertir l'image en niveaux de gris
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Appliquer une égalisation adaptative de l'histogramme local (CLAHE)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
clahe_image = clahe.apply(gray)

# Obtenir les landmarks à partir de l'image traitée
get_landmarks(clahe_image)

# Si des landmarks sont détectés
if data['landmarks_vectorised'] != "error":
    # Convertir les landmarks en tableau numpy
    prediction_data = np.array(data['landmarks_vectorised'])

    # Prédire les étiquettes émotionnelles
    predicted_labels = pickle_model.predict(prediction_data.reshape(1, -1))

    # Afficher l'émotion prédite sur l'image en rouge
    emotion_text = emotions[predicted_labels[0]]
    cv2.putText(frame, f"Emotion: {emotion_text}", (10, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
else:
    # Afficher un message si aucun visage n'est détecté
    cv2.putText(frame, "Aucun visage détecté.", (10, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

# Afficher l'image avec la prédiction
cv2.imshow("image", frame)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

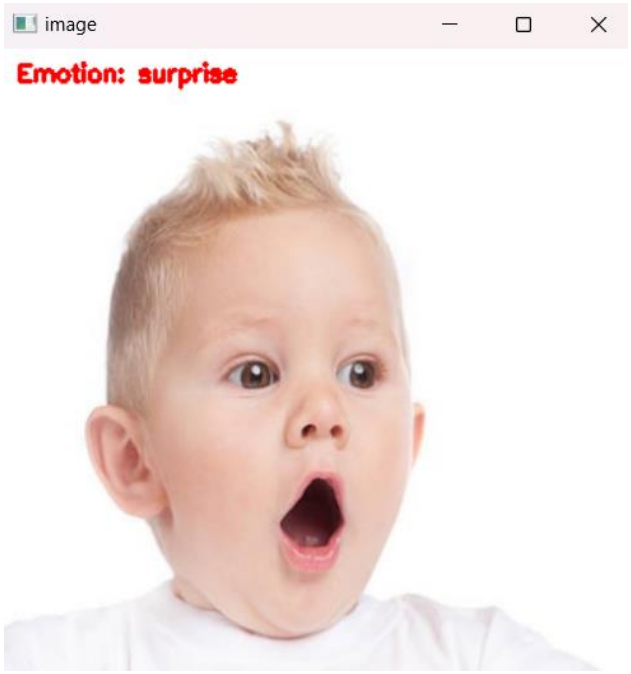


Figure 9: Emotion Détecté : La surprise

Nous avons soumis différentes images au modèle pour évaluer sa capacité à prédire différentes émotions en fonction d'entrées statiques :

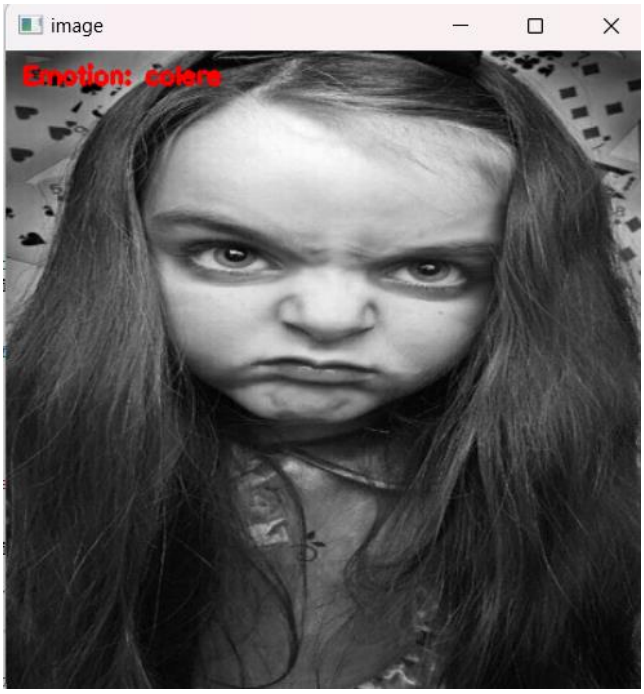


Figure 10: Emotion Détecté : La colère

Figure 11: Emotion Détecté : La tristesse

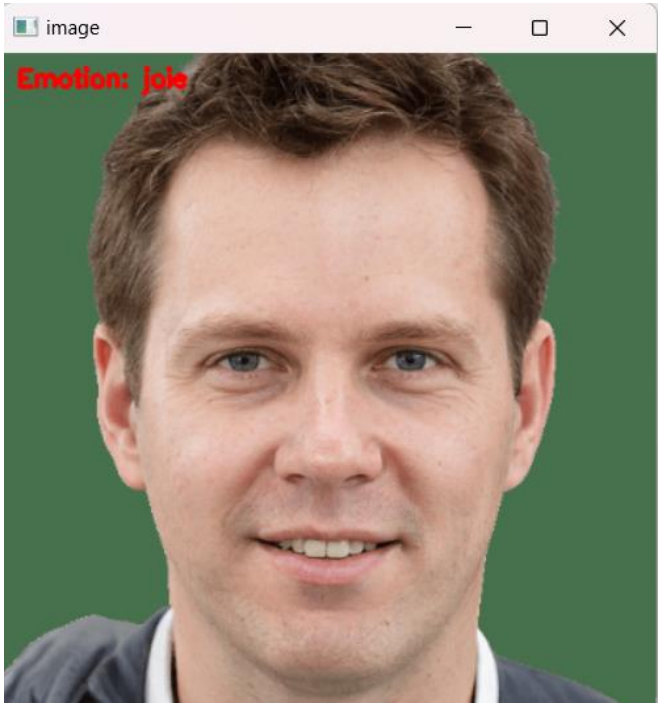


Figure 12: Emotion Détecté : La joie

Figure 13: Emotion Détecté : La peur

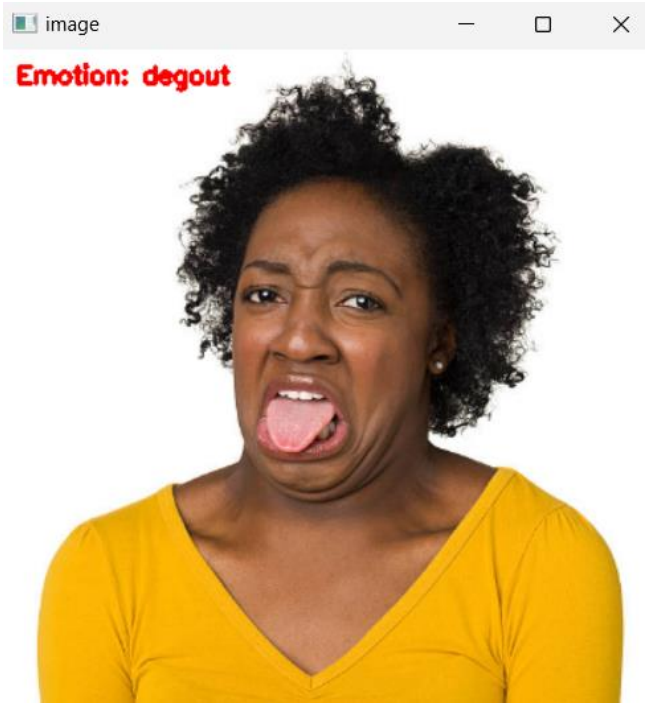


Figure 14: Emotion Détecté : Le dégout

VI. Conclusion

La mise en œuvre de la vision par ordinateur pour la détection de l'état émotionnel des personnes, en utilisant des outils tels que Python, OpenCV, DLib et scikit-learn, a abouti à un projet significatif et prometteur. En exploitant les 68 landmarks faciaux universellement reconnus, le code développé a réussi à extraire des caractéristiques cruciales permettant la classification des six expressions émotionnelles fondamentales : la joie, la tristesse, la surprise, la colère, la peur et le dégoût.

L'utilisation d'un modèle de machine learning, en l'occurrence le Support Vector Machine (SVM), a démontré une précision raisonnable dans la prédiction des émotions à partir des landmarks extraits. Le processus de formation du modèle, basé sur un ensemble de données d'entraînement comprenant diverses expressions émotionnelles, a permis d'obtenir des résultats encourageants lors des tests.

Cependant, des défis subsistent, notamment la gestion des variations individuelles et culturelles dans l'expression faciale, ainsi que l'optimisation continue des performances du modèle. Des améliorations futures pourraient inclure l'exploration de techniques avancées de prétraitement d'image, l'augmentation de la diversité des données d'entraînement, et l'investigation d'architectures de modèles plus complexes.

VII. Bibliographie

https://paulvangent1.rssing.com/chan-67522377/article7.html

https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer/data?select=train