1. preprocess_data.py

Ce script charge et pré-traite les données du dataset FER2013, qui est fourni sous forme de fichier CSV.

```
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split

def load_fer2013_dataset():
    data = pd.read_csv('data/fer2013.csv')

    # Extraction des données
    pixels = data['pixels'].apply(lambda x: np.array(x.split(),
dtype='float32').reshape(48, 48))
    images = np.stack(pixels.values) / 255.0  # Normalisation des
valeurs des pixels
    labels = data['emotion'].values

    # Division du dataset
    X_train, X_test, y_train, y_test = train_test_split(
        images, labels, test_size=0.2, stratify=labels,
random_state=42
    )

    # Reshape pour compatibilité avec CNN
    X_train = X_train.reshape(-1, 48, 48, 1)
    X_test = X_test.reshape(-1, 48, 48, 1)

    return X_train, X_test, y_train, y_test
```

Explication :

- Le dataset FER2013 contient une colonne `pixels` où chaque ligne représente une image sous forme d'une liste de valeurs de pixels.
- Le code convertit chaque ligne en un tableau NumPy, puis reformate les données en 48x48 pixels (images en niveaux de gris).
- Les valeurs des pixels sont normalisées dans l'intervalle `[0, 1]`.
- Les données sont divisées en ensembles d'entraînement (80%) et de test (20%)

- La forme `(48, 48, 1)` est utilisée pour s'assurer que le modèle CNN puisse traiter les images.

---

2. train_model.py

Ce script définit l'architecture du modèle CNN, entraîne le modèle et sauvegarde la meilleure version.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint
from preprocess_data import load_fer2013_dataset

# Charger les données
X_train, X_test, y_train, y_test = load_fer2013_dataset()

# Définir le modèle
def build_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48,
1)),
        MaxPooling2D((2, 2)),
        Dropout(0.25),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Dropout(0.25),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(7, activation='softmax')  # 7 classes d'émotions
    ])
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

model = build_model()

# Sauvegarde du meilleur modèle
```

```
checkpoint = ModelCheckpoint('data/models/emotion_model.h5',
save_best_only=True, monitor='val_accuracy')

# Entraîner le modèle
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=30,
    batch_size=64,
    callbacks=[checkpoint]
)
```

Explication :

- **Architecture du modèle** :
  - **Conv2D** : Extrait les caractéristiques spatiales des images à l'aide de filtres.
  - **MaxPooling2D** : Réduit la taille des cartes de caractéristiques pour éviter le surapprentissage et diminuer le calcul.
  - **Dropout** : Désactive aléatoirement certains neurones pour prévenir le surapprentissage.
  - **Dense** : Couches entièrement connectées pour classifier les émotions.
- La fonction `ModelCheckpoint` garantit que seul le meilleur modèle (selon la précision de validation) est sauvegardé.
- L'entraînement est réalisé sur 30 époques avec une taille de batch de 64.

---

3. convert_model.py

Ce script convertit le modèle Keras entraîné en un modèle TensorFlow Lite compatible avec un Raspberry Pi.

```
import tensorflow as tf

# Charger le modèle entraîné
model = tf.keras.models.load_model('data/models/emotion_model.h5')

# Convertir en TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Sauvegarder le modèle
with open('data/models/fer2013_model.tflite', 'wb') as f:
```

```
    f.write(tflite_model)
```

Explication :

- `TFLiteConverter` convertit le modèle Keras en un modèle TensorFlow Lite optimisé pour les appareils à faibles ressources, comme le Raspberry Pi.
- Le modèle `.tflite` généré est sauvegardé pour une utilisation future en reconnaissance d'émotions en temps réel.

---

4. camera.py

Gère l'accès à la caméra et la capture d'images en temps réel.

```
import cv2

def capture_frame():
    cap = cv2.VideoCapture(0)  # Utilise le module caméra
    if not cap.isOpened():
        print("Erreur : Caméra non détectée !")
        return None
    return cap
```

Explication :

- Ce script utilise OpenCV pour accéder au flux vidéo de la caméra.
- `cv2.VideoCapture(0)` se connecte à la caméra par défaut (ID caméra `0`).
- Si la caméra n'est pas détectée, un message d'erreur est affiché.

---

5. face_detection.py

Ce module détecte les visages dans les images capturées à l'aide des cascades de Haar.

```
import cv2

def detect_faces(frame):
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    faces = face_cascade.detectMultiScale(gray_frame,
scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
    return faces
```

Explication :

- Les cascades de Haar détectent les visages en analysant les motifs de lumière et d'ombre.
- L'image est convertie en niveaux de gris pour un traitement plus rapide.
- `detectMultiScale` identifie les visages et renvoie leurs coordonnées sous forme de rectangles.

---

6. emotion_model.py

Ce module charge le modèle TensorFlow Lite et effectue la classification des émotions.

```
import numpy as np
import tensorflow as tf

def load_tflite_model():
    interpreter =
tf.lite.Interpreter(model_path='data/models/fer2013_model.tflite')
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    return interpreter, input_details, output_details

def classify_emotion(interpreter, input_details, output_details,
face_image):
    face_image = cv2.resize(face_image, (48, 48)) / 255.0
    face_image = np.expand_dims(face_image,
axis=0).astype(np.float32)

    interpreter.set_tensor(input_details[0]['index'], face_image)
    interpreter.invoke()
    predictions = interpreter.get_tensor(output_details[0]['index'])
    emotions = ['angry', 'happy', 'sad', 'Surprise', 'Neutral',
'feaed', 'Desgusted']
    return emotions[np.argmax(predictions)]
```

Explication :

- **Chargement du modèle** : L'interpréteur TensorFlow Lite est utilisé pour charger le modèle.
- **Prétraitement** : Les visages sont redimensionnés à `(48, 48)` et normalisés.
- **Inférence** : L'interpréteur effectue des prédictions, et l'émotion avec la probabilité la plus élevée est renvoyée.

---

7. emotion_recognition.py

Le pipeline principal intègre le flux vidéo, la détection des visages et la reconnaissance des émotions.

```
from camera import capture_frame
from face_detection import detect_faces
from emotion_model import load_tflite_model, classify_emotion
import cv2

def run_emotion_recognition():
    cap = capture_frame()
    interpreter, input_details, output_details = load_tflite_model()

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        faces = detect_faces(frame)
        for (x, y, w, h) in faces:
            face = frame[y:y+h, x:x+w]
            emotion = classify_emotion(interpreter, input_details,
output_details, face)
            cv2.putText(frame, emotion, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

        cv2.imshow("Reconnaissance d'Émotions", frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
```

```
    cv2.destroyAllWindows()
```

Explication :

- Ce script combine tous les modules précédents pour détecter et afficher les émotions en temps réel.
- Les rectangles sont tracés autour des visages, et l'émotion détectée est affichée au-dessus de chaque visage.

Le fichier `main.py` :

Le fichier `main.py` contiendra les ajustements nécessaires pour gérer les boutons et afficher les résultats de la reconnaissance des émotions. Voici le code de `main.py` modifié pour répondre à vos besoins.

python

Copier le code

```
import RPi.GPIO as GPIO

import time

import cv2

from src.camera import initialize_camera, release_camera

from src.face_detection import detect_faces

from src.emotion_recognition import recognize_emotion


# Définition des émotions (FER2013)

EMOTIONS = ["Angry", "Disgust", "Fear", "Happy", "Sad", "Surprise",
"Neutral"]


# Configuration GPIO pour les boutons

START_STOP_BUTTON_PIN = 17  # Pin GPIO pour le bouton de
démarrage/arrêt
```

```
POWER_BUTTON_PIN = 27       # Pin GPIO pour le bouton d'alimentation


GPIO.setmode(GPIO.BCM)  # Utilisation de la numérotation BCM

GPIO.setup(START_STOP_BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
# Bouton de démarrage/arrêt

GPIO.setup(POWER_BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
# Bouton d'alimentation


def display_emotion_result(frame, emotion_scores, x, y, width=200,
height=20):

    """

    Affiche les résultats des émotions sous forme de pourcentages et
met en surbrillance l'émotion la plus probable.

    """

    max_index = emotion_scores.argmax()  # Trouve l'indice de
l'émotion avec le score le plus élevé

    for i, (emotion, score) in enumerate(zip(EMOTIONS,
emotion_scores)):

        bar_length = int(width * score)  # Calcul de la longueur de
la barre pour chaque émotion

        bar_color = (0, 255, 0) if i == max_index else (255, 0, 0)
# Couleur verte pour l'émotion dominante



        # Positionner la barre pour chaque émotion

        bar_y = y + i * (height + 5)

        cv2.rectangle(frame, (x, bar_y), (x + bar_length, bar_y +
height), bar_color, -1)

        cv2.putText(frame, f"{emotion}: {int(score * 100)}%",
```

```
                    (x, bar_y + height - 2),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)


def main():

    # Initialiser la caméra

    cap = initialize_camera()


    # Déclaration de la variable d'état

    is_running = False

    alert_message = "Appuyez sur le bouton de démarrage/arrêt pour
commencer."


    print(alert_message)


    try:

        while True:

            # Vérification du bouton d'alimentation

            if GPIO.input(POWER_BUTTON_PIN) == GPIO.LOW:  # Bouton
d'alimentation pressé

                print("Bouton d'alimentation pressé. Extinction...")

                break  # Quitter le programme


            # Vérification du bouton de démarrage/arrêt

            if GPIO.input(START_STOP_BUTTON_PIN) == GPIO.LOW:

                if not is_running:
```

```
                    print("Démarrage de la reconnaissance des
émotions...")
                    alert_message = "Reconnaissance des émotions
démarrée."
                    is_running = True
                else:
                    print("Arrêt de la reconnaissance des
émotions...")
                    alert_message = "Reconnaissance des émotions
arrêtée."
                    is_running = False

                time.sleep(1)  # Délai pour éviter la répétition
multiple de la pression

            # Si la reconnaissance des émotions est en cours
            if is_running:
                ret, frame = cap.read()
                if not ret:
                    print("Erreur : Impossible de capturer le flux
vidéo.")
                    break

                # Détection des visages
                faces = detect_faces(frame)
                for (x, y, w, h) in faces:
                    face_image = frame[y:y+h, x:x+w]
```

```
                    # Reconnaître les émotions
                    emotion_scores = recognize_emotion(face_image)

                    # Affichage des résultats des émotions
                    display_emotion_result(frame, emotion_scores, x
+ w + 10, y)

                # Affichage du message d'alerte
                cv2.putText(frame, alert_message, (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

                # Affichage du flux vidéo avec les résultats
                cv2.imshow("Reconnaissance des émotions faciales",
frame)

            else:
                # Si la reconnaissance est arrêtée
                cv2.putText(frame, alert_message, (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
                cv2.imshow("Reconnaissance des émotions faciales",
frame)

            # Délai pour éviter une utilisation excessive du CPU
            time.sleep(0.1)
```

```
    finally:
        # Libérer la caméra et fermer les fenêtres proprement
        release_camera(cap)
        cv2.destroyAllWindows()
        GPIO.cleanup()  # Nettoyage des GPIO

if __name__ == "__main__":
    main()
```

Explication du code :

1. **Initialisation de la caméra** : La caméra est initialisée à partir du fichier `camera.py`. Cela permet de capturer des images en temps réel.
2. **Détection des visages** : La détection des visages est gérée par le fichier `face_detection.py`. Il utilise un détecteur de visages pour localiser les visages dans chaque image capturée.
3. **Reconnaissance des émotions** : La reconnaissance des émotions est effectuée par `emotion_recognition.py`. Ce fichier prend une image de visage, la prétraite, puis l'envoie au modèle de classification pour obtenir les scores de probabilité pour chaque émotion.
4. **Affichage des résultats** : Le programme affiche les résultats sous forme de barres de pourcentages pour chaque émotion, en surlignant l'émotion avec le score le plus élevé.
5. **Gestion des boutons** :
    - **Bouton de démarrage/arrêt** : Quand ce bouton est pressé, il commence ou arrête la reconnaissance des émotions en modifiant l'état de la variable `is_running`.
    - **Bouton d'alimentation** : Quand ce bouton est pressé, le programme s'arrête complètement (la caméra est désactivée et le programme quitte proprement).

Ce qui se passe à chaque itération :

1. Une image est capturée.
2. Les visages sont détectés dans l'image.
3. Les émotions sont classifiées pour chaque visage détecté.
4. Les résultats sont affichés sur l'image.

5. L'image avec les résultats est montrée à l'utilisateur.
6. Le processus se répète.

Le programme traite donc **en continu** une séquence d'images à une fréquence relativement rapide, proche du temps réel (en millisecondes), et affiche les résultats immédiatement après chaque traitement d'image. Ce n'est pas un traitement par intervalles éloignés, mais plutôt un traitement de chaque image dès qu'elle est capturée.

Explication du traitement :

- **Capture d'images en temps réel** : La caméra est en continu et capture des images à chaque itération de la boucle `while True` dans la fonction `main()`. À chaque itération, une nouvelle image est lue à partir de la caméra via la fonction `cap.read()`. Si la lecture de l'image réussit (`ret` est `True`), l'image est ensuite traitée.
- **Détection et reconnaissance** :
    - Les visages sont détectés immédiatement dans chaque image capturée par la fonction `detect_faces()`. Cela est fait pour chaque nouvelle image qui est lue en temps réel.
    - Une fois les visages détectés, chaque visage est extrait et envoyé au modèle pour la reconnaissance des émotions via `recognize_emotion()`. Ce modèle fait une prédiction à chaque image et retourne les scores des émotions.
- **Affichage en temps réel** : Les résultats sont immédiatement affichés sur l'image capturée. Le programme utilise OpenCV pour dessiner les barres de pourcentage des émotions sur chaque image en temps réel, juste après la reconnaissance.
    - Chaque émotion est représentée par une barre horizontale, dont la longueur est proportionnelle au score de cette émotion dans l'image actuelle.
    - L'image avec les résultats est ensuite affichée dans une fenêtre à chaque itération du cycle, donnant ainsi l'impression d'un traitement en temps réel.
- **Fréquence de traitement** : Le programme utilise un `time.sleep(0.1)` pour ralentir légèrement le traitement à chaque itération. Cela réduit la charge CPU et permet de traiter les images à une fréquence raisonnable, mais il traite néanmoins les images à un rythme relativement rapide.

comment deployer ce projet vers une carte Raspberry Pi ?

1. Préparer le Raspberry Pi

- **Installer Raspberry Pi OS** :

- Utilisez l'outil **Raspberry Pi Imager** pour installer Raspberry Pi OS sur la carte SD.

Activer SSH (facultatif, pour une gestion à distance) :

```
sudo raspi-config
# Sélectionnez 'Interfacing Options' -> 'SSH' -> Activer
```

-

Mettre à jour le système :

```
sudo apt update
sudo apt upgrade
sudo apt dist-upgrade
```

-

Installer les dépendances :

```
sudo apt install python3-opencv
sudo apt install python3-pip
sudo pip3 install tensorflow keras numpy imutils
```

-

---

2. Transférer le code depuis le PC vers le Raspberry Pi

Utiliser `scp` pour copier le code :

```
scp -r /chemin/vers/ton/projet
pi@<adresse_ip_du_raspberry_pi>:/home/pi/
```

-

Utiliser `rsync` pour synchroniser les fichiers :

```
rsync -avz /chemin/vers/ton/projet
pi@<adresse_ip_du_raspberry_pi>:/home/pi/
```

-
- **Utiliser SFTP** (via FileZilla) pour transférer les fichiers.

---

3. Configurer la caméra sur le Raspberry Pi

Activer la caméra (si vous utilisez une caméra RPi) :

```
sudo raspi-config
```

```
# Sélectionnez 'Interfacing Options' -> 'Camera' -> Activer
```

-

Tester la caméra pour vérifier qu'elle fonctionne :

```
raspistill -o test_image.jpg
```

-

Accéder à la caméra avec OpenCV dans le script Python :

```
import cv2
cap = cv2.VideoCapture(0)  # Utiliser 0 pour la caméra intégrée ou 1
pour la caméra USB
```

-

---

4. Adapter le code pour le Raspberry Pi

Capture d'image depuis la caméra :

```
ret, frame = cap.read()
if not ret:
    print("Impossible de capturer l'image.")
```

-

Exécuter votre script Python pour la reconnaissance des émotions :

```
import cv2
# Ajouter le code de reconnaissance des émotions ici
while True:
    ret, frame = cap.read()
    if not ret:
        break
    # Traitement de l'image et reconnaissance des émotions
```

-

---

5. Exécuter le programme sur le Raspberry Pi

- **Lancer le script Python** :

Allez dans le répertoire du projet :

```
cd /home/pi/mon_projet
```

-

Exécutez votre script :
```
python3 main.py
```

-
- Assurez-vous que la caméra et la reconnaissance fonctionnent correctement en temps réel.

---

6. Déboguer et ajuster

- **Vérifier les erreurs dans le terminal** si le programme ne fonctionne pas comme prévu.
- **Vérifier les logs de la caméra** et l'accès aux périphériques.
- **Tester les boutons GPIO** et ajuster la configuration si nécessaire.