Rapport
Fezoui Yacine
Introduction
Projet d'identification par l'iris de l'œil. La réalisation de ce projet a été faite avec la bibliothèque Opencv qui est spécialisée dans le traitement d'image, utile donc pour la comparaison des matches entre une photo insérée et la base de données. Pour l'interface, j'ai choisi le framework Flask. Nous avons une base de données qui contient 3 photos de chaque œil de 64 personnes, donc 384 images.

Environnement
pip install django opencv-python numpy scipy Pillow Or execute install_env.bat
py manage.py makemigrations && py manage.py migrate Or exécuter migration.bat pour effectuer les migration de la Base de donnée

Réalisation
Outre les lignes de codes qui concerne la communication entre l'interface et la partie traitement, plusieur points sont à retenir de cette dernière

1. Réception de l'image : L'utilisateur valide le formulaire de la page d'accueil. On reçoit ensuite le fichier. Après vérification du formulaire, on supprime la photo de l'exécution précédente, puis on sauvegarde la nouvelle image dans la base de données.

```
if request.method == "POST":
    form = OeilForm(request.POST, request.FILES)
    if form.is_valid():
        file_path = os.path.join(settings.MEDIA_ROOT, 'images/probleme.png')

        # Vérifier si le fichier existe avant de le supprimer
        if os.path.exists(file_path):
            os.remove(file_path)
        image = form.save(commit=False)
        image.image.name = 'probleme.png'
        image.save()
```

2. Augmenter le contraste global de l'image : on lit l'image en niveau de gris, puis augmenter le contraste, pour rendre les détailles plus lisible et les points clés facile à détecter

```
probleme_c = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)# En gris
probleme_c = cv2.equalizeHist(probleme_c) # Contraste
```

3. Cette fonction ensuite vas lire l'image introduite et aussi chaque image de la bdd, pour extraire les keypoints, la Boucle a servie a prendre uniquement les keypoints qui se situe suffisamment loins du centre (383,287) ce qui représente la pupille, et pas assez pour ne pas dépasser l'iris, les valeurs choisit (250 et 70) ont été sélectionnés après plusieurs testes.

```
def extract_sift_features(image):
    sift = cv2.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(image, None)
    K=[]
    D=[]
    for i, j in zip(keypoints, descriptors):
        x, y = i.pt
        distance = euclidean((x, y), (383, 287))
        if distance < 250 and distance > 70 :
            K.append(i)
            D.append(j)
    return K, D
```

4. La boucle ci-dessous, permet de lire chaque photo de la BDD, et d'appliquer les même traitement que la photo insérée. ensuite a chaque itération, nous gardons celle avec le plus de matchs (Explication en 5).

```python
for database in database_images:
    # Traitement de chauqe image du database
    image_c2 = cv2.imread(database, cv2.IMREAD_GRAYSCALE)
    database_c = cv2.equalizeHist(image_c2)
    database_k, database_d = extract_sift_features(database_c)
    # Matche
    matches = match_features(probleme_d, database_d)
    print(database,len(matches))
    # On prend celle qui a le plus de matches
    if len(matches) > max_matches:
        match_sol, max_matches = matches, len(matches)
        solution, solution_c, solution_k = database, database_c, database_k
```

5. Cette fonction est appelée à chaque itération, elle prend deux images avec leurs keypoints, elle renvoie ensuite ce qui est considéré comme étant des good matches. En prenant les deux meilleurs matchs pour chaque keypoint de l'image insérée, on calcule la différence entre les deux distances. Un vrai good matches sera unique, donc il ne peut pas y avoir 2 bon matches, alors le rapport entre les 2 est forcément plus petit que 0,76, cette valeur est inspirée des nombreux travaux cités en bibliographie.

```python
def match_features(probleme_d, database_d, ration_distance=0.76):
    bf = cv2.BFMatcher()
    probleme = np.array(probleme_d)
    database = np.array(database_d)
    matches = bf.knnMatch(probleme, database, k=2)
    good_matches = []
    for m, n in matches:
        if (m.distance/n.distance) < ration_distance:
            good_matches.append(m)
    return good_matches
```

Sources

La partie du code ou nous calculons le rapport entre les 2 matches est inspiré de plusieurs projet trouvés sur le net, plus exactement en cherchant des mot clé lié au sujet du projet sur github et youtube, même chose pour l'utilisation de cv2.equalizeHist.

Guide

En exécutant la commande py manage.py runserver, le lien vers le site web sera affiché dans le terminal: http://127.0.0.1:5000 ou http://127.0.0.1:8000, ou en exécutant le fichier run.bat. 2 scénarios peuvent être testés dans notre site web, la réussite et l'échec :

La réussite :
Vous devez extraire une image de la base de données, puis l'introduire dans la page d'accueille, pour que le code puisse détecter sa ressemblance avec les 2 autres photos du même oeil

L'échec :
Insérer n'importe quelle photo, d'un oeil non présent dans la base de donné, voir même une image qui n'as rien avoir, au dessous d'un certain seuil de match, il n'y a pas assez pour une identification

Conclusion

Ce projet m'a permis de prendre connaissance de différentes techniques d'identification des individus, l'iris étant plus précis que l'identification par l'index. La recherche bibliographique m'a permit ensuite d'avoir une idée sur la conception du code python et des différentes fonctionnalités qu'offre la bibliothèque opencv. et enfin, la réalisation d'une interface web a permis de rendre l'identification simple à utiliser.

Références web

Iris Recognition Based on SIFT Features : https://arxiv.org/pdf/2111.00176

Efficient Iris Recognition Based on Optimal Subfeature Selection and Weighted Subregion Fusion : https://www.hindawi.com/journals/tswj/2014/157173/

Enhancing Computer Vision with SIFT Feature Extraction in OpenCV and Python

FLANN Feature Matching Example with OpenCV

andreibercu / iris-recognition : https://github.com/andreibercu/iris-recognition

Django - cours ✅ - YouTube