esprit
Ecole Supérieure Privée
d'Ingénierie et de Technologies

RECONNAISSANCE FACIALE

RAPPORT STAGE D'ÉTÉ 14

réalisé par :
Belkis Baccar
Walid Tayeche

CONTENU

Liste des figures

Répartition des tâches

Introduction

01.Compréhension des données

02.Préparation des donneés

03.Détection facial

04.Reconnaissance faciale

Conclusion

03

RÉPARTITION DES TÂCHES

Ce travail a été réalisé grâce à la collaboration de deux étudiants Belkis Baccar et Walid Tayeche. Le projet a été divisé en deux parties.
La première partie qui a compris la préparation des données et détection a été accomplit dans un environnement d'équipe et de recherche.
La deuxième partie a été entamé par chaque membre de l'équipe. Belkis Baccar a travaillé sur les siamese networks et Walid Taycehce a travaillé sur le modèle facenet

LISTE DES FIGURES

Figure 1.2: Répartition d'image par personne
Figure 1.1: Nombre de personnes en fonction du nombre d'images
Figure 1.3: Répartition des photos pour les personnes avec 2 à 5 photos
Figure 1.4: Pourcentage des photos pour les personnes avec 2 à 5 photos
Figure 2.1: Factionnement des données
Figure 2.1: Création des dossiers train et test
Figure 2.2: Arborescence du dossier des données
Figure 2.3: Création des dossiers train et test
Figure 2.4: Arborescence du dossier 0
Figure 2.5: Fonction de création des listes de comparaisons intra classe
Figure 2.6: Fonction de création des listes de comparaisons inter classe
Figure 2.7: Création des listes de comparaison
Figure 2.8: Equilibre des listes
Figure 2.9: Enregistrement des listes sous format texte
Figure 2.10: Téléchargement des listes en dataframes
Figure 3.1: Détection facial avec CNN et dlib
Figure 3.2: Détection facial avec CNN et dlib
Figure 3.3: Détection facial avec OpenCV
Figure 3.4: Reconstruction des listes avec les dimensions
Figure 3.5: Reconstruction d'une partie des listes avec les dimensions
Figure 3.6: Reconstruction des listes avec les dimensions
Figure 3.7: Retranchement des visages avec MTCNN
Figure 3.8: Retranchement des visages du trainset avec MTCNN
Figure 3.9: Retranchement des visages du testset avec MTCNN
Figure 4.1: modèle siamese
Figure 4.2: Plot du modèle siamese
Figure 4.3: Sommaire du modèle siamese

LISTE DES FIGURES

Figure 4.4: Epoch 1 du modèle siamese
Figure 4.5: Plot du train_loss et accurance_list
Figure 4.6: Répartition des train et test set
Figure 4.7: Répartition des train et test set
Figure 4.8: Epoch 10 du modèle 2 siamese
Figure 4.9: Plot du train_loss et accurance_list
Figure 4.10: Fonction pour extraire des embeddings
Figure 4.11: Extraction des embeddings
Figure 4.12: Fonction pour calculer les embeddings
Figure 4.13: Distance euclidienne des trainset
Figure 4.14: Distance cosine des trainset
Figure 4.15: Groupement des images par les méthode TSNE
Figure 4.16: Groupement des images par les méthode PCA
Figure 4.17: Recherche des meilleurs paramètres
Figure 4.18: score de présicion

06

INTRODUCTION GENERALE

La technologie de reconnaissance faciale est un système utilisé pour détecter la présence d'une personne en comparant une image numérique ou une vidéo du visage d'une personne à des données préexistantes. La reconnaissance faciale et la comparaison faciale peuvent être utilisées pour vérifier l'identité d'une personne en enregistrant et en analysant une image ou une vidéo de la structure faciale d'une personne et en la comparant à une image préexistante pour déterminer s'il existe une correspondance. L'image préexistante peut provenir d'une base de données privée ou publique ou de l'image d'une pièce d'identité émise par le gouvernement.

La technologie de reconnaissance faciale est en développement depuis des décennies, mais les progrès de ces dernières années ont rendu ces solutions courantes dans notre vie quotidienne. Des outils de reconnaissance faciale sont désormais disponibles sur les smartphones et les institutions financières commencent à adopter la comparaison faciale pour la vérification d'identité numérique dans le cadre de l'ouverture de compte numérique.

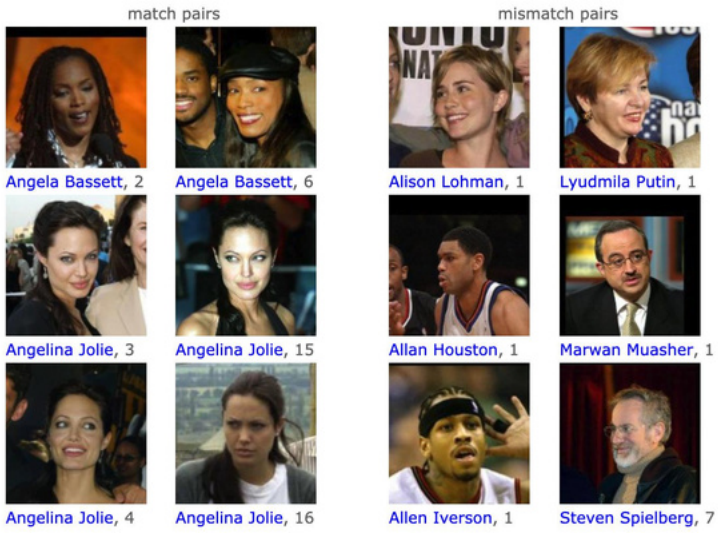
La technologie de reconnaissance faciale se révèle utile dans une variété de cas d'utilisation pour une variété d'organisations. Parmis ces domaines on cite quelques-uns dont la reconnaissance faciale est utilisée pour une vérification d'identité efficace et efficiente aujourd'hui: sécurité de l'appareil, mesures de prévention antivol, achat d'alcool, sécurité scolaire, sécurité aéroportuaire, forces de l'ordre ...

01.COMPRÉHENSION DES DONNÉES

"Labeled Faces in the Wild " ou en abréviation LFW, est une base de données de photographies de visage désigné pour l'étude du problème de reconnaissance de visage non contraint. Le dataset comprend plus de 13000 images de visages collectés depuis le web. Chaque visage est labellisé avec le nom de la personne en photo. 1680 de ces personnes ont un nombre de photos supérieur a 2 Ces visages ont été détectés par the Viola-Jones face detector.

lien vers la base de données :

https://arxiv.org/abs/1901.05903



01.COMPRÉHENSION DES DONNÉES

Afin d'analyser la répartition des images par personnes dans la base de données, on visualise quelques tableau, barplot et piechart. On remarque la distribution non équilibrée du dataset.

1- On élimine les personnes ayant une seule photo puisqu'on n'a pas d'autres supports pour faire les comparaisons.

2- On élimine les personnes ayant un nombre de photos supérieur à cinq vu qu'on remarque la grande variation dans le nombres de photos qui emmènera a un déséquilibre dans l'entrainement de nos modèles dans des futurs étapes

photo	
2	779
3	291
4	187
5	112
6	55
7	39
8	33
9	26
10	15
11	16
12	10
13	11
14	10
15	11
16	3
17	8
18	5
19	7
20	5
21	4
22	5
23	3
24	3
25	1
26	2
27	1
28	2
29	2
30	2
31	2
32	3
33	3
35	1
36	1
37	1
39	2
41	2
42	2
44	1
48	1
49	1
50	12

Figure 1.1: Nombre de personnes en fonction du nombre d'images

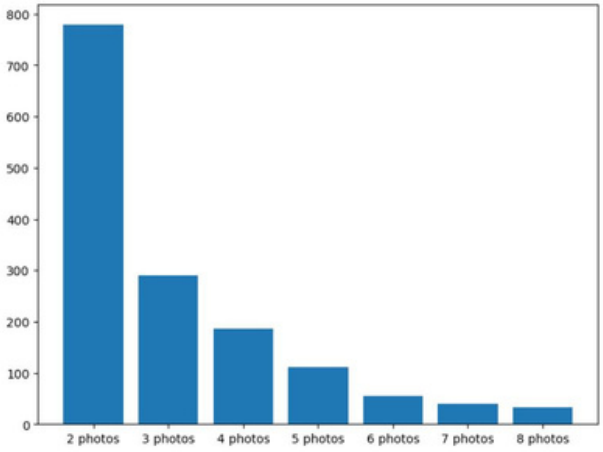


Figure 1.2: Répartition d'image par personne

01.COMPRÉHENSION DES DONNÉES

La répartition des images par personne se distribue comme suivant:

- 56.9% : 779 personnes ayant 2 photos
- 23.3% : 291 personnes ayant 3 photos
- 13.7% : 187 personnes ayant 4 photos
- 8.2% : 112 personnes ayant 5 photos

Le dataset retenu est composé de 1369 personnes.

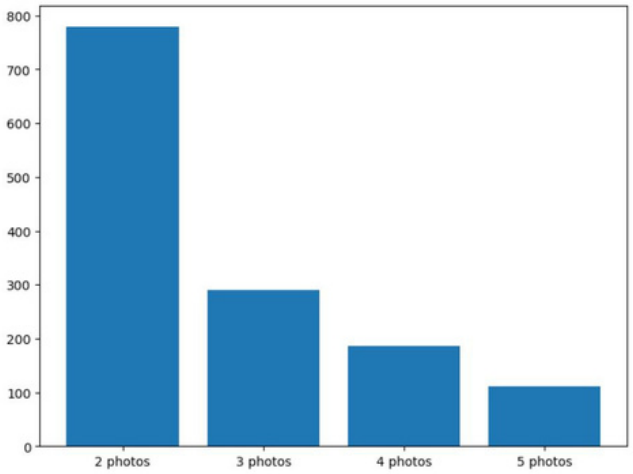


Figure 1.3: Répartition des photos pour les personnes avec 2 à 5 photos

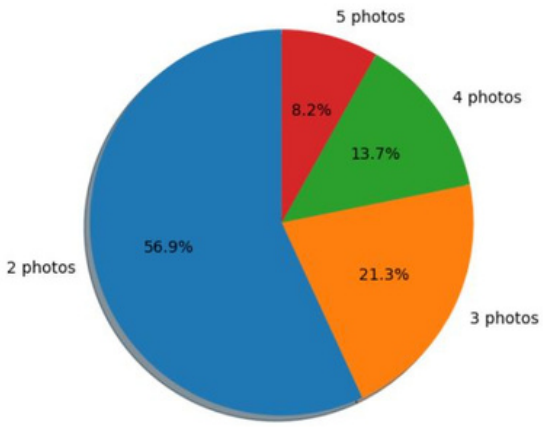


Figure 1.4: Pourcentage des photos pour les personnes avec 2 à 5 photos

02.PRÉPARATION DES DONNEÉS

1-Fractionnement des données:

Le dataset a été diviser en 70% trainset et 30% testset. Le trainset comporte 958 personnes avec leurs images et le testset comporte 411 personnes.

```
from sklearn.model_selection import train_test_split

training_data, testing_data = train_test_split(df_filtered, test_size=0.3, random_state=25)

print(f"No. of training examples: {training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")
```

```
No. of training examples: 958
No. of testing examples: 411
```

Figure 2.1: Factionnement des données

2- Les listes de comparaisons:

Afin d'unifier les comparaisons dans les futurs étapes, on a créé des listes de comparaisons. Ces listes comportent les comparaisons intra classes; c'est à dire au sein de la même personne, ou inter classes; c'est à dire entre deux personnes différentes. La composition regroupe le numéro d'identité de la première personne, le numéro de son image, le numéro d'identité de la deuxième personne et le numéro de son image.

02.PRÉPARATION DES DONNEÉS

Le dataset est composé de plusieurs sous-dossiers surnommés avec le nom de chaque personnes dont on trouve leurs images. Pour simplifier les taches, on associe une id a chaque personne comme nom de dossier et id a chaque une de leurs images.

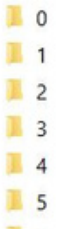


Figure 2.2: Arborescence du dossier des données

Figure 2.3: Arborescence du dossier 0

Le dossier comprenenat les sous-dossier a été divisé en deux noveaux dossiers train et test chaqun avec la listes des personnes du trainset et testset.

```
[21]: for directory in os.listdir(flw_trim):
          path_src=os.path.join(flw_trim, directory)
          if directory in train_name:
              path_dest=os.path.join(flw_train, directory)
              #print(path_dest)
              destination = shutil.copytree(path_src, path_dest)
          else:
              path_dest=os.path.join(flw_test, directory)
              #print(path_dest)
              destination = shutil.copytree(path_src, path_dest)
```

Figure 2.4: Création des dossiers train et test

12

02.PRÉPARATION DES DONNEÉS

Le choix des images dans les listes de comparaisons intra classes train et inter classes test ont été attribuées pour avoir toutes les compositions possibles entres les images de la même personne.

```
def create_pairs_intra(ROOT):
    num_folders = len(os.listdir(ROOT))
    similar_data_id1 = []
    similar_data_id2 = []
    similar_data_img1 = []
    similar_data_img2 = []

    for i in os.listdir(ROOT):
        files=os.listdir(os.path.join(ROOT, i))

        # Creating pairs of similar faces 1
        for pair in list(combinations(files, 2)):
            #similar_data.append(((str(i), pair[0]), (str(i), pair[1]), 1))
            similar_data_id1.append(str(i))
            similar_data_img1.append(pair[0])
            similar_data_id2.append(str(i))
            similar_data_img2.append(pair[1])
        dicti1 = {'id1': similar_data_id1, 'img1': similar_data_img1, 'id2': similar_data_id2, 'img2': similar_data_img2}
        dfsim = pd.DataFrame(dicti1)

    return dfsim
```

Figure 2.5: Fonction de création des listes de comparaisons intra classe

Le choix des images dans les listes de comparaisons inter classes train et inter classes test ont été atribuées pour avoir toutes les compositions possibles entres les images de deux personnes differentes. Aprés, on tranche un échantillon aléatoire avec la meme longeur des listes intra classes pour avoir une comparaison équilibré.

13

02.PRÉPARATION DES DONNEÉS

```python
def create_pairs_inter(ROOT):
    num_folders = len(os.listdir(ROOT))
    dissimilar_data_id1 = []
    dissimilar_data_id2 = []
    dissimilar_data_img1 = []
    dissimilar_data_img2 = []
    ids=[]
    for i in os.listdir(ROOT):
        ids.append(int(i))
    #print(ids)
    for i in os.listdir(ROOT):
        files=os.listdir(os.path.join(ROOT, i))
        #print(i)

        # Creating pairs of different faces 0
        for j in range(int(i), min(int(i)+80, num_folders)):
            #print(j)
            if j in ids:
                #print(j)
                #dissimilar_data.append(((str(i), "0.jpg"), (str(j), "0.jpg"), 0))
                dissimilar_data_id1.append(i)
                dissimilar_data_img1.append("0.jpg")
                dissimilar_data_id2.append(str(j))
                dissimilar_data_img2.append("0.jpg")
        dicti2 = {'id1': dissimilar_data_id1, 'img1': dissimilar_data_img1, 'id2': dissimilar_data_id2, 'img2': dissimilar_data_img2}
        dfdsim = pd.DataFrame(dicti2)
    return dfdsim
```

Figure 2.6: Fonction de création des listes de comparaisons inter classe

```python
intra_train=create_pairs_intra(flw_train)
inter_train=create_pairs_inter(flw_train)
intra_test=create_pairs_intra(flw_test)
inter_test=create_pairs_inter(flw_test)
```

Figure 2.7: Création des listes de comparaison

```python
inter_train_trim=inter_train.sample(len(intra_train))
inter_test_trim=inter_test.sample(len(intra_test))
```

Figure 2.8: Equilibre des listes

14

02.PRÉPARATION DES DONNEÉS

```
np.savetxt(r'C:/Users/Bolbol/Desktop/stage/lfw/intraclasse_train.txt',intra_train, fmt='%s')
np.savetxt(r'C:/Users/Bolbol/Desktop/stage/lfw/interclasse_train.txt',inter_train_trim, fmt='%s')
np.savetxt(r'C:/Users/Bolbol/Desktop/stage/lfw/intraclasse_test.txt',intra_test, fmt='%s')
np.savetxt(r'C:/Users/Bolbol/Desktop/stage/lfw/interclasse_test.txt',inter_test_trim, fmt='%s')
```

Figure 2.9: Enregistrement des listes sous format texte

```
intra_train = pd.read_csv('C:/Users/Bolbol/Desktop/stage/lfw/intraclasse_train.txt', sep=" ", header=None)
intra_train.columns = ["id1", "img1", "id2", "img2"]
inter_train = pd.read_csv('C:/Users/Bolbol/Desktop/stage/lfw/interclasse_train.txt', sep=" ", header=None)
inter_train.columns = ["id1", "img1", "id2", "img2"]
intra_test = pd.read_csv('C:/Users/Bolbol/Desktop/stage/lfw/intraclasse_test.txt', sep=" ", header=None)
intra_test.columns = ["id1", "img1", "id2", "img2"]
inter_test = pd.read_csv('C:/Users/Bolbol/Desktop/stage/lfw/interclasse_test.txt', sep=" ", header=None)
inter_test.columns = ["id1", "img1", "id2", "img2"]
```

Figure 2.10: Téléchargement des listes en dataframes

15

03.DÉTECTION FACIAL

La détection facial comprend l'identification des dimensions sur les axes (x,y) ,w la largeur et h la hauteur des visages pour chaque photo par personne. Ca résulte en 4 dimensions (x, y, w, h) par photo.
Dans nos travaux, on a utilisé les méthodes OpenCV avec les cascades Haar, CNN et dlib pour la détection des visages et pour le retranchement des visages, on a utilisé MTCNN.

1-*Dlib et CNN:*

Selon la page github de dlib, dlib est une boîte à outils permettant de créer des applications d'apprentissage automatique et d'analyse de données dans le monde réel en C++. Bien que la bibliothèque soit écrite à l'origine en C++, elle possède de bonnes liaisons Python faciles à utiliser. Nous utiliserons un modèle CNN pour la détection des visages. C'est un modèle pré-entraîné que nous allons charger lors de l'exécution de nos scripts Python.

L'un des principaux avantages de ce détecteur est qu'il peut utiliser la puissance de calcul de nos GPU, s'il en existe un. Cela peut rendre le pipeline de détection un peu plus rapide et alléger la charge du processeur qui peut alors se concentrer sur d'autres tâches.

16

03.DÉTECTION FACIAL

```python
# Load trained model
cnn_face_detector = dlib.cnn_face_detection_model_v1(r'C:\Users\Bolbol\Desktop\stage\lfw\mmod_hu

# Function to detect and show faces in images
def detect_face_dlib(img_path, ax):
    # Read image and run algorithm
    img = io.imread(img_path)
    dets = cnn_face_detector(img, 1)
    # If there were faces detected, show them
    if len(dets) > 0:
        for d in dets:
            rect = patches.Rectangle(
                (d.rect.left(), d.rect.top()),
                d.rect.width(),
                d.rect.height(),
                fill=False,
                color='b',
                lw='2')
            ax.add_patch(rect)
            ax.imshow(img)
            ax.set_title(os.path.split(os.path.split(img_path)[0])[1])
            print(d.rect.left(), d.rect.top(),d.rect.width(), d.rect.height())

# Path to images
images = list(Path(r'C:\Users\Bolbol\Desktop\stage\lfw\Face Dataset Train\0').glob('*.jpg'))

# Show results
fig = plt.figure(figsize=(15, 5))
for i, img in enumerate(images):
    ax = fig.add_subplot(1, len(images), i+1)
    detect_face_dlib(img, ax)
```

Figure 3.1: Détection facial avec CNN et dlib

03.DÉTECTION FACIAL

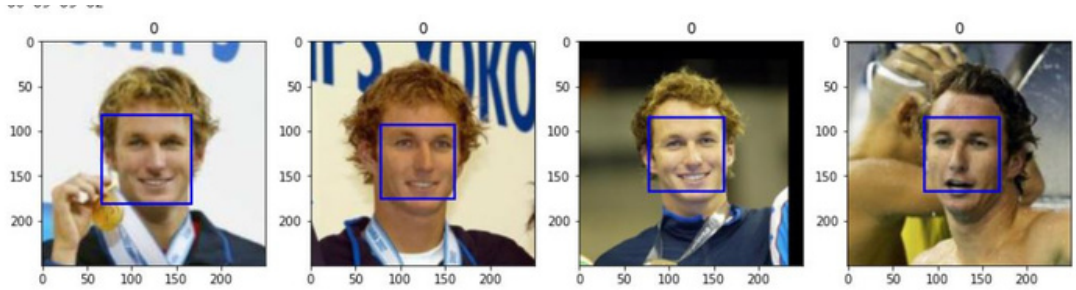


Figure 3.2: Détection facial avec CNN et dlib

2-OpenCV et Haar:

La détection de visage à l'aide de cascades Haar est une approche basée sur l'apprentissage automatique dans laquelle une fonction en cascade est entraînée avec un ensemble de données d'entrée. OpenCV contient déjà de nombreux classificateurs pré-entraînés pour le visage, les yeux, les sourires, etc.

```
for directory in os.listdir(train_path):
    #print(directory)
    path=os.path.join(train_path, directory)
    for filename in os.listdir(path):
        image=os.path.join(path, filename)
        # Load the cascade
        face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
        # Read the input image
        img = cv2.imread(image)
        # Convert into grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # Detect faces
        faces=face_cascade.detectMultiScale(gray, 1.1, 4)
        # Draw rectangle around the faces
        for (x, y, w, h) in faces:
            cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
            #dimensions.append((x,y,w,h))
```

Figure 3.3: Détection facial avec OpenCV

03.DÉTECTION FACIAL

Après la détection, on a intégré les dimensions facial dans les listes de comparaisons. La nouvelle formule comprend alors les ids de deux images, les numéros de deux images et les dimensions (x, y, w, h) pour chaque images

```
b = pd.merge( intra_train,df, left_on=["id1","img1"], right_on=["ID","IMG"], how='left')
b.drop('id2', inplace=True, axis=1)
b.drop('img2', inplace=True, axis=1)
b.drop('ID', inplace=True, axis=1)
b.drop('IMG', inplace=True, axis=1)
b = b.rename(columns={'X': 'x1'}, index={'ONE': 'Row_1'})
b = b.rename(columns={'Y': 'y1'}, index={'ONE': 'Row_1'})
b = b.rename(columns={'W': 'w1'}, index={'ONE': 'Row_1'})
b = b.rename(columns={'H': 'h1'}, index={'ONE': 'Row_1'})
```

Figure 3.4: Reconstruction des listes avec les dimensions

	id1	img1	x1	y1	w1	h1
0	0	0.jpg	69.0	68.0	114.0	114.0
1	0	0.jpg	69.0	68.0	114.0	114.0
2	0	0.jpg	69.0	68.0	114.0	114.0
3	0	1.jpg	69.0	68.0	115.0	115.0
4	0	1.jpg	69.0	68.0	115.0	115.0
...	...	...	...	...	...	...
2773	997	1.jpg	69.0	67.0	114.0	114.0
2774	998	0.jpg	69.0	67.0	114.0	114.0
2775	999	0.jpg	75.0	71.0	107.0	107.0
2776	999	0.jpg	75.0	71.0	107.0	107.0
2777	999	1.jpg	78.0	122.0	100.0	100.0

2778 rows × 6 columns

Figure 3.5: Reconstruction d'une partie des listes avec les dimensions

03.DÉTECTION FACIAL

	id1	img1	x1	y1	w1	h1	id2	img2	x1	y1	w1	h1
0	29	0.jpg	67.0	65.0	117.0	117.0	58	0.jpg	69.0	68.0	110.0	110.0
1	361	0.jpg	70.0	69.0	113.0	113.0	365	0.jpg	69.0	67.0	113.0	113.0
2	527	0.jpg	63.0	67.0	118.0	118.0	554	0.jpg	73.0	69.0	110.0	110.0
3	481	0.jpg	65.0	65.0	120.0	120.0	514	0.jpg	55.0	60.0	134.0	134.0
4	43	0.jpg	68.0	68.0	112.0	112.0	94	0.jpg	69.0	68.0	114.0	114.0
...	...	...	...	...	...	...	...	...	...	...	...	...
2773	745	0.jpg	72.0	70.0	113.0	113.0	778	0.jpg	64.0	66.0	121.0	121.0
2774	279	0.jpg	68.0	68.0	114.0	114.0	310	0.jpg	69.0	70.0	111.0	111.0
2775	706	0.jpg	70.0	70.0	111.0	111.0	756	0.jpg	68.0	68.0	115.0	115.0
2776	463	0.jpg	70.0	68.0	112.0	112.0	527	0.jpg	63.0	67.0	118.0	118.0
2777	563	0.jpg	70.0	70.0	110.0	110.0	638	0.jpg	68.0	69.0	116.0	116.0

2778 rows × 12 columns

Figure 3.6: Reconstruction des listes avec les dimensions

3- MTCNN

MTCNN est une bibliothèque python (pip) écrite par l'utilisateur de Github ipacz, qui implémente l'article Zhang, Kaipeng et al. « Détection et alignement conjoints des visages à l'aide de réseaux convolutifs multitâches en cascade. »
Dans cet article, ils proposent un cadre multitâche en cascade profond utilisant différentes fonctionnalités de «sous-modèles» pour chacun augmenter leurs forces de corrélation.

20

03.DÉTECTION FACIAL

```python
def crop_face_and_save(path, new_path=None, model=MTCNN, transformer=None, params=None):
    """
    Detect face on each image, crop them and save to "new_path"
    :param str path: path with images will be passed to  datasets.ImageFolder
    :param str new_path: path to locate new "aligned" images, if new_path is None
                    then new_path will be path + "_cropped"
    :param model: model to detect faces, default MTCNN
    :param transformer: transformer object will be passed to ImageFolder
    :param params: parameters of MTCNN model
    """
    if not new_path:
        new_path = path + '_cropped'

    # in case new_path exists MTCNN model will raise error
    if os.path.exists(new_path):
        shutil.rmtree(new_path)

    # it is default parameters for MTCNN
    if not params:
        params = {
            'image_size': 160, 'margin': 0,
            'min_face_size': 10, 'thresholds': [0.6, 0.7, 0.7],
            'factor': 0.709, 'post_process': False, 'device': device
            }

    model = model(**params)

    if not transformer:
        transformer = transforms.Lambda(
            lambd=lambda x: x.resize((1280, 1280)) if (np.array(x) > 2000).all() else x
        )
    # for convenience we will use ImageFolder instead of getting Image objects by file paths
    dataset = datasets.ImageFolder(path, transform=transformer)
    dataset.samples = [(p, p.replace(path, new_path)) for p, _ in dataset.samples]

    # batch size 1 as long as we havent exact image size and MTCNN will raise an error
    loader = DataLoader(dataset, batch_size=1, collate_fn=training.collate_pil)
    for i, (x, y) in enumerate(tqdm.tqdm(loader)):
        model(x, save_path=y)

    # spare some memory
    del model, loader, dataset
```

Figure 3.7: Retranchement des visages avec MTCNN

03.DÉTECTION FACIAL

Figure 3.8: Retranchement des visages du trainset avec MTCNN

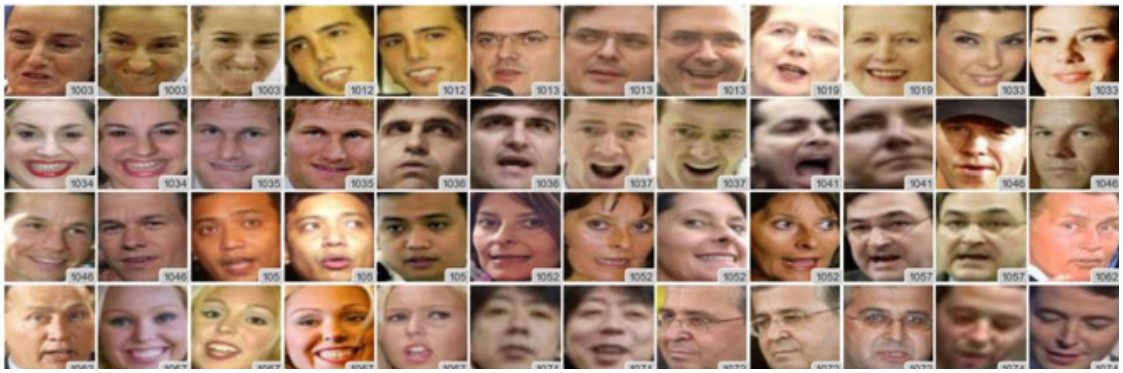


Figure 3.9: Retranchement des visages du testset avec MTCNN

04.RECONNAISSANCE FACIAL

La reconnaissance faciale comprend l'identification positive des personnes selon des modèles entrainé sur le trainset et testé sur le testset. Dans un premier lieu on a travaillé sur les siamese networks avec les listes de comparaisons prédéfinies dans l'étape de préparation des données. Le score de précision est égale à 0.65

1-Siamese Networks:

Dans l'ère moderne de l'apprentissage profond, les réseaux de neurones sont presque excellents dans toutes les tâches, par contre ces réseaux de neurones s'appuient sur plus de données pour bien fonctionner. Mais, pour certains problèmes comme la reconnaissance faciale, nous ne pouvons pas toujours compter sur l'obtention de plus de données, pour résoudre ce genre de tâches, nous avons un nouveau type d'architecture de réseau neuronal appelé Siamese Networks.

Un réseau de neurones siamese est une classe d'architectures de réseaux de neurones qui contiennent au moins deux sous-réseaux identiques. « identique » signifie ici qu'ils ont la même configuration avec les mêmes paramètres et poids. La mise à jour des paramètres est reflétée sur les deux sous-réseaux. Il est utilisé pour trouver la similitude des entrées en comparant ses vecteurs de caractéristiques, de sorte que ces réseaux sont utilisés dans de nombreuses applications

23

04.RECONNAISSANCE FACIAL

```python
def get_siamese_model(input_shape):
    # Define the tensors for the two input images
    left_input  = Input(input_shape, name="input1")
    right_input = Input(input_shape, name="input2")

    # Convolutional Neural Network
    base_model = tf.keras.applications.Xception(
        input_shape=input_shape,
        weights='imagenet',
        include_top=False,
        pooling='max',
    )

    for i in range(len(base_model.layers)-7):
        base_model.layers[i].trainable = False

    model = Sequential([
        base_model,
        Flatten(),
        Dense(2048, activation='sigmoid')
    ])
     # Generate the encodings (feature vectors) for the two images
    encoded_l = model(left_input)
    encoded_r = model(right_input)

    # Add a Subtract layer to compute the absolute difference between the encodings
    L1_layer = Lambda(lambda tensors: backend.abs(tensors[0] - tensors[1]), name='Distance')
    L1_distance = L1_layer([encoded_l, encoded_r])

    # Add a dense layer with a sigmoid unit to generate the similarity score
    prediction = Dense(1, activation='sigmoid', name='Prediction')(L1_distance)

    # Connect the inputs with the outputs
    siamese_net = Model(inputs=[left_input, right_input], outputs=prediction)

    return siamese_net
```

Figure 4.1: Modèle siamese

04.RECONNAISSANCE FACIAL

```
siamese_model = get_siamese_model((128,128,3))
plot_model(siamese_model, show_shapes=True, show_layer_names=True)
```

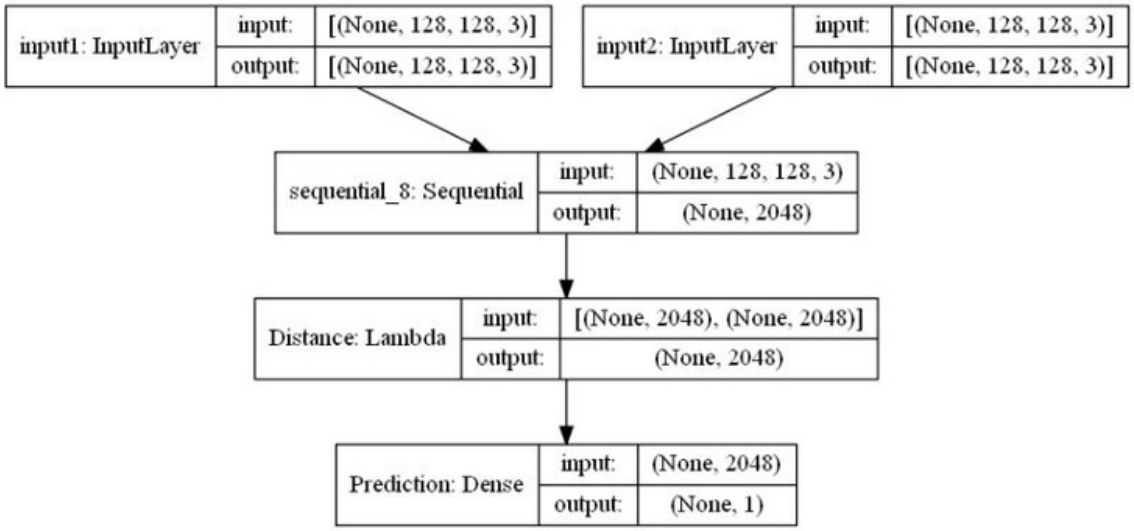


Figure 4.2: Plot du modèle siamese

```
optimizer = Adam(learning_rate=0.01)
siamese_model.compile(loss='binary_crossentropy', optimizer=optimizer)
siamese_model.summary()
```

```
Model: "model_8"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input1 (InputLayer)             [(None, 128, 128, 3) 0
__________________________________________________________________________________________________
input2 (InputLayer)             [(None, 128, 128, 3) 0
__________________________________________________________________________________________________
sequential_8 (Sequential)       (None, 2048)         25057832    input1[0][0]
                                                                 input2[0][0]
__________________________________________________________________________________________________
Distance (Lambda)               (None, 2048)         0           sequential_8[0][0]
                                                                 sequential_8[1][0]
__________________________________________________________________________________________________
Prediction (Dense)              (None, 1)            2049        Distance[0][0]
==================================================================================================
Total params: 25,059,881
Trainable params: 8,947,201
Non-trainable params: 16,112,680
__________________________________________________________________________________________________
```

Figure 4.3: Sommaire du modèle siamese

04.RECONNAISSANCE FACIAL

```
-------------------------------  START OF EPOCH - 1  -------------------------------
Batch 10        | Done in 79.08 secs    | Loss = 0.8613788962364197
Batch 20        | Done in 78.90 secs    | Loss = 0.6797506034374237
Batch 30        | Done in 77.71 secs    | Loss = 0.661037003993988
Batch 40        | Done in 70.82 secs    | Loss = 0.6589286804199219
Batch 50        | Done in 75.11 secs    | Loss = 0.617213636636734
Batch 60        | Done in 75.04 secs    | Loss = 0.5967756807804108
Batch 70        | Done in 72.53 secs    | Loss = 0.6148975372314454
Batch 80        | Done in 79.79 secs    | Loss = 0.5857959866523743
INFO:tensorflow:Assets written to: Model1-epoch-1\assets
INFO:tensorflow:Assets written to: Model1-epoch-1\assets
C:\Users\Bolbol\anaconda3\lib\site-packages\keras\utils\generic_utils.py:494: CustomMaskW
  warnings.warn('Custom mask layers require a config and must override '
Epoch took 843.95 secs
Accuracy on test = 0.6487455197132617

Confusion Matrix:
 [[683 433]
 [351 765]]

Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.61      0.64      1116
           1       0.64      0.69      0.66      1116

    accuracy                           0.65      2232
   macro avg       0.65      0.65      0.65      2232
weighted avg       0.65      0.65      0.65      2232
```

Figure 4.4: Epoch 1 du modèle siamese

```
plot_metrics(train_loss, acc_list)
```

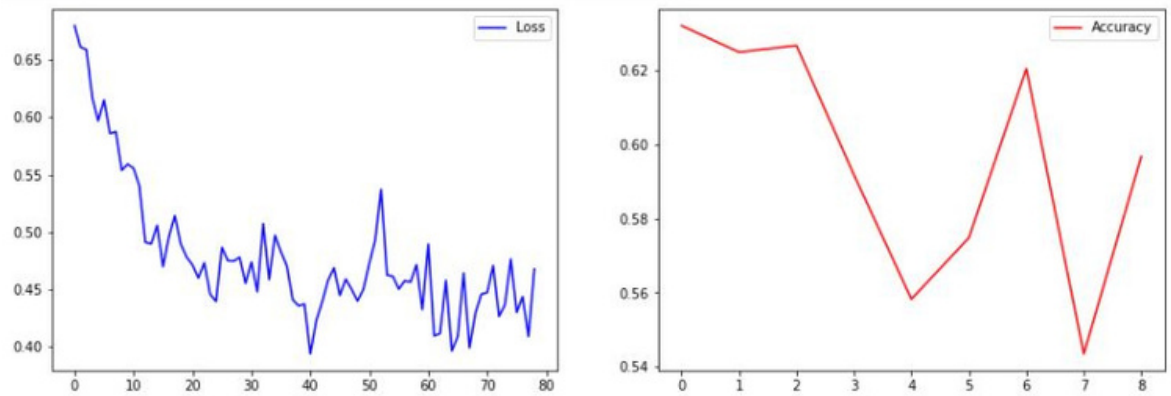


Figure 4.5: Plot du train_loss et accurance_list

04.RECONNAISSANCE FACIAL

Ensuite, on a changé les pourcentages de répartition des listes de comparaisons afin de tester l'effet de la répartition sur le score de précision. Le changement en un trainset de taille 95% du datadset a emmené a un score de précision meilleur égale à 0.98

```
ROOT=flw
train_pairs, test_pairs = train_test_split(create_pairs(), test_size=0.05, random_state=5)
len(train_pairs), len(test_pairs)
```

Figure 4.6: Répartition des train et test set

```
epochs = 10
batch_size = 1024

acc_list   = []
train_loss = []


for epoch in range(1, epochs+1):
    print(f"\n\n------------------------------  START OF EPOCH - {epoch}  ------------------------------")

    t = time.time()
    bt = time.time()
    i = 1
    loss = []

    # Training on Train data
    for batch_x, batch_y in get_batch(train_pairs, batch_size=batch_size):
        loss.append(siamese_model.train_on_batch(batch_x, batch_y))
        if i%10==0:
            print(f"Batch {i} \t| Done in {(time.time()-bt):.2f} secs \t| Loss = {sum(loss)/len(loss)}")
            train_loss.append(sum(loss)/len(loss))
            bt = time.time()
            loss = []
        i+=1

    # Testing on Test data
    y_true = None
    y_pred = None
    for batch_x, batch_y in get_batch(test_pairs, batch_size=batch_size):
        y_pred_now = siamese_model.predict(x=batch_x)
        if y_true is None:
            y_true = batch_y
            y_pred = y_pred_now
        else:
            y_true = np.vstack([y_true, batch_y])
            y_pred = np.vstack([y_pred, y_pred_now])
    y_pred = np.where(y_pred>=0.5,1,0)
    accuracy = accuracy_score(y_true, y_pred)
    acc_list.append(accuracy)

    # Saving the model if accuracy is better than previous ones.
    if accuracy>=max(acc_list):
        siamese_model.save(f"Model2-epoch-{epoch}")
```

Figure 4.7: Répartition des train et test set

04.RECONNAISSANCE FACIAL

```
--------------------------------  START OF EPOCH - 10  --------------------------
Batch 10        | Done in 657.90 secs   | Loss = 0.026120374258607627
Batch 20        | Done in 659.89 secs   | Loss = 0.02458469392731786
Batch 30        | Done in 659.36 secs   | Loss = 0.03088292321190238
Batch 40        | Done in 657.99 secs   | Loss = 0.023377724923193455
Batch 50        | Done in 656.31 secs   | Loss = 0.02401175331324339
Batch 60        | Done in 656.85 secs   | Loss = 0.019636730756610633
Batch 70        | Done in 657.79 secs   | Loss = 0.01888498682528734
Batch 80        | Done in 692.42 secs   | Loss = 0.02216341495513916
Batch 90        | Done in 962.16 secs   | Loss = 0.023147747106850147
Batch 100       | Done in 975.38 secs   | Loss = 0.025948972068727018
Batch 110       | Done in 1081.32 secs  | Loss = 0.023634669464081527
INFO:tensorflow:Assets written to: Model2-epoch-10\assets
INFO:tensorflow:Assets written to: Model2-epoch-10\assets
C:\Users\Bolbol\anaconda3\lib\site-packages\keras\utils\generic_utils.py:494: C
m mask layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '
Epoch took 8993.75 secs
Accuracy on test = 0.9810593362386859

Confusion Matrix:
 [[3257   43]
 [  70 2596]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      3300
           1       0.98      0.97      0.98      2666

    accuracy                           0.98      5966
   macro avg       0.98      0.98      0.98      5966
weighted avg       0.98      0.98      0.98      5966
```

Figure 4.8: Epoch 10 du modèle 2 siamese

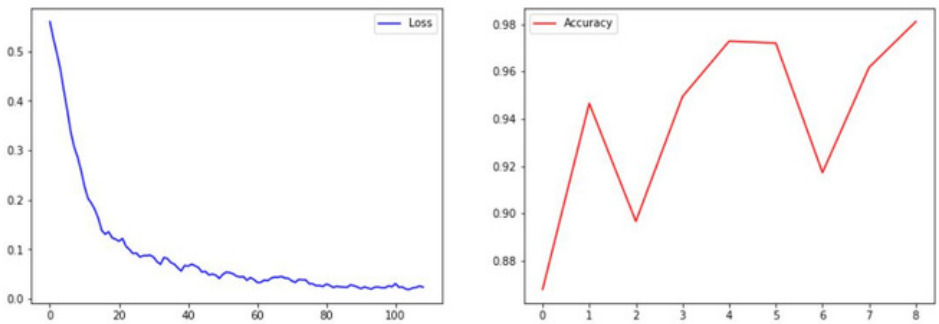


Figure 4.9: Plot du train_loss et accurance_list

04.RECONNAISSANCE FACIAL

2-Facenet

Enfin, on a utilisé le modèle facenet sans les listes de comparaisons. Cette modélisation a résulté en un score de précision égale à 0.61

FaceNet est un système de reconnaissance faciale développé en 2015 par des chercheurs de Google qui a obtenu des résultats à la pointe de la technologie sur une gamme d'ensembles de données de référence de reconnaissance faciale. Il est basé sur la couche initiale. FaceNet utilise des modules de création en blocs pour réduire le nombre de paramètres pouvant être entraînés. Ce modèle prend des images RVB de 160×160 et génère un plongement de taille 128 pour une image. Le système FaceNet peut être largement utilisé grâce à plusieurs implémentations open source tierces du modèle et à la disponibilité de modèles pré-entraînés.

Dans ce projet, on a développée un système de détection de visage à l'aide de FaceNet et d'un classificateur SVM pour identifier des personnes à partir de photographies

04.RECONNAISSANCE FACIAL

Enfin, on a utilisé le modèle facenet sans les listes de comparaisons. Cette modélisation a résulté en un score de précision meilleur 0.61

```python
def fixed_denormalize(image):
    """ Restandartize images to [0, 255]"""
    return image * 128 + 127.5

def getEmbeds(model, n, loader, imshow=False, n_img=5):
    model.eval()
    # images to display
    images = []
    embeds, labels = [], []
    for n_i in tqdm.trange(n):
        for i, (x, y) in enumerate(loader, 1):
            # on each first batch get 'n_img' images
            if imshow and i == 1:
                inds = np.random.choice(x.size(0), min(x.size(0), n_img))
                images.append(fixed_denormalize(x[inds].data.cpu()).permute((0, 2, 3, 1)).numpy())

            embed = model(x.to(device))
            embed = embed.data.cpu().numpy()
            embeds.append(embed), labels.extend(y.data.cpu().numpy())
    if imshow:
        plot(images=np.concatenate(images))

    return np.concatenate(embeds), np.array(labels)
```

Figure 4.10: Fonction pour extraire des embeddings

```python
# 3. Get embeddings
# Train embeddings
trainEmbeds, trainLabels = getEmbeds(model, 1, trainL, False)
#trainEmbeds_aug, trainLabels_aug = getEmbeds(model, 50, trainL_aug, imshow=True, n_img=3)
#trainEmbeds = np.concatenate([trainEmbeds, trainEmbeds_aug])
#trainLabels = np.concatenate([trainLabels, trainLabels_aug])

# Test embeddings
testEmbeds, testLabels = getEmbeds(model, 1, testL, False)
```

```
100%|██████████| 1/1 [02:15<00:00, 135.20s/it]
100%|██████████| 1/1 [01:03<00:00, 63.23s/it]
```

Figure 4.11: Extraction des embeddings

04.RECONNAISSANCE FACIAL

```python
from sklearn.metrics import pairwise_distances
import pandas as pd
import seaborn as sns
sns.set()

def getDist(x, metric='euclidean', index=None, columns=None):
    dists = pairwise_distances(x, x, metric=metric)
    return pd.DataFrame(dists, index=index, columns=columns)

def heatmap(x, title='', cmap='Greens', linewidth=1):
    plt.figure(figsize=(17, 12))
    plt.title(title)
    sns.heatmap(x, cmap=cmap, square=True)
    plt.show()
```

Figure 4.12: Fonction pour calculer les embeddings

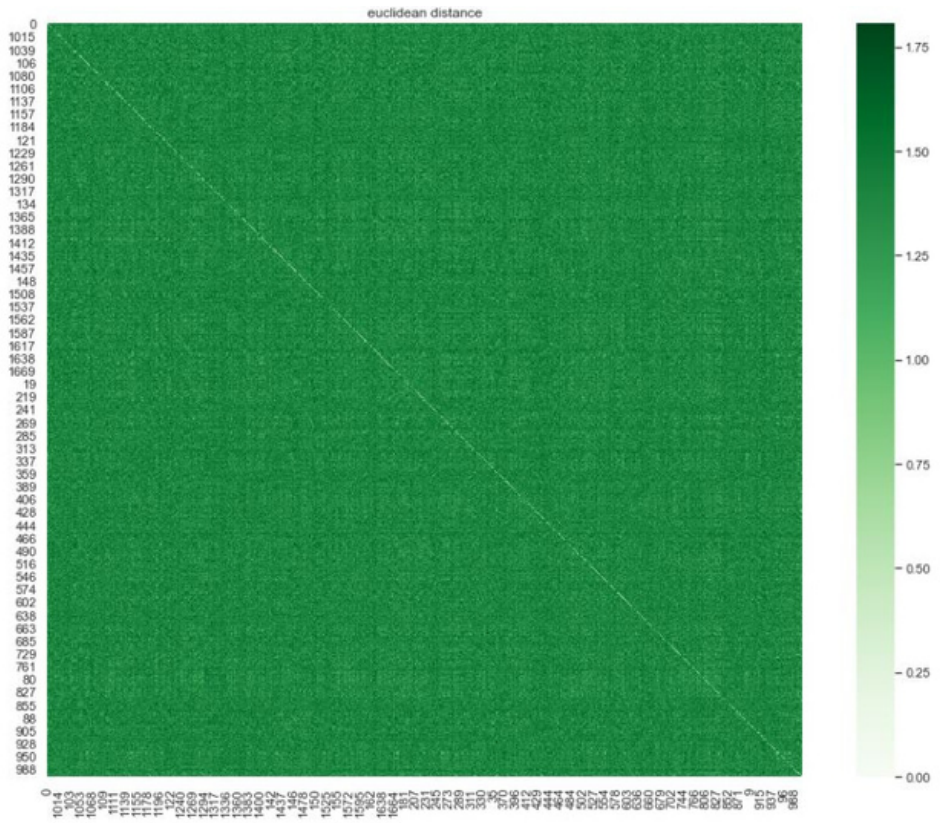


Figure 4.13: Distance euclidienne des trainset

31

04.RECONNAISSANCE FACIAL

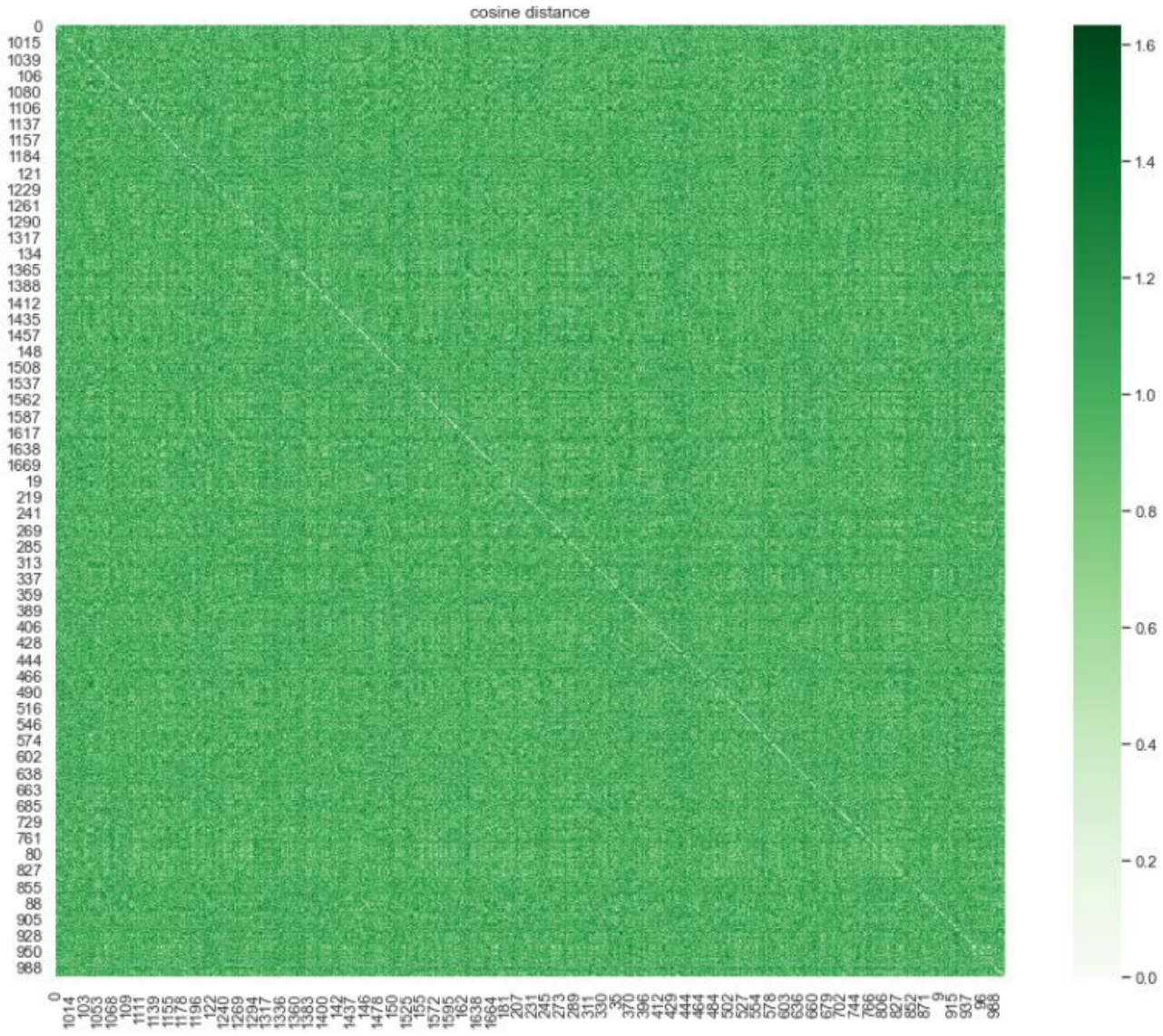


Figure 4.14: Distance cosine des trainset

04.RECONNAISSANCE FACIAL

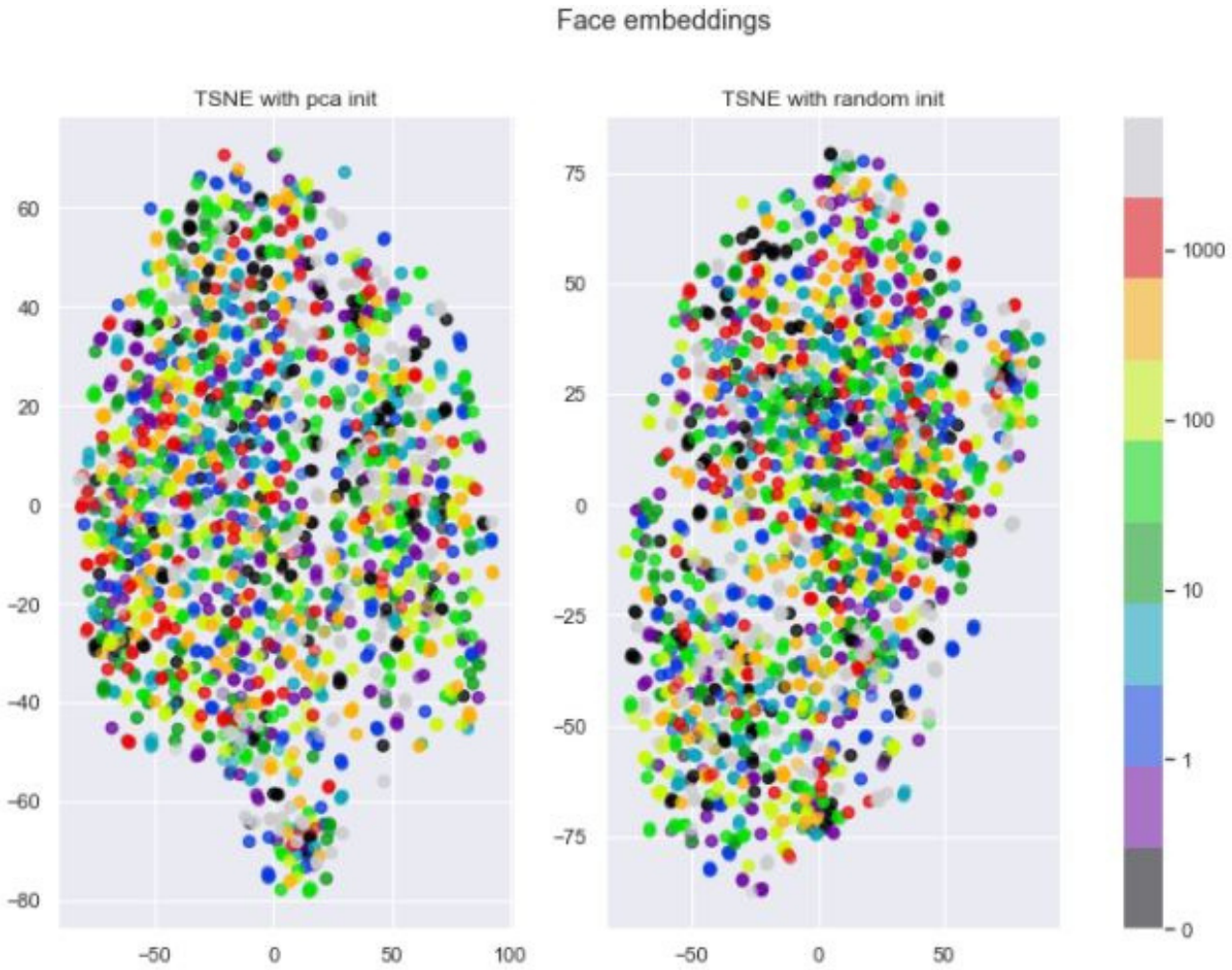


Figure 4.15: Groupement des images par les méthode TSNE

04.RECONNAISSANCE FACIAL

Figure 4.16: Groupement des images par les méthode PCA

34

04.RECONNAISSANCE FACIAL

```
%%time

from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
import warnings

warnings.filterwarnings('ignore', 'Solver terminated early.*')

param_grid = {'C': [1, 10, 100, 1e3, 5e3],
              'gamma': [0.0001, 0.0005, 0.001, 'auto'],
              'kernel': ['rbf', 'sigmoid', 'poly']}
model_params = {'class_weight': 'balanced', 'max_iter': 10, 'probability': True, 'random_state': 3}
model = SVC(**model_params)
clf = GridSearchCV(model, param_grid)
clf.fit(X, y)

print('Best estimator: ', clf.best_estimator_)
print('Best params: ', clf.best_params_)
```

```
C:\Users\Bolbol\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: UserWarning: The lea
  warnings.warn(("The least populated class in y has only %d"
Best estimator:  SVC(C=1000.0, class_weight='balanced', gamma=0.0001, max_iter=10,
    probability=True, random_state=3)
Best params:  {'C': 1000.0, 'gamma': 0.0001, 'kernel': 'rbf'}
Wall time: 3h 44min 22s
```

Figure 4.17: Recherche des meilleurs paramètres

```
from sklearn.metrics import accuracy_score

inds = range(2600)
train_acc = accuracy_score(clf.predict(X[inds]), y[inds])
print(f'Accuracy score on train data: {train_acc:.3f}')

test_acc = accuracy_score(clf.predict(X_test), y_test)
print(f'Accuracy score on test data: {test_acc:.3f}')
```

```
Accuracy score on train data: 0.611
```

Figure 4.18: score de précision

35

CONCLUSION

Dans ce projet on a testé plusieurs méthodes de détection comme OpenCV, MTCNN, CNN et dlib et de reconnaissance facial comme siamese networks et facenet. Les résultats trouvés varient selon les modèles et les listes de comparaisons. Citons, le siamese network ayant le meilleur score de précision entre les modèle. On conclue aussi que le score de précision peut s'améliorer en fonction des listes de comparaisons.