

STAGE D'ETE 14

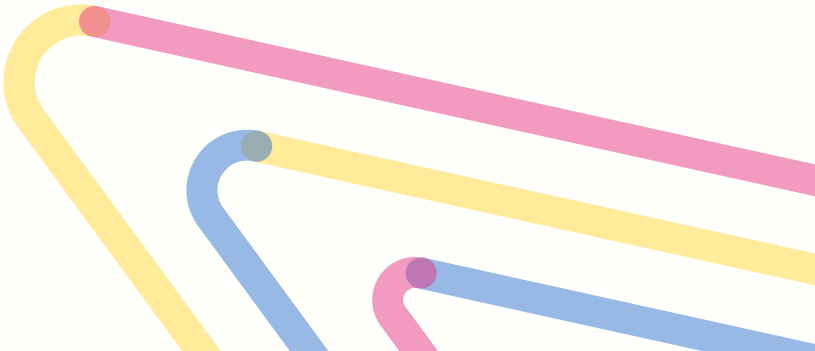
Reconnaissance facial

BACCAR BELKIS
TAYECHE WALID





Plan

- 01.Introduction
 - 02.Compréhension des données
 - 03.Préparation des données
 - 04.Détection faciale
 - 05.Reconnaissance faciale
 - 06.Conclusion
- 



01.Introduction



Biométrie et reconnaissance facial

Biométrie: authentifier une personne sur la base d'un ensemble de données reconnaissables et vérifiables, uniques et spécifiques à celles-ci.

Reconnaissance facial: identification positive d'un visage dans une photo vis à vis des images pre-existant dans la BD des visages

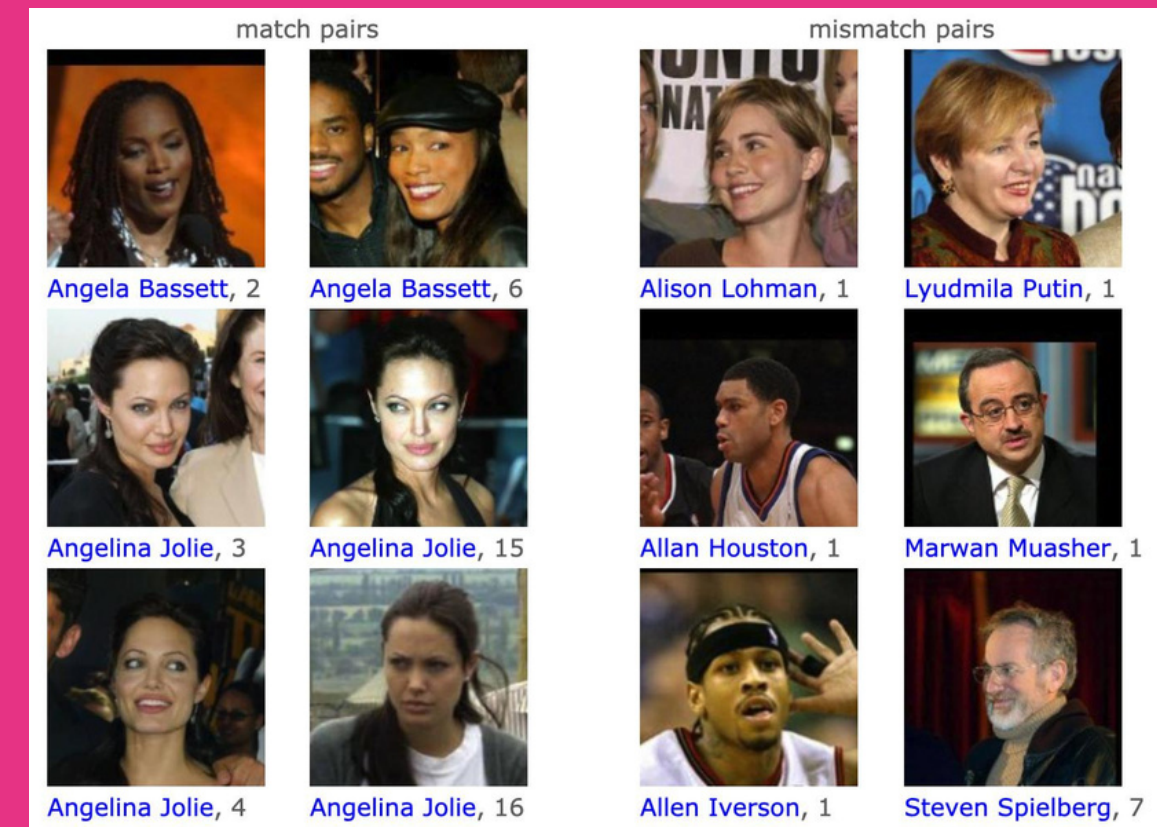


02.Compréhension des données

LFW (Labeled Faces in the Wild)

- 13,233 images de visages collectés depuis le web
- 5749 identitiés avec 1680 personnes avec 2 images ou plus

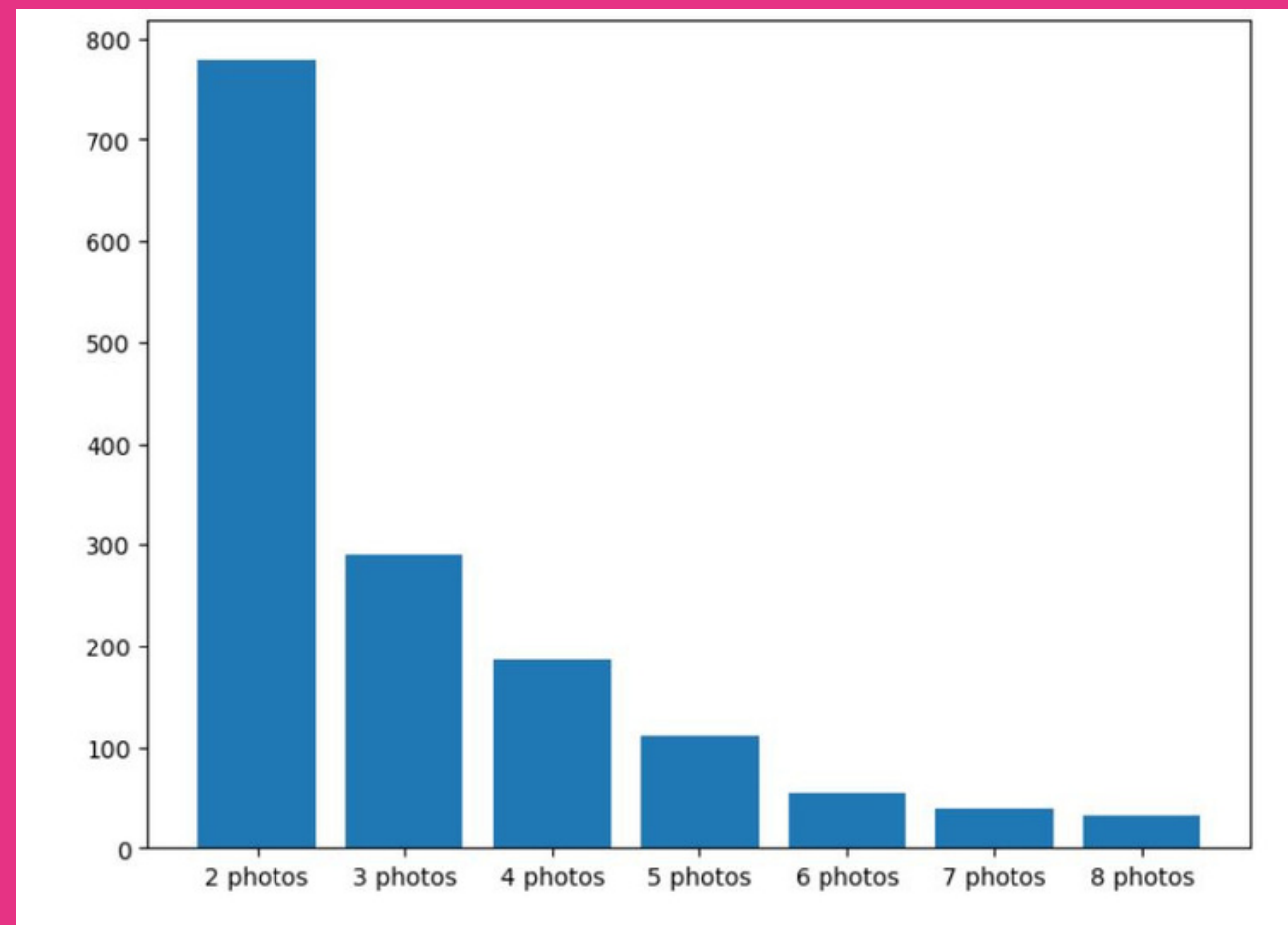
lien: <https://arxiv.org/abs/1901.05903>



Usage 

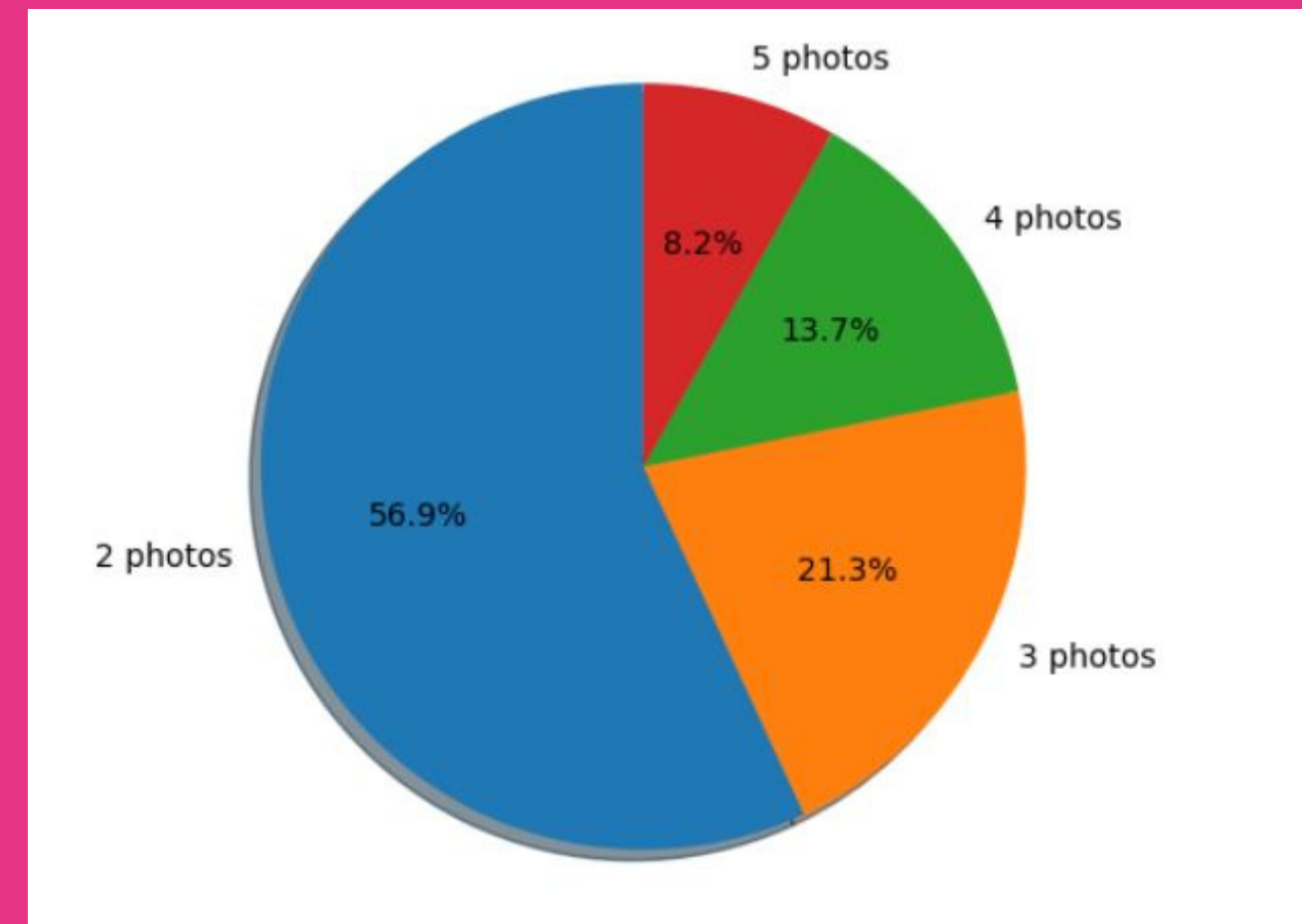


LFW (Labeled Faces in the Wild)



Répartition d'image par personne

Pourcentage des photos pour les personnes avec 2 à 5 photos



Nombre de personnes en fonction du nombre d'images

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



03.Préparation des données

Préparation des données

1-Fractionnement des données :
70% trainset et 30% testset
trainset: 958 personnes
testset: 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

2- Les listes de comparaisons:
regroupe id1, img1, id2, img2
intraclasse train
interclasse train
intraclasse test
interclasse test

```
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
```

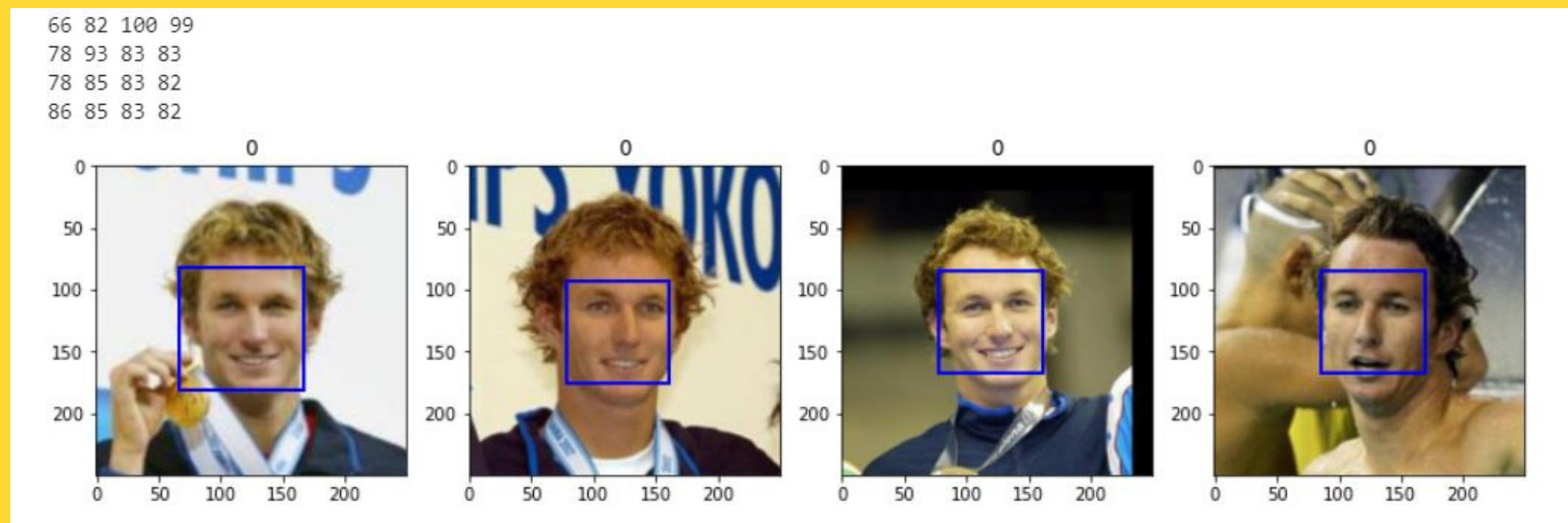


04.Détection facial

4.1-Dlib et CNN:

- boîte à outils permettant de créer des applications d'apprentissage automatique et d'analyse de données dans le monde réel.

L'un des principaux avantages de ce détecteur est qu'il peut utiliser la puissance de calcul de nos GPU



4.1-Dlib et CNN:

Détection facial avec dlib et CNN

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

# 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)
```


4.2-OpenCV et Haar:

- une approche basée sur l'apprentissage automatique
- 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.harcascades + '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))
```

4.2-OpenCV et Haar:

d_inter_train

	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

les listes de comparaisons avec les dimensions



4.3- MTCNN

- une bibliothèque python
- un cadre multitâche en cascade profond utilisant différentes fonctionnalités de «sous-modèles» pour chacun augmenter leurs forces de corrélation.

```
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(
            lambda x: x.resize((1280, 1280)) if (np.array(x) > 2000).all() else x
        )

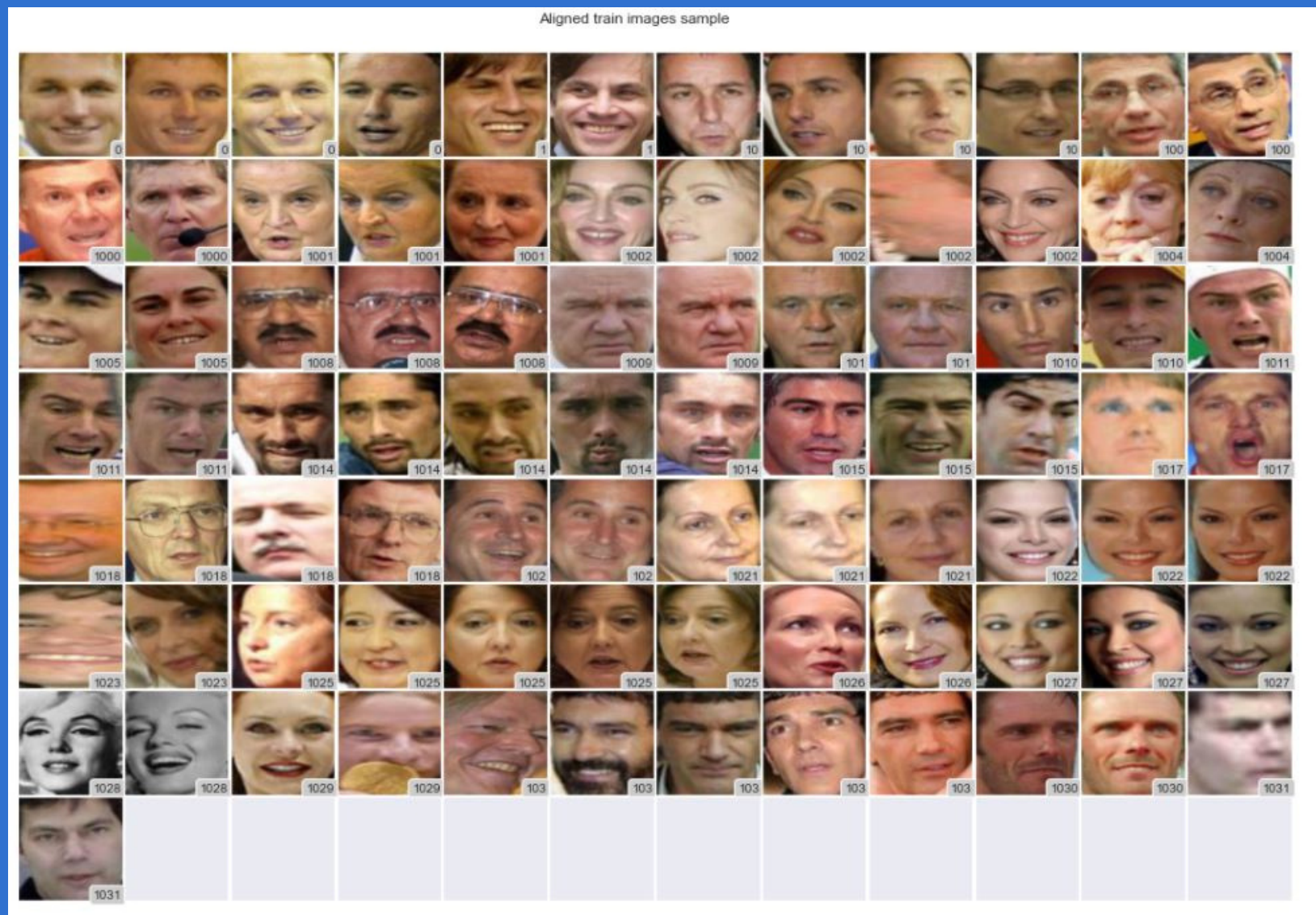
    # for convenience we will use ImageFolder instead of getting Image objects by file paths
    dataset = datasets.ImageFolder(path, transform=transformer)
    dataset.samples = [(p, p.replace(path, new_path)) for p, _ in dataset.samples]

    # 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)

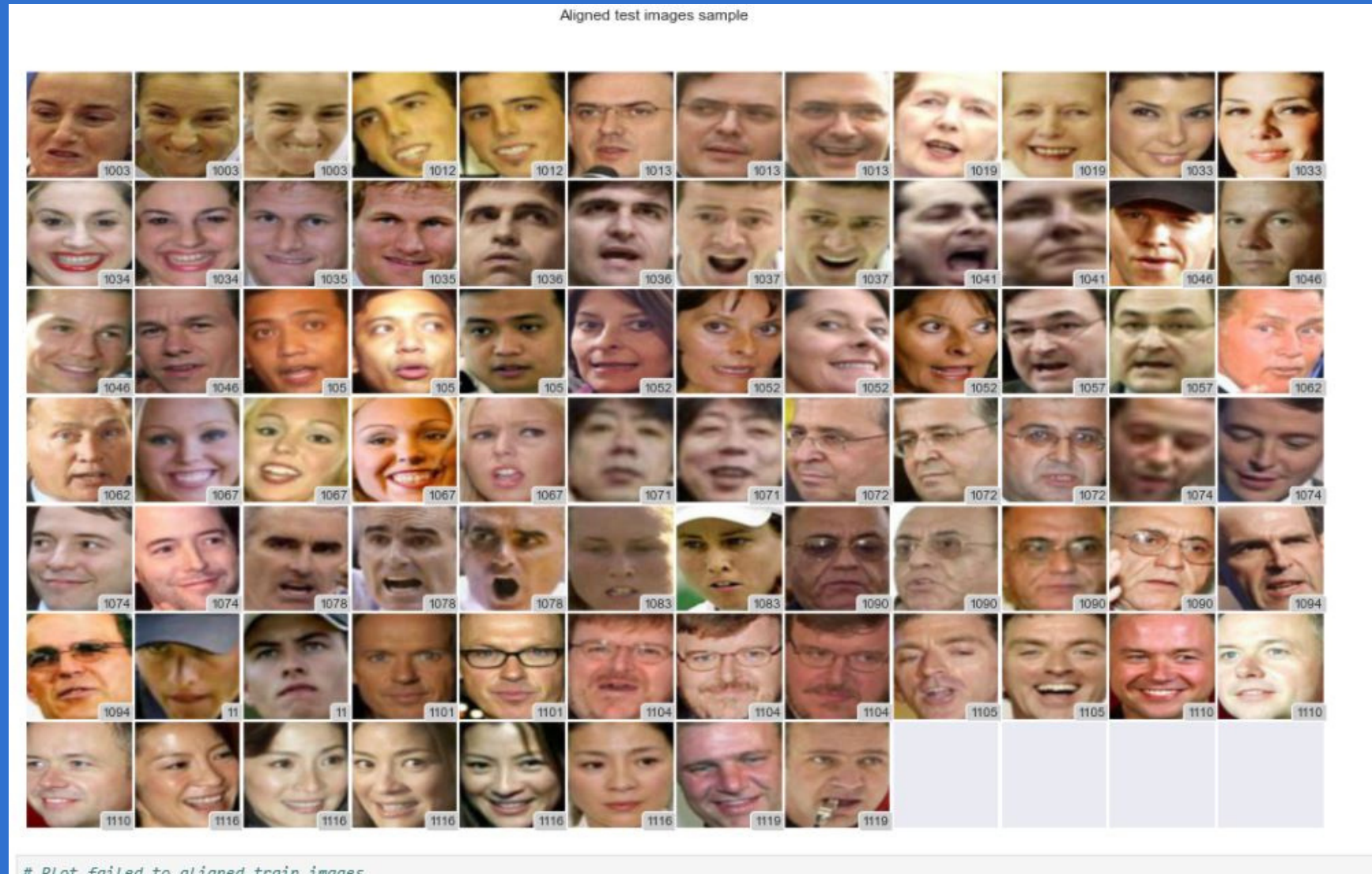
    # spare some memory
    del model, loader, dataset
```

Retranchement des visages avec MTCNN

4.3- MTCNN



Retranchement des visages du trainset avec MTCNN



Retranchement des visages du testset avec MTCNN

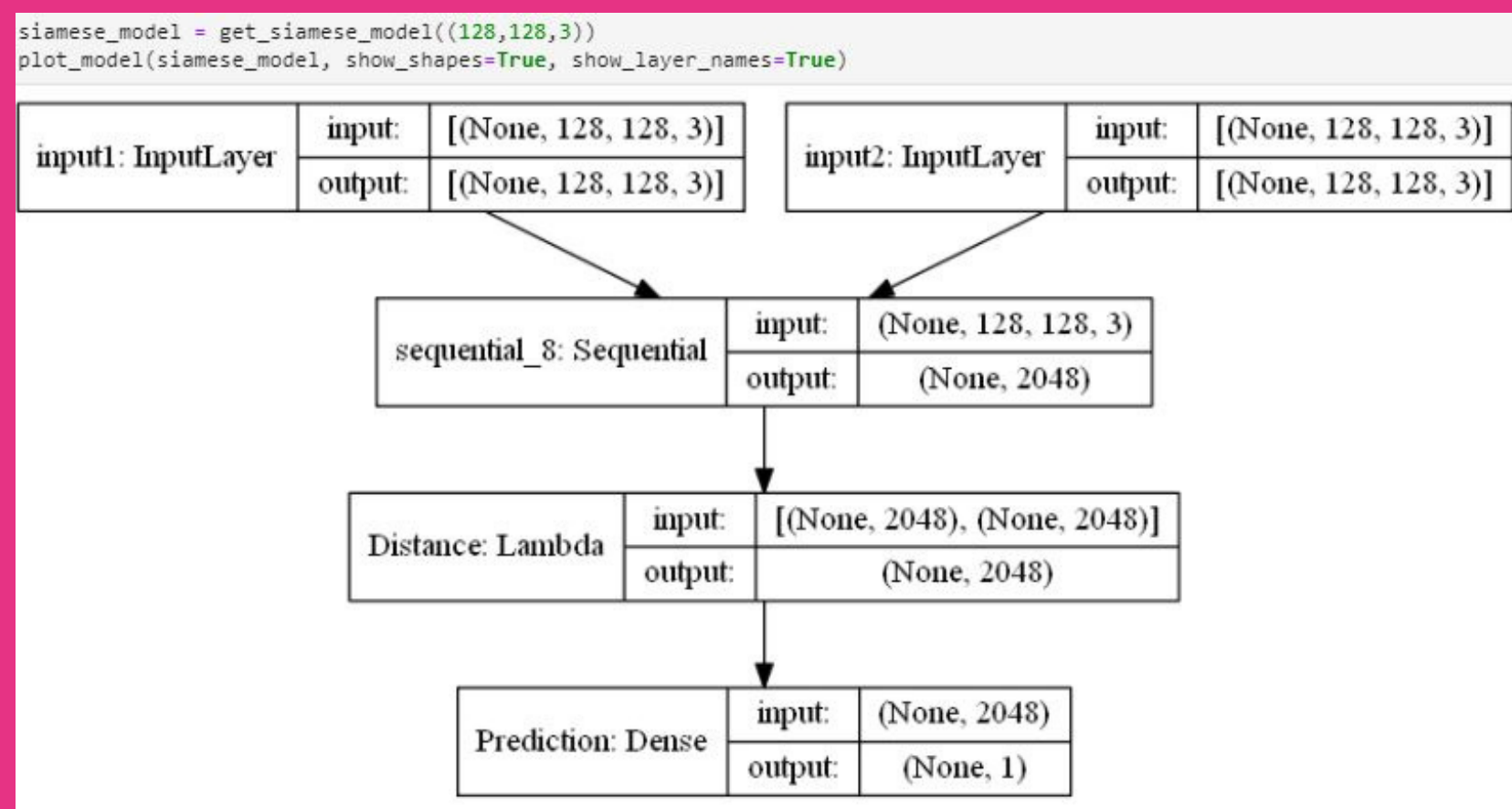




05.Reconnaissance faciale

5.1-Siamese Networks:

- une classe d'architectures de réseaux de neurones qui contiennent au moins deux sous-réseaux identiques
- La mise à jour des paramètres est reflétée sur les deux sous-réseaux
- utilisé pour trouver la similitude des entrées en comparant ses vecteurs de caractéristiques



Plot du modèle siamese

5.1-Siamese Networks:

Modèle siamese

```
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
```



5.1-Siamese Networks:

Epoch 1 du modèle siamese

```
Accuracy on test = 0.6487455197132617

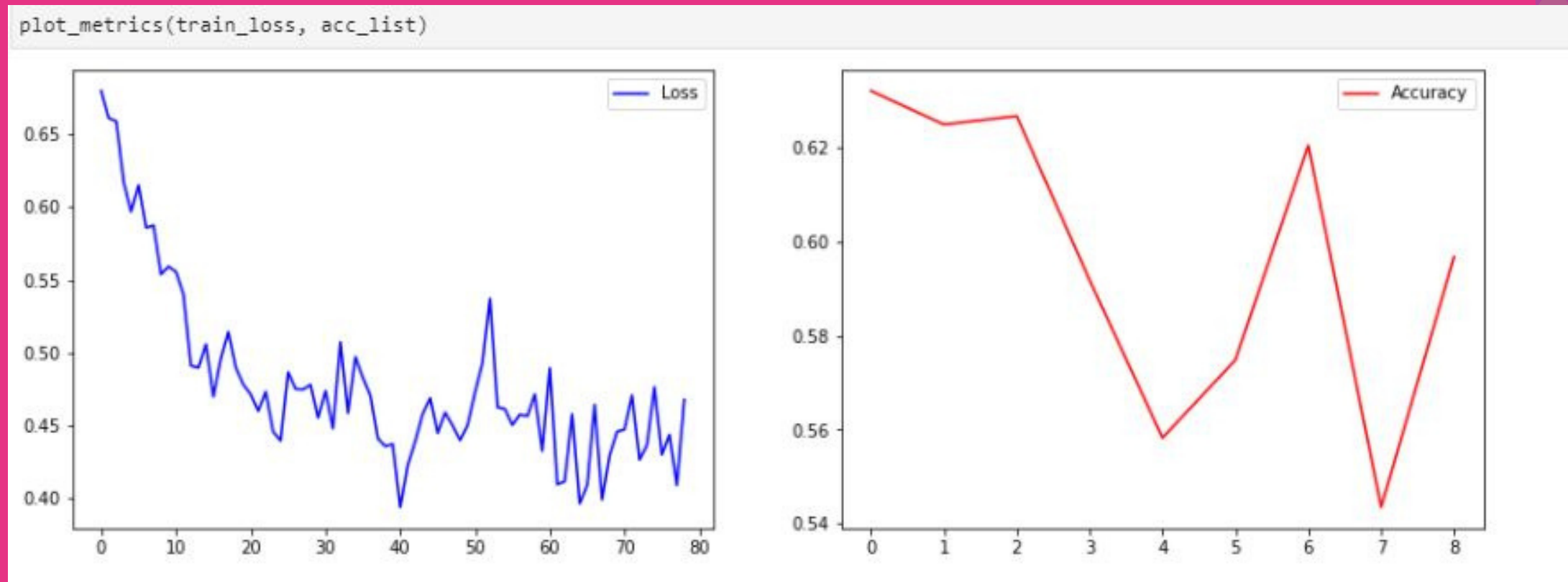
Confusion Matrix:
[[683 433]
 [351 765]]

Classification Report:
              precision    recall  f1-score   support

     0       0.66       0.61       0.64       1116
     1       0.64       0.69       0.66       1116

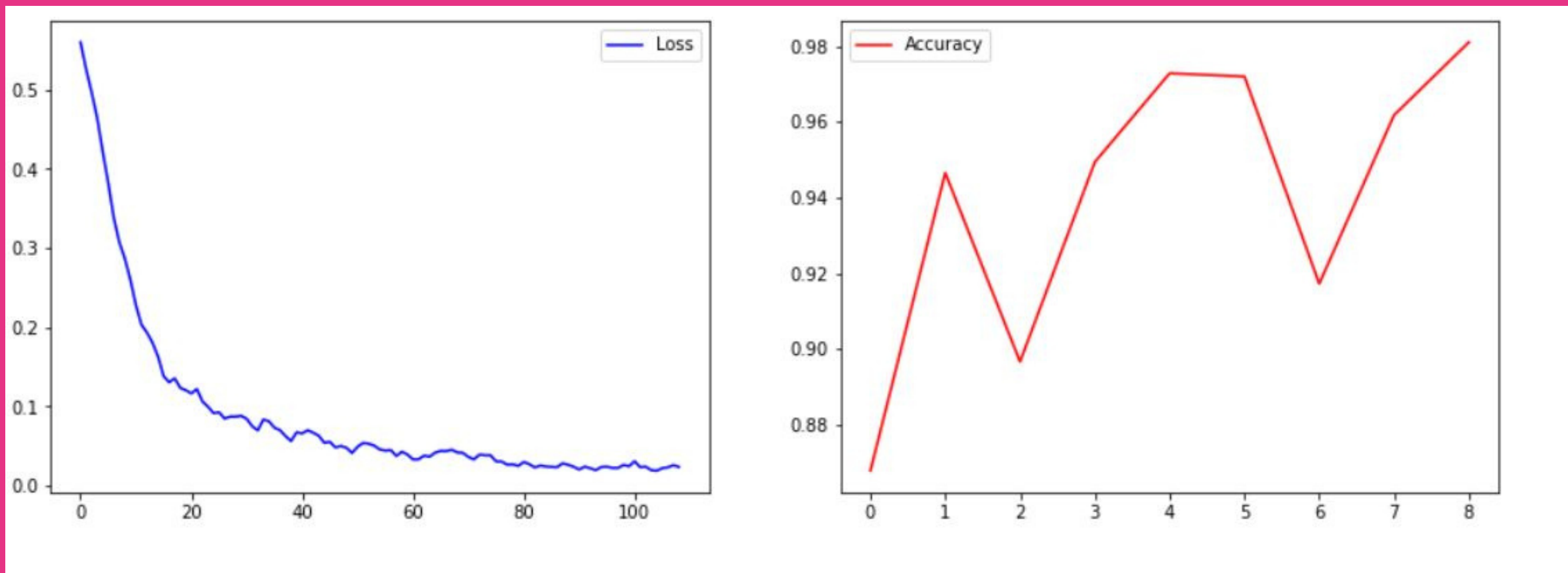
 accuracy          0.65
 macro avg         0.65
 weighted avg      0.65
```

Plot du train_loss et
accurance_list



5.1-Siamese Networks:

- 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 dataset a emmené a un score de précision meilleur égale à 0.98
-



Plot du train_loss et accuracy_list

5.2-Facenet

- un système de reconnaissance faciale développé en 2015 par des chercheurs de Google
- utilise des modules de création en blocs pour réduire le nombre de paramètres pouvant être entraînés
- prend des images RVB de 160×160 et génère un plongement de taille 128 pour une image
- 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

5.2-Facenet

```
def fixed_denormalize(image):
    """ Restandardize 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)
```

Extraction des embeddings

```
%%time

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

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

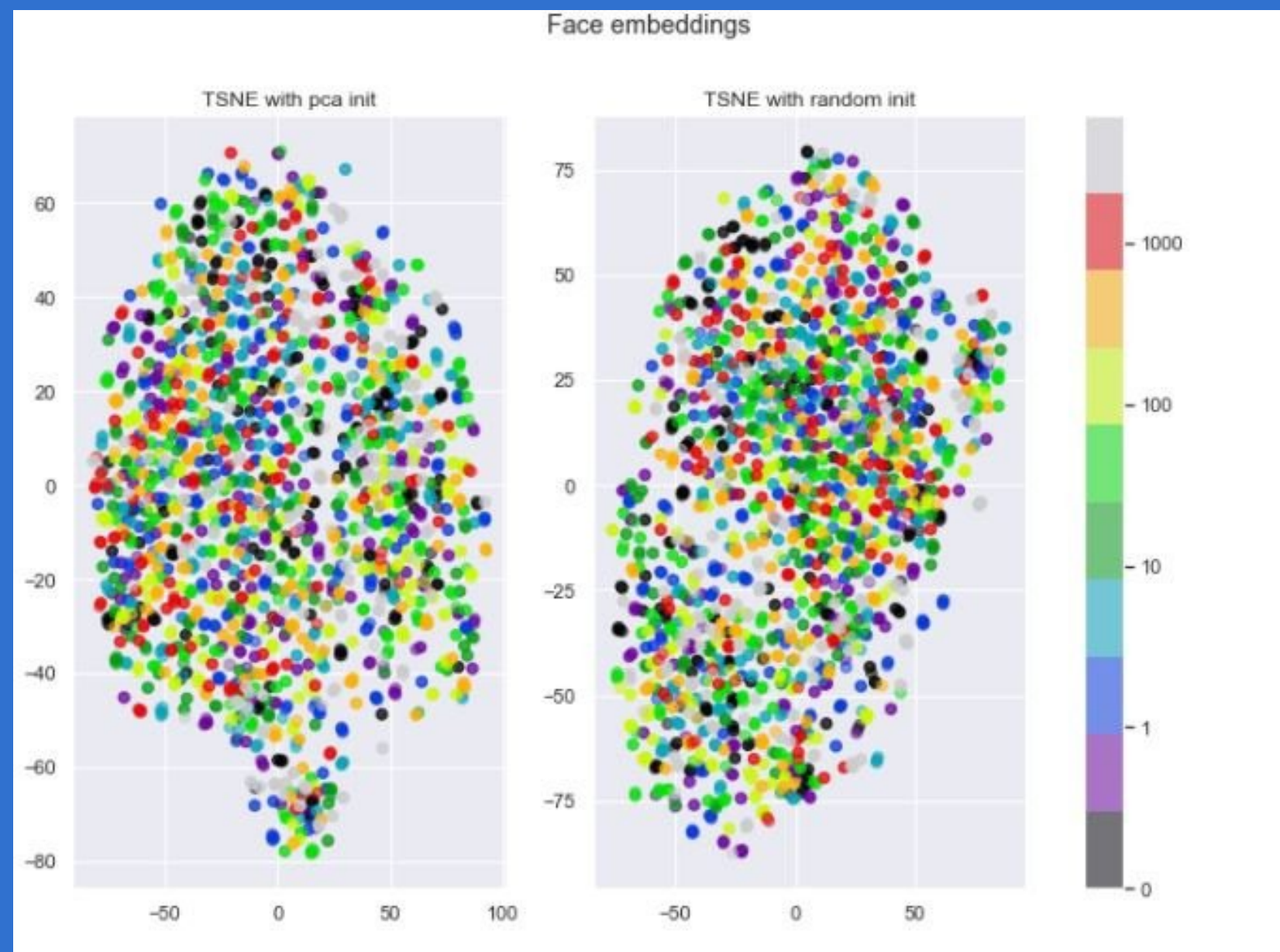
param_grid = {'C': [1, 10, 100, 1e3, 5e3],
              'gamma': [0.0001, 0.0005, 0.001, 'auto'],
              'kernel': ['rbf', 'sigmoid', 'poly']}
model_params = {'class_weight': 'balanced', 'max_iter': 10, 'probability': True, 'random_state': 3}
model = SVC(**model_params)
clf = GridSearchCV(model,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

Recherche des meilleurs paramètres

5.2-Facenet



```
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

score de précision

Groupement des images par les méthode TSNE



06.Conclusion



Merci pour votre
attention