UMMTO
TIZI-OUZOU

RAPPORT

SYSTÈME DE RECONNAISSANCE FACIALE

// BIOMÉTRIE SÉCURISÉE 2024

Sommaire

Introduction

La biométrie sécurisée, un domaine en constante évolution, repose sur l'utilisation de caractéristiques physiologiques ou comportementales uniques pour identifier et authentifier les individus. Cette approche offre un niveau de sécurité supérieur aux méthodes traditionnelles de sécurité, telles que les mots de passe ou les cartes d'identité, en raison de sa capacité à garantir une identification précise et fiable.

Dans un monde où la protection des données et la sécurité des informations sensibles sont des préoccupations majeures, la biométrie sécurisée trouve des applications variées, allant de l'accès aux bâtiments et aux appareils électroniques à la gestion des identités en ligne et à la prévention de la fraude. Parmi les différentes modalités biométriques, la reconnaissance faciale se distingue par sa commodité, sa non-intrusivité et sa capacité à fournir des résultats rapides et précis.

L'objectif principal de ce travail est de créer un un système de reconnaissance faciale capable de détecter et d'identifier les visages avec précision. Ce projet offre une occasion unique d'appliquer les concepts théoriques appris en cours à un problème concret, tout en acquérant une expérience pratique précieuse dans ce domaine .

Les principales difficultés de la reconnaissance de visages :

La reconnaissance faciale pose des défis significatifs, en particulier lorsque l'on cherche à automatiser un processus que le cerveau humain accomplit naturellement avec aisance. Pour les humains, reconnaître des visages est une tâche visuelle complexe, mais la construction d'un système automatique capable d'accomplir cette tâche est particulièrement difficile, surtout dans des conditions d'acquisition d'images très variables. Les difficultés sont exacerbées par deux types de variations associées aux images de visages : les variations inter-sujet, limitées en raison de la ressemblance physique entre les individus, et les variations intra-sujet, bien plus étendues. Ces dernières sont influencées par divers facteurs, que nous examinerons ci-dessous.

1 - Changement d'illumination :

La variation de l'éclairage lors de la capture des visages rend la tâche de la reconnaissance plus ardue, un visage de la même personne avec deux différents niveau d'éclairage peut être reconnue comme deux différentes personnes.

2 - La variation de pose

elle représente un défi majeur pour les systèmes de reconnaissance faciale. Lorsque le visage est en profil dans le plan image (avec une orientation inférieure à 30°), il peut être normalisé en identifiant au moins deux traits faciaux (passant par les yeux). Cependant, au-delà d'une rotation de 30°, la normalisation géométrique devient impraticable.

3 -Expressions faciales

Les défis associés aux expressions faciales sont également considérables dans les systèmes de reconnaissance faciale. Les variations d'expressions, telles que les sourires, les froncements de sourcils, ou d'autres mouvements faciaux, peuvent considérablement altérer l'apparence du visage. La reconnaissance précise devient alors complexe en raison de la diversité des expressions possibles. Ces variations constituent un obstacle majeur pour les systèmes automatisés, nécessitant des techniques avancées pour la détection et l'interprétation des expressions faciales.

4 - Présence ou absence des composants structurels

Des aspects particuliers tels que la barbe, la moustache, les lunettes, le style et la couleur des cheveux(Figure suivante) provoquent des changements importants dans les composants structuraux du visage, notamment la forme, la couleur, la taille, etc....

Etats de l'arts des techniques de reconnaissance des visages :

Un système automatique de reconnaissance de visages se décompose en trois sous-systèmes: détection de visage, extraction des caractéristiques et reconnaissance de visages. La mise en œuvre d'un système automatique et fiable de reconnaissance faciale est un verrou technologique qui n'est toujours pas résolu.

Plusieurs méthodes de reconnaissance de visages ont été proposées durant les vingt dernières années. La reconnaissance de visage est un axe de recherche ouvert attirant des chercheurs venant de disciplines différentes: psychologie, reconnaissance de formes, réseaux neuraux, vision artificielle et infographie. Dans ce paragraphe, nous présenterons les approches de la reconnaissance de visage les plus connues. Ces dernières peuvent être subdivisées en trois catégories: les approches holistiques ou globales, les approches locales et les approches hybrides.

Approches hybrides :

Les approches basées sur l'apparence, également connues sous le nom de méthodes basées sur l'apparence, se concentrent sur l'identification des visages en utilisant l'image complète comme entrée pour le système de reconnaissance. Chaque image de visage est représentée par un vecteur résultant de la concaténation des valeurs de niveau de gris de ses pixels. Cette représentation, dans l'espace images, conserve implicitement les informations de texture et de forme cruciales pour la reconnaissance faciale, offrant une meilleure capture de l'aspect global du visage par rapport aux représentations locales. Ces approches peuvent être classées en deux catégories : linéaire et non linéaire.

Les méthodes globales :

Le principe de ces approches est d'utiliser toute la surface du visage comme source d'information sans tenir compte des caractéristiques locales comme les yeux, la bouche, etc... L'une des méthodes la plus largement utilisée pour la représentation du visage dans son ensemble est l'ACP. Les algorithmes globaux s'appuient sur des propriétés statistiques bien connues et utilisent l'algèbre linéaire. Ils sont relativement rapides à mettre en œuvre, mais sont sensibles aux variations d'illumination, de pose et d'expression faciale. Parmi les approches les plus importantes réunies au sein de cette classe on trouve:

L'Analyse en Composantes Principales (PCA ou Eigen Faces),
L'Analyse Discriminante Linéaire (LDA),
Machine à Vecteurs de Support (SVM),
Les Réseaux de Neurones (RNA)

Mélange de Gaussiennes (GMM),
Modèle Surfacique du Visage (3D),
L'approche statistique et probabiliste.

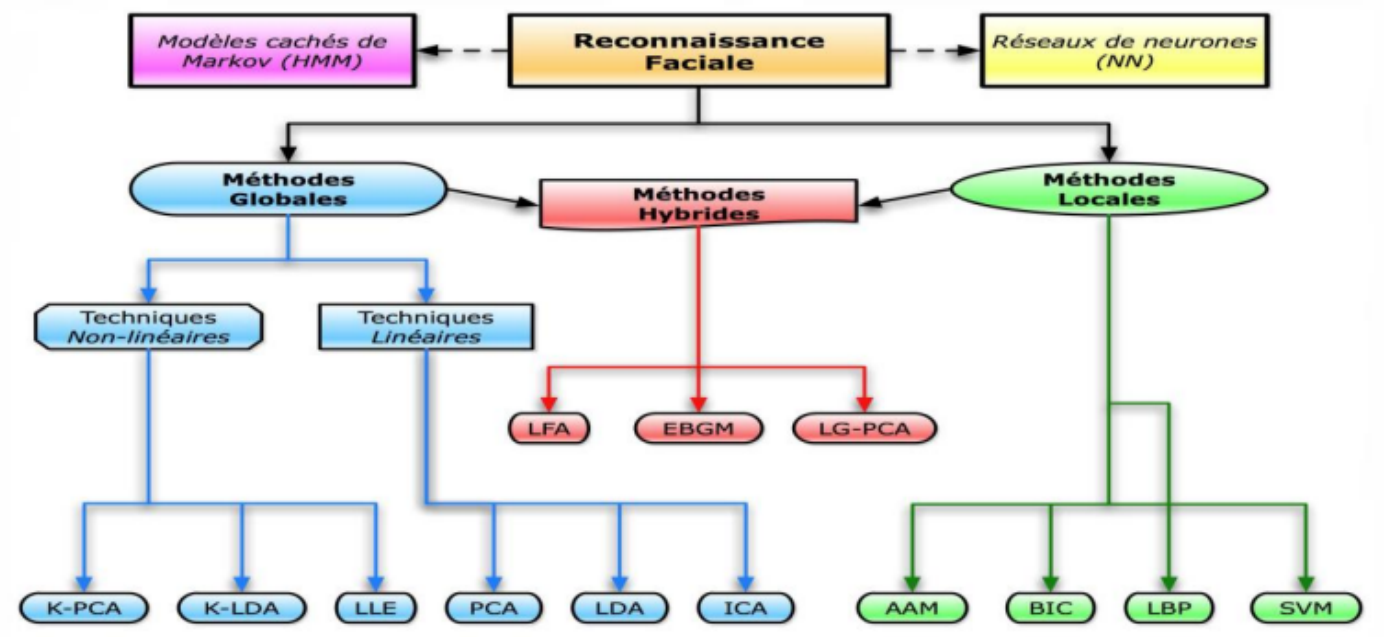
Approches locales :

Les méthodes locales, aussi appelées méthodes à traits ou géométriques, ciblent des points spécifiques de l'image, souvent autour de caractéristiques clés du visage. Elles visent à minimiser le bruit induit par des éléments perturbateurs tels que cheveux, lunettes ou chapeaux en accordant une attention particulière aux détails locaux. Cependant, leur complexité se manifeste dans la prise en compte de multiples perspectives du visage et la précision limitée lors de l'extraction des points. Ces approches extraient des caractéristiques locales, comme les yeux, le nez et la bouche, utilisant ensuite leur géométrie ou apparence comme entrée pour le classificateur.
Deux pratiques distinctes émergent :

- La première repose sur l'extraction de régions entières du visage, elle est souvent implémentée avec une approche globale de reconnaissance de visage.
- La deuxième pratique extrait des points particuliers des différentes régions caractéristiques du visage, tels que les coins des yeux, de la bouche et du nez.

Parmi ces approches on peut citer :
Modèles de Markov Cachés (Hidden Markov Models (HMM)),
L'Algorithme Elastic Bunch Graph Matching (EBGM),
Eigen Object (EO),
L'appariement de gabarits.

Classification des algorithmes principaux utilisés en reconnaissance faciale:



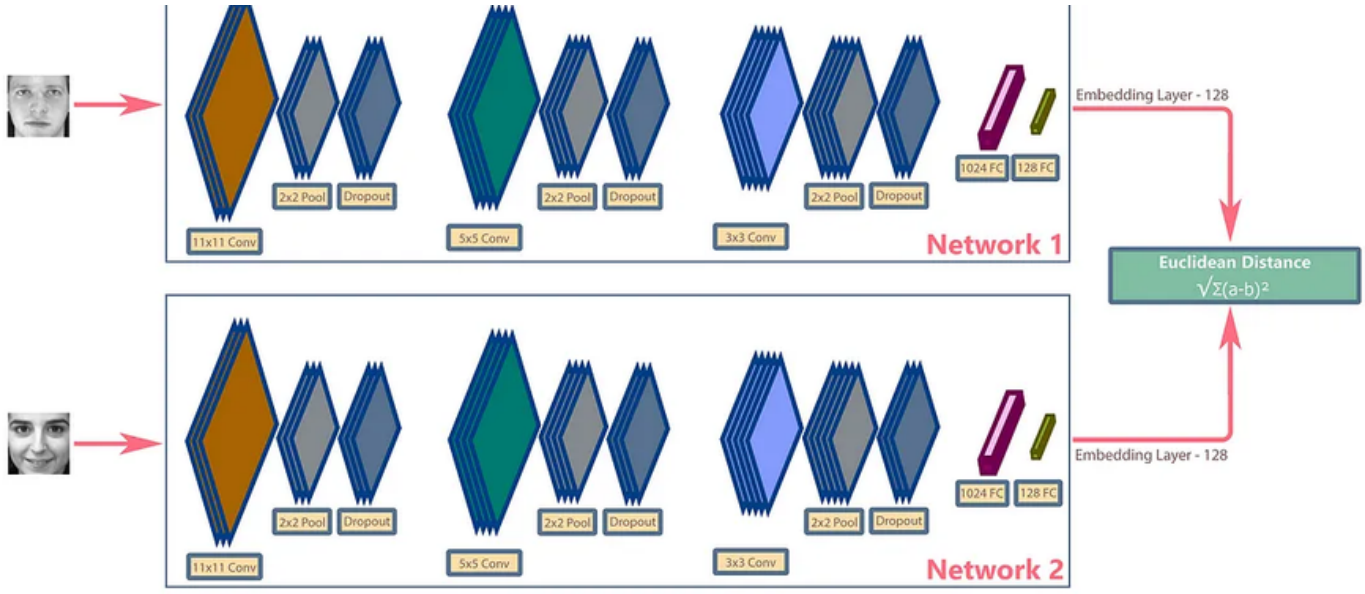
Architecture choisi

Avec le modele CNN nous avons rencontré un problème majeur. Après l'entraînement et la validation du modèle, si nous voulons ajouter une nouvelle personne à notre modèle, nous devons le re-entraîner intégralement.

C'est pourquoi nous avons opté pour la création d'un modèle basé sur les réseaux Siamese. Ce dernier est composé d'une paire de réseaux de neurones identiques entre eux. La principale différence entre les réseaux CNN et Siamese réside dans le fait que ce dernier ne classe pas les images dans des catégories ou étiquettes spécifiques. Il se contente de découvrir la distance entre deux images données.

Si les images ont la même étiquette, le réseau Siamese doit apprendre les paramètres, les poids et les biais de manière à produire une distance plus petite entre les deux images. En revanche, si les images appartiennent à des étiquettes différentes, la distance doit être plus grande.

Cette approche Siamese offre l'avantage de ne pas nécessiter le re-entraînement complet du modèle lors de l'ajout d'une nouvelle personne, ce qui constitue une solution efficace à notre problème initial avec les modèles CNN.



L’utilisation de ce modele dans notre cas est pour l’extraction des caracteristique de l’image. Comme le montre la photo, la paire de réseaux est la même.

Pour former un réseau Siamese, une paire d'images est sélectionnée dans l'ensemble de données, chacune traitée par l'un des réseaux.

Les réseaux ont la même structure, donc les mêmes opérations seront effectuées sur les images respectives.

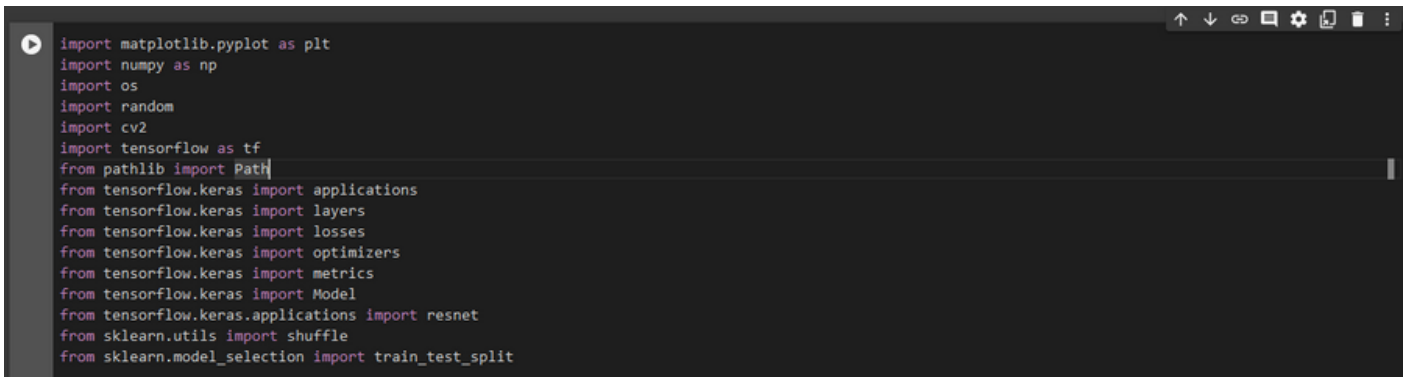
Les réseaux de neurones à la fin ont des couches entièrement connectées, la dernière étant composée de 128 nœuds. Cette couche est la dernière caractéristique produite lorsque le réseau est appliqué à l’image. Ainsi, les deux images de la paire traitées par le réseau Siamese produisent deux représentations de l’image.

Le réseau trouve ensuite la distance euclidienne entre les deux vecteures Si les images représentent la même personne, on s'attend à ce que les vecteures soient très similaires, la distance devrait donc être plus petite. Cependant, si les images représentent des personnes différentes, la distance devrait alors être une valeur plus élevée.

Une fonction sigmoïde est appliquée sur la valeur de distance pour l'amener à la plage 0–1.
Une fonction de perte est appliquée au résultat sigmoïde, pour pénaliser le réseau pour mettre à jour ses poids et ses biais. J'utilise Binary Cross Entropy pour la fonction de perte. La mise à jour des poids et des biais effectués sur les deux réseaux est exactement la même.

Préparation de l'Environnement pour le Modèle de Reconnaissance Faciale:

Importation des Bibliothèques Requises :



```
import matplotlib.pyplot as plt
import numpy as np
import os
import random
import cv2
import tensorflow as tf
from pathlib import Path
from tensorflow.keras import applications
from tensorflow.keras import layers
from tensorflow.keras import losses
from tensorflow.keras import optimizers
from tensorflow.keras import metrics
from tensorflow.keras import Model
from tensorflow.keras.applications import resnet
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
```

Fonction de Génération de Paires d'Images pour l'Entraînement du Modèle

```
# Function to generate image pairs
def generate_image_pairs(images, labels):
    # Generate index for each label
    unique_labels = np.unique(labels)
    label_wise_indices = dict()
    for label in unique_labels:
        label_wise_indices.setdefault(label,
                                      [index for index, curr_label in enumerate(labels) if
                                       label == curr_label])

    # Generate image pairs and labels
    pair_images = []
    pair_labels = []
    for index, image in enumerate(images):
        pos_indices = label_wise_indices.get(labels[index])
        pos_image = images[np.random.choice(pos_indices)]
        pair_images.append((image, pos_image))
        pair_labels.append(1)

        neg_indices = np.where(labels != labels[index])
        neg_image = images[np.random.choice(neg_indices[0])]
        pair_images.append((image, neg_image))
        pair_labels.append(0)

    return np.array(pair_images), np.array(pair_labels)
```

Cette fonction a pour objectif de générer des paires d'images pour l'entraînement d'un modèle de reconnaissance faciale. Elle prend en entrée un ensemble d'images et les étiquettes correspondantes.

Génération des Indices pour Chaque Étiquette:

label_wise_indices cette partie de la fonction crée un dictionnaire qui associe à chaque étiquette (label) une liste d'indices correspondant aux images ayant cette étiquette.

Génération des Paires d'Images et des Étiquettes:

Cette section de la fonction itère à travers les images et génère des paires positives et négatives. Pour chaque image, une image positive (de la même étiquette) et une image négative (d'une étiquette différente) sont sélectionnées de manière aléatoire. Les paires d'images sont stockées dans pair_images et les étiquettes correspondantes dans pair_labels.

Retour des Résultats:

La fonction retourne finalement les paires d'images et les étiquettes sous forme de tableaux NumPy.

Chargement des Données du Répertoire :

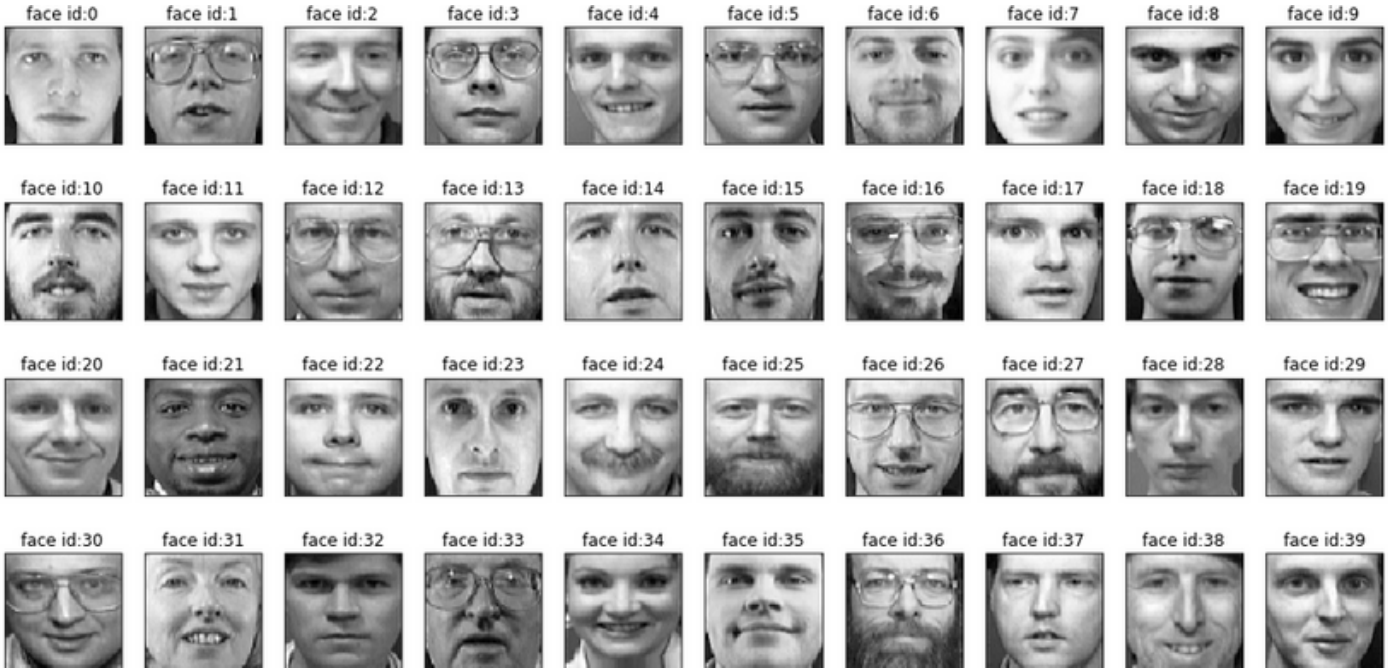
```
# Get the data directory paths
faces = 'olivetti_faces.npy'
faces_targets = 'olivetti_faces_target.npy'
```

Cette partie du code spécifie les chemins vers les fichiers de données. Les fichiers 'olivetti_faces.npy' et 'olivetti_faces_target.npy' contiennent respectivement les images du visage (faces) et les étiquettes (targets) associées à chaque visage dans l'ensemble de données Olivetti Faces.

Chargement des Images et des Étiquettes :

```
# Load the images and labels
face_images = np.load(faces)
face_labels = np.load(faces_targets)
```

Ces deux lignes de code chargent les images du visage et les étiquettes associées à partir des fichiers spécifiés. Ces données seront utilisées pour l'entraînement ou l'évaluation du modèle de reconnaissance faciale. La variable face_images contient les images, et face_labels contient les étiquettes correspondantes.



Détermination de la Forme Cible des Images :

```
[ ] target_shape = face_images[0].shape
```

Cette ligne de code détermine la forme cible des images en accédant à la première image dans le tableau face_images et en récupérant sa forme à l'aide de la propriété shape. La forme de la première image est ensuite assignée à la variable target_shape.

Génération de Paires d'Images et Mélange des Données :

```
images_dataset, labels_dataset = generate_image_pairs(face_images, face_labels)
images_dataset, labels_dataset = shuffle(images_dataset, labels_dataset)
```

Ces deux lignes de code accomplissent deux étapes importantes. Tout d'abord, elles génèrent des paires d'images à partir des images du visage et de leurs étiquettes. Ensuite, elles mélangent ces paires d'images et les étiquettes correspondantes, préparant ainsi les données pour l'entraînement du modèle de reconnaissance faciale. Les données mélangées sont stockées dans images_dataset et labels_dataset.

Fonction de Visualisation des Paires d'Images :

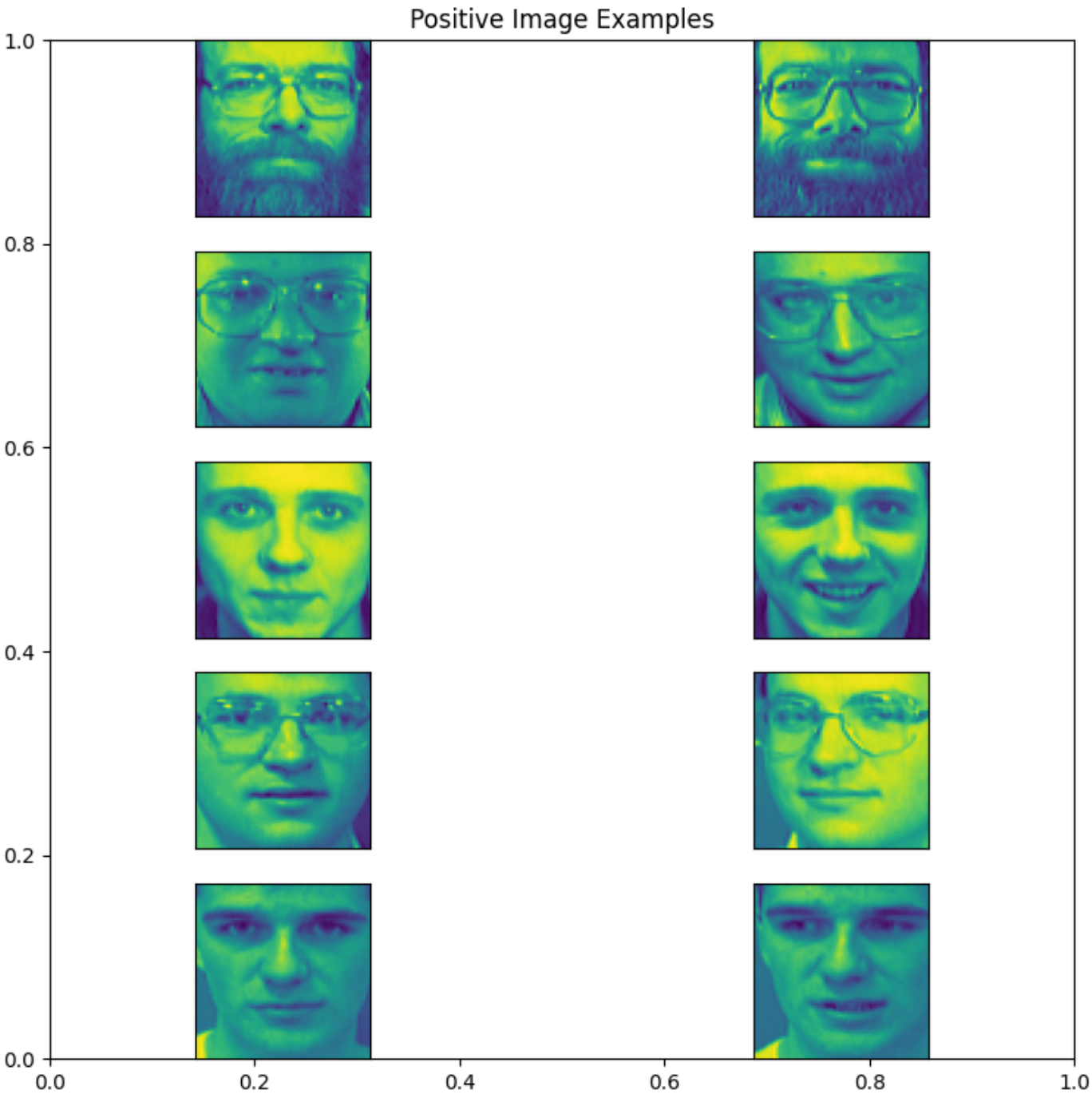
```
def visualize(image_pairs, labels, n = 5, title = "Image Pair Examples"):
    """ Visualize a few pairs """

    def show(ax, image):
        ax.imshow(image)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

    fig = plt.figure(figsize=(9, 9))
    plt.title(title)
    axs = fig.subplots(n, 2)
    for i in range(n):
        show(axs[i, 0], image_pairs[i][0])
        show(axs[i, 1], image_pairs[i][1])
```

Cette fonction est définie pour visualiser quelques paires d'images du jeu de données. Elle prend en entrée les paires d'images (image_pairs), les étiquettes correspondantes (labels), le nombre de paires à afficher (n), et un titre optionnel (title) pour la figure.

- **Paramètres:**
  - image_pairs: Les paires d'images à visualiser.
  - labels: Les étiquettes correspondantes aux paires d'images.
  - n: Le nombre de paires d'images à afficher (par défaut, 5).
  - title: Le titre de la figure (par défaut, "Image Pair Examples").
- **Fonction Interne:**
  - show(ax, image): Une fonction interne utilisée pour afficher une image dans un sous-plot (ax) en supprimant les axes x et y.
- **Création de la Figure et des Sous-Plots:**
  - fig = plt.figure(figsize=(9, 9)): Crée une figure de dimension 9x9 pouces pour afficher les paires d'images.
  - axs = fig.subplots(n, 2): Crée une grille de sous-plots de taille n par 2 pour afficher les paires d'images.
- **Boucle de Visualisation:**
  - La boucle for itère sur les n premières paires d'images et les affiche dans les sous-plots correspondants.

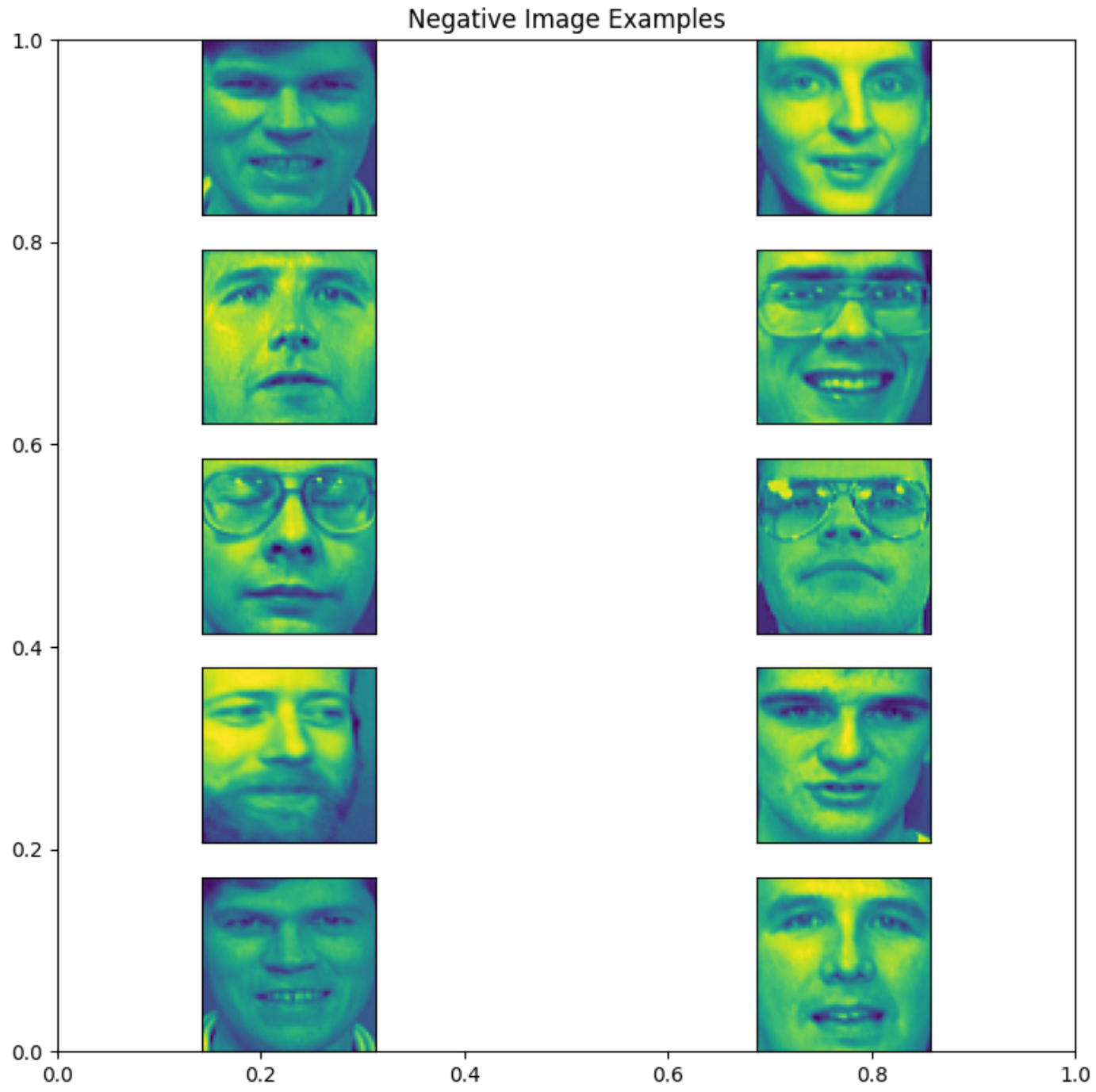
Visualisation d'Images Positives :

```
# Visualize Positive Images
n = 5
indices = np.where(labels_dataset == 1)[:n]
visualize(images_dataset[indices], labels_dataset[indices],
          title = "Positive Image Examples")
```



Visualisation d'Images Négatives :

```
# Visualize Negative Images
n = 5
indices = np.where(labels_dataset == 0)[:n]
visualize(images_dataset[indices], labels_dataset[indices],
          title = "Negative Image Examples")
```



Définition du Modèle d'Embedding pour la Reconnaissance Faciale:

```
inputs = layers.Input((64, 64, 1))

x = layers.Conv2D(64, (10, 10), padding="same", activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(128, (7, 7), padding="same", activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(128, (4, 4), padding="same", activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Dropout(0.3)(x)

x = layers.Conv2D(256, (4, 4), padding="same", activation="relu")(x)
fcOutput = layers.Flatten()(x)
fcOutput = layers.Dense(4096, activation = "relu")(fcOutput)
outputs = layers.Dense(1024, activation = "sigmoid")(fcOutput)

embedding = Model(inputs, outputs, name = "Embedding")
```

Ce code définit un modèle d'embedding pour la reconnaissance faciale. Le modèle est construit en utilisant la bibliothèque TensorFlow/Keras et suit une architecture de réseau convolutionnel (CNN) avec des couches de pooling et de dropout.

- **Architecture du Modèle:**
  - La première couche (inputs) prend des images en entrée avec une taille de (64, 64, 1), indiquant une image en niveaux de gris de 64x64 pixels.
  - Les couches convolutionnelles (Conv2D) suivies de max pooling (MaxPooling2D) sont utilisées pour extraire des caractéristiques de l'image.
  - Des couches de dropout (Dropout) sont utilisées pour régulariser le modèle en réduisant le surajustement.
  - Les couches fully connected (Dense) avec des activations "relu" sont utilisées pour la partie dense du réseau.
  - La dernière couche a une activation "sigmoid" avec une sortie de taille 1024, suggérant qu'il s'agit d'un modèle d'embedding où chaque vecteur d'embedding a une dimension de 1024.
- **Création du Modèle:**
  - embedding = Model(inputs, outputs, name="Embedding"): Le modèle est construit en spécifiant les couches d'entrée (inputs) et de sortie (outputs) pour créer un modèle global appelé "Embedding".

Couche de Distance pour le Calcul des Distances entre les Embeddings :

```python
# Distance Layer
class DistanceLayer(layers.Layer):
    """
    This layer is responsible for computing the distance
    between the embeddings
    """

    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def call(self, anchor, compare):
        sum_squared = K.sum(K.square(anchor - compare), axis=1, keepdims=True)
        return K.sqrt(K.maximum(sum_squared, K.epsilon()))
```

Cette classe définit une couche personnalisée appelée DistanceLayer qui est responsable du calcul de la distance entre les embeddings générés par le modèle. Cette couche prend deux embeddings en entrée (l'ancre et la comparaison) et renvoie la distance euclidienne entre eux.

- **Méthode __init__:**
  - super().__init__(**kwargs): Appelle le constructeur de la classe parente.
- **Méthode call:**
  - def call(self, anchor, compare): La méthode call est appelée lors de l'exécution de la couche sur des données d'entrée.
  - sum_squared = K.sum(K.square(anchor - compare), axis=1, keepdims=True): Calcule la somme des carrés des différences entre les éléments de l'ancre et de la comparaison sur l'axe 1 (somme par dimension) tout en maintenant les dimensions avec keepdims=True.
  - return K.sqrt(K.maximum(sum_squared, K.epsilon())): Calcule la racine carrée de la somme des carrés avec une régularisation minimale (K.epsilon() est ajouté pour éviter la division par zéro).

Création du Modèle Siamese pour la Reconnaissance Faciale :

```
anchor_input = layers.Input(name="anchor", shape=target_shape + (1,))
compare_input = layers.Input(name="compare", shape=target_shape + (1,))

distances = DistanceLayer()(
    embedding(anchor_input),
    embedding(compare_input),
)

outputs = layers.Dense(1, activation = "sigmoid") (distances)

siamese_model = Model(
    inputs=[anchor_input, compare_input], outputs=outputs
)
```

Ce code définit un modèle siamois pour la reconnaissance faciale, utilisant l'idée de partage de poids entre les deux branches du modèle pour les images ancrées et de comparaison.

- **Définition des Entrées:**
  - anchor_input et compare_input: Les entrées du modèle représentant les images ancrées et de comparaison. La forme de ces entrées est définie par target_shape
- **Calcul des Distances:**
  - distances = DistanceLayer() : Utilisation de la couche DistanceLayer pour calculer la distance entre les embeddings générés par le modèle pour les images et de comparaison.
- **Couche de Sortie:**
  - outputs = layers.Dense(1, activation="sigmoid")(distances): Une couche dense avec une activation sigmoïde est utilisée pour prédire si les paires d'images sont similaires ou non (classification binaire).
- **Création du Modèle Siamese:**
  - siamese_model = Model(inputs=[anchor_input, compare_input], outputs=outputs): Création du modèle siamois en spécifiant les entrées et les sorties. Les poids du modèle d'embedding sont partagés entre les deux branches du modèle siamois.

Compilation et Entraînement du Modèle Siamese :

```
[ ] from keras.callbacks import ModelCheckpoint
    model_checkpoint_callback = ModelCheckpoint(
        filepath="model.h5",
        monitor='val_accuracy',
        mode='max',
        save_best_only=True)

    siamese_model.compile(loss="binary_crossentropy", optimizer="rmsprop", metrics=["accuracy"])
    history = siamese_model.fit([X_train[:, 0, :], X_train[:, 1, :]], y_train,
      epochs=150, validation_split = 0.2, batch_size = 64,callbacks=[model_checkpoint_callback])
```

Entraînement du modèle siamois pour la reconnaissance faciale.

- **Compilation du Modèle:**
  - siamese_model.compile(loss="binary_crossentropy", optimizer="rmsprop", metrics=["accuracy"])", metrics=["accuracy"]): Compilation du modèle avec une fonction de perte binaire (binary crossentropy) pour la tâche de classification binaire, l'optimiseur rmsprop, et la métrique d'accuracy pour évaluer les performances.
- **Entraînement du Modèle:**
  - history = siamese_model.fit(...): Entraînement du modèle en fournissant les paires d'images (X_train[:, 0, :] et X_train[:, 1, :]) en tant qu'entrées et les étiquettes (y_train) en tant que sorties. L'entraînement se fait sur 150 époques avec une validation de 20% des données (validation_split=0.2) et une taille de lot de 64 et un callback de type checkpoint pour sauvegarder le meilleure modele obtenu lors de l'entrainement selon val_Accuracy.

```
:==========] - 1s 114ms/step - loss: 0.0872 - accuracy: 0.9844 - val_loss: 0.2973 - val_accuracy: 0.9297
:==========] - 1s 117ms/step - loss: 0.0719 - accuracy: 0.9922 - val_loss: 0.2442 - val_accuracy: 0.9375
:==========] - 1s 114ms/step - loss: 0.0692 - accuracy: 0.9980 - val_loss: 0.2737 - val_accuracy: 0.9453
:==========] - 1s 114ms/step - loss: 0.0573 - accuracy: 1.0000 - val_loss: 0.2967 - val_accuracy: 0.9453
:==========] - 1s 113ms/step - loss: 0.0597 - accuracy: 0.9980 - val_loss: 0.2755 - val_accuracy: 0.9375
```

Les résultats indiquent une précision de 93% lors de l'entrainement, ce qui suggère une performance excellente du modèle.

```
evaluation = siamese_model.evaluate([X_test[:, 0, :], X_test[:, 1, :]], y_test)
print("Test accuracy:", evaluation[1])
```

```
5/5 [==============================] - 1s 17ms/step - loss: 0.3663 - accuracy: 0.8875
Test accuracy: 0.887499988079071
```

Les résultats indiquent une précision de 88.75% lors de validation de modele sur les donnees de test, ce qui suggère une performance excellente du modèle.

Sauvegarde du Modèle Siamese:

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[ ] siamese_model.save("/drive/MyDrive/BioMetrie/model.h5")

    /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
      saving_api.save_model(
```

Conclusion

En conclusion, l'importance d'un système de reconnaissance faciale est indéniable dans divers domaines tels que la sécurité, la gestion des identités et la facilitation des processus automatisés.
En renforçant la sécurité, en simplifiant les procédures d'identification et en améliorant l'efficacité des systèmes automatisés, la reconnaissance faciale offre des avantages significatifs.
Cependant, il est crucial de trouver un équilibre entre les avantages de cette technologie et les préoccupations liées à la protection de la vie privée et à l'utilisation éthique des données. Un cadre réglementaire solide et une transparence accrue sont nécessaires pour garantir que les systèmes de reconnaissance faciale sont déployés de manière responsable et respectueuse des droits individuels.