ANDRIANIRAISANTSOA

Tefy Maherison

M1 MIASHS

2023-2024

PROJET DEEP LEARNGING

Face verification using Siamese Networks

1. Introduction

Actuellement, il est nécessaire de présenter une pièce d'identité dans de nombreuses situations, telles que l'entrée dans un aéroport, l'accès à des zones militaires, le retrait d'argent à un distributeur automatique, le paiement en magasin, l'ouverture d'un compte bancaire, la location de véhicules, l'enregistrement à l'hôtel ou encore la demande de services sociaux. Cela entraîne souvent la nécessité de mémoriser et de gérer de nombreux codes et mots de passe, ce qui peut devenir contraignant et augmenter les risques de sécurité liés à la gestion de ces informations.

Ces mesures d'identification visent à confirmer l'identité unique d'une personne en utilisant des caractéristiques corporelles immuables ou difficilement quantifiables. La biométrie comportementale, en particulier, se concentre sur l'analyse des mouvements et des comportements musculaires, tels que la signature manuscrite, la démarche, la voix, ainsi que les habitudes de frappe au clavier et les gestes sur un écran tactile.

La reconnaissance faciale est un sous-ensemble des méthodes biométriques largement utilisées pour l'identification et l'authentification des individus. En plus de la reconnaissance faciale, d'autres techniques biométriques incluent la lecture des empreintes digitales, la reconnaissance de l'iris, la reconnaissance de la rétine, et l'analyse de la configuration des veines de la main. Chacune de ces technologies offre

différents niveaux de précision et de sécurité, et leur utilisation dépend du contexte et des exigences spécifiques en matière de sécurité.

Les réseaux siamois révolutionnent la sécurité avec la reconnaissance faciale, faisant de votre visage la clé ultime. Cette technologie révolutionnaire promet une efficacité et une sécurité inégalées, en remodelant l'accès et la vérification de l'identité. Nous utilisons déjà cette méthode pour déverrouiller nos téléphones par exemple.

Une question se pose alors : comment fonctionne ce modèle ?

Ce rapport de l'état de l'art scientifique, également appelé revue critique de la littérature a pour objectif :

Dans un premier temps :

- Former un réseau à la similarité des visages à l'aide de réseaux siamois
- L'augmentation des données de travail, générateurs et exploitation minière négative
- Utiliser le modèle sur une photo prise par la caméra de l'ordinateur

Dans un second temps :

- Entraîner un réseau à la similarité des visages en utilisant la perte de triplet
- D'augmenter les données de travail, générateurs et exploitation minière négative

Tout au long du projet, nous utiliserons l'ensemble de données Labeled Faces in the Wild (LFW) disponible librement sur http://vis-www.cs.umass.edu/lfw/

NB : L'absence de GPU sur l'ordinateur nous empêche de réaliser différentes actions

Ce projet approfondit ces concepts importants : la détection d'objets, l'extraction de fonctionnalités, l'API fonctionnelle de TensorFlow et l'architecture de réseau neuronal siamois.

II- Revue de la littérature :

Les réseaux siamois ont été introduits par Bromley et LeCun pour résoudre la vérification de signature en tant que problème de correspondance d'images.

L'idée de base des réseaux siamois est d'avoir deux réseaux de neurones identiques (également appelés réseaux jumeaux ou réseaux frères) avec des poids et une architecture partagée. Chaque réseau prend l'un des deux échantillons d'entrée et produit une intégration (une représentation de faible dimension) pour cet échantillon dans un espace d'intégration commun. Les intégrations des deux échantillons sont ensuite comparées pour calculer un score de similarité ou une métrique de distance. Le réseau est formé à l'aide d'une fonction de perte contrastive/fonction de perte triplet, qui encourage les échantillons similaires à avoir des intégrations proches les unes des autres et des échantillons différents à avoir des intégrations éloignées les unes des autres.

Les réseaux de neurones triplets ont été introduits par Schroff, Kalenichenko et Philbin (2015) dans leur travail fondateur intitulé *FaceNet: A Unified Embedding for Face Recognition and Clustering*. FaceNet propose une méthode novatrice pour la reconnaissance faciale en apprenant directement une représentation euclidienne des visages à partir de photographies. L'architecture repose sur l'apprentissage d'un espace d'embeddings où les images similaires sont rapprochées et les images dissimilaires sont éloignées. Ce concept est mis en œuvre à l'aide de la *triplet loss*, qui mesure la distance relative entre un ancrage, un positif (une image de la même classe que l'ancrage) et un négatif (une image d'une classe différente).

Exemple d’utilisation :

1. **Système d'identification numérique**

Tout d’abord, l'utilisateur prend sa photo et enregistre ses informations lors de l'inscription.

Par la suite, le système obtient le codage facial à l’aide du réseau siamois.

Les encodages seront enregistrés avec leurs identités dans la base de données.

Par ailleurs, lors de la saisie, le visage/les données biométriques de l'utilisateur sont collectées et les encodages biométriques sont calculés à l'aide du réseau siamois.

De ce fait, la distance entre les nouvelles valeurs de codage et chaque valeur de codage dans la base de données est calculée.

L'encodage avec la plus petite distance est récupéré et l'identité de l'utilisateur sera reconnue.

2. Le clustering de documents

Les réseaux Siamois peuvent détecter les faux documents en double ou contrefaits. Lorsqu'il s'agit d'un référentiel central de documents, l'approche conventionnelle nécessite des comparaisons NxN (comparer les documents de chaque utilisateur avec tous les autres utilisateurs) pour identifier les faux documents. Cependant, les réseaux siamois rationalisent ce processus en regroupant efficacement les documents similaires. Cela nous permet de détecter efficacement les cas où l'identité d'un utilisateur a été falsifiée par une autre personne.

Méthodologie :

1. Utilisation GPU/CPU

On commence par la configuration du GPU à utiliser pour le projet. Malheureusement, par absence de GPU sur l'ordinateur, nous allons utiliser le CPU.

2. Prétraitement des données :

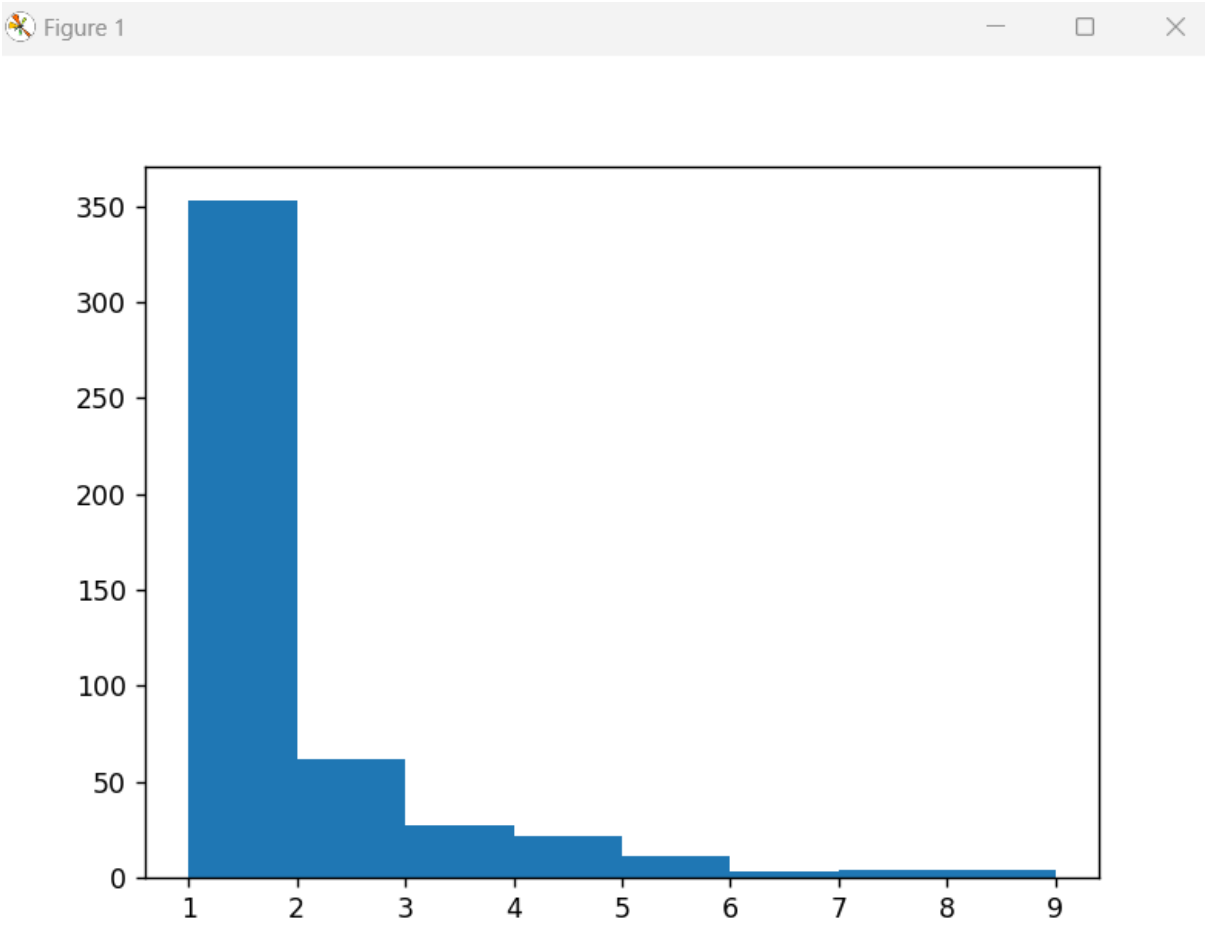
Le jeu de données de visages est chargé à partir du répertoire spécifié. Une seule partie du jeu de données est utilisée pour accélérer les expérimentations.

- Les noms des classes (personnes) sont mappés à des identifiants numériques.
- Les chemins des images sont collectés et mappés à des identifiants.

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input, Concatenate, Lambda, Dot
from tensorflow.keras.layers import Conv2D, MaxPool2D, GlobalAveragePooling2D, Flatten, Dropout
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

#processing the dataset
PATH = "lfw-deepfunneled/"
USE_SUBSET = True

dirs = sorted(os.listdir(PATH))
if USE_SUBSET:
    dirs = dirs[:500]

name_to_classid = {d: i for i, d in enumerate(dirs)}
classid_to_name = {v: k for k, v in name_to_classid.items()}
num_classes = len(name_to_classid)

print("number of classes: ", num_classes)

#mapping all images
# read all directories
img_paths = {c: [PATH + subfolder + "/" + img
                 for img in sorted(os.listdir(PATH + subfolder))]
             for subfolder, c in name_to_classid.items()}
```

3. Visualisation des données :

Une distribution des images par classe est visualisée pour comprendre la répartition des données.



4. Construction des paires d'images :

Nous avons défini des fonctions pour construire des paires 'images positives (de même classe) et négatives (classes différentes)

```
# build pairs of positive image ids for a given classid
def build_pos_pairs_for_id(classid, max_num=50):
    imgs = classid_to_ids[classid]

    if len(imgs) == 1:
        return []

    pos_pairs = list(itertools.combinations(imgs, 2))

    random.shuffle(pos_pairs)
    return pos_pairs[:max_num]

# build pairs of negative image ids for a given classid
def build_neg_pairs_for_id(classid, classes, max_num=20):
    imgs = classid_to_ids[classid]
    neg_classes_ids = random.sample(classes, max_num+1)

    if classid in neg_classes_ids:
        neg_classes_ids.remove(classid)

    neg_pairs = []
    for id2 in range(max_num):
        img1 = imgs[random.randint(0, len(imgs) - 1)]
        imgs2 = classid_to_ids[neg_classes_ids[id2]]
        img2 = imgs2[random.randint(0, len(imgs2) - 1)]
        neg_pairs += [(img1, img2)]

    return neg_pairs

build_pos_pairs_for_id(5, max_num=10)
build_neg_pairs_for_id(5, list(range(num_classes)), max_num=6)
```

5. Chargement et redimensionnement des images :

Maintenant que nous avons un moyen de calculer les paires, chargeons tous les fichiers image compressés JPEG possibles dans un seul tableau numpy en RAM Les images sont chargées et redimensionnées pour être utilisées par le modèle.

```
from skimage.io import imread
from skimage.transform import resize


def resize100(img):
    return resize(
        img, (100, 100), preserve_range=True, mode='reflect', anti_aliasing=True
    )[20:80, 20:80, :]


def open_all_images(id_to_path):
    all_imgs = []
    for path in id_to_path.values():
        all_imgs += [np.expand_dims(resize100(imread(path)), 0)]
    return np.vstack(all_imgs)
```

6. Données d’entraînement et de tests :

On commence par créer crée des ensembles de données d'entraînement et de test en générant des paires d'images positives (même classe) et négatives (classes différentes).

La proportion de données à utiliser pour l'entraînement est de 80% pour l’entrainement et 20% pour les tests.

Pour chaque classe des données d’entrainement, nous effectuons une boucle pour générer les paires positives et négatives.

Par la suite, nous ajoutons les paires aux listes en respectant leur label respectif.

Entraînement :

```
def build_train_test_data(split=0.8):
    listX2 = []
    listY = []
    split = int(num_classes * split)

    # train
    for class_id in range(split):
        pos = build_pos_pairs_for_id(class_id)
        neg = build_neg_pairs_for_id(class_id, list(range(split)))
        for pair in pos:
            listX1 += [pair[0]]
            listX2 += [pair[1]]
            listY += [1]
        for pair in neg:
            if sum(listY) > len(listY) / 2:
                listX1 += [pair[0]]
                listX2 += [pair[1]]
                listY += [0]
    perm = np.random.permutation(len(listX1))
    X1_ids_train = np.array(listX1)[perm]
    X2_ids_train = np.array(listX2)[perm]
    Y_ids_train = np.array(listY)[perm]

    listX1 = []
    listX2 = []
    listY = []
```

Test :

```
    #test
    for id in range(split, num_classes):
        pos = build_pos_pairs_for_id(id)
        neg = build_neg_pairs_for_id(id, list(range(split, num_classes)))
        for pair in pos:
            listX1 += [pair[0]]
            listX2 += [pair[1]]
            listY += [1]
        for pair in neg:
            if sum(listY) > len(listY) / 2:
                listX1 += [pair[0]]
                listX2 += [pair[1]]
                listY += [0]
    X1_ids_test = np.array(listX1)
    X2_ids_test = np.array(listX2)
    Y_ids_test = np.array(listY)
    return (X1_ids_train, X2_ids_train, Y_ids_train,
            X1_ids_test, X2_ids_test, Y_ids_test)
```

```
X1_ids_train, X2_ids_train, train_Y, X1_ids_test, X2_ids_test, test_Y = build_train_test_data()
X1_ids_train.shape, X2_ids_train.shape, train_Y.shape
np.mean(train_Y)
X1_ids_test.shape, X2_ids_test.shape, test_Y.shape
np.mean(test_Y)
```

7. Augmentation des données:

Des augmentations d'images sont appliquées pour améliorer la robustesse du modèle. Un générateur Keras est utilisé pour produire des lots d'images.

Concrètement, ici nous avons

- Recadrer certaines images de 0 à 10 % de leur hauteur/largeur
- Retourner horizontalement 50 % des images
- Retourner verticalement 20 % de toutes les images
- Améliorer ou aggraver le contraste des images.

```python
#data Augmentation and generator
from imgaug import augmenters as iaa
import numpy as np

import imgaug.augmenters as iaa

# Sometimes(0.5, ...) applies the given augmenter in 50% of all cases,
# e.g. Sometimes(0.5, GaussianBlur(0.3)) would blur roughly every second
# image.
sometimes = lambda aug: iaa.Sometimes(0.5, aug)

seq = iaa.Sequential([
    iaa.Fliplr(0.5), # horizontally flip 50% of the images
    #iaa.Flipud(0.2), # vertically flip 20% of all images
    # Improve or worsen the contrast of images.
    iaa.LinearContrast((0.5, 2.0), per_channel=0.5),
    # crop some of the images by 0-10% of their height/width
    sometimes(iaa.Crop(percent=(0, 0.1))),



    # You can add more transformation like random rotations, random change of luminance, etc.
])
```

Le générateur de données quant à lui permet de produire des lots de données augmentées de manière efficace pendant l'entraînement.

A la sortie, nous avons les affichages des images augmentées avec leurs labels.

8. Modèle Siamois :

Comme expliqué auparavant, le modèle siamois utilise un réseau convolutionnel partagé pour encoder les images. La similarité est calculée via un produit scalaire normalisé.

Ce modèle utilise une fonction de perte contrastive qui encourage les représentations des images similaires à être proches et celles des images différentes à être éloignées.

```
#Simple convolutional model
@tf.function
def contrastive_loss(y_true, y_pred, margin=0.25):
    '''Contrastive loss from Hadsell-et-al.'06
    http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf
    '''
    y_true = tf.cast(y_true, "float32")
    return tf.reduce_mean( y_true * tf.square(1 - y_pred) +
                           (1 - y_true) * tf.square(tf.maximum(y_pred - margin, 0)))
```

Cette précision est calculée en utilisant un seuil fixe sur la similarité.

```
@tf.function
def accuracy_sim(y_true, y_pred, threshold=0.5):
    '''Compute classification accuracy with a fixed threshold on similarity.
    '''
    y_thresholded = tf.cast(y_pred > threshold, "float32")
    return tf.reduce_mean(tf.cast(tf.equal(y_true, y_thresholded), "float32"))
```

Afin de permettre l'extraction des caractéristiques des images, ce modèle utilise une convulsion partagée.

```python
class SharedConv(tf.keras.Model):
    def __init__(self):
        super(SharedConv, self).__init__(name="sharedconv")

        # Define the layers
        self.conv1 = Conv2D(32, (3, 3), activation='relu')
        self.pool1 = MaxPool2D((2, 2))
        self.conv2 = Conv2D(64, (3, 3), activation='relu')
        self.pool2 = MaxPool2D((2, 2))
        self.conv3 = Conv2D(128, (3, 3), activation='relu')
        self.pool3 = MaxPool2D((2, 2))
        self.flatten = Flatten()
        self.dropout = Dropout(0.5)
        self.dense = Dense(128, activation='relu')

    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.pool2(x)
        x = self.conv3(x)
        x = self.pool3(x)
        x = self.flatten(x)
        x = self.dropout(x)
        x = self.dense(x)
        return x

shared_conv = SharedConv()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	?	896
max_pooling2d (MaxPooling2D)	?	0
conv2d_1 (Conv2D)	?	18,496
max_pooling2d_1 (MaxPooling2D)	?	0
conv2d_2 (Conv2D)	?	73,856
max_pooling2d_2 (MaxPooling2D)	?	0
flatten (Flatten)	?	0
dropout (Dropout)	?	0
dense (Dense)	?	409,728

Total params: 502,976 (1.92 MB)
Trainable params: 502,976 (1.92 MB)
Non-trainable params: 0 (0.00 B)

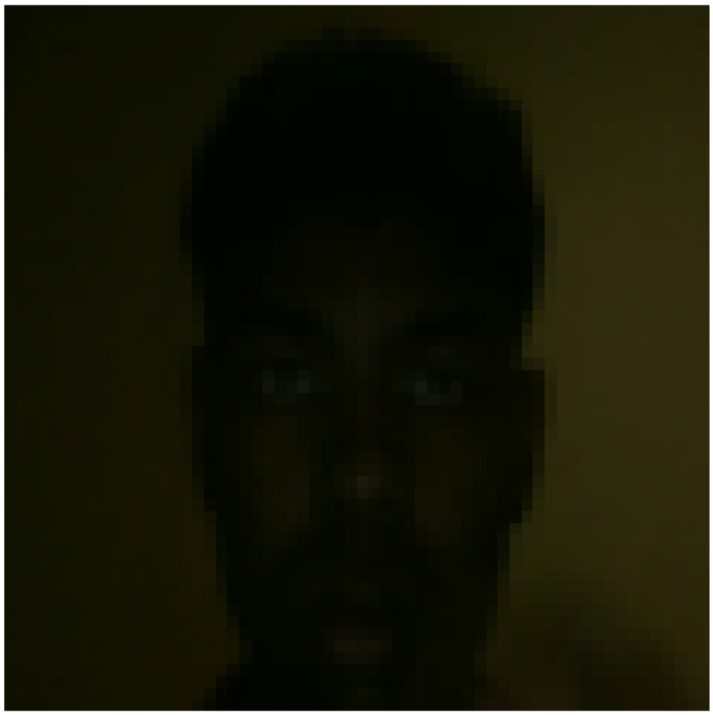
```
query: Amelia_Vega 462
nearest matches
Amelia_Vega 462 1.0
Alanis_Morissette 199 0.984093
Alvaro_Silva_Calderon 406 0.9820434
Ari Fleischer 800 0.9806597
```

De ce fait, le modèle siamois compare deux images en utilisant les caractéristiques extraites par la convulsion partagé et calcule leur similarité avec un produit scalaire normalisé.

9. Utilisation de la caméra pour récupérer la photo de l'utilisateur

```python
import cv2

def camera_grab(camera_id=0, fallback_filename=None):
    camera = cv2.VideoCapture(camera_id)
    try:
        # take 10 consecutive snapshots to let the camera automatically tune
        # itself and hope that the contrast and lightning of the last snapshot
        # is good enough.
        for i in range(10):
            snapshot_ok, image = camera.read()
        if snapshot_ok:
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        else:
            print("WARNING: could not access camera")
            if fallback_filename:
                image = imread(fallback_filename)
    finally:
        camera.release()
    return image

image = camera_grab(camera_id=0,
                    fallback_filename='test_images/olivier/img_olivier_0.jpeg')
x = resize100(image)
out = shared_conv(np.reshape(x, (1, 60, 60, 3)))
print("query image:")
```

Figure 1



```
import cv2

def camera_grab(camera_id=0, fallback_filename=None):
    camera = cv2.VideoCapture(camera_id)
    try:
        # take 10 consecutive snapshots to let the camera automatically tune
        # itself and hope that the contrast and lightning of the last snapshot
        # is good enough.
        for i in range(10):
            snapshot_ok, image = camera.read()
        if snapshot_ok:
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        else:
            print("WARNING: could not access camera")
            if fallback_filename:
                image = imread(fallback_filename)
    finally:
        camera.release()
    return image

image = camera_grab(camera_id=0,
                    fallback_filename='test_images/olivier/img_olivier_0.jpeg')
x = resize100(image)
out = shared_conv(np.reshape(x, (1, 60, 60, 3)))
print("query image:")
```

10. Explication des résultats :

Cette section montre les résultats de la recherche de correspondances pour une image de requête donnée :

- Requête :

  query: Amelia_Vega 462 : L'image de requête est l'image numéro 462 appartenant à la classe Amelia_Vega.
- Correspondances les Plus Proches :

  Amelia_Vega 462 1.0 : L'image de requête elle-même, avec une similarité de 1.0 (ce qui est attendu car elle est identique à la requête).

  Alanis_Morissette 199 0.984093 : Image de la classe Alanis_Morissette avec une similarité de 0.984093.

  Alvaro_Silva_Calderon 406 0.9820434 : Image de la classe Alvaro_Silva_Calderon avec une similarité de 0.9820434.

  Ari_Fleischer 800 0.9806597 : Image de la classe Ari_Fleischer avec une similarité de 0.9806597.

TRIPLET LOSS :

Méthodologie :

Nous utiliserons les mêmes données qu'auparavant. Nous mettrons en avant les différentes méthodes de ce modèle.

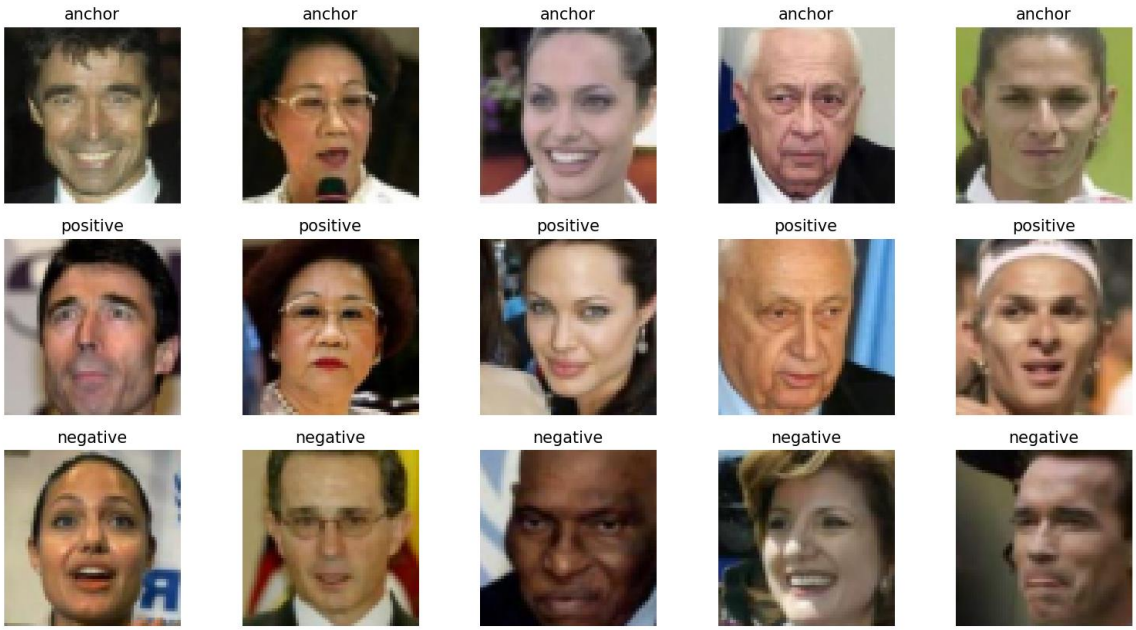
Contrairement à la méthode précédente, ici, le réseau siamois prend un triplé en entrée (une image d'ancrage, une image positive, une image négative). L'ancre est l'échantillon d'entrée pour lequel nous voulons apprendre son intégration.

L'image positive est un échantillon similaire à l'image d'ancrage et appartenant à la même classe (par exemple, même personne, pose différente).

L'image négative est un échantillon d'une classe différente, et elle est différente de l'image d'ancrage (par exemple, une personne différente). Le réseau est entraîné à l'aide de la fonction de perte triplet de telle sorte que la distance avec l'image positive soit minimisée et la distance avec l'image négative soit maximisée.

Génération des triplés :

```python
class TripletGenerator(tf.keras.utils.Sequence):
    def __init__(self, Xa_train, Xp_train, batch_size, all_imgs, neg_imgs_idx):
        self.cur_img_index = 0
        self.cur_img_pos_index = 0
        self.batch_size = batch_size

        self.imgs = all_imgs
        self.Xa = Xa_train  # Anchors
        self.Xp = Xp_train
        self.cur_train_index = 0
        self.num_samples = Xa_train.shape[0]
        self.neg_imgs_idx = neg_imgs_idx

    def __len__(self):
        return self.num_samples // self.batch_size

    def __getitem__(self, batch_index):
        low_index = batch_index * self.batch_size
        high_index = (batch_index + 1) * self.batch_size

        imgs_a = self.Xa[low_index:high_index]  # Anchors
        imgs_p = self.Xp[low_index:high_index]  # Positives
        imgs_n = random.sample(self.neg_imgs_idx, imgs_a.shape[0])  # Negatives

        imgs_a = seq.augment_images(self.imgs[imgs_a])
        imgs_p = seq.augment_images(self.imgs[imgs_p])
        imgs_n = seq.augment_images(self.imgs[imgs_n])

        # We also a null vector as placeholder for output, but it won't be needed:
        return ([imgs_a, imgs_p, imgs_n], np.zeros(shape=(imgs_a.shape[0])))
```



En résumé, dans cette partie, le modèle utilise un réseau de neurones convolutifs pour extraire des caractéristiques des images de visages, et entraîne un modèle de triplets

pour apprendre des représentations discriminantes des visages en minimisant la perte triplet.

Références :

oneshot1.pdf (cmu.edu)

Learn to Build a Siamese Neural Network for Image Similarity (projectpro.io)

lectures-labs/labs/09_triplet_loss/Face_Verification_Using_Triplet_Loss.ipynb at master · m2dsupsdlclass/lectures-labs (github.com)

lectures-labs/labs/09_triplet_loss/Face_Verification_Using_Siamese_Nets.ipynb at master · m2dsupsdlclass/lectures-labs (github.com)

Building a Siamese Neural Network for Face Verification: A Comprehensive Guide | by Issam Jebnouni | Medium

Siamese Network – your face is the key – Fingerprints