

Modern face recognition

MANG Nathan
IA et Deep Learning
ESIEE - 2024

France

nathan.mang@edu.esiee.fr

LECOIN Nathan
IA et Deep Learning
ESIEE - 2024

France

nathan.lecoin@edu.esiee.fr

Abstract— Ce document rapporte la mise en œuvre d'un système de reconnaissance faciale. L'objectif final de ce projet est d'utiliser des méthodes de Deep Learning afin de concevoir un système bien entraîné. La difficulté est d'arriver à entraîner notre modèle pour qu'il fasse les bonnes prédictions à partir d'un faible jeu de données. Dans cette étude la reconnaissance faciale s'effectuera par le biais d'un réseau de neurones convolutifs. Toutefois, cette tâche ne sera pas aisée. En effet, nous serons confrontés à différentes contraintes comme la détection du visage, l'estimation ou encore l'encodage du visage. Nous verrons donc comment tacler ses contraintes et obtenir un modèle performant, notamment en utilisant des algorithmes comme l'estimation des points de repère.

Index Terms — Deep Learning, landmarks, pose estimation, face encoding, ConvNet.

I. INTRODUCTION

La reconnaissance faciale est une technologie fascinante et en constante évolution qui joue un rôle croissant dans de nombreux aspects de notre vie quotidienne. De la sécurité à la surveillance, en passant par la gestion de l'identité et l'authentification biométrique, cette technologie offre un large éventail d'applications prometteuses et suscite un intérêt croissant dans les domaines de la recherche, de l'industrie et de la société en général.

Au cœur de la reconnaissance faciale réside la capacité des systèmes informatiques à identifier et à vérifier les individus en se basant sur les caractéristiques uniques de leur visage. Cette technologie repose sur des algorithmes sophistiqués qui analysent les caractéristiques faciales telles que la forme des yeux, du nez et de la bouche, ainsi que la disposition des traits distinctifs, pour créer des modèles numériques représentant les visages des individus.

La reconnaissance faciale est un des enjeux majeurs de notre société actuelle. En effet, aujourd'hui de nombreuses grandes entreprises mondiales comme Facebook utilisent des algorithmes de Deep Learning pour reconnaître des visages avec une très haute précision.

Dans ce rapport, nous verrons comment élaborer un modèle de reconnaissance faciale en quatre étapes clefs. Pour chacune de ces étapes, nous utiliserons des méthodes de Deep Learning bien précises pour améliorer les performances de notre modèle

II. RELATED WORKS

La reconnaissance faciale a gagné en importance ces dernières années en raison de ses applications potentielles dans la sécurité, la surveillance et la gestion des identités. Alors que les progrès de l'apprentissage profond ont révolutionné le domaine, les approches traditionnelles basées sur des caractéristiques définies à la main continuent de jouer un rôle crucial.

[1] Zhao et al. (2003) ont proposé une méthode de reconnaissance faciale utilisant les motifs binaires locaux (LBP) et l'analyse en composantes principales (ACP). Le LBP extrait des caractéristiques texturales locales des images faciales, capturant les relations spatiales entre les pixels. Enfin, un classificateur par plus proche voisin est utilisé pour l'identification. Les résultats sont d'une précision de 95.8 sur la base de données de FERET. Cependant ce modèle est sensible aux variations de pose et les expressions faciales peuvent affecter l'extraction des caractéristiques.

[2] Sun et al. (2014) ont utilisé une architecture d'apprentissage profond pour la reconnaissance faciale, combinant des réseaux de neurones convolutifs (CNN) et des réseaux de neurones récurrents (RNN). Cette architecture permet d'améliorer la robustesse aux variations de pose. Cette étude obtient des performances supérieures avec une précision de 98.3 sur la base de données LFW. L'inconvénient de ce modèle est qu'il nécessite beaucoup de données d'entraînement.

[3] Martinez et Acuña (2002) utilisent une méthode basée sur l'apparence locale pour la reconnaissance faciale avec une seule image d'apprentissage par personne. Le principe est de diviser le visage en six régions. Pour chaque région, un modèle probabiliste capture la variabilité au sein

de cette région. Cette technique a obtenu des résultats prometteurs sur un ensemble de données de 2600 images, démontrant une robustesse aux variations d'expression. Cependant, une limitation potentielle réside dans le coût de calcul associé à l'apprentissage des modèles complexes.

notre cas les noms des personnages. Pour ce faire, nous parcourons le chemin d'accès aux images, nous lisons l'image et nous extrayons le visage ainsi que le nom. Nous ajoutons le label de l'image souhaité dans une nouvelle liste. L'étape suivante consiste à normaliser les images (diviser chaque pixel par 255) afin d'améliorer les performances du modèle.

III. PROPOSITION

A. Face detection

Avant de pouvoir différencier les visages, il serait intéressant de pouvoir les localiser. Dans un premier temps, notre travail consiste à détecter le visage sur l'image. Il sera beaucoup plus simple pour notre système de travailler avec des images où le visage est bien localisé. La détection de visage est souvent

La fonction *face_locations* pour détecter les visages nous est donnée. Cette fonction prend une image en entrée et retourne une liste de rectangles encadrant les visages détectés dans cette image, en utilisant soit le détecteur HOG soit le détecteur CNN selon le modèle spécifié.

La seconde fonction importante est *extract_faces*. Cette fonction est conçue pour extraire et redimensionner les visages détectés dans une image en utilisant la fonction *face_locations* définie précédemment. Ensuite elle extrait les régions correspondant aux visages détectés, les redimensionne à une taille fixe de 128x128 pixels, puis retourne la liste des visages extraits.

Enfin la fonction *list_images* parcourt une structure de répertoires à partir d'un chemin de base donné, trouve tous les fichiers d'images ayant des extensions valides spécifiées et éventuellement contenant une chaîne spécifique dans leur nom et retourne une liste de chemins d'accès vers ces fichiers d'images.

Pour commencer nous avons récupéré notre dataset dans un dossier nommé 'data'. En le téléchargeant sur l'ordinateur nous avons compris qu'il s'agissait d'images de personnages de la licence de films Jurassic Park. Nous avons commencé à parcourir la liste des noms associés aux images et nous les avons stockées dans une liste nommée *list_names*. En affichant la taille de cette liste, il y a 218 images.

Maintenant que nous connaissons notre dataset nous allons extraire les visages de nos images et les labels associés, dans



Maintenant que nos images sont cropées et normalisées, il nous reste plus qu'à créer notre modèle. Avant de le créer il est nécessaire de séparer les données en deux jeux différents. Le premier sera celui d'entraînement (70%) tandis que le second sera celui de validation (30%). Pour séparer notre jeu de données nous utiliserons la fonction *train_test_split* de la librairie sklearn.

Egalement nous importons la librairie pickle pour sauvegarder nos jeux de données dans un fichier sans avoir à les télécharger à chaque ouverture du notebook.

Pour augmenter la taille de nos données nous utilisons la technique de data augmentation. Cette technique vue en cours est très efficace pour améliorer les performances des modèles. Pour notre cas d'application ce sera très utile pour augmenter le nombre d'images.

La dernière étape avant d'écrire le modèle est de transformer nos images en batch pour l'entraînement. Nous créons notre générateur de batch à partir des données d'entraînement et des étiquettes encodées. La taille de nos batch sera de 32.

Le plus gros du travail pour cette partie est fait, il nous reste maintenant à définir notre modèle. Les ConvNet sont particulièrement adaptés à l'analyse d'images, même sur de petits ensembles de données. C'est pourquoi nous avons commencé par former un petit ConvNet pour avoir une base de référence avec une seule couche alternée 32 et se terminant avec une couche 6 unités avec activation = 'softmax' puisque nous sommes dans une classification catégorielle. Nous avons également modifié la taille des couches.

```

# Define the network
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128, (3,3), activation = 'relu'))
#model.add(layers.Dropout(0.5))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation = 'relu'))
model.add(layers.Dense(6, activation = 'softmax'))

model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(learning_rate=1e-4), metrics=['acc'])

# Train the network. Hint: use .fit(...)

history = model.fit(train_generator, epochs=50, validation_data = (np.array(X_validation), Y_validation_encoded))

# Evaluation du modèle sur les données de validation et obtention de la perte (loss) et de la précision (accuracy)
test_loss, test_acc = model.evaluate(np.array(X_validation), Y_validation_encoded)

print('Validation accuracy: {:.2f}%'.format(test_acc*100))

```

Nous avons utilisé un optimizer « `categorical_crossentropy` » et nous avons entraîné le modèle avec 50 itérations. Pour une première version, le modèle affiche une précision de **70%** et nous n'observons pas d'overfitting. Ce score d'accuracy peut être augmenté.

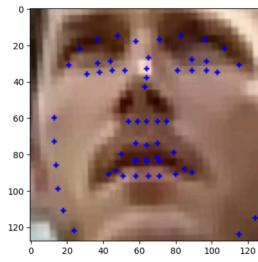
B. Pose estimation

Dans la première partie du projet nous avons extrait les visages. Cependant tourner les visages dans des directions différentes à un aspect totalement différent pour un ordinateur. L'idée dans cette seconde partie est de déformer chaque image de manière à ce que les yeux et les lèvres se trouvent toujours au même endroit dans l'image.

Nous allons donc utiliser un algorithme nommé "estimation des points de repère du visage" (face landmark estimation). Cet algorithme consiste à localiser 68 points spécifiques (appelés points de repère) qui existent sur chaque visage : le haut du menton, le bord extérieur de chaque œil, le bord intérieur de chaque sourcil, etc. Il suffit ensuite de faire pivoter, de mettre à l'échelle et de cisailler l'image pour que les yeux et la bouche soient centrés le mieux possible. La reconnaissance des visages sera ainsi plus précise.

La première fonction utile qui nous est donnée est `face_landmarks`. Cette fonction extrait les landmarks (points caractéristiques) du visage à partir d'images en utilisant la bibliothèque dlib.

La deuxième fonction que nous utiliserons est `align_faces`. Cette fonction prend une liste d'images de visages et une liste de landmarks correspondants, puis aligne les visages en utilisant une transformation affine basée sur les landmarks spécifiés. Nous parcourons donc nos images extraites et nous leur appliquons ces deux méthodes.



Le premier jeu de données sera celui d'entraînement (70%) tandis que le second sera celui de validation (30%).

Pour la compilation, nous avons repris notre même modèle que dans la première partie.

Résultat obtenu : Nous avons utilisé un optimizer « `categorical_crossentropy` » et nous avons entraîné le modèle avec 50 itérations. Pour ce modèle, on obtient une précision de **71,4%**

Cette légère amélioration est donc liée à la prise en compte de la pose du visage.

Les visages ont tous une pose similaire suite à l'estimation de la pose.

De ce fait, il est plus facile au ConvNets de rapprocher deux visages d'une même personne puisque les différences de pose sont atténuées. C'est la raison pour laquelle on obtient une meilleure précision.

C. Face encoding

Jusqu'à présent notre reconnaissance faciale consistait à classer directement un visage inconnu à l'aide d'un ConvNet formé sur une base de données de personnes étiquetées. Le souci de cette approche est qu'il serait difficile d'entraîner le réseau pour des milliards d'êtres humains.

Le face encoding est une technique utilisée en reconnaissance faciale pour représenter numériquement les caractéristiques distinctives du visage d'une personne. L'avantage du face encoding est qu'il permet une reconnaissance plus efficace des visages car il compare des vecteurs numériques compacts plutôt que de comparer directement les images de visages qui peuvent varier en termes de luminosité, d'angle de vue et de qualité d'image.

Nous allons utiliser ici un réseau créé par OpenFace déjà entraîné à générer 128 mesures pour chaque visage.

Après avoir encodé nos images sous la forme d'un vecteur de 128 mesures, nous utilisons un réseau de couches entièrement connectées pour entraîner ce vecteur.

```

Epoch 59/70
5/5 [=====] - 0s 21ms/step - loss: 0.0017 - acc: 1.0000 - val_loss: 0.0415 - val_acc: 0.9841
Epoch 60/70
5/5 [=====] - 0s 32ms/step - loss: 0.0017 - acc: 1.0000 - val_loss: 0.0414 - val_acc: 0.9841
Epoch 61/70
5/5 [=====] - 0s 24ms/step - loss: 0.0016 - acc: 1.0000 - val_loss: 0.0417 - val_acc: 0.9841
Epoch 62/70
5/5 [=====] - 0s 49ms/step - loss: 0.0016 - acc: 1.0000 - val_loss: 0.0418 - val_acc: 0.9841
Epoch 63/70
5/5 [=====] - 0s 52ms/step - loss: 0.0015 - acc: 1.0000 - val_loss: 0.0418 - val_acc: 0.9841
Epoch 64/70
5/5 [=====] - 0s 37ms/step - loss: 0.0015 - acc: 1.0000 - val_loss: 0.0417 - val_acc: 0.9841
Epoch 65/70
5/5 [=====] - 0s 43ms/step - loss: 0.0014 - acc: 1.0000 - val_loss: 0.0418 - val_acc: 0.9841
Epoch 66/70
5/5 [=====] - 0s 36ms/step - loss: 0.0014 - acc: 1.0000 - val_loss: 0.0418 - val_acc: 0.9841
Epoch 67/70
5/5 [=====] - 0s 36ms/step - loss: 0.0014 - acc: 1.0000 - val_loss: 0.0415 - val_acc: 0.9841
Epoch 68/70
5/5 [=====] - 0s 35ms/step - loss: 0.0013 - acc: 1.0000 - val_loss: 0.0414 - val_acc: 0.9841
Epoch 69/70
5/5 [=====] - 0s 38ms/step - loss: 0.0013 - acc: 1.0000 - val_loss: 0.0412 - val_acc: 0.9841
Epoch 70/70
5/5 [=====] - 0s 48ms/step - loss: 0.0013 - acc: 1.0000 - val_loss: 0.0412 - val_acc: 0.9841
2/2 [=====] - 0s 8ms/step - loss: 0.0412 - acc: 0.9841

```

On peut voir cette fois que notre modèle est très performant puisqu'il atteint un score de précision de **98%**. Au vu du graphique, les paramètres ont bien été choisis

D. Face recognition

Dans cette partie, nous avons utilisé et entraîné les classifiers suivants logistic régression, SVM (Support vector machine), kNN (K-nearest neighbors avec 5 voisins) et neural network.

L'idée est de pouvoir appliquer la reconnaissance faciale sur des séquences vidéo.

La librairie cv2 rencontrait des soucis avec Google Colab. On a donc utilisé l'import suivant et adapté notre fonction `process_movie` pour la vidéo.

```
from google.colab.patches import cv2_imshow
```

Nous avons ensuite utilisé les différents classifiers afin de déterminer lequel était le plus performant à notre cas d'étude.

```

Epoch 59/70
10/10 [=====] - 0s 6ms/step - loss: 0.1964 - acc: 0.9931 - val_loss: 0.2286 - val_acc: 1.0000
Epoch 60/70
10/10 [=====] - 0s 6ms/step - loss: 0.1888 - acc: 0.9931 - val_loss: 0.2196 - val_acc: 1.0000
Epoch 61/70
10/10 [=====] - 0s 6ms/step - loss: 0.1833 - acc: 0.9931 - val_loss: 0.2069 - val_acc: 1.0000
Epoch 62/70
10/10 [=====] - 0s 6ms/step - loss: 0.1766 - acc: 0.9931 - val_loss: 0.2032 - val_acc: 1.0000
Epoch 63/70
10/10 [=====] - 0s 6ms/step - loss: 0.1652 - acc: 0.9931 - val_loss: 0.1916 - val_acc: 1.0000
Epoch 64/70
10/10 [=====] - 0s 6ms/step - loss: 0.1580 - acc: 0.9931 - val_loss: 0.1829 - val_acc: 1.0000
Epoch 65/70
10/10 [=====] - 0s 6ms/step - loss: 0.1519 - acc: 0.9931 - val_loss: 0.1740 - val_acc: 1.0000
Epoch 66/70
10/10 [=====] - 0s 8ms/step - loss: 0.1443 - acc: 0.9931 - val_loss: 0.1675 - val_acc: 1.0000
Epoch 67/70
10/10 [=====] - 0s 8ms/step - loss: 0.1395 - acc: 0.9931 - val_loss: 0.1607 - val_acc: 1.0000
Epoch 68/70
10/10 [=====] - 0s 6ms/step - loss: 0.1329 - acc: 0.9931 - val_loss: 0.1567 - val_acc: 1.0000
Epoch 69/70
10/10 [=====] - 0s 8ms/step - loss: 0.1273 - acc: 0.9931 - val_loss: 0.1492 - val_acc: 1.0000
Epoch 70/70
10/10 [=====] - 0s 8ms/step - loss: 0.1224 - acc: 0.9931 - val_loss: 0.1430 - val_acc: 1.0000
2/2 [=====] - 0s 9ms/step - loss: 0.1430 - acc: 1.0000
Test accuracy: 100.0

```

A partir de ces résultats nous avons donc pu relever les classifieurs les plus pertinents. Le classifieur Logistic regression était celui le moins précis. Le classifieur KNN présentait un très bon score de précision, cependant nous voulions voir si ce score pouvait être amélioré.

Finalement nous avons retenu le Neural network qui était le meilleur au niveau de la précision, mais il était un peu plus

lent que les autres. Si nous voulions un bon rapport entre la performance et le temps utilisé, le SVM était le meilleur classifieur. Notre choix était d'avoir les meilleures performances possibles, il était préférable de choisir le Neural network.

```

Logistic Regression - Validation Accuracy: 98.62% - Time: 0.02s
SVM - Validation Accuracy: 99.31% - Time: 0.00s
kNN - Validation Accuracy: 99.31% - Time: 0.01s
Neural Network - Validation Accuracy: 100.00% - Time: 0.54s

```

E. Personal dataset

Pour créer notre dataset, nous avons sélectionné des photos personnelles et celles de nos amis. Nous avons essayé d'avoir le plus d'images le plus diversifié possible afin de limiter le biais.

Pour créer notre modèle nous avons repris celui de la partie 1 et 2 tout en les adaptant. Cependant, lors de l'importation de nos photos pour l'implémenter dans notre modèle, nous avons rencontré des problèmes. Nous supposons que le problème provient du format de nos photos: "HEIC". Nous avons essayé tant bien que mal de les convertir en "JPG" pour ensuite l'implémenter dans le modèle, une erreur persistait lors du traitement de nos images par la fonction "cvtColor".

Après avoir fixé ce problème nous avions des soucis lors de la compilation de l'extraction des visages. En effet, il arrivait que la RAM du Google Colab saute et donc nous devions tout redémarrer. Également nous étions freinés par la restriction de GPU imposée par Google. Finalement, nous avons quand même élaboré notre modèle de reconnaissance faciale et nous avons pu mettre en œuvre les méthodes utilisées comme l'algorithme de l'estimation des points de repères, sans pour autant que le programme marche.

F. Extra - Bias analysis

Le biais est une distorsion systématique qui peut affecter les résultats d'un modèle d'apprentissage automatique. Ce biais peut provenir des données d'entraînement utilisées, des choix algorithmiques ou encore de décisions humaines prises lors du développement (collecte des données, sélection des caractéristiques).

Le biais est problématique car il conduit à des modèles injustes et inexacts. Si le système est majoritairement entraîné sur des visages clairs, il risque de mal reconnaître les personnes à la peau plus foncée. Cela peut avoir des conséquences graves, comme des arrestations injustifiées.

L'impact de la diversité des données sur la performance des systèmes de reconnaissance faciale peut être calculé. En calculant des statistiques comme le taux de fausses détections, le taux de fausses identifications et le taux de réussite globale pour différents groupes, on peut identifier d'éventuels biais et mesurer l'amélioration de la précision et de l'équité du modèle suite à la diversification des données. Concernant notre base de données, nous disposons d'images avec des éclairages, netteté, zoom... différents afin de limiter les biais.

IV. CONCLUSION

Le développement de ce système de reconnaissance faciale nous a plongés au cœur des défis et des opportunités que présente cette technologie prometteuse. En explorant différentes approches et en analysant les résultats obtenus, nous avons acquis des connaissances précieuses.

Le fait d'affiner itérativement un modèle nous a permis de cerner les éléments clés influençant la performance et d'optimiser l'architecture du réseau de manière efficace. De plus, ce projet montre l'importance de la qualité et de la préparation des données d'apprentissage sur la performance du modèle.

REFERENCES

- [1] Zhao, W., Huang, R., & Wang, M. (2003). Face recognition using a combined model based on Gabor transform and PCA. *Pattern Recognition*, 36(4), 803-813.
- [2] Sun, Y., Wang, X., & Tang, X. (2014). Deep learning face representation by combining convolutional and recurrent networks. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* (Vol. 36, No. 11, pp. 2485-2498). IEEE.
- [3] A.M. Martinez, Recognizing imprecisely localized, partially occluded, and expression variant faces from a single sample per class, *IEEE Trans. Pattern Anal. Mach. Intell.* 25 (6) (2002) 748–763.