

IDENTIFICATION FACIALE VIA UNE INTERFACE

Simplon.co

Introduction

Ce projet consiste en la requête d'informations préalablement stockées dans une base de données, à partir d'une photo de visage. Il s'agissait d'étudier et d'améliorer une brique IA de détection de visage pour lui ajouter de la ré-identification et la mettre à disposition via une interface.

Contexte

La mission porte sur le projet d'une chaîne de boîtes de nuit qui veut constituer et mettre à profit un fichier de ses clients. Elle souhaite implémenter un système qui pourra scanner les visages de ses clients pour les lier à un historique (fréquentation, violence, sur-consommation,...). L'entreprise n'en étant qu'à une phase d'expérimentation, elle demande de produire un POC et votre avis sur le sujet. N'ayant pour l'instant pas de plateforme, elle souhaite une url sur lequel elle puisse uploader une photo et observer les résultats.

Votre produit devra donc **comporter une interface simple** permettant d'**uploader une photo en input du réseau de neurones** et d'**afficher le résultat de la requête sur l'interface**.

Cette requête sollicitera donc le réseau de neurones pour savoir quelles sont les personnes ré-identifiées sur la photo et les informations associées à l'aide d'une base de données. Ces dernières seront à afficher sur l'interface.

Pour le contenu de cette base de données, votre client vous interroge sur que vous jugeriez intéressant dans un tel cas. **Le client vous fournira une brique IA de détection de visages, celle-ci est à améliorer** en la complétant notamment par une partie de ré-identification faciale.

Aussi **la détection de visage doit fonctionner pour les photos avec un seul individu ou sur un groupe de personnes**.

Reprise du code pré-existant

```
# Use opencv to process images and detect faces thanks to haarcascade
import cv2

# load the image with faces
image = cv2.imread('dataset/test1.jpg')

# load the pre-trained model - Haar cascade
classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# Process the image to perform face detection
bboxes = classifier.detectMultiScale(image)

# Display bounding box for each detected face
for box in bboxes:
    # extract
    x, y, width, height = box
    x2, y2 = x + width, y + height
    # Draw a rectangle over the image to display the face area
    cv2.rectangle(image, (x, y), (x2, y2), (0,0,255), 1)

# Display the resulting image
cv2.imshow('face detection', image)

# Until we press a key keep the window opened
cv2.waitKey(0)

# Close the window and return to terminal
cv2.destroyAllWindows()

# What are the parameters to improve
# What are the specifications that you can give
# Advantages and problems
```

Ce code utilise une modèle de détection de visage nommé "Cascade de Haar". Ce dernier est capable de prédire la position des yeux, des lèvres, du nez contenu dans une photo, puis d'enregistrer le positionnement du visage repéré à partir de ces caractéristiques (`classifier.detectMultiScale(image)`) dans un rectangle. Ces rectangles obtenus sont alors juxtaposés sur l'image (`cv2.rectangle()`), permettant ainsi à un opérateur de vérifier la pertinence de la détection.

L'image est alors affichée (`cv2.imshow()`), puis l'appui d'une touche suffit à éteindre le programme (`cv2.waitKey(0)` `cv2.destroyAllWindows()`)

Limite du code pré-existant

Bien que fonctionnel, ce code entraîne plusieurs problématique :

- Il n'inclut pas une interface graphique facilitant le changement d'image à analyser.
- Par son imprécision, la méthode Haar-Cascade est aujourd'hui obsolète comme nous le démontreront plus loin dans cette étude.

HaarCascade vs MTCNN

HaarCascade est une méthode inventé en 2001 par Paul Viola et Michael Jones dans la revue scientifique International Journal of Computer Vision. Cette méthode est depuis dépassée depuis la démocratisation des réseaux de neurone, à l'instar du Multi-Task Cascaded Convolutional Neural Network (MTCNN). Une étude réalisée par *datawow.io* (<https://datawow.io/blogs/face-detection-haar-cascade-vs-mtcnn>) semble le démontrer, ces derniers ayant tester la tendance de ces deux modèles à extraire des visages supplémentaires sur des images n'en possédant qu'un. Le résultat semble démontrer que HaarCascade produit plus de faux positif qu'un MTCNN :

HaarCascade

- Number of images in UTK Face : **24,111**
- Number of cropped faces : **19,915**
- Total number of extra faces from a single image : **947**

$$\begin{aligned}\mathbf{Recall} &= (19915 / 24111) * 100 \\ &= \mathbf{82.60\%}\end{aligned}$$

$$\begin{aligned}\mathbf{Precision} &= (18968 / 19915) * 100 \\ &= \mathbf{95.24\%}\end{aligned}$$

MTCNN

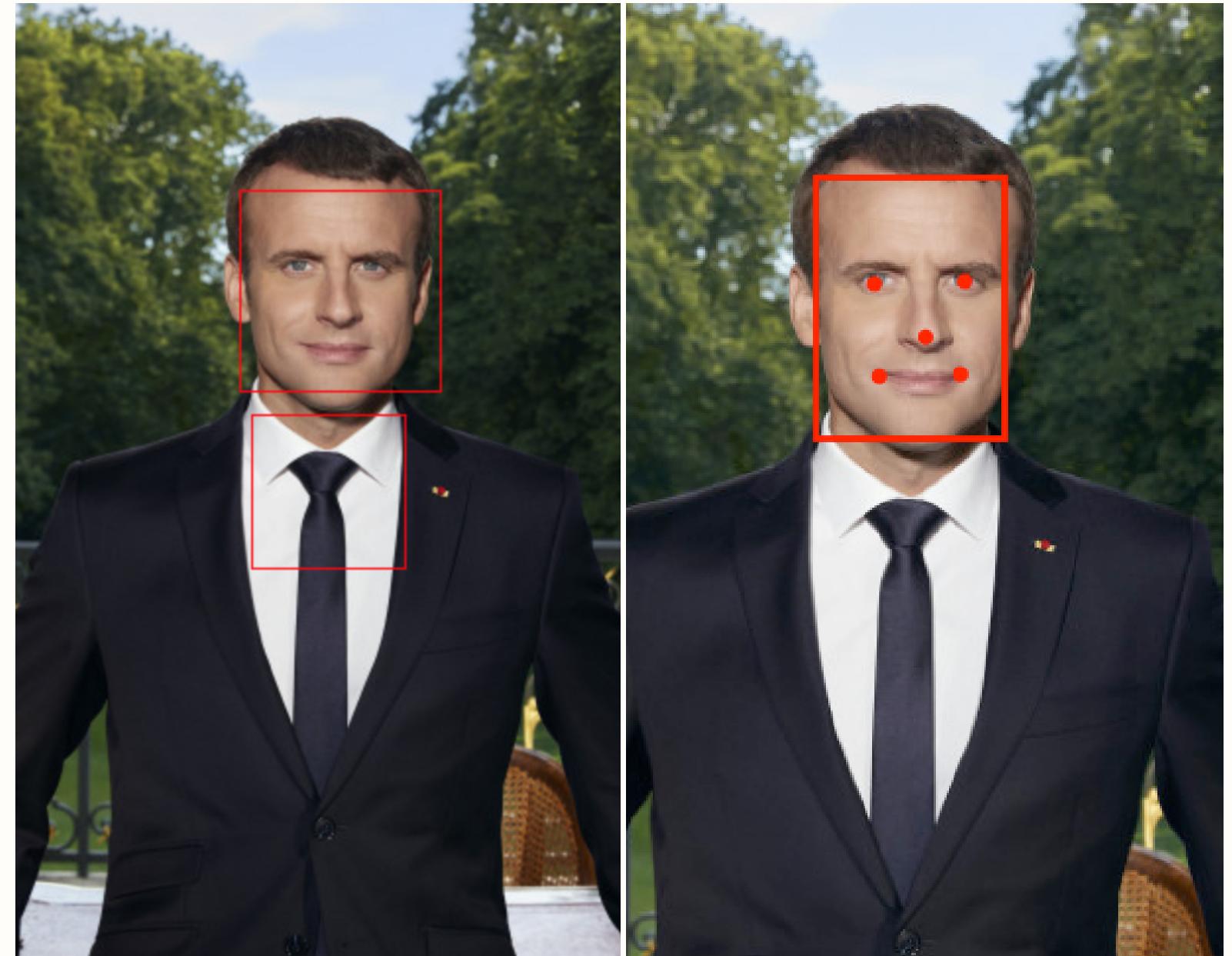
- Number of images in UTK Face : **24,111**
- Number of cropped faces : **21,666**
- Total number of extra faces from a single image : **428**

$$\begin{aligned}\mathbf{Recall} &= (21666 / 24111) * 100 \\ &= \mathbf{89.85\%}\end{aligned}$$

$$\begin{aligned}\mathbf{Precision} &= (21238 / 21666) * 100 \\ &= \mathbf{98.02\%}\end{aligned}$$

Test du MTCNN

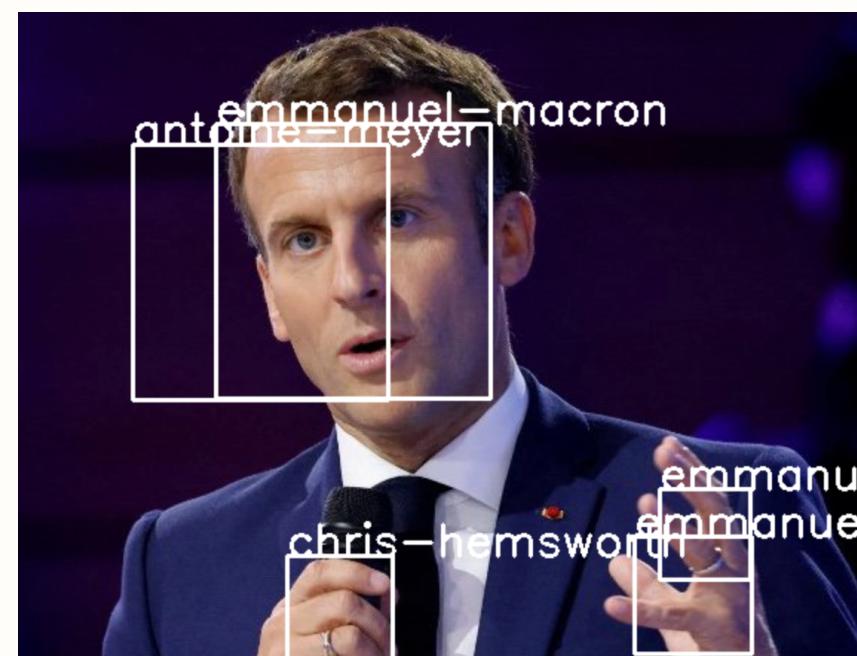
Bien que capable de détecter les visages, HaarCascade démontre rapidement ses limites face au MTCNN. La tendance de HaarCascade a produire des faux positifs devient rapidement problématique. Ainsi, il semble préférable de remplacer la brique de détection utilisant les cascades de Haar avec un MTCNN.



HaarCascade semble confondre visage et cravate...

... tandis que MTCNN ne produit pas cette erreur

Bien qu'ayant moins de faux positif, ce dernier présente un désavantage notable comparé à HaarCascade, celui du temps de traitement. Dans l'étude réalisée par *datawow.io*, HaarCascade est capable de traiter 25 images par secondes, tandis que le MTCNN était limité à 3 images/seconde. On choisi ainsi de sacrifier de la rapidité de traitement pour gagner en précision. La méthode d'HaarCascade n'est donc pas totalement à rejeter, elle peut toujours trouver son utilité dans une application qui privilégie la rapidité du traitement à la précision, comme de la vidéo ou de l'analyse en temps réel.



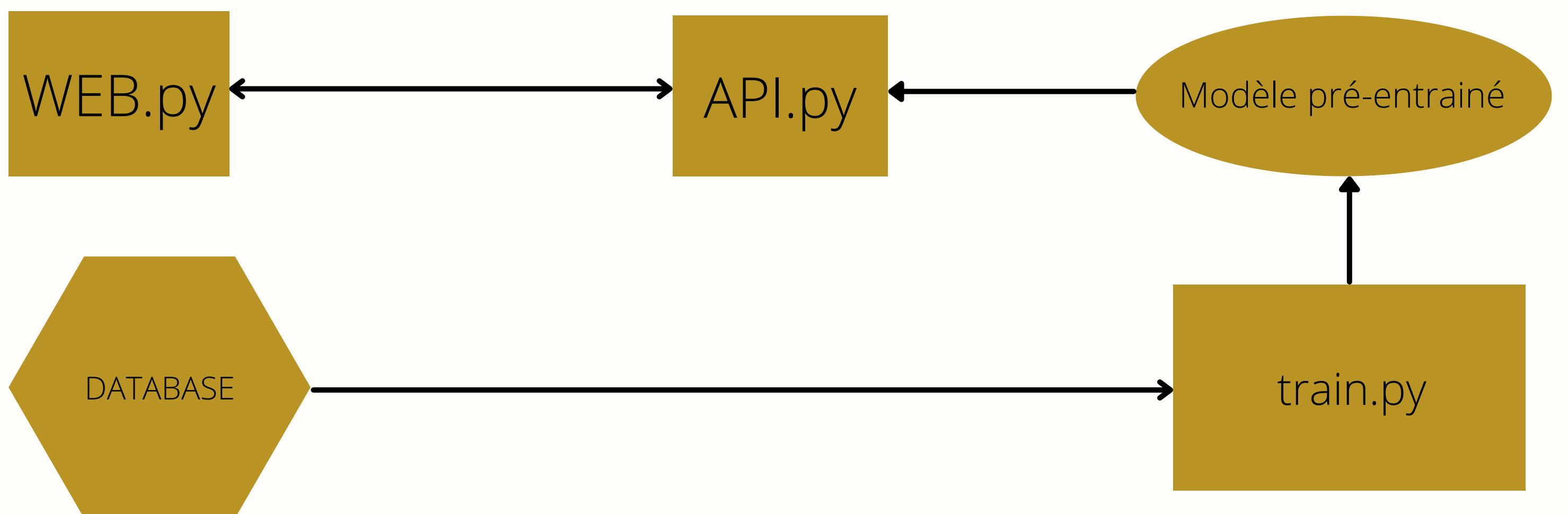
Le nombre de faux positif avec HaarCascade peut rapidement devenir problématique...



... problème bien moins présent pour le MTCNN.

Proposition

Nous proposons ainsi une amélioration du code pré-existant en la composition comme suit :



Le bloc WEB correspond à l'interface graphique web dont les fonctions sont :

- Uploader une image.
- Afficher l'image traitée.
- Afficher le nom des personnes prédites.
- Afficher l'indice de confiance quant aux prédictions.

L'API correspond au traitement de l'image : elle récupère l'upload de l'utilisateur, détecte les visages, puis les compare avec le modèle pré-entraîné, pour enfin retourner dans l'interface WEB une nouvelle image et les résultats du traitement.

Le modèle pré-entraîné est quant à lui une sauvegarde du modèle prédictif généré par la partie Entrainement, elle même étant le générateur d'une IA configurée à partir des informations de la base de donnée (visages et labels associés).

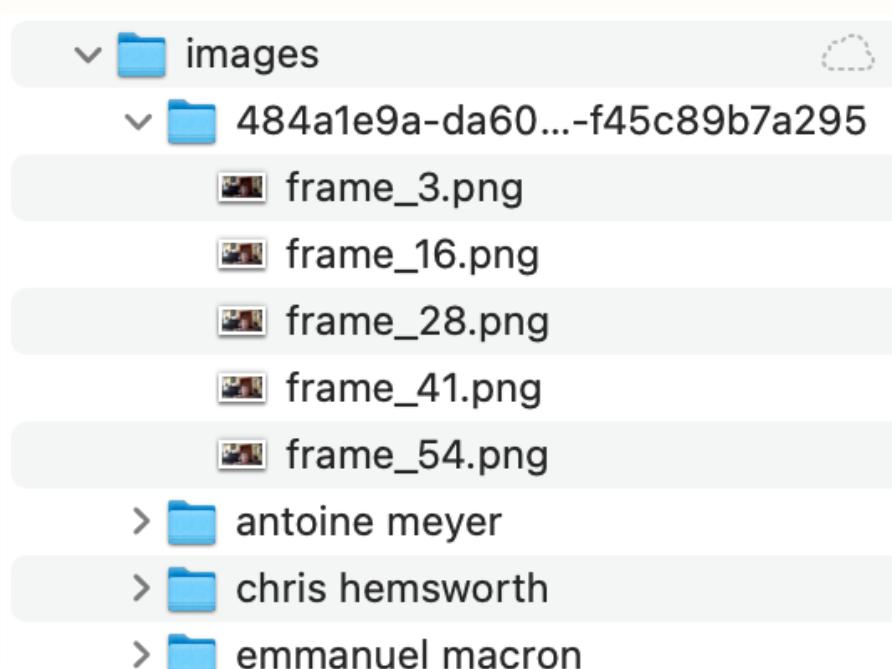
Dans ce prototype, on remarque qu'il n'existe aucune connexion directe entre l'interface WEB et la base de donnée, cette dernière ne servant qu'à construire une IA ayant la capacité d'associer un label à une série de caractéristique tirée des visages contenus dans les photos qui la constitue.

Database

S'il avait été préférable d'avoir une base de donnée structurée sous la forme d'une base de donnée SQL, nous avons opté ici pour un format simple consistant en une série de dossier contenant chacun les images associées à une personne. L'avantage est la facilité d'implémentation et d'extraction de l'information, chaque nom de dossier faisant office de label et les images faisant office de donnée d'entraînement. Cette base de donnée d'image n'étant appelée que durant l'entraînement, ce format suffit pour l'établissement d'un POC. L'avantage d'une telle méthode est également la rapidité de création d'un nouveau label : il suffit de créer un dossier dont le nom représentera le futur label et d'y insérer les photos que nous souhaitons labéliser. De la sorte, il est aussi aisément d'ajouter ou de retirer des images pour un même label, l'unicité nominative inhérente aux dossiers évitant les doublons.

Le désavantage d'un tel format se trouve dans sa limite :

- L'absence de table et de la possibilité de clé primaire/étrangère réduit notre capacité à ajouter de l'information à nos labels. Là où une base de donnée SQL aurait permis d'ajouter aisément des catégories de labels (rentable, dangereux, etc...), notre format impose l'ajout d'une nouvelle instance, tel un dictionnaire contenu dans un .pickle, pour produire le même fonctionnement.
- Les dossiers contiennent des images sous format *jpeg*, *png* ou *jpg*, ce qui peut s'avérer problématique si ces dernières sont trop nombreuses ou trop lourdes. Un encodage des images dans une base de donnée SQL permettrait sans aucun doute de gagner en espace mémoire.



Une base de donnée rudimentaire mais fonctionnelle ...

Entrainement du modèle

Dans cette étude, HaarCascade fut utilisé pour déterminer les ROIs (Région Of Interest) des images, c'est à dire des rectangles contenant chacun un visage. Bien que nous avons établi que HaarCascade possède une fâcheuse tendance à produire des faux positifs, ceci n'est néanmoins pas problématique dans le cas de l'entraînement. En effet, si de faux positifs s'insèrent durant l'entraînement, on peut avancer l'idée que bien que ces faux positifs alourdissent le modèle de donnée déviante, ces données déviantes n'interviennent que très peu dans l'analyse de part l'utilisation d'un MTCNN dans l'API qui fera office de filtre.

L'idée derrière le bloc "Entrainement" est d'associer ainsi à une ROI son label correspondant, le tout dans un format lisible par un algorithme de machine learning. Pour ce faire, le code se comporte comme suit :

1. On instancie un classifier (`cv2.CascadeClassifier()`) et un recognizer (`cv2.face.LBPHFaceRecognizer_create()`)
2. On parcourt les différents dossiers pour en soutirer les noms des labels, et on produit un identifiant unique pour chacun d'entre eux.
3. Pour chaque image étudiée, on converti l'image en noir et blanc, on les redimensionne sous format carré, puis on les transforme en `numpy.array` (`uint8`).
4. On détecte les visages contenus dans l'image grâce à `classifier.detectMultiScale()`
5. Pour chaque visage détecté, on en soutire le positionnement du visage, puis on découpe l'image en cette position. On rempli alors deux listes de manière synchrone : une liste sera nos images transformés et découpés, nos ROIs (`x_train`), tandis que l'autre sera nos labels, exprimé sous format d'ID (`y_labels`).
6. On crée un fichier "labels.pickle" qui est un dictionnaire servant à reconnaître un label à partir de son ID.
7. Enfin, on entraîne notre recognizer avec `x_train` et `x_labels`, puis on le sauvegarde.

API

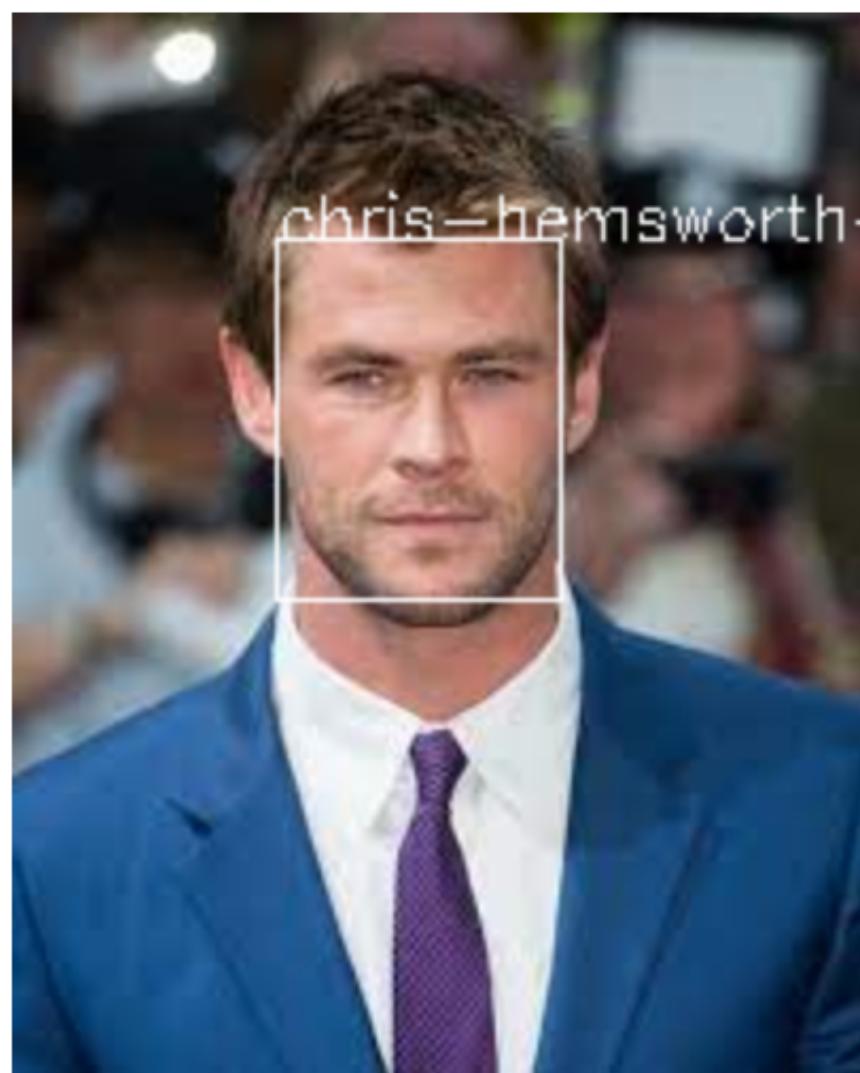
L'API consiste en une fonction unique dénommée "process" qui peut être appelé à souhait par l'interface. Elle récupère en argument le contenu envoyé par le bouton d'upload du framework Dash, qui est encodé en base64.

1. La première étape consiste ainsi à décoder cette chaîne de caractère afin d'en récupérer l'image qu'elle contient. Cette image sera par la suite convertie en gris pour limiter la quantité de donnée à traiter.
2. Puis, le détecteur MTCNN est instancié. Il sera utilisé pour repérer les visages contenus dans la photo (MTCNN().detect_faces()).
3. On instancie ensuite le modèle produit avec l'entraînement.
`(read("trainner.yml"))`
4. Il s'agit à présent de lire notre "labels.pickle" contenant la correspondance labels-ID.
5. Pour chaque visage reconnu, on en extrait la ROI grâce au MTCNN, et on soumet cette région au recognizer (recognizer.predict(ROI)) ce qui nous renvoie l'ID dont les vecteurs sont les plus proches de ceux appris par notre modèle LBPH, ainsi qu'un indice de confiance (distance moyenne entre les vecteurs).
6. On ajoute le nom correspondant à l'ID récupéré en 5 ainsi que son indice de confiance associé dans une variable.
7. La mesure est alors affichée au sommet de la ROI (`cv2.putText()`) et on trace un rectangle autour de cette dernière (`cv2.rectangle`).
8. Un fichier temporaire 'temp.png' est créé, ce dernier aura vocation à être affiché dans l'interface web.

Interface Web

L'interface web est générée grâce au framework Dash. Ce dernier permet la génération d'une page html et d'un serveur pour la supporter, le tout en restant dans l'écriture Python. Notre page se compose d'un unique bouton "Upload" qui permet à l'utilisateur de charger une image. Le processus de traitement est immédiatement démarré à la fin de l'upload.

Une fois le traitement terminé, "temp.png" est affiché ainsi que les noms des personnes détecté et l'indice de confiance étant associé à cette détection.



◦ chris-hemsworth85.60137415930619



Pour un même niveau de confiance l'IA peut tomber juste comme se tromper....

VERSION V2

La qualité de la prédiction de notre IA étant clairement inconsistante, il fallait conclure cette étude avec une piste d'amélioration. L'algorithme de reconnaissance LBPH (Local Binary Patterns Histogram) est presque dépassé de par son age (2006), il existe aujourd'hui des alternatives plus efficace en performance. C'est le cas par exemple des VGG-Face CNN descriptor disponible avec la librairie deepface, une librairie dédié spécialement à l'utilisation du deep learning pour la reconnaissance faciale.

Cette librairie propose une méthode .verify qui permet de vérifier si deux images contiennent la même personne. Quelques tests rapides permettent de confirmer un gain très net en précision. Cependant cette dernière se fait au prix de la rapidité : étant forcés de parcourir chaque image de la base de donnée pour les comparer à l'image contenant l'inconnu, notre temps de calcul se voit directement augmenté proportionnellement à la taille de notre base de donnée.



- chris hemsworth
- 0.11010088515672767



- emmanuel macron
- 0.15622551747827806

La précision à drastiquement augmenté au détriment du temps de traitement