
Master 2 Imagine

# Projet Image: Reconnaissance faciale

## Compte Rendu 5

DEROUBAIX Renaud, SERVA Benjamin

17 novembre 2024

## Table des matières

# 1. Comparaison des vecteurs caractéristiques

## 1.1 Introduction

On va tester ici les méthodes de comparaisons de vecteurs caractéristiques. Pour rappel grâce à LBPHFaceRecognizer on va créer un vecteur caractéristique pour chaque image de la BDD et on cherche ensuite à trouver une correspondance avec une image autres et une images de la BDD si il y en a une.
On étudiera les statistiques qui sont fait exclusivement avec des images de personnes présentent dans la BDD, c'est à dire que si notre méthode est "parfaite" alors on est censé obtenir 100% de bonne reconnaissance.

## 1.2 Comparaison par distance euclédienne

| BDD | 1 | 2 | 3 |
|---|---|---|---|
| Moyenne | 0.97 | 1.03 | 0.95 |
| Minimum | 0.68 | 0.83 | 0.68 |
| Maximum | 1.47 | 1.50 | 1.36 |
| Médiane | 0.94 | 0.99 | 0.93 |
| Similitude | 76 | 79 | 77 |
| Différence | 4 | 1 | 3 |

TABLE 1 – Comparaison des méthodes

1. comparaisons avec vecteurs moyens de chaque label
2. comparaison avec les vecteurs de chaque images de la bdd
3. utilisation des deux

## 1.3 Classifieur : KNN

Après différents tests effectués c'est avec ces valeurs de voisinages que l'on obtient les meilleurs résultats.

### 1.3.1 Tests

| | 1 | 2 |
|---|---|---|
| Moyenne | 0.56 | 0.47 |
| Minimum | 0.29 | 0.14 |
| Maximum | 1. | 0.86 |
| Médiane | 0.57 | 0.43 |
| Similitude | 63 | 70 |
| Différence | 17 | 10 |

TABLE 2 – 7 voisins

| | 1 | 2 |
|---|---|---|
| Moyenne | 0.85 | 0.75 |
| Minimum | 0.33 | 0.33 |
| Maximum | 1. | 1. |
| Médiane | 1. | 0.67 |
| Similitude | 70 | 69 |
| Différence | 10 | 11 |

TABLE 3 – 3 voisins

1. comparaison avec les vecteurs de chaque images de la BDD
2. utilisation du 1 ainsi que les vecteurs moyens de chaque label

### 1.3.2 Discussion

Pour la création du classifieur on a cherché à optimiser sa précision (en chercher à être le plus proche de 100%), malheureusement on ne peut pas entraîné ce classifieur un nombre n de fois pour augmenter sa précision donc il faut jouer avec ses paramètres en l'occurrence ici c'est le facteur **n_neighbors** qu'il faut faire varier.
En chercher à optimiser sa précision on obtient au maximum 85% de précision ce qui fais sur 80 tests quand même 10 erreurs.

## 1.4 Conclusion

On obtient pour l'instant de meilleur résultats avec la distance euclidienne pour la comparaison entre vecteurs qu'avec le classifieur mis en place.
Il faut donc qu'on implémente un autre classifieur qui lui peut être entraîné un nombre n fois pour augmenter sa précision.

De plus il faut quand même tester ces deux méthodes sur notre propre BDD car sur celle où on fait les tests actuellement toutes les photos sont similaires en termes de variations de lumière et de pose, ce qui peux expliquer en partie la confusion dans la recherche de similitude.

# 2. CNN : Deepface

## 2.1 Changement depuis le CR4 :

Lors de l'implémentation de notre modèle de reconnaissance faciale, nous avons rencontré des difficultés d'installation avec la bibliothèque **InsightFace**, que ce soit sur nos PC personnels ou sur les ordinateurs du bâtiment 16. Après plusieurs tentatives pour résoudre les conflits, nous avons décidé de changer de modèle vers une bibliothèque qu'on a déjà utilisé pour nos tests précédents, qu'on a vu lors de notre état de l'art et qu'on sait fonctionnel sans conflits : **DeepFace**.

InsightFace et DeepFace offrent des fonctionnalités similaires, ce qui permet de continuer dans l'optique envisagée précédemment.
La seul véritable différence est la perte d'accès aux modèles spécifiques SubCenter ArcFace et MobileFaceNet, mais DeepFace dispose d'autres modèles (comme VGG-Face, OpenFace, ArcFace, Dlib et FaceNet) qui permettent de tout de même réaliser une reconnaissance faciale satisfaisante. Nous avons décidé d'utiliser dlib.

## 2.2 Choix de dlib

**Dlib** propose une implémentation de réseau de neurones convolutifs (CNN) pour la reconnaissance faciale avec :

- **Modèle CNN pré-entraîné** : `dlib` propose un modèle CNN pré-entraîné pour la reconnaissance faciale qui peut être utilisé directement avec notre base de données.
- **Précision et robustesse** : Le modèle CNN de `dlib` est particulièrement robuste face aux variations d'éclairage et de pose.
- **Performances sur appareils mobiles** : `dlib` est adapté aux appareils mobiles.

## 2.3 Prétraitement de DeepFace et Dlib

Lors de l'utilisation de **DeepFace** avec **dlib**, plusieurs prétraitement sont effectuées automatiquement sur l'image avant d'extraire les caractéristiques :

- **Redimensionnement de l'image** : Taille standard des modèles de DeepFace et dlib.
- **Détection et alignement du visage** : Dlib applique une détection des visages et un alignement facial automatique(ajustement de l'orientation du visage).

## 2.4 Processus de reconnaissance faciale :

Pour le moment, l'idée de l'implémentation que l'on souhaite mettre en place est la suivante :

- On utilise un modèle cnn pré-entraîné pour faire de la détection de visage (dlib).

- On récupère un vecteur caractéristique de ce visage.
- On labellise ce vecteur caractéristique obtenu et on le sauvegarde dans un YML.
- On utilise ce YML afin d'obtenir un classifier.

Ensuite dans une application mobile :

- On utilise le même modèle pour obtenir le vecteur caractéristique de la personne à identifier.
- On donne le vecteur caractéristique au CNN lié au classifier.
- On reconnaît (ou pas) le visage.

## 2.5 YML :

Les vecteurs caractéristiques sont rangés dans le YML de la façons suivante :
Label :
-image : `id_image`
vector : `data`

Adam :
- image : Adam_1.jpg
  vector :
  - −0.7747194766998291
  - 0.2503308951854706
  - 1.7457960844039917
  - −0.4501965343952179
  - 1.3083291053771973
  - −1.3216145038604736
  - 0.5944512486457825
  - −1.540221095085144
  - −0.4274892508983612
  - −1.0483371019363403

  ...

  - 2.8276870250701904
  - −2.1867430210113525
  - −0.6057721376419067
  - −0.12380097061395645
  - −0.04700630158185959
  - −0.40768831968307495
  - −1.0282247066497803
  - 1.7080134153366089
  - −2.1447370052337646
  - −0.2742111086845398
  - −2.408881425857544
  - −0.44323474168777466
  - 0.038504332304000854

```
Adam:
- image: Adam 1.jpg
- image: Adam 2.jpg
- image: Adam 3.jpg
- image: Adam 4.jpg
- image: Adam 5.jpg
- image: Adam 6.jpg
- image: Adam 7.jpg
- image: Adam 8.jpg
- image: Adam 9.jpg
- image: Adam 10.jpg
Arthur:
- image: Arthur 1.jpg
- image: Arthur 2.jpg
- image: Arthur 3.jpg
- image: Arthur 4.jpg
- image: Arthur 5.jpg
- image: Arthur 6.jpg
- image: Arthur 7.jpg
Benjamin:
- image: Benjamin 1.jpg
- image: Benjamin 2.jpg
- image: Benjamin 3.jpg
- image: Benjamin 4.jpg
- image: Benjamin 5.jpg
- image: Benjamin 6.jpg
- image: Benjamin 7.jpg
- image: Benjamin 8.jpg
- image: Benjamin 9.jpg
- image: Benjamin 10.jpg
- image: Benjamin 11.jpg
- image: Benjamin 12.jpg
Brian:
- image: Brian 1.jpg
- image: Brian 2.jpg
- image: Brian 3.jpg
- image: Brian 4.jpg
- image: Brian 5.jpg
- image: Brian 6.jpg
- image: Brian 7.jpg
Emmanuel:
- image: Emmanuel 1.jpg
- image: Emmanuel 2.jpg
- image: Emmanuel 3.jpg
- image: Emmanuel 4.jpg
- image: Emmanuel 5.jpg
- image: Emmanuel 6.jpg
- image: Emmanuel 7.jpg
- image: Emmanuel 8.jpg
- image: Emmanuel 9.jpg
- image: Emmanuel 10.jpg
JB:
- image: JB 1.jpg
- image: JB 2.jpg
- image: JB 3.jpg
JL:
- image: JL 1.jpg
```

FIGURE 1 – Extrait du YML

Voir sur le github dans dir : Code/CNN/DeepFace.

Exemple de détection de visage :

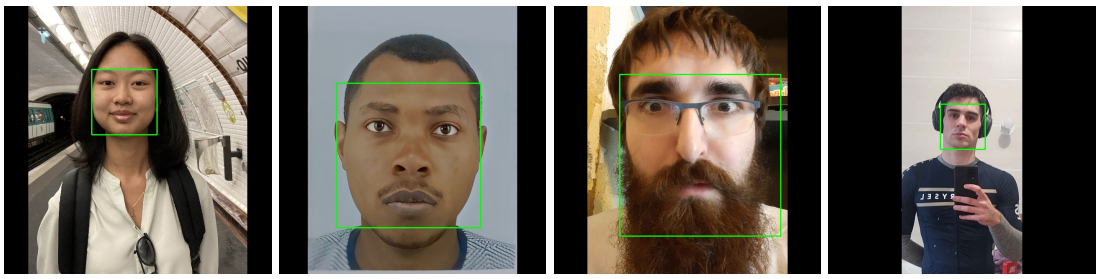

FIGURE 2 – 4 détections de visage dlib

Normalement dlib donne un vecteur caractéristique de la même taille quelle que soit la taille de la détection du visage.

# 3. Portage application mobile

## 3.1 Planification

À l'origine, on voulait utiliser Qt pour faire une application mobile tout en restant en python grâce à **PySide6**. PySide6 supporte le développement multi-plateforme, le problème est que l'intégration à Android ne fonctionne pas correctement sous Windows (ou alors je n'ai pas trouvé la solution). Ayant des problèmes avec ma machine Linux, je n'ai pas eu le choix que de regarder d'autres solutions.

De là, j'ai regardé quelles sont les autres solutions envisageables. **BeeWare** et le framework **Toga** permettent facilement de créer une application et de la déployer sur mobile. Cependant, BeeWare limite le portage seulement aux bibliothèques 100% Python, or nous avons des bibliothèques avec des composantes C++ telles que dlib ou numpy.

La dernière solution est l'utilisation de **Kivy** avec **Buildozer** pour le portage Android. Nous n'avons pas encore mis en place Buildozer, mais nous avons essayé Kivy avec la webcam comme caméra, les premiers résultats sont intéressants. À voir si le portage mobile fonctionne bien niveau performance.

Maintenant à voir si on reste sur Qt et on développe l'application à la fac ou si on change et on se lance sur Kivy.

## 4. Objectif pour la semaine #7 - CR6 :

Nos objectifs d'ici le prochain compte rendu sont :
— Améliorer le classifieur et le tester avec la méthode CNN (pour l'extraction de caractéristiques).
— Tester nos méthodes sur la BDD des étudiants du master et autres que l'on a mis en place.
— Démarrer l'implémentation de l'application avec détection par camera téléphone et/ou prise de photo.

## Sources & Liens

2. DeepFace code source
3.1 PySide6 Description du module Qt.
3.1 BeeWare Site du projet.
3.1 Toga Description du widget toolkit
3.1 Kivy Site du FrameWork.
3.1 Buildozer Description du packager.

## Github

Lien Github avec tout le code et compte rendu du projet.