Comparaison d'Algorithmes de Reconnaissance Faciale Principal Component Analysis VS Machine Learning

Thomas Bédrine, Nathan Rassié
Université de Technologie de Belfort Montbéliard, Belfort
Secteur Vision Artificielle

17 Octobre 2022

Résumé — *Dans ce papier, nous nous intéresserons à la comparaison entre la méthode d'Analyse par Composantes Principales (ACP, PCA en anglais) et la méthode par Machine Learning, utilisées dans le but de reconnaître les visages d'un ensemble d'images de test à partir d'un ensemble distinct d'images d'entraînement. Notre support sera le langage Python. On relèvera notamment les temps d'exécution par sous-fonction de chaque algorithme, la consommation de ressources ainsi que la précision des résultats obtenus.*

I. Introduction

Lorsque l'on parle de reconnaissance faciale, on pense très souvent à l'intelligence artificielle et donc au *machine learning*. C'est une méthode qui est simple à appliquer, et de nombreux outils sont disponibles afin de la mettre en place aisément.

Dans le cadre de nos enseignements à l'UTBM, nous avons étudié l'utilisation de l'Analyse par Composantes Principales (ACP/PCA). Il s'agit d'une méthode qui, semble-t-il, avait disparu pendant quelques années face au Machine Learning, mais qui a refait surface récemment.

L'objectif était donc le suivant : mettre en place les deux méthodes en parallèle, comparer leur efficacité, leur coût, et leur complexité. Ainsi, on pourrait déterminer les avantages et les inconvénients de chaque méthode dans diverses circonstances.

Nous avons travaillé à partir de photographies des membres de notre classe, réalisées sous divers angles de vue et avec différentes mimiques, afin d'obtenir une base de données aussi complète que possible. Après traitement des images, nous avons quatre à huit images par personne, pour onze personnes. Une image de chaque personne a été extraite comme ensemble de test, et le reste sera notre ensemble d'entraînement. L'utilisation de ces images étant restreinte à notre expérience, nous respecterons la consigne qui nous a été donnée de ne pas faire apparaître nos photographies.

II. Méthode ACP

1. Problématique

Pour identifier une personne sur une image, nous avons à notre disposition un ensemble de photos de personnes quelconques. Il faut donc trouver des points de comparaison entre l'image d'entrée et la base d'images afin d'établir une correspondance entre elles et identifier la personne parmi la base de données.

Si l'on décide de comparer les images pixel par pixel, le temps de calcul et le coût

d'exécution seraient extrêmement élevés. Il faudrait en plus prendre en compte le décalage potentiel des images les unes par rapport aux autres: le visage n'est pas toujours centré, la personne n'a pas la même expression, ou bien elle tourne la tête dans une autre direction.

Chaque pixel peut se voir attribuer un coefficient de corrélation, qui définit à quel point il est proche d'un autre pixel auquel il est comparé. On a donc $k(k-1)/2$ coefficients à traiter pour une simple comparaison d'une image à une autre. Il faut trouver un moyen de réduire l'espace de recherche.

2. Intérêt de l'ACP

Le fonctionnement de l'ACP est aussi complexe à saisir qu'à expliquer, surtout pour un problème avec autant de dimensions: il y a, au plus, N dimensions car il y a N images. Toutefois, l'idée est la suivante: en calculant les valeurs propres et les vecteurs propres d'une matrice de covariance formée à partir de toutes les images de la base, on peut extraire des axes autour desquels les images sont particulièrement sensibles à la variation. On appelle ces axes les "composantes principales" du problème.

Par exemple, on remarquera souvent que le fond de l'image est le même pour tous nos échantillons, ainsi il y a peu de variation pour chaque image sur ce point. En revanche, pour la forme du visage, l'emplacement des yeux/du nez/de la bouche ou encore la longueur des cheveux, chaque image aura des variations importantes par rapport aux autres, et les axes qui concernent ces changements seront les plus proéminents.

Le but de l'ACP est donc de créer ces composantes et de maximiser la variance sur un minimum d'axes. Puisque cette méthode projette notre espace de départ sur un espace strictement plus petit, on risque de perdre de l'information si on retire trop de dimensions. L'objectif est d'atteindre environ 95% de variance afin de perdre le moins d'informations possible.

S'il s'avère que les trois axes les plus importants cumulent plus de 95% de variance, alors on aura ramené notre problème à seulement trois dimensions, sans pour autant impacter l'intégrité de notre espace d'images. Pour notre étude, on comparera donc plusieurs passes possibles, avec un nombre de composantes variables.

3. Mise en pratique en Python

Avec l'aide de la librairie OpenCV et de glob, on charge les bases d'entraînement et de test, et on crée une matrice unique contenant toutes les images d'entraînements, chacune réduites à un vecteur de longueur M.

On cherche maintenant à obtenir la matrice des vecteurs propres nécessaire à notre méthode. En calculant la matrice de covariance puis ses vecteurs propres, on obtient donc les composantes principales du problème.

```
def findEigenvectors(mat):
    # Substracting the mean vector to all vectors
    subMatrix = mat - mat.mean(axis=0, keepdims=True)

    # Pseudo-covariance matrix : transposed C dot C (CtC)
    covarMatrix = subMatrix.transpose()@subMatrix
    # w : eigenvalues of CtC, v : eigenvectors of CtC
    w, v = np.linalg.eig(covarMatrix)
    # Eigenvectors of CCt : C dot v
    eigenV = subMatrix@v
    normEigenV = eigenV/np.linalg.norm(eigenV)

    return normEigenV
```

Figure 1. - Détermination des vecteurs propres.

À partir d'ici, nous n'étions pas sûrs de la marche à suivre et il nous manquait des étapes, nous nous sommes donc tournés vers le package *scikit-learn*, qui propose une fonction de PCA déjà établie. Le résultat est une base formée de vecteurs propres portant suffisamment d'informations pour projeter toutes les images d'origine sans perdre en qualité. En pratique, on choisit L composantes parmi au plus N vecteurs propres obtenus.

On calcule ensuite les poids associés à chaque image, à partir de la nouvelle base créée, puis on récupère l'image à reconnaître. Après avoir soustrait la valeur moyenne de la base à cette image, on calcule son poids. Il ne reste qu'à comparer celui-ci avec les poids des images d'entraînement, et choisir celle pour laquelle la distance entre les deux poids est minimale.

```
def identifyFace(testSet, weights, pca, e_faces):

    # query = getImageInputByName() #if you want to test one image at a time

    best_matches = []

    for query in testSet:
        query = query.reshape(1, -1)
        q_weight = e_faces @ (query - pca.mean_).T

        euclidean_distance = np.linalg.norm(weights - q_weight, axis=0)
        best_matches.append(np.argmin(euclidean_distance))

    return best_matches
```

Figure 2. - Identification du visage.

Enfin, nous pouvons afficher les résultats et comparer la personne qui a été identifiée par rapport à la personne qui est réellement sur l'image.

III. Méthode du Machine Learning

1. Problématique et intérêt

Le Machine Learning est une branche de l'intelligence artificielle basée sur l'apprentissage et la reconnaissance (de formes, d'images, de texte…).

Il possède l'avantage de pouvoir traiter une large variété de données différentes, sans pour autant complexifier la mise en œuvre du système.

Basé sur un réseau de neurones souvent réparti en plusieurs couches, le système va interconnecter chaque information qu'il possède sur un jeu d'entrée et évaluer à l'aide de poids ses corrélations avec d'autres informations.

Nous souhaitons utiliser cette technique pour évaluer l'efficacité des associations nom - visage que le réseau pourra produire par rapport au PCA précédemment évoqué.

De ce fait, en prenant en compte son temps de construction, d'entraînement, de reconnaissance ainsi que sa fiabilité, il nous sera possible d'établir des hypothèses concernant l'abandon justifié – ou non – du PCA au cours des dernières années au profit du machine learning.

2. Mise en oeuvre avec Python

Pour réaliser ce réseau de neurones, nous travaillerons via un modèle en apprentissage supervisé. En effet, tandis que certains types d'applications du machine learning impliquent que celui-ci apprenne tout de lui-même, nous préférons fournir nous-même l'association nom - visage de chaque image pour entraîner le modèle.

Nous utiliserons la librairie TensorFlow, qui intègre un modèle de réseau neuronal via son API Keras. Utilisée dans sa version séquentielle, elle correspond au besoin de notre expérience.

L'implémentation est découpée en deux fichiers différents:

Le premier sert à la construction, la compilation ainsi qu'à l'entraînement du modèle à partir des mêmes jeux d'essai et d'entraînement utilisés pour le PCA. De ce fait, il nous est possible de faire varier les paramètres d'entraînement du modèle, puis de le sauvegarder pour pouvoir l'exploiter ailleurs lors de la phase de reconnaissance.

Le second fichier sert justement à construire un modèle de probabilité basé sur le modèle pré-entrainé de notre choix, pour pouvoir effectuer une reconnaissance faciale (i.e une association nom - visage) et afficher le résultat à l'utilisateur via une interface graphique gérée par OpenCV.

IV. Benchmarking des méthodes

1. Environnement et paramètres des simulations

Par mesure d'homogénéité des résultats, tous les benchmark ont été réalisés sur la même machine, à savoir un MacBook Pro 2019 doté d'un Intel Core i7-9750H 6 coeurs cadencés à 2,6 Ghz ainsi que de 16Go de DDR4 cadencée à 2664 Mhz. Le système d'exploitation est macOS 12.6, et la version de Python utilisée est la 3.10.8.

Le profiling des temps d'exécution par fonction a été réalisé via la librairie *line_profiler*, tandis que la mesure de la mémoire a été effectué via *memory_profiler* ainsi qu'en parallèle avec l'outil *Instruments* d'Apple. Il est important de noter que la version de TensorFlow (2.10.0) ne tourne que sur CPU.

Concernant les images d'entraînement et d'essai, nous avons travaillé avec une base de visages en nuances de gris, au format JPEG et de taille 2000 x 2000. Le jeu d'entrainement est composé de 11 pesonnes différentes réaprties sur 55 images, tandis que le jeu d'essai contient une image par personne, donc 11.

Nous avons fait varier le nombre de composantes du PCA selon la liste suivante: {1, 3, 5, 10, 20, 30, 50}. De la même façon, ces valeurs serviront de paramètre au machine learning pour définir son nombre d'itérations d'apprentissage.

Le réseau de neurones du machine learning est quant à lui composé de trois couches:

- une couche d'entrée correspondant au nombre de pixels d'une image, soit 2000 x 2000 = 4 000 000
- une couche à 128 noeuds
- une couche de sortie avec autant de noeuds que de personne différentes dans la BDD, soit 11

2. Résultats par Sous-fonctions du PCA

La reconnaissance faciale à l'aide du PCA a été découpée en trois sous-étapes dans notre programme.

La première consiste à "aplatir" les images, autrement dit réduire chaque matrice de pixels qui définissent une image en vecteur unidimensionnel. En effet, le PCA travaille sur des vecteurs de dimension une, cette étape de prétraitement est donc nécessaire.

La deuxième, le cœur de la méthode en elle-même, consiste à trouver et construire la matrice des vecteurs propres, qui sera utilisée comme matrice de passage vers le sous-espace propre.

Enfin, la dernière permet d'utiliser la matrice précédemment calculée pour identifier les visages en fonction des requêtes.

Nous obtenons donc les résultats suivants:

Étape	**Temps d'exécution moyen (s)**	**Mémoire Vive [RAM] consommée (Mb)**
1	0,227	650,988
2	50,518	2391,840
3	1,366	2418,758

Table 1. - Temps d'exécutions moyens et consommation RAM par sous-étape du traitement PCA

Nombre de composantes du PCA (Étape 2)	Fidélité de la reconnaissance (%)
1	9,1
3	54,5
5	81,8
10	81,8
20	90,9
30	81,8
50	90,9

Table 2. - Fidélité de la reconnaissance en fonction du nombre de composantes du PCA

Si on remarque assez aisément une amélioration de la reconnaissance avec davantage de composantes pour le PCA, on remarque néanmoins une chute à 30 composantes (9 visages sur 11 au lieu de 10), avant de remonter à 90,9% pour 50 composantes - le maximum pour notre échantillon. L'amélioration de la reconnaissance avec l'augmentation du nombre de composantes souligne bien le principe des axes portant le plus d'informations : avec 5 à 10 axes, on a déjà une précision acceptable, qui n'évoluera que très peu.

Notre hypothèse pour la chute de précision autour de 30 composantes est l'introduction possible de données superflues parmi les vecteurs propres, qui induisent alors un décalage important et faussent les prédictions.

3. Résultats par Sous-fonctions du Machine Learning

Pour la reconnaissance par entraînement supervisé d'un réseau de neurones, nous avons segmenté les mesures en cinq sous-étapes, similaire aux traitements séquentiels du machine learning.

Les deux premières correspondent respectivement à la construction ainsi qu'à la compilation du modèle. Comme expliqué en III. 2., il faut donner au modèle un jeu d'entrainement, donc un couple nom - visage. De la même façon que pour le PCA, les images doivent être réduites sur une seule dimension. La libraire TensorFlow inclut donc une étape dans la construction d'un réseau de neurones pour faire cela.

La troisième étape, la plus importante, consiste à entraîner le modèle. Pour cela, plusieurs "passes" peuvent être réalisées (i.e le nombre de fois que le réseau parcourra le jeu d'entraînement pour apprendre). Nous définirons les mêmes paramètres d'essais que pour le nombre de composantes du PCA.

Enfin, les deux dernières étapes permettent de construire un modèle de probabilité et de l'utiliser pour tenter de prédire le nom associé à un visage soumis par l'utilisateur.

Les résultats obtenus sont les suivants:

Étape	Temps d'exécution moyen (s)	Mémoire Vive [RAM] consommée (Mb)
1	2,883	2424,000
2	0,010	2856,500
4	0,027	2662,500
5	0,470	2784,000

Table 3. - Temps d'exécutions moyens et consommation RAM par sous-étape du traitement ML

Nombre de passes d'entraînement (Étape 3)	Temps d'exécution moyen (s)	Fidélité de la reconnaissance (%)
1	23,526	9,1
3	32,825	18,2
5	42,994	18,2
10	80,929	27,3
20	120,130	45,5
30	176,082	81,8
50	285,942	90,9

Table 4. - Temps d'exécutions et fidélité de la reconnaissance en fonction du nombre de passes d'entraînement du ML

Nous remarquons que les pré-traitements (étapes 1 et 2) ainsi que la phase de reconnaissance (étapes 4 et 5) sont particulièrement rapides, leur temps d'exécution ainsi que la consommation mémoire sont approximativement constants (autour de 2,6Gb pour la mémoire).

Concernant l'entraînement, nous avons une fiabilité croissante mais jamais maximale malgré un nombre relativement élevé de passes d'entraînement.

En revanche pour ce-dernier, les temps d'exécutions augmentent linéairement, et deviennent rapidement importants. Si l'on souhaite avoir minimum 80% de fiabilité sur notre reconnaissance, il faudrait entre 20 et 30 passes d'entraînement, soit déjà plus de 2 minutes de calcul.

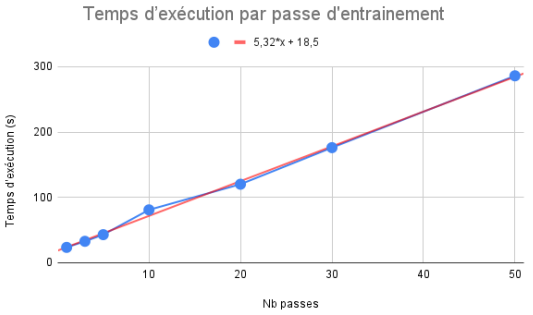

Figure 3. - Courbe de tendance associée aux temps d'exécutions par passes d'entraînement du ML

4. Comparaison des temps d'exécutions et de la fidélité par rapport au nombre de composantes / de passes

Si l'on compare désormais nos valeurs obtenues avec le PCA en faisant varier son nombre de composantes, ainsi qu'avec le ML en incrémentant ses passes d'entraînement, nous obtenons les graphiques suivants:

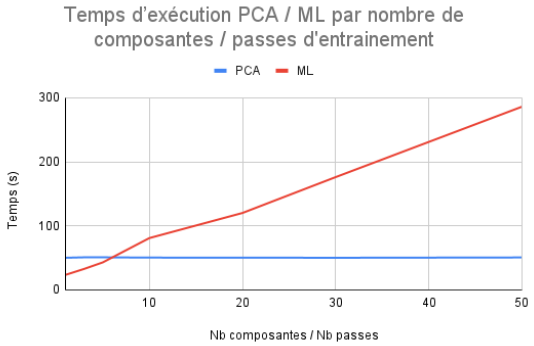

Figure 4. - Temps d'exécution PCA / ML par nombre de composantes / passes d'entraînement

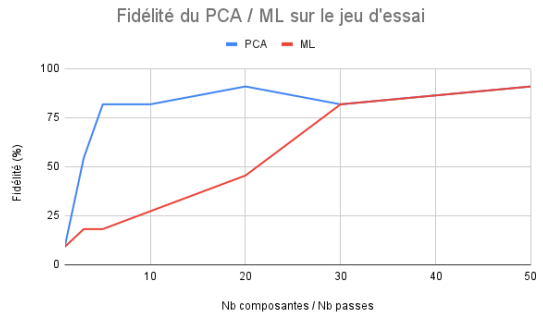

Figure 5. - Fidélité de reconnaissance du PCA / ML sur le jeu d'essai

En négligeant les temps de pré-traitement et de reconnaissance pour les deux méthodes, nous pouvons clairement observer que, par rapport au temps de traitement quasiment constant du PCA, la méthode de machine learning est plus lente à partir d'environ 5 passes d'entraînement.

C'est aussi pour la valeur de 5 composantes établies dans le PCA que l'on atteint une stabilité proche du taux de fiabilité le plus haut du modèle.

V. Conclusion

Il est nécessaire de prendre en compte que nos jeux d'essais et d'entraînement sont très petits comparés aux plusieurs centaines voir milliers d'images qui sont utilisées dans l'industrie pour entraîner des algorithmes à la reconnaissance.

De plus, un profiling de temps d'exécution réaliste effectuerait plusieurs dizaines de fois les mesures de chaque sous-fonction pour chacun des paramètres de la simulation, dans le but d'en faire une moyenne et une estimation plus fine. Ceci n'a pas été réalisé dans cette étude, exceptée pour les valeurs constantes (relevées autant de fois que de variation du paramètre principal de chaque méthode, soit 7 fois).

Enfin, les mesures de mémoire vive faites à partir de la librairie *memory_profiler* fluctuent légèrement et ne sont pas suffisamment fiables pour être considérées comme réalistes. De plus, un profiling avec l'outil *Instruments* intégré à macOS nous montre une consommation réelle du processus Python bien plus importante que celle annoncée, soit environ 5,3Gb pour la version PCA, et 14Gb (avec des pics à 16Gb) pour la version Machine Learning.

Ceci nous permet malgré tout de noter une bien plus grande consommation de mémoire pour le machine learning que pour le PCA.

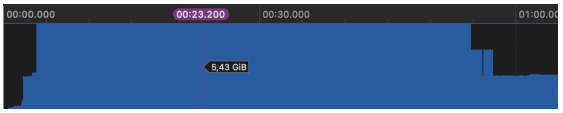

Figure 6. - Consommation RAM réelle du PCA via Instruments (macOS)

Figure 7. - Consommation RAM réelle du ML via Instruments (macOS)

Nous arrivons donc à la conclusion qu'un algorithme de PCA n'est clairement pas dépassé par les nouvelles branches de l'intelligence artificielle telle que le machine learning, puisque sur notre petite base d'images il a su se montrer plus rapide, plus efficace et bien moins gourmand en ressources matérielles que le réseau de neurones.

VI. Bibliographie

Chollet, François. 2017. “Basic classification: Classify images of clothing | TensorFlow Core”. February 25, 2022. https://www.tensorflow.org/tutorials/keras/classification?hl=fr

VanderPlas, Jake. 2016. “Profiling and Timing Code | Python Data Science Handbook”. November, 2016. https://jakevdp.github.io/PythonDataScienceHandbook/01.07-timing-and-profiling.html

Tam, Adrian. 2021. “Face Recognition using Principal Component Analysis”. October 28, 2021. https://machinelearningmastery.com/face-recognition-using-principal-component-analysis/