La reconnaissance faciale par Analyse en Composantes Principales (ACP)

Eloi Navet - 13637 - CPGE MP Lycée Berthollet - 2020/2021

I - Principe de l'Analyse en Composantes Principales

II - Programmation en Python d'un algorithme d'ACP pour la reconnaissance faciale

III - Les limites de l'ACP

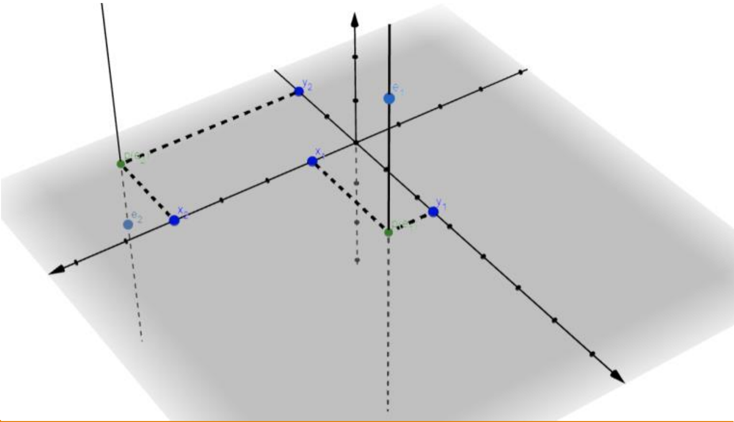
IV - Quelques améliorations et alternatives

I - Principe de l'ACP

- Données : *n* visages.
- But de l'ACP : chercher des *directions principales* pour maximiser l'information.
- Outils mathématiques :
  - Matrice de variance-covariance :

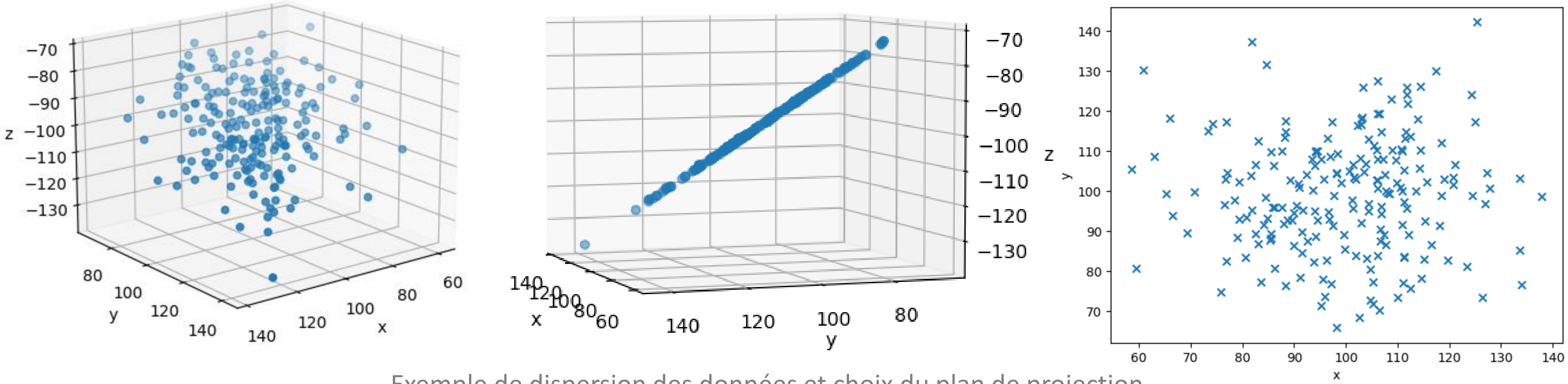
$$S = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^t(X_i - \bar{X})$$

  - Projection orthogonale.
- Dimension du sous-espace : **k**
- Intérêt : ***k*** << *n* variables expliquent la plupart de l'information.



I - Principe de l'ACP

- Idée : exprimer une image en un **petit** nombre de combinaisons linéaires des *n* images de départ.
- Il faut **maximiser la variance** à chaque itération.
- Construction d'une **base orthonormale** de vecteurs propres.



Exemple de dispersion des données et choix du plan de projection

I - Principe de l’Analyse en Composantes Principales

II - Programmation en Python d’un algorithme d’ACP pour la reconnaissance faciale

III - Les limites de l’ACP

IV - Quelques améliorations et alternatives

II - Programmation Python

Phase d'apprentissage

moyenne

$$\bar{V} = \frac{1}{n}\sum_{i=1}^{n} V_i$$

construitA

$$W_i = V_i - \bar{V} \longrightarrow A = (W_1, \ldots, W_n)$$

$$\forall \lambda_i \in Sp({}^tAA),\ \forall f_i \in E_{\lambda_i}({}^tAA),\ e_i = Af_i \in E_{\lambda_i}(A^tA)$$

contruitEi

$$(e_1, \ldots, e_n)$$

Base orthonormée de vecteurs propres de A^tA

Phase de reconnaissance

projectionVisage

$$\omega_i = {}^te_i\left(\tilde{V} - \bar{V}\right)$$

construitOmega

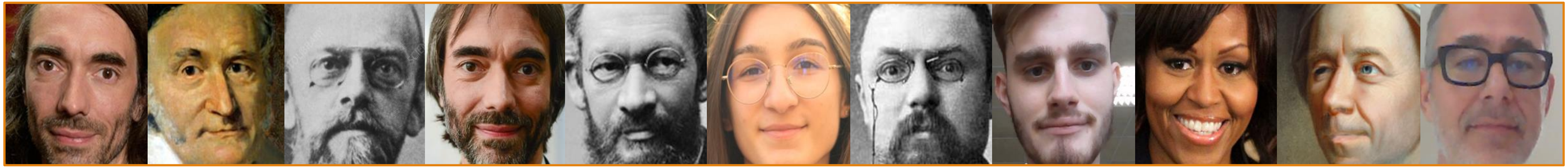
$$\Omega_k(V) = (\omega_1(V), \ldots, \omega_k(V)) \in \mathbb{R}^k \text{ où } k << n$$

distances

$$d_i = \left\|\Omega(\tilde{V}) - \Omega(V_i)\right\|$$

II - Visages et fantômes

V_1 V_2 V_3 V_4 V_5 V_6 V_7 V_8 V_9 V_{10} V_{11}



e_1 e_2 e_3 e_4 e_5 e_6 e_7 e_8 e_9 e_{10} e_{11}

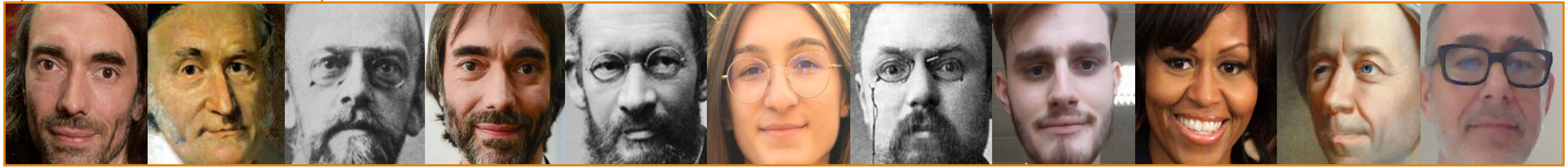
$\bar{V}$

II - Reconnaissance d’un visage

$\tilde{V}$

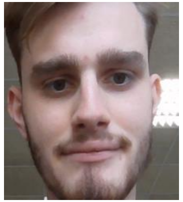
Nouveau visage à reconnaitre

Base de données des visages



14,4%	47,2%	7,1%	0%	13,1%	45,3%	9,4%	**72,4%**	15,8%	51,1%	22,9%

Facteur de reconnaissance



Visage identifié

II - Facteur de reconnaissance

- But : maximiser le facteur de reconnaissance.
- Facteur d'identification de $\tilde{V}$ au visage *i* :

$$R(\widetilde{V}, i) = 100\left(1 - \frac{d_i}{\underset{j\in[\![1;n]\!]}{max}\ d_j}\right)$$

  - Expérimentalement : on « reconnait » au dessus de 70%.
- Facteur de sureté de l'algorithme vis-à-vis d'une base de données :

$$R(\widetilde{V}) = 100\left(1 - \frac{\underset{j\in[\![1;n]\!]}{min}\ d_j}{\underset{j\in[\![1;n]\!]}{max}\ d_j}\right)$$

  - Si $R(\tilde{V}) < 60$: l'image n'est pas dans la banque.

I - Principe de l’Analyse en Composantes Principales

II - Programmation en Python d’un algorithme d’ACP pour la reconnaissance faciale

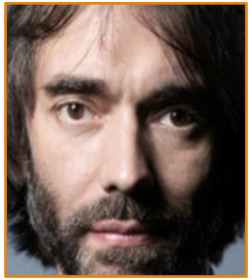
III - Les limites de l’ACP

IV - Quelques améliorations et alternatives

III - Limites de l'ACP

expressions différentes

luminosité variée



masque

limite temporelle
(complexité en O($k.n.N$))

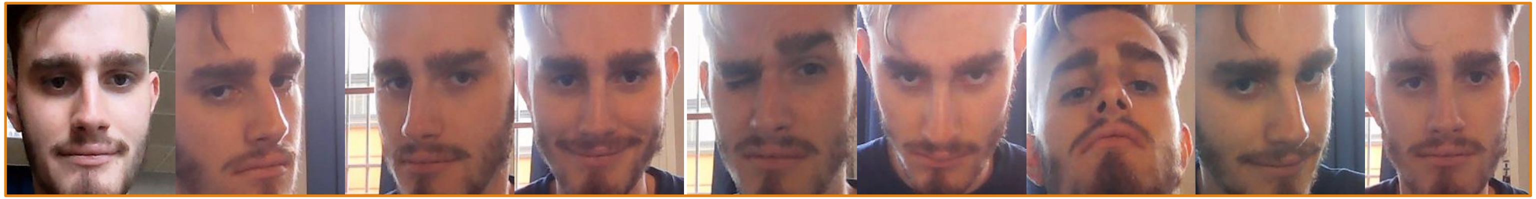
I - Principe de l’Analyse en Composantes Principales

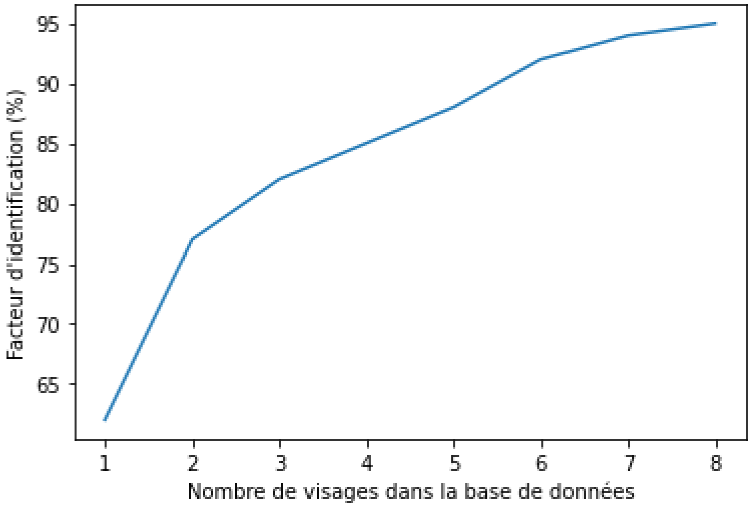
II - Programmation en Python d’un algorithme d’ACP pour la reconnaissance faciale

III - Les limites de l’ACP

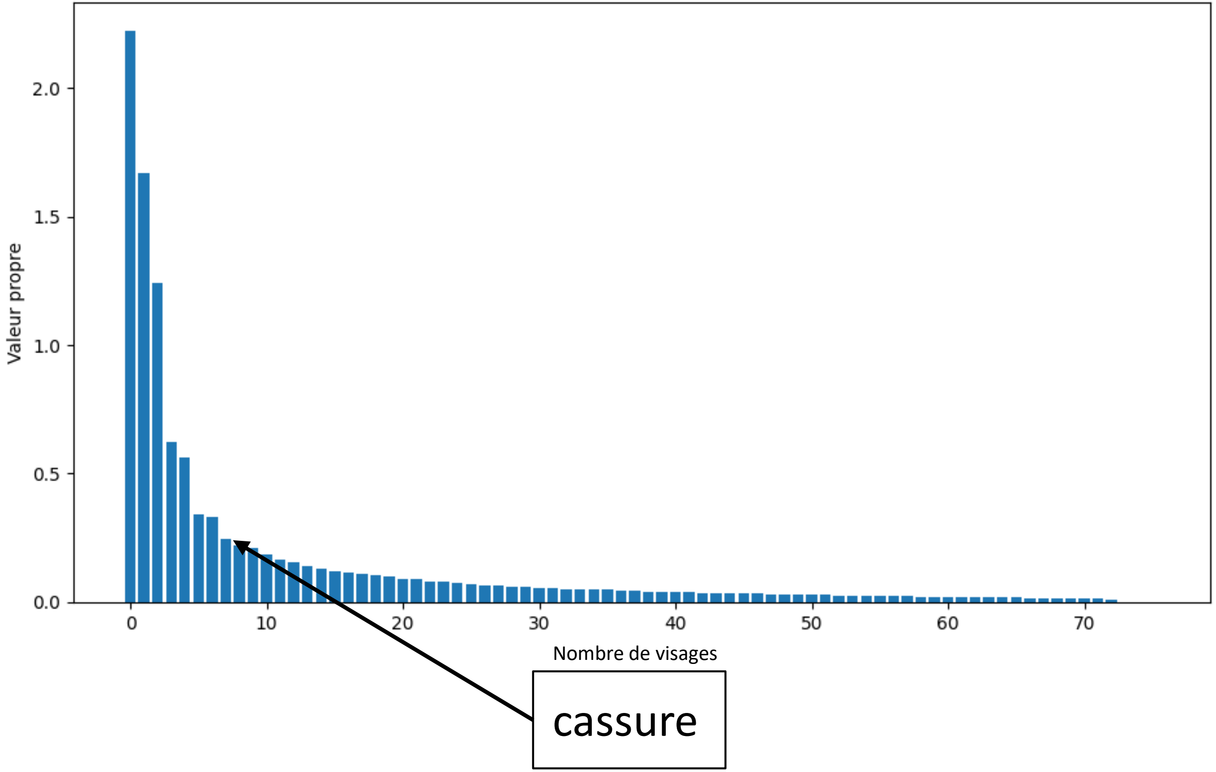
IV - Quelques améliorations et alternatives

IV - Efficacité de l'ACP en fonction du nombre de visages



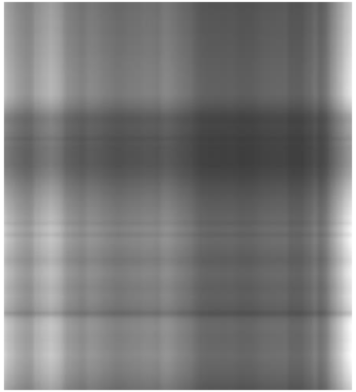


IV - Eboulement des valeurs propres



IV - Image de projection : méthode de Wu Zhou

Image originale



Carte de projection

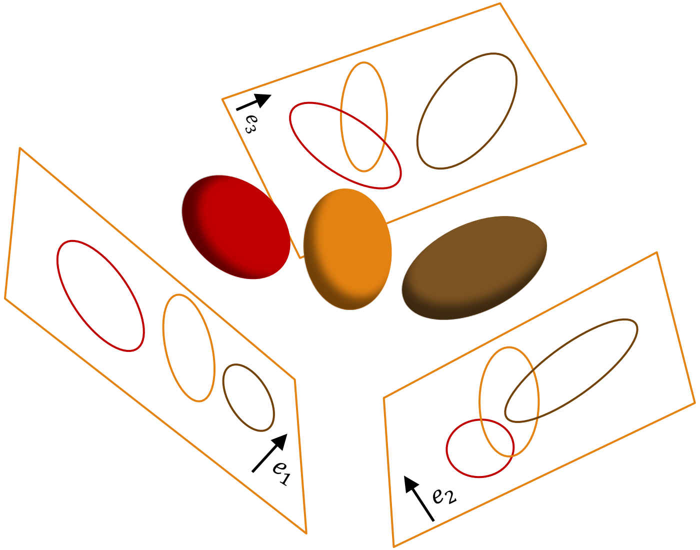
Image combinée

$$HI(x) = \sum_{y=1}^{p} I(x,y)$$

$$VI(y) = \sum_{x=1}^{m} I(x,y)$$

IV - Analyse Discriminante Linéaire (ADL)

- Principe et contexte :
  - Division de la base de données en différentes classes de personnes.
- Alternative à l'ACP :
  - Résout les problèmes de variation de pose, d'expression et de luminosité.
- Possède encore des limites :
  - Nécessite au moins deux visages par personne.
  - Besoin d'organiser la base de données.




SUV
灰色
轿车
下身黑色
白色
轿车
上身蓝色
下身黑色
白色
轿车
非机动车
黑色
轿车
成人
长裤
男
上身蓝色
下身深蓝色
黑色
轿车
灰色
面包车
红色
轿车
成人
短袖
女
上身灰色
下身黑色
非机动车

Conclusion

Annexe : démonstration vecteurs propres de $A^t A$

$$\boxed{\forall \lambda_i \in Sp({}^t AA),\ \forall f_i \in E_{\lambda_i}({}^t AA),\ e_i = Af_i \in E_{\lambda_i}(A^t A)}$$

Démonstration : *On pose* $L = {}^t AA$ *et* $Sp(L) = \lambda_1, \ldots, \lambda_n$. *Alors,* $\forall \lambda_i \in Sp(L), \exists X_i \in \mathbb{R}^n \setminus \{0\} / LX_i = \lambda_i X_i \Rightarrow {}^t AAX_i = \lambda_i X_i \Rightarrow A^t A(AX_i) = \lambda_i(AX_i)$
$\Rightarrow \forall \lambda_i \in Sp(L), \exists e_i \in \mathbb{R}_N \setminus \{0\} / A^t Ae_i = \lambda_i e_i$ *donc* $\lambda_i \in Sp({}^t AA) \Rightarrow \lambda_i \in Sp(A^t A)$.

Annexe : construction des composantes principales

On cherche à s'assurer que les axes principaux choisis à chaque itération sont :

1) De sorte que la variabilité soit maximale;
2) Orthogonaux (i.e. indépendants des autres, totalement décorrélés des autres variables);
3) Normés.

1) $Y_i = {}^t v_i \; X \quad v_1 = \underset{\underline{v} \in \Sigma^{N-1}}{\operatorname{argmax}}(Var({}^t \underline{v} \; X)) \;\; \forall k \in [\![2, N]\!], \; v_k = \underset{\underline{v} \in \Sigma^{N-1} \cap T_{k-1}}{\operatorname{argmax}} (Var({}^t \underline{v} \; X))$

2) $T_k = \left\{ \underline{v} \in \mathbb{R}^N / \, \forall i \in [\![1, k]\!], \; Cov({}^t \underline{v}_i \; \underline{X}, {}^t \underline{v} \; \underline{X}) = 0 \right\}$

3) $\Sigma^{N-1} = \{ \underline{v} \in \mathbb{R} / \, \|\underline{v}\| = 1 \}$

Annexe : programme

```
def moyenne(visages):
    "Calcule le vecteur colonne visage moyen."
    taille = np.shape(visages[0])[0]
    somme = np.zeros((taille,1))
    for vecteur in visages:
        somme+=vecteur
    return somme//len(visages)
```

```
def construitA(visages,moy):
    "Construit A en concaténant les n vecteurs colonnes des visages auxquels on a
soustrait le visage moyen."
    N = np.shape(visages[0])[0]
    A = np.zeros((N,0))
    for vecteur in visages:
        A = np.concatenate((A,vecteur-moy),axis=1)
    return A
```

Annexe : programme

```
def construitEi(visages,moy):
    """Renvoie le liste de n vecteurs propres de taille (N,) de AtA.
    Ils forment une base orthonormée de l'espace des visages.
    Pour chaque f_i vecteur propre de tAA, on construit e_i = A*f_i vecteur propre
de AtA."""
    A = construitA(visages,moy)
    L = np.dot(np.transpose(A),A)
    valP,vectP = np.linalg.eig(L)
    "valP est une liste de n valeurs propres, vectP est une liste de n vecteurs
propres de dimension (N,)."
    liste_ei = [np.dot(A,vectP[:,i]) for i in range(len(vectP))]
    #vectP[:,i] siginifie que l'on prend le i-ème vecteur propre de la matrice tAA
    liste_ei = [e/np.linalg.norm(e) for e in liste_ei] #Pour avoir une BON (normée)
de vecteurs propres
    return liste_ei
```

Annexe : programme

```
def projectionVisage(V,moy,e_i):
    """Projette le visage i, auquel on a soustrait le visage moyen, sur le vecteur
propre e_i. On fait le produit entre la transposée de e_i et le visage pour
renvoyer un flottant."""
    V_Moy = (V-moy)
    return float(np.dot(np.transpose(e_i),V_Moy))
```

```
def construitOmega(V,moy,k):
    "Concatène k <= n projections pour renvoyer un vecteur colonne de taille (k,)."
    "On choisit en pratique k << n."
    liste_projections = [projectionVisage(V,moy,liste_ei[i]) for i in range(k)]
    return np.array(liste_projections)
```

Annexe : programme

```
def distance(visages,V,k,moy):
    """Renvoie une liste de taille n où chaque élément est la distance_i entre
    Omega(V) calculé précédement et le V_i."""
    distances = []
    OmegaV = construitOmega(V,moy,k)
    for i in range(len(visages)):
        diff = OmegaV-construitOmega(visages[i],moy,k)
        d_i = np.linalg.norm(diff,2)
        distances.append(d_i)
    return distances
```

Annexe : programme

```
def trouveVisageCorrespondant(V,visages,k):
    "Renvoie le visage correspondant au visage V."
    t0=time.time()
    distances = distance(visages,V,k,moy)
    distance_mini,bonVisage = minimum(distances)
    t = round(time.time()-t0,4)
    texte = "Meilleur correspondance trouvée : image{}.png. Fait en {}
secondes.".format(bonVisage, t)
    facteur = 100*(1-distance_mini/max(distances)) #Facteur de reconnaissance.
    return texte,facteur
```

Annexe : programme

```
def montreFantomes(visages,moy,size):
    "Renvoie une liste des n visages propres c'est-à-dire les vecteurs propres de
nos visages."
    fantomes = [Image.fromarray((liste_ei[i]+np.reshape(moy,
(size[0]*size[1],))).reshape((size[1],size[0]),order="F")).convert("RGB") for i in
range(len(visages))]
    return fantomes
```

Annexe : programme

```
def analyseValp(visages):
    """Permet de constater l'éboulement des valeurs propres afin de choisir k,
    la dimension du sous-espace."""

    A = construitA(visages,moy)
    L = np.dot(np.transpose(A),A)
    valP,vectP = np.linalg.eig(L)

    y = valP.tolist()
    print(y)

    y.sort()
    y.reverse()
    x = list(range(len(y)))
    plt.xlabel("N° image")
    plt.ylabel("Valeur propre")
    plt.title("Eboulement des valeurs propres")
    plt.bar(x,y)
    plt.show()
```

Annexe : programme

```
def uniformiseLumiere(M, on=False):
    "Méthode de Wu Zhou : fait ressortir les composantes saillantes."

    if not on:
        return M

    J = np.mean(M)
    hauteur,largeur = np.shape(M)
    Mbis = np.zeros((hauteur,largeur))

    listehix, listeviy = [],[]

    for x in range(largeur):
        hix = 0
        for y in range(hauteur):
            hix += M[y,x]
        listehix.append(hix)

    for y in range(hauteur):
        viy = 0
        for x in range(largeur):
            viy += M[y,x]
        listeviy.append(viy)

    for x in range(hauteur):
        for y in range(largeur):
            Mbis[x,y] = listehix[y]*listeviy[x]/J *M[x,y]
    return (Mbis*(256/np.max(Mbis))).astype(int)
```

Annexe : Complexité de l'algorithme

Images de taille N = m*p (nous : 55000)

- moyenne : O(n)
- construitA : O(n)
- construitEi : construitA + 2*O(n^2) + O(N*n)
- projectionVisage : liste_ei (chargée une seule fois) + O(N)
- construitOmega : k*projectionVisage
- distance : (n+1)*construitOmega
- minimum : O(k)
- trouveVisageCorrespondant : distance + minimum + max → **O($k.n.N$)**

- updateVisageTexte : O(n) + listeVisages + moyenne + construitEi

Annexe : construction des composantes principales (lien avec la matrice de variance-covariance)

Avec la matrice variance-covariance $L = A^t A$, les vecteurs propres doivent vérifier les 3 conditions précédentes.
L induit un produit scalaire sur $\mathbb{R}^N$ défini par :

$$\forall(\underline{v}, \underline{w}) \in (\mathbb{R}^N)^2,\ \langle \underline{v}, \underline{w} \rangle_L = {}^t\underline{v} L \underline{w}.$$

Donc $\underline{w} \in T_k \iff \forall i \in [\![1, k]\!],\ \langle \underline{w}, \underline{v}_i \rangle = 0.$
La recherche de composantes principales revient à rechercher les extremas de la fonction :

$$f_{Var} : \underline{v} \in \Sigma^{N-1} \longmapsto \langle \underline{v}, \underline{v} \rangle_L \in \mathbb{R}$$

Supposons que L est symétrique définie positive (ce qui est le cas pour nous par construction). Notons $\lambda_1 \geq \cdots \geq \lambda_N$ ses valeurs propres et $(e_1, \ldots, e_N)$ une base orthonormale de vecteurs propres correspondantes (qui existe et est unique d'après le théorème spectral). Alors :

$$\underset{\underline{v} \in \Sigma^{N-1}}{max} f_{Var}(\underline{v}) = \lambda_1 \quad \text{et} \quad \underset{\underline{v} \in \Sigma^{N-1}}{argmax} f_{Var}(\underline{v}) = e_1$$

$$\forall k \in [\![2, N]\!], \quad \underset{\underline{v} \in \Sigma^{N-1} \cap \{e_1, \ldots, e_{k-1}\}^{\perp}}{max} f_{Var}(\underline{v}) = \lambda_k \quad \text{et} \quad \underset{\underline{v} \in \Sigma^{N-1} \cap \{e_1, \ldots, e_{k-1}\}^{\perp}}{argmax} f_{Var}(\underline{v}) = e_k$$

Annexe : construction des composantes principales (lien avec la matrice de variance-covariance, suite)

Démonstration : D'après la théorème spectral, comme L est symétrique, elle peut être diagonalisée en base orthonormale. Soit $P \in \mathcal{O}_N(\mathbb{R})$ la matrice de passage contenant les vecteurs propres $(e_N, \ldots, e_1)$ et $D = diag(\lambda_1 \ldots \lambda_N)$ la matrice diagonale contenant les valeurs propres ordonnées tel que $L = PD^tP$. On cherche à maximiser la forme

$$f_{Var}(\underline{v}) = {}^t\underline{v}L\underline{v} = {}^t({}^tP\underline{v})D({}^tP\underline{v}) = {}^t\underline{u}D\underline{u} = \sum_{i=1}^{N} \lambda_i u_i^2 = \frac{1}{\sum_{i=1}^{N} u_i^2} \sum_{i=1}^{N} \lambda_i u_i^2 \leq \lambda_1.$$

Ce maximum est atteint pour $\underline{u} = {}^t(1, 0, \ldots, 0)$ c'est-à-dire en $\underline{v} = P\underline{u} = e_1$.

Pour le second point, sous la contrainte ${}^t\underline{v}e_i = 0,\ \forall i \in [\![1, k-1]\!]$, on trouve : ${}^t\underline{v}e_i = 0 \iff {}^t(P\underline{u})e_i = 0 \iff {}^t\underline{u}\,{}^tPe_i = 0 \iff \underline{u}_i = 0$.

Ainsi, on a $f_{var}(\underline{v}) = {}^t\underline{v}L\underline{v} = \sum_{i=1}^{N} \lambda_i u_i^2 \leq \lambda_k$, valeur qui est atteinte pour $\underline{v} = e_k$.

Ainsi, ce dernier théorème nous prouve que les composantes principales d'un vecteur X sont les variables $Y_i = {}^te_iX$ où les e_i sont les vecteurs propres ordonnés de la matrice variance-covariance L de X.