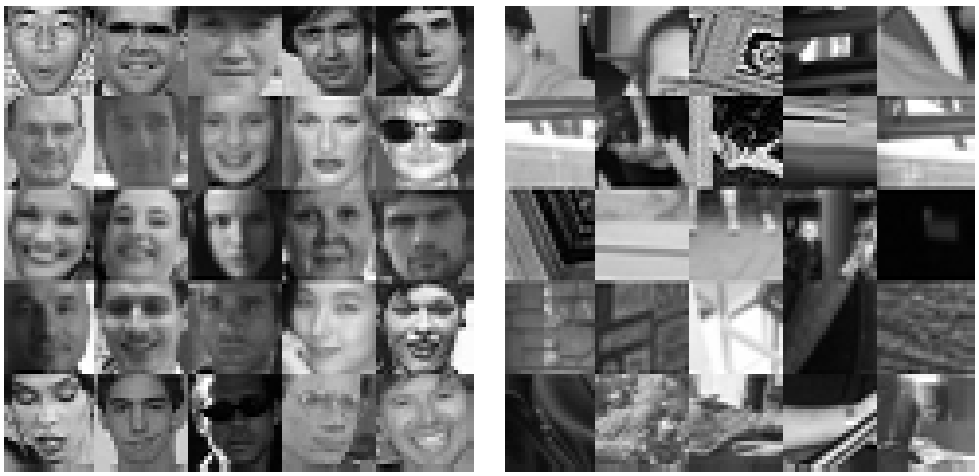
# SY32 – TD Apprentissage 03 : Classification de visages

## Exercice 1 : Classification de visages

L'archive fournie avec le TD contient un dossier `imageface` avec 4 000 images de visages et 17 256 images de fond réparties en un ensemble d'apprentissage (sous-dossier `train`) et un ensemble de test (sous-dossier `test`) de la manière suivante :

| | # positifs | # négatifs |
|---|---|---|
| apprentissage | 3 000 | 12 000 |
| test | 1 000 | 5 256 |

La Figure 1 montre un sous-échantillon de ces images. Toutes les images sont de taille $24 \times 24$.



(a) Visages. (b) Non visages.

FIGURE 1 – Exemples d'images.

**1 .** La commande ci-dessous permet de lire l'image `imageface/train/pos/00001.png`, de convertir les valeurs des pixels à valeurs entières $[[0; 255]]$ en valeurs flottantes $[0; 1]$ et d'afficher l'image.

```
from skimage import io, util
i = 1
I = io.imread('imageface/train/pos/%05d.png'%i)
I = util.img_as_float(I)
io.imshow(I)
```

**2 .** On souhaite utiliser directement les pixels comme représentation des images. La commande `I.flatten()` permet d'aplatir l'image qui est un tableau de taille $24 \times 24$ en un vecteur de longueur $576 = 24 \times 24$. Charger toutes les images de l'ensemble d'apprentissage sous la forme d'un tableau `X_train` comportant $15000 = 3000 + 12000$ lignes, une pour chaque image et 576 colonnes.

**3 .** Construire le vecteur `y_train` associé aux images, on utilisera la valeur $+1$ pour les visages et $-1$ pour les non visages.

**4 .** Utiliser la librairie Scikit-learn pour apprendre un classifieur Adaboost utilisant 10 itérations.

```
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(n_estimators=10)
clf.fit(X_train, y_train)
```

**5 .** Calculer le taux d'erreur du classifieur sur les images de test.

**6 .** Reprendre l'apprentissage avec 50 itérations.

**7 .** Essayer d'autres classifieurs de la librairies :

- $k$-plus proches voisins : `https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification`
- Arbre de décision : `https://scikit-learn.org/stable/modules/tree.html#classification`

(a) Visage d'origine. (b) Inversion *left-right*. (c) Inversion *up-down*.

FIGURE 2 – Images inversées.

— Forêt aléatoire : `https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees`

— SVM : `https://scikit-learn.org/stable/modules/svm.html#classification`

**8 .** Créer une copie des images de **test** mais inversées horizontalement, le regard dans la direction opposée (utiliser la fonction `numpy.fliplr`). Le rendu est illustré en Figure 2 (b). Tester les classifieurs entraînés précédemment sur ces données inversées.

**9 .** Créer une copie des images de **test** mais inversées verticalement, la tête en bas (utiliser la fonction `numpy.flipud`). Le rendu est illustré en Figure 2 (c). Tester les classifieurs entraînés précédemment sur ces données inversées.

**10 .** Reprendre l'exercice en utilisant les caractéristiques HOG (Histogram of Oriented Gradients) pour représenter les images : `https://scikit-image.org/docs/stable/api/skimage.feature.html?highlight=hog#skimage.feature.hog`

```
from skimage.feature import hog
I = io.imread('imageface/train/pos/00001.png')
X = hog(I)
```

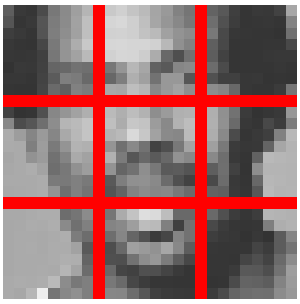


FIGURE 3 – Visage découpé en $3 \times 3$ blocs.

**11 .** Reprendre l'exercice en utilisant les LBP (Local Binary Patterns), suivant les indications suivantes. En classification, les histogrammes des LBP sont de bonnes représentations pour classifier des textures répétitives. Les visages ne correspondent pas à ce cas d'usage, mais on sait qu'il présentent des yeux, un nez, une bouche, etc, selon une organisation fixe (à des positions données). Une solution simple consiste à calculer un petit histogramme pour chaque bloc, puis à les concaténer en un histogramme de l'ensemble, toujours dans le même ordre.
Implémenter la fonction de calcul des features histogrammes concaténés de LBP d'une image, avec les paramètres suivants :

— Couper et traiter l'image en $3 \times 3$ blocs, comme illustré en Figure 3.

— Fixer les paramètres $P = 8$, $R = 1$, $bins = 10$ intervalles pour chaque sous-histogramme.

— Normaliser les sous-histogrammes (`density=True`).

LBP : `https://scikit-image.org/docs/stable/api/skimage.feature.html?highlight=lbp#skimage.feature.local_binary_pattern`

**Liste de fonctions utiles :**

- `numpy.flatten`
- `numpy.ravel`
- `skimage.io.imread`
- `skimage.util.img_as_float`
- `skimage.io.imshow`
- `numpy.zeros`
- `numpy.count_nonzero`
- `numpy.mean`
- `numpy.sum`
- `numpy.full`
- `sklearn.ensemble.AdaBoostClassifier`
- `sklearn.neighbors.KNeighborsClassifier`
- `sklearn.tree.DecisionTreeClassifier`
- `sklearn.ensemble.RandomForestClassifier`
- `sklearn.svm.SVC`
- `numpy.fliplr`
- `numpy.flipud`
- `skimage.feature.hog`
- `skimage.feature.local_binary_pattern`
- `skimage.feature.multiblock_lbp`
- `numpy.array_split`
- `numpy.histogram`