TP 3 - Apprentissage non-supervisé de descripteurs

Le but de ce TDM est de s'intéresser à la classification d'une collection d'image en mettant en place un processus d'apprentissage non supervisée de descripteurs.

Nous préconisons l'utilisation de KMeans pour la construction des descripteurs, mais vous êtes également libre de proposer en bonus des implémentations alternatives en passant par les auto-encodeurs implémentables en utilisant Keras comme vu dans le UE "DL"

Le TDM se découpera en trois parties reflétant les étapes d'apprentissage non-supervisé de descripteurs et des processus de classification sur la base de ces descripteurs appris.

Extraction des patchs

La première partie consiste en la décomposition de l'image en unités (patch) sur lesquelles et à partir des quelles nous apprendrons les descripteurs. En explorant une grande quantité des patchs nous essayerons d'identifier des patrons qui apparaissent fréquemment et qui, par conséquent, peuvent renseigner sur certains caractéristiques des images.

A partir d'une base d'image, il vous sera demande de générer une série de patch qui constitueront la base de l'apprentissage supervisé. Nous travaillerons sur des images couleurs et nous adopterons donc une structuration des patchs de la forme suivante : `(w - largeur, w - hauteur, 3 - canaux RGB)`. Certaines images sont en niveau de gris dans la base, il faudra faire attention à les traiter en tant qu'images couleurs.

Afin de faciliter la suite du travail, les patchs seront tous applatis en étant représentés comme des vecteurs de taille w*w*3

Nous continuerons de travailler sur la base *caltech101_subset.files*

Q1/ Il vous est demandé d'écrire une fonction qui permet de générer un numpy.array (patchs) sauvegardé dans un fichier patchs_w_nbp.npy qui rassemble un ensemble de patchs aléatoires extraits à partir des images de la base caltech101_subset.

- `w` indique la largeur et la hauteur des patchs
- `nbp` indique le nombre de patchs aléatoires extraites à partir d'une image

La forme de patchs (`patchs.shape`) doit correspondre à `(nbp*nb_images,w*w*3)`

Vous pouvez utiliser la fonction `sklearn.feature_extraction.image.extract_patchs_2d` pour l'extraction des patchs.

Avant de sauvegarder les patchs, il faut aussi veiller à normaliser la representation. En effet, des conditions de lumière et de contrastes différentes pourraient venir influer sur la qualité des patchs en dénaturant leur nature. Un processus de normalisation (soustraction de moyenne et division par l'écart type de l'ensemble des valeurs par composant au sein des patchs) qui s'appuie sur `sklearn.preprocessing.StandardScaler` permet de réduire en partie ces problèmes.

Générer les fichiers contenant les patchs normalisés:

- `patchs_6_10.npy`, `patchs_6_25.npy`, `patchs_6_50.npy`
- `patchs_20_10.npy`,`patchs_20_25.npy`,`patchs_6_50.npy`
- `patchs_40_10.npy`, `patchs_40_25.npy`, `patchs_40_50.npy`

Construire les descripteurs

Utilisation de *KMeans* pour regrouper les `patchs` en plusieurs `clusters`. Cette structuration de l'espace constituera la base du descripteur qu'on va apprendre. Intuitivement, nous pourrions nous dire que les patchs se retrouvant dans le même cluster partage la même caractéristique. Il y aura autant de caractéristiques que des clusters.

Ainsi, nous construirons la fonction `f(patch)=(x_1,..,x_K)` qui associera à chaque patch une caractéristique `x_i` décrivant le lien entre le patch et le `cluster_i`

Selon le choix de `k` et selon la manière dont nous souhaiterions caractériser le lien entre un `patch` et le `cluster`, nous définirons une fonction `f` et, par conséquent, un descripteur spécifique. La définition de la fonction `f` se fera dans la section suivante (*Affecation*). Pour le moment, on se concentre sur le calcul de `clusters` et de `centres`.

Q2/ Construire une fonction qui charge un ensemble de patchs et construit un modèle `sklearn.cluster.KMean` en précisant le `nombre de centres`.

Pour chacun des ensembles de patch de la Q1, générer et sauvegarder plusieurs modèle KMeans en considérant 8, 16, 32, 64, 128 centres.

Par exemple, pour le fichier `patchs_6_10.npy` vous devez générer les modèles KMeans suivants : `model_6_10_8.pkl`, `model_6_10_16.pkl`, `model_6_10_32.pkl`, `model_6_10_64.pkl`, `model_6_10_128.pkl`

Afin de rendre le calcul des centres plus rapide, vous pouvez réduire la valeur par défaut du nombre d'initialisations de différents centres considérées en utilisant le paramètre n_init=1 ou 3 lors de l'appel à `KMeans`.

La sauvegarde des fichiers peut être fait avec la commande :

```
import pickle pickle.dump(model, open(dirpath+"/model.pkl", "wb"))
```

Q3/ Les centres de chaque cluster constituent une représentation visuelle d'une caractéristique du descripteur. Pour quelques uns de modèles construits à la Q2, réalisez une visualisation sous forme d'image des caractéristiques apprises.

Par exemple, si vous avez calculé 16 centres, vous pouvez coller `4 par 4` les caractéristiques correspondant au 16 centres. Vous obtiendrez ainsi une image ayant une taille de `(4*w+3*sp,4*w+3*sp,3)` où `w` - taille du patch, `sp` - taille de la séparation entre deux patch sur l'image générée.

Associer un descripteur à un patch

Premièrement, en **Q4**, vous écrirez une fonction qui permet d'écrire l'ensemble de patchs d'une image. Ensuite, en `Q5-Q7`, vous coderez plusieurs fonctions qui permettent d'associer un descripteur sur la base des modèles `KMeans` calculés en **Q2**.

Q4/ Il vous est demandé d'écrire une fonction qui permet de générer un `numpy.array` (contenant des `patchs`) sauvegardé dans un fichier `nom_w.npy` qui rassemble tous les patchs possibles (sans recouvrement) extraits à partir d'une image de la base `caltech101_subset`.

- `nom` indique le nom de l'image originelle
- `w` indique la largeur et la hauteur des patchs

Générer pour chaque image (`nom.png`) de la base d'images, les fichiers suivant:

- nom_6.npy
- nom_20.npy
- nom_40.npy

Q5/ Soit m un modèle KMeans avec k clusters, écrivez une fonction (*hard_assignement*) qui associe à un patch un vecteur x de taille k tel que x_i=1 si le patch est associé au cluster_i, x_i=0 sinon.

Générer pour chaque image (nom.png) de la base d'images, les fichiers suivant:

- nom_hard_6.npy
- nom_hard_20.npy
- nom_hard_40.npy

Q6/ Soit m un modèle KMeans avec k clusters, écrivez une fonction (*soft_assignement*) qui associe à un patch un vecteur x de taille k tel que x_i=1/(1+exp(-d)) où d-dist entre le patch et le centre du cluster_i.

Générer pour chaque image (nom.png) de la base d'images, les fichiers suivant:

- nom_soft_6.npy
- nom_soft_20.npy
- nom_soft_40.npy

Q7/ Soit m un modèle KMeans avec k clusters, écrivez une fonction (*soft_assignement*) qui associe à un patch un vecteur x de taille k tel que x_i = centre_i.T * patch. Ce mode d'affectation est fortement inspiré des convolutions que nous retrouvons dans les réseaux convolutionnels.

Générer pour chaque image (nom.png) de la base d'images, les fichiers suivant:

- nom_conv_6.npy
- nom_conv_20.npy
- nom_conv_40.npy

Uniformiser la description de l'image à travers le pooling spatial

Pour le moment, une image est décrite comme la somme des patchs la composant.

Si nous appliquons les fonctions **Q5-Q7** sur les patchs tels qu'extraits en Q4, nous nous trouverons en fonction de la taille de l'image avec un nombre variable de descripteurs d'une image à un autre.

Afin, d'uniformiser la représentation de l'image, nous imposerons une structuration en grille de l'image.

Nous mettrons en place un mécanisme de pooling qui associera à chaque cellule de la grille une valeur calculée en fonction des patchs recouverts par la cellule.

Afin de simplifier les opérations de pooling, nous considérons que l'image est représentée comme un vecteur à plat :

```
[   descr_patch1_x1, , , , ,descr_patch1_xd,   descr_patch2_x1, , , ,
,descr_patch2_xd,   descr_patch3_x1, , , , ,descr_patch3_xd,   ...
descr_patchm_x1, , , , ,descr_patchm_xd, ]
```

L'opération de `pooling` générera un nouveau `vecteur` à plat contenant les `descripteurs` de chaque `cellule` de la `grille`.

```
[   descr_cell1_x1, , , , ,descr_patch1_xd,   descr_cell2_x1, , , ,
,descr_cell2_xd,   descr_cell3_x1, , , , ,descr_cell3_xd,   ...
descr_celln_x1, , , , ,descr_celln_xd, ]

descr_celli_x1 =
pooling(descr_patch(i*cell_width)_x1,descr_patch((i+1)*cell_width)_x1)
```

avec `cell_width=m/n` en faisant attention à ne pas déborder du tableau si division avec reste

Q8/ Codez une méthode de pooling qui garde le max de la fenêtre de pooling pour chaque composant: `[descr_patch(i*cell_width),descr_patch((i+1)*cell_width)]`.

Appliquer la méthode de pooling à chaque fichier généré aux question **Q5-Q7** en générant des nouveaux fichiers :

- `nom_conv_max_6_{nbcells}.npy`
- `nom_conv_max_20_{nbcells}.npy`
- `nom_conv_max_40_{nbcells}.npy`
- `nom_soft_max_6_{nbcells}.npy`
- `nom_soft_max_20_{nbcells}.npy`
- `nom_soft_max_40_{nbcells}.npy`
- `nom_hard_max_6_{nbcells}.npy`
- `nom_hard_max_20_{nbcells}.npy`
- `nom_hard_max_40_{nbcells}.npy`

Nous vous recommandons d'utiliser les valeurs suivantes pour `{nbcells}` : 8, 16, 32, 64, 128, 256.

Le fichier nom_hard_max_6_{nbcells}.npy contient le descripteur associé à l'image (nom.png) dans son intégralité en utilisant :

- une décomposition de l'image en patch de taille `w=6`
- une affectation des caracteristiques au niveau de patchs de type `hard_assignement` (**Q5**)
- un pooling de type `max` vers une grille de taille `{nbcells}`

Q9/ Codez une méthode de pooling qui garde le somme de la fenêtre de pooling pour chaque composant : [descr_patch(i*cell_width),descr_patch((i+1)*cell_width)].

Appliquer la méthode de pooling à chaque fichier généré aux question **Q5-Q7** en générant des nouveaux fichiers :

- `nom_conv_sum_6_{nbcells}.npy`
- `nom_conv_sum_20_{nbcells}.npy`
- `nom_conv_sum_40_{nbcells}.npy`
- `nom_soft_sum_6_{nbcells}.npy`
- `nom_soft_sum_20_{nbcells}.npy`
- `nom_soft_sum_40_{nbcells}.npy`
- `nom_hard_sum_6_{nbcells}.npy`
- `nom_hard_sum_20_{nbcells}.npy`
- `nom_hard_sum_40_{nbcells}.npy`

Nous vous recommandons d'utiliser les valeurs suivantes pour `{nbcells}` : 8, 16, 32, 64, 128, 256.

Le fichier nom_hard_sum_6_{nbcells}.npy contient le descripteur associé à l'image (nom.png) dans son intégralité en utilisant : a) une décomposition de l'image en patch de taille `w=6` b) une affectation des caracteristiques au niveau de patchs de type `hard_assignement` (**Q5**) c) un pooling de type `sum` vers une grille de taille `{nbcells}`

Q10/ Codez une méthode de pooling qui garde la moyenne de la fenêtre de pooling pour chaque composant : `[descr_patch(i*cell_width),descr_patch((i+1)*cell_width)]`.

Appliquer la méthode de pooling à chaque fichier généré aux question **Q5-Q7** en générant des nouveaux fichiers :

- `nom_conv_mean_6_{nbcells}.npy`
- `nom_conv_mean_20_{nbcells}.npy`
- `nom_conv_mean_40_{nbcells}.npy`
- `nom_soft_mean_6_{nbcells}.npy`
- `nom_soft_mean_20_{nbcells}.npy`
- `nom_soft_mean_40_{nbcells}.npy`
- `nom_hard_mean_6_{nbcells}.npy`
- `nom_hard_mean_20_{nbcells}.npy`
- `nom_hard_mean_40_{nbcells}.npy`

Nous vous recommandons d'utiliser les valeurs suivantes pour `{nbcells}` : 8, 16, 32, 64, 128, 256.

Le fichier nom_hard_mean_6_{nbcells}.npy contient le descripteur associé à l'image (nom.png) dans son intégralité en utilisant :

- une décomposition de l'image en patch de taille `w=6`
- une affectation des caracteristiques au niveau de patchs de type `hard_assignement` (**Q5**)
- un pooling de type `mean` vers une grille de taille `{nbcells}`

Classification

Mettez en place un protocole de validation croisée pour évaluer les performances des nouveaux descripteurs images (Q8 à Q10).

Pour chaque image (`nom.png`) du dataset chargez le contenu du fichier `nom_{type_assignemet}_{type_pooling}_{w}_{nbcells}.npy`

Assemblez ainsi l'ensemble des descripteurs pour l'ensemble du dataset.

Utilisez `sklearn.model_selection.cross_val_score` avec `5-folds`.

Rapportez les résultats sur les différents configurations calculées : `{type_assignemet}_{type_pooling}_{w}_{nbcells}`