RDFIA TME 1-2
Description d'images par SIFT et Bag of Words

Thomas Robert Remi Cadene Matthieu Cord

19 & 26 septembre

Comptes rendus à rendre

— Un compte rendu de suivi dans la soirée après la séance (par mail à `remi.cadene@lip6.fr` et `thomas.robert@lip6.fr`). Ce compte rendu contiendra en quelques lignes : (1) les réponses aux questions précédées par ★ ; (2) votre avancement dans le sujet, ce que vous avez réussi ou non ; (3) si vous voulez, des commentaires, remarques sur ce que vous avez compris ou non, etc.

— Un compte rendu global une semaine après la dernière séance (par mail), dont le but est de reprendre au propre le travail effectué pendant l'intégralité des séances de TP, en prenant du recul sur ce qui a été fait. Ce compte rendu représentera la majorité de la note de TP.

Les données et une version numérique du sujet sont accessibles
à l'adresse `http://webia.lip6.fr/~robert`

Objectifs

Pendant ces premiers TP de RDFIA, nous allons préparer le développement d'un premier modèle de classification d'images. L'objectif est de produire un algorithme capable de classer les images du jeu de données 15-Scenes, qui contient 4485 images appartenant à 15 catégories de scènes intérieures et extérieurs (cuisine, chambre, rue, *etc.*).

Pour cela, nous allons dans un premier temps **(Partie 1)** utiliser des **SIFT (Scale-invariant feature transform)**, qui est un descripteur visuel local et qui permet donc de caractériser numériquement un petit *patch* d'image. On cherchera ensuite **(Partie 2)** des descripteurs-types représentant des motifs fréquents parmi tous les SIFT de notre jeu d'images pour constituer un **dictionnaire visuel**. Puis, on aggrégera **(Partie 3)** les SIFT de chaque image avec la technique du **BoW (Bag of Words)** qui permet de représenter numériquement et de manière condensée une image.

Dans le TP suivant, on apprendra a classer chaque image à partir de sa représentation condensée à l'aide d'un **SVM (Support Vector Machine)**.

Données · Code · Python

Pendant nos TP, nous utiliseront Python 3 ainsi que divers *packages* très courant pour le calcul scientifique et le machine learning (numpy, matplotlib, scikit-learn, pytorch, ...). Les bases du langage Python sont présentées ici : `http://webia.lip6.fr/~robert/rdfia-python/`

Commencez par télécharger les données et le code fournis. Dans un terminal, placez-vous dans le dossier contenant le code, et tapez `jupyter notebook` pour lancer l'outil de notebook, il vous permet de prototyper de façon itérative. La commande `python` ou `python3` lance un terminal Python et vous indique la version utilisée, verifiez qu'il s'agit de la version 3. Suivez cette commande d'un nom de fichier (*e.g.* `python main.py`) pour lancer un script Python.

Partie 1 – SIFT

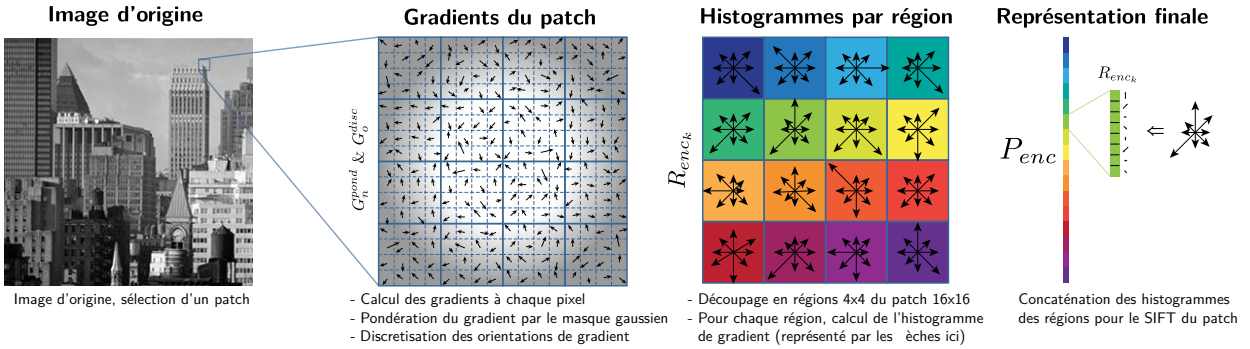

FIGURE 1 – Illustration du processus de calcul du SIFT pour un patch

Un SIFT (Scale-invariant feature transform) est un descripteur visuel local. Il va permettre de transformer un petit *patch* d'image de taille 16×16 pixels en une représentation numérique (un vecteur de dimension 128 ici) qui sera robuste à des perturbations et transformations, c'est à dire que deux patchs similaires auront des SIFT très proches.

1.1 Calcul du gradient d'une image

Dans un premier temps, on s'intéresse au calcul du gradient d'une image au pixel (x, y) :

$$G(x,y) = \left[\frac{\partial I}{\partial x} \; \frac{\partial I}{\partial y}\right]^\top = [I_x \; I_y]^\top \tag{1}$$

En pratique, on approximera les derivées partielles par des différences finies : dans ce cas les dérivées partielles I_x et I_y peuvent s'obtenir en calculant la convolution (notée $\star$) de l'image par un masque : $I_x = I \star M_x$, $I_y = I \star M_y$. On utilisera ici les masques de Sobel 3×3 suivants :

$$M_x = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2}$$

A partir de I_x et I_y on peut ensuite calculer la norme du gradient $G_n = ||G|| = \sqrt{I_x^2 + I_y^2}$ et l'orientation G_o (code fourni).

Questions

1. Montrer que les masques M_x et M_y sont séparables, c'est à dire qu'ils peuvent s'écrire $M_x = h_y \times h_x^\top$ et $M_y = h_x h_y^\top$ avec h_x et h_y deux vecteurs colonne de taille 3 a déterminer
2. Quel est l'intêret de séparer le filtre de convolution ?

1.2 Calcul de la représentation SIFT d'un patch

On s'intéresse maintenant à l'algorithme permettant d'obtenir la représentation SIFT d'un patch d'image P de 16×16 pixels. Le processus global décrit ci-après est présenté sur la figure **??**.

Le patch P va être décomposé en 16 sous-régions R_i de 4×4 pixels chacune. Chaque région va être encodée sous la forme d'un vecteur $R_{enc_k} \in \mathbb{R}^8$. Pour cela, on utilisera le descripteur SIFT (Scale Invariant Feature Transform) qui consiste à calculer un histogramme d'orientation des gradients.

Le patch complet sera décrit par un vecteur $P_{enc} \in \mathbb{R}^{128}$ qui sera la concatenation des encodages de ses régions.

Algorithme

L'algorithme du SIFT est donc le suivant :

— Pour chaque pixel du patch P on calcule le gradient (au sens gradient d'image) à chaque pixel sous la forme d'une matrice d'orientation des gradients $G_o \in \mathbb{R}^{16\times16}$ et d'une matrice de norme des gradients $G_n \in \mathbb{R}^{16\times16}$.

— On discrétise la matrice d'orientation des gradients en 8 valeurs. La valeur 0 sera associée à l'orientation $0°$ (nord), la valeur 1 à l'orientation $45°$ (nord-est), la valeur 2 à l'orientation $90°$, etc. On obtiendra donc une matrice $G_o^{disc} \in N^{16\times16}$.

— On calcule une gaussienne $N \in \mathbb{R}^{16\times16}$ centrée sur P d'écart type $\sigma = 0.5 \times width$ avec $width = 16$ pixels. On pondère G_n par N pour obtenir G_n^{pond}.

— Pour chaque région de taille 4×4, on calcule un histogramme des orientations des gradients $R_{enc_k} \in \mathbb{R}^8$. Chaque coordonnée de l'histogramme est la somme des normes des gradients pondérés pour une orientation. Par exemple, $R_{enc_k}[0]$ est la somme des valeurs de $G_n^{pond}[i,j]$ tel que (i,j) soit dans la region k et que $G_o^{disc}[i,j] = 0$.

— On concatène nos 16 vecteurs R_{enc_k} pour former le descripteur $P_{enc} \in \mathbb{R}^{128}$.

On effectuera ensuite un **post-processing** sur P_{enc} comme suit :

— On calcule la norme 2 du descripteur P_{enc}. Si elle est inférieure à un seuil de 0.5, P_{enc} est fixé au vecteur nul et est retourné immédiatement.

— Sinon, on normalise le descripteur pour avoir une norme euclidienne unité ($||P_{enc}||_2 = 1$).

— Enfin, les valeurs supérieurs à 0.2 sont seuillées (ramenées à 0.2), puis le descripteur est normalisé à nouveau.

Pratique

On vous fourni dans le fichier `tools.py` un certain nombre de fonctions qui vous seront utiles. Testez vos fonctions dans le *notebook* `main.ipynb` pré-rempli avant de les copier dans le fichier `sift.py` lorsqu'elles fonctionnent.

— Ecrire la fonction `compute_grad(I)` qui calcule le gradient de l'image en entrée et retourne I_x et I_y. Vous utiliserez la `conv_separable(I, ha, hb)` qui calcule $I \star M$ avec $M = h_a \times h_b^\top$.

— Ecrire la fonction `compute_grad_mod_ori(I)` qui retourne le module G_n et l'orientation discrétisée G_o^{disc} de l'image en entrée. On utilisera `compute_grad(im)` et `compute_grad_ori(Gn, Ix, Iy)` et retourne G_o^{disc}.

— Ecrire la fonction `compute_sift_region(Gn, Go, mask=None)` qui retourne la représentation P_{enc} d'un patch dont on aura fourni G_n et G_o^{disc} ainsi qu'un masque gaussian optionnel en entrée qui

permet de produire G_n^{pond} le cas échéant.
Le masque gaussien a fournir en entrée est généré par la fonction `gaussian_mask()` fournie.
Testez votre fonction manuellement et visuellement grace à la fonction `display_sift_region(I, compute_grad_mod_ori, compute_sift_region, x, y)` qui prend en entrée une image, des coordonnées x, y où découper un patch et vos fonctions de calcul.

Questions

3. Quel est le rôle de la pondération par masque gaussien ?
4. ★ Expliquez le rôle de la discrétisation des orientations
5. Justifiez l'intéret des differents post-processings appliqués au SIFT
6. ★ Expliquez en quoi le principe du SIFT est une façon raisonnable de décrire numériquement un patch d'image pour faire de l'analyse d'image
7. Interprétez les résultats que vous avez obtenus dans cette partie.

1.3 Calcul des SIFT sur la base d'images

Pour calculer les représentations SIFT d'une image, on doit choisir des points autour desquels calculer nos SIFT. Il existe différentes facon de facon, mais une technique classique est de simplement faire un *sampling* dense c'est à dire de prendre un patch tous les n pixels. Ici, on propose de prendre un patch 16×16 tous les 8 pixels dans chaque direction. Notez que nos patchs se recouvrent donc partiellement.

— Ecire la fonction `compute_sift_image(I)` qui calcule les SIFT d'une image.
La fonction `x, y = dense_sampling(I)` fourni deux listes `x` et sur `y`. En prennant toutes les paires (x_i, y_i) on a toutes les coordonnées en haut à gauche des patchs a représenter. Les SIFT seront sauvegardées dans un array numpy de taille $n_x \times n_y \times d_{SIFT}$ avec n_x nombre de coordonées sur l'axe x, n_y nombre de coordonnées sur l'axe y, et d_{SIFT} la taille d'un SIFT.
— Sauvez vos fonctions dans le fichier `sift.py` puis lancez le fichier `compute_sift_image.py` pour calculer les SIFT sur toute la base de données.

Partie 2 – Dictionnaire visuel

Théorie

A partir de tous les descripteurs SIFT P_{enc} extraits de notre base de données, on veut calculer un "dictionnaire visuel". Un "mot" de ce dictionnaire $c_m \in \mathbb{R}^{128}$ est une sorte de descripteur SIFT-type. L'objectif du dictionnaire est de représenter au mieux l'ensemble des SIFT de la base de données en un nombre fixé de SIFT-type qui correspondront à des patterns fréquents.

Concretement, l'objectif est de pouvoir determiner un ensemble de K centres (clusters) c_m de l'espace SIFT minimisant la distorsion des données x_i (les SIFT). On veut résoudre un problème de quantification optimale et trouver les centres qui permettent de résoudre la minimisation suivante :

$$\min_{c_m} \sum_{m=1}^{M} \sum_{x_i \in C_m} ||x_i - c_m||_2^2 \tag{3}$$

Une facon classique de résoudre ce problème est d'utiliser l'argorithme K-Means dont le fonctionnement est assez simple :

— Initialiser M centres c_m sur des points x_i tirés au hasard
— Alterner jusqu'à convergence entre :
  — Assigner chaque point x_i au centre le plus proche
  — Recalculer les centres c_m comme étant la moyennes des points qui lui sont assignés

Pratique

Grâce a la fonction `compute_load_sift_dataset`, on obtient une liste dont chaque élément correspond à une list des SIFT d'une image.

Complétez la fonction `compute_visual_dict` pour qu'elle calcule le dictionnaire visuel, c'est-à-dire les clusters, des SIFT de la base. Pour cela, on utilisera scikit-learn : `http://scikit-learn.org/stable/modules/clustering.html#k-means`
Pensez à ajouter un vecteur de zéros comme "cluster" qui servira aux SIFT nuls.

Vous pouvez ensuite utiliser la fonction `compute_or_load_vdict` pour calculer les clusters sur la base de SIFT (*cf* notebook).

La fonction `get_regions_and_sifts` vous permet d'obtenir des patchs d'image ainsi que les SIFT correspondants. Pour analyser les éléments du dictionnaire visuel, on propose de chercher des patchs d'image qui leur sont proches dans l'espace SIFT. Manipulez ces éléments pour tenter d'étudier le contenu de notre dictionnaire visuel. (Ex : chercher les 50 patchs les plus proches d'un cluster, afficher pour chaque cluster le patch le plus proche, etc.) Pour cela, la fonction `display_images` vous permet d'afficher des régions.

Mettez votre fonction `compute_visual_dict` dans le fichier `kmeans.py`.

Questions

8. ★ Justifiez la nécessité du dictionnaire dans le processus général de reconnaissance d'image que nous sommes en train de mettre en place.
9. Considérant les points $\{x_i\}_{i=1..n}$ assignés à un cluster c, montrer que le centre du cluster qui minimise la dispersion est bien le barycentre (moyenne) des points x_i :

$$\min_c \sum_i ||x_i - c||_2^2 \tag{4}$$

10. En pratique, comment choisir le nombre de clusters "idéal" ?
11. Pourquoi l'analyse des éléments du dictionnaire doit se faire à travers des exemples de patchs et pas directement ?
12. Commentez les résultats que vous aurez obtenus.

Partie 3 – Bag of Words (BoW)

Théorie

Dans cette partie, l'objectif est d'obtenir une représentation numérique de chaque image qui permette de caractériser chaque image et qui permettra par la suite de classer l'image.

Pour l'instant, on dispose pour chaque image d'un ensemble de descripteurs locaux (eventuellement pas le même nombre pour chaque image), et un dictionnaire visuel des descripteurs moyens fréquents.

L'objectif de la méthode BoW est de synthétiser l'ensemble des descripteurs locaux en un seul descripteur global avec l'aide du dictionnaire.

Pour une image I donnée, on a tous les descripteurs x_i de ses patchs dans une matrice $\mathrm{X} \in \mathbb{R}^{n_{patch} \times d}$. On dispose également du dictionnaire visuel sous forme matricelle $\mathbf{C} \in \mathbb{R}^{M \times d}$.

Le calcul du descripteur BoW de l'image I se fait en deux étapes :

— Le *coding* : chaque descripteurs x_i est codé sous la forme d'un vecteur $h_i \in \mathbb{R}^M$ par "projection" sur le dictionnaire visuel. On obtient une nouvelle matrice $\mathbf{H} \in \mathbb{R}^{n_{patch} \times M}$.
Dans notre cas, on utilise un codage "plus proche voisin" : le vecteur $h_i \in \mathbb{R}^M$ est un vecteur one-hot indiquant quel mot du dictionnaire est le plus proche :

$$h_i[j] = \begin{cases} 1 & \text{si } j = \arg\min_k ||x_i - c_k||^2 \\ 0 & \text{sinon} \end{cases} \tag{5}$$

— Le *pooling* : on aggrège les h_i sur l'ensemble des descripteurs locaux pour obtenir un vecteur $z \in \mathbb{R}^M$ qui décrit globalement l'image.
Dans notre cas, z sera simplement la somme des h_i.

Les descriteurs seront ensuite normalisés avec une norme euclidenne (L_2).

Pratique

Ecrire une fonction `compute_feats` qui prend en entrée la matrice de dictionnaire visuel C et la matrice de l'ensemble des SIFT d'une image X, et retourne la représentation BoW de l'image z.

Utilisez le code fourni dans le notebook pour visualiser votre représentation BoW.

Mettez votre fonction `compute_feats` dans le fichier `bow.py` une fois vérifiée.

Questions

13. ★ Finalement, que représente concretement notre vecteur z pour une image ?
14. Montrez et discutez les résultats visuels obtenus
15. Quelle est l'intérêt du codage au plus proche voisin ? Quel(s) autre codage pourrait-on utiliser ?
16. Quelle est l'intérêt du pooling somme ? Quel(s) autre pooling pourrait-on utiliser ?
17. Quelle est l'intérêt de la normalisation L_2 ? Pourrait-on utiliser une autre normalisation ?