Calcul de descripteurs locaux et de dictionnaires visuels

Travaux sur Machines Encadrés

À toutes fins utiles, se reporter à l'url http://webia.lip6.fr/∼thomen/Teaching/RDFIA.html. On s'intéressera à la base "15-scenes" contenant 4485 images de 15 catégories de scènes intérieurs et extérieures, dont certains exemples sont montrés à la figure 1.

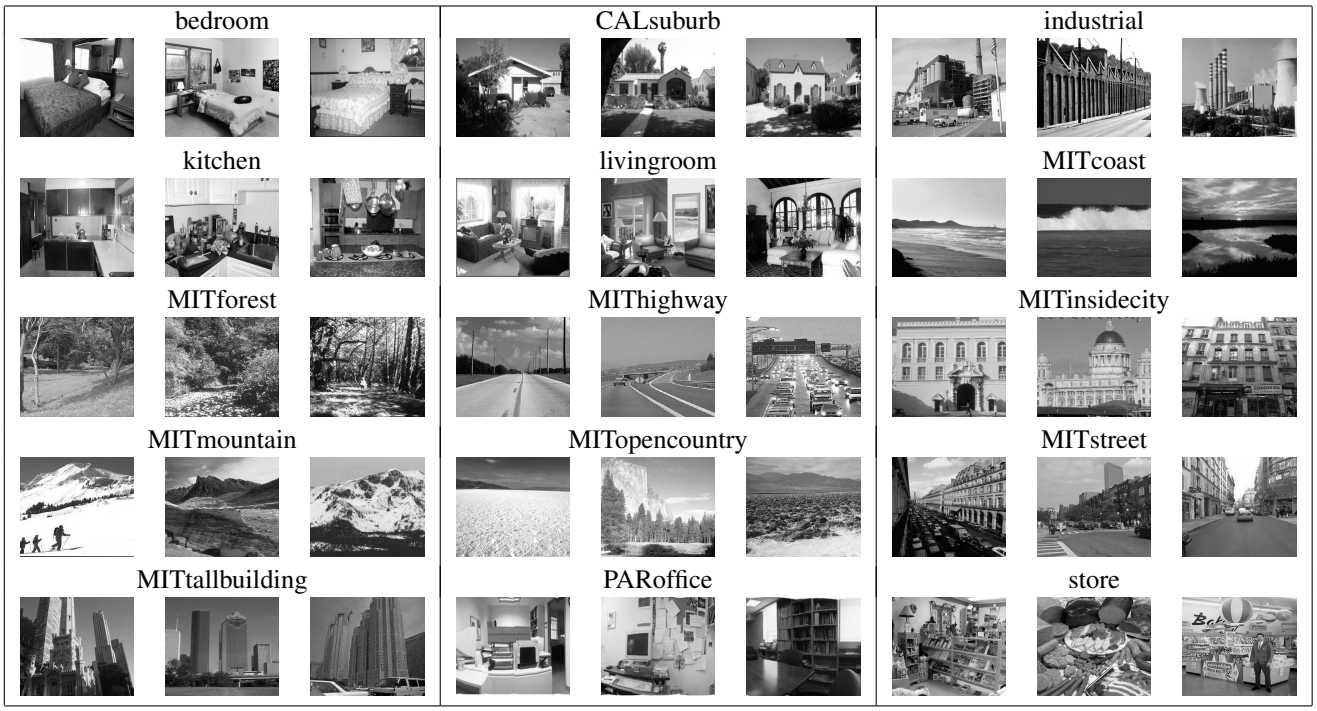


FIGURE 1 – Base de données

L'objectif final des deux séances consiste à mettre en place un système de classification d'images par apprentissage supervisé. Une fois le système appris, il sera en mesure d'affecter une des 15 catégories sémantiques à une image de test donnée.

L'objectif de ce premier TME consiste à calculer un dictionnaire visuel représentant des motifs courant dans des régions d'images de la base de la figure 1. L'exercice 1 consiste à extraire des descripteurs locaux discriminants de chaque région de l'image. L'exercice 2 consiste à mettre en place un algorithme de clustering (K-Means) dans l'espace des descripteurs afin de produire un dictionnaire visuel permettant ensuite de comparer efficacement différentes images.

Exercice 1 Extraction de descripteurs locaux : SIFT

On considère une image dans laquelle on va extraire des descripteurs locaux, c'est à dire qu'on calcule une caractéristique visuelle dans un ensemble de sous-régions. On considérera ici le cas simple où les régions sont des patchs carrés de taille 16×16 pixels, échantillonnées avec un pas contant de $\delta = 8$ pixels. Utiliser la fonction `r=denseSampling(I,s,delta)` pour effectuer l'échantillonnage, et visualiser les régions correspondantes avec `drawRectsImage(I,r,s)`.

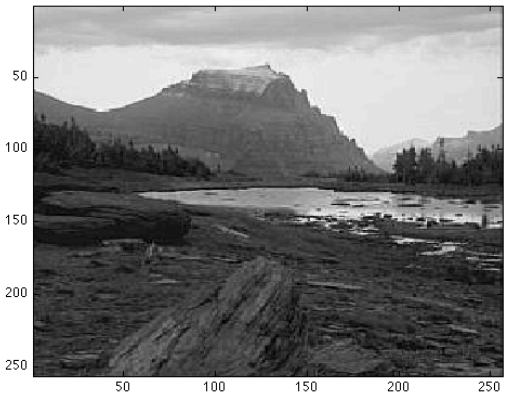
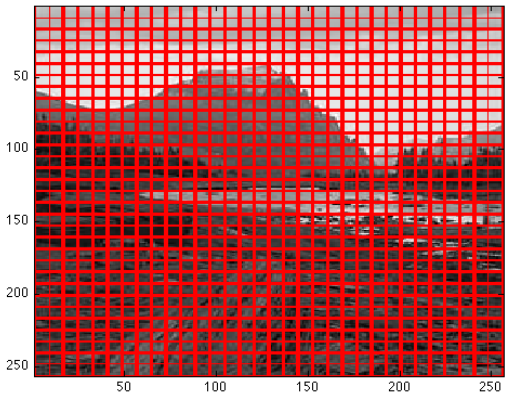


Image source



Échantillonage dense de régions

Pour chacune des régions, on va extraire un descripteur très utilisé en vision par ordinateur, le descripteur SIFT (**S**cale **I**nvariant **F**eature **T**ransform) [1]. Son principe consiste à calculer un histogramme d'orientation de gradients, comme illustré à la figure 1.

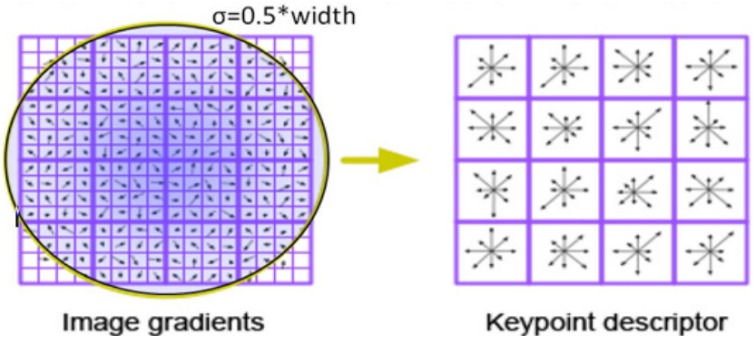


FIGURE 2 – Le descripteur SIFT

Plus précisemment, voici les grandes étapes de l'algorithme proposé pour calculer un descripteur SIFT (légèrement simplifié) pour un patch P :

— On décompose le voisinage de P en 16 blocs de 4×4 pixels (figure 1).
— Dans chaque bloc on forme un histogramme d'orientation de gradients, en discrétisant l'orientation en 8 bins.
— La formation de l'histogramme est effectué comme suit :

$$H\left[\Theta^i(x,y)\right] = H\left[\Theta^i(x,y)\right] + ||\overrightarrow{G}(x,y)|| \cdot M_\sigma(x,y) \quad (1)$$

où $\Theta^i(x,y)$ est l'orientation du gradient discrétisée au pixel (x,y). Ainsi, on incrémente le bin de l'histogramme correspondant à $\Theta^i(x,y)$ de la valeur $||\overrightarrow{G}(x,y)|| \cdot M_\sigma(x,y)$, où $||\overrightarrow{G}(x,y)||$ est le module du gradient au pixel (x,y) et $M_\sigma(x,y)$ la pondération obtenue au pixel (x,y) en appliquant un masque Gaussien centré sur le point d'intérêt et d'écart type $\sigma = 0.5 \cdot width = 8$ pixels (figure 1).

Enfin, un ensemble de post-traitements est appliqué pour produire le descripteur final :

— Normalisation ℓ_2 du descripteur pour avoir une norme euclidienne unité. Ceci a pour but de réduire l'influence des fortes valeurs de gradient afin d'être robuste aux variations non linéaires d'intensité, et peut être vu comme une normalisation du descripteur par rapport au contraste.
— Les valeurs > 0.2 sont seuillées (saturation à 0.2), puis le descripteur est re-normalisé.

1 Calcul du descripteurs SIFT pour une région (patch)

On demande ici :

1. De mettre au point une fonction `sift=computeSIFT(s,Ig,Ior,Mg)` qui calcule le descripteur SIFT pour un patch de taille s (ici $s = 16$), à partir de l'image du module du gradient dans le patch Ig (image de taille 16×16), de l'image de l'orientation du gradient discrétisée en 8 bins (image de taille 16×16), et du masque Gaussien Mg (image de taille 16×16). Le vecteur sift, de taille 128, sera obtenu en concaténant en ligne chacun des histogrammes calculés en chaque bloc du découpage 4×4.
2. Mettre en place un script `testExtractionSIFT` pour tester la fonction `computeSIFT` précédente sur un patch d'une image donnée. Ce script mettra en place les étpaes suivantes :
   (a) Chargement d'une image de test I
   (b) Calcul du vecteur gradient $\overrightarrow{G}(x,y) = \left(\frac{\partial I}{\partial x} \quad \frac{\partial I}{\partial y} \right)^T = \left(Ix \quad Iy \right)^T$ en chaque pixel de l'image I. En pratique, on approximera les dérivées partielles par des différences finies : dans ce cas les dérivées partielles I_x et I_y peuvent s'obtenir en calculant la convolution 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 : $M_x = \frac{1}{4} \cdot \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ et $M_y = \frac{1}{4} \cdot \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$.
   — Montrer que les masques M_x (resp. M_y) sont séparables, c'est à dire qu'ils peuvent s'écrire comme $M_x = h_y \times h_x$, où h_y est un masque 1d sur les lignes (taille 3×1) et h_x est un masque 1d sur les colonnes (taille 1×3).
   — Utiliser la fonction `If = convolution_separable(I,hx,hy)` pour calculer I_x et I_y.
   — En déduire le module du gradient $I_g = ||\overrightarrow{G}|| = \sqrt{Ix^2 + Iy^2}$.
   — Calcul de l'orientation du gradient en chaque pixel. Utiliser pour cela la fonction `Ior = orientation(Ix,Iy,Ig)` fournie, qui calcule une image où l'orientation du gradient (discrétisée à 8 bins) en chaque pixel est calculée.
   — Calcul du masque gaussien pour pondérer le calcul du SIFT : utiliser la fonction `Mg = gaussSIFT(s)` fournie.
   — Calcul le descripteur SIFT en un patch déterminé en appelant la fonction `computeSIFT` précédente.
   — Visualisation du calcul du descripteur en utilisant la fonction `visuSIFT( I, Ig,Ior, patch ,nom, s,sift )` fournie (patch contient les coordonnées du point haut gauche du patch).

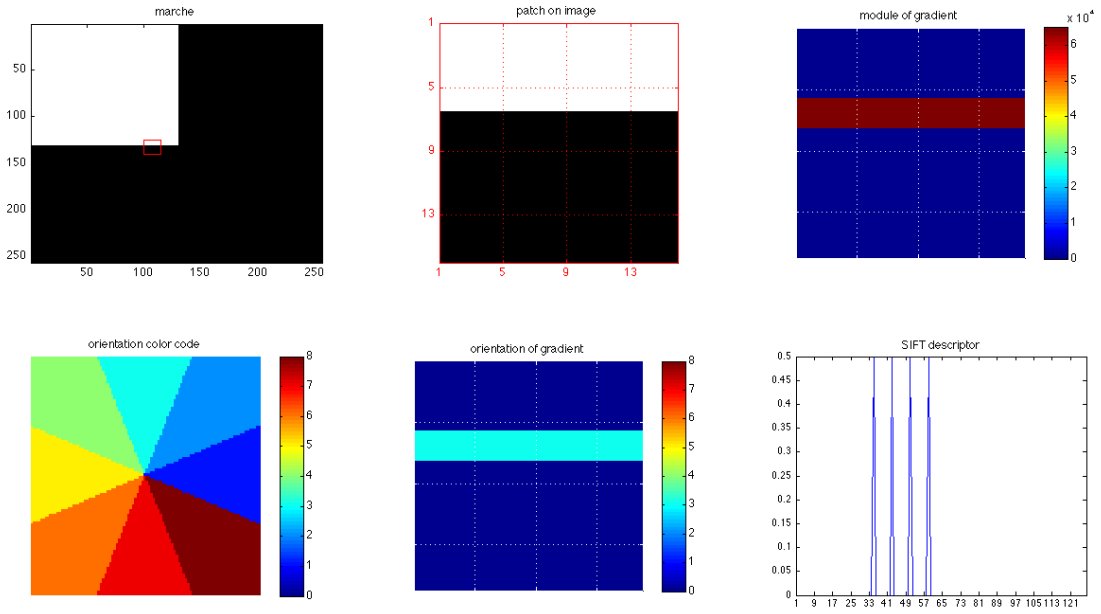


FIGURE 3 – Calcul et visualisation de descripteurs SIFT pour l'image jouet "Marche"

3. Tester le script `testExtractionSIFT` sur différentes images et visualiser son fonctionnement sur les images suivantes :

— Image jouet ”marche” (utiliser la fonction `marche()` fournie). Pour un patch situé en $[125, 100]$, vérifier le calcul du descripteur montré à la figure 3. Analyser et justifier la forme du descripteur obtenu.
  — Tester également pour les patchs de coordonnées $[97; 121]$ et $[121; 121]$, et valider la forme du descripteur.
— Tester également le calcul du descripteur SIFT pour l'image 'tools.gif' fournie, aux patchs de coordonnées $[80; 200]$ et $[173; 250]$. Analyser la forme du descripteur obtenu.
— Enfin, tester avec une image aléatoire de la base 15-scene. Utiliser pour cela la fonction `randomImage(dir)` fournie. Utiliser la fonction `denseSampling` pour échantillonner uniformément l'image et sélectionner (aléatoirement) un des patchs pour calculer et visualiser le descripteur. La figure 4 donne un exemple de visualisation. Présenter des résultats pour diverses images de la base.

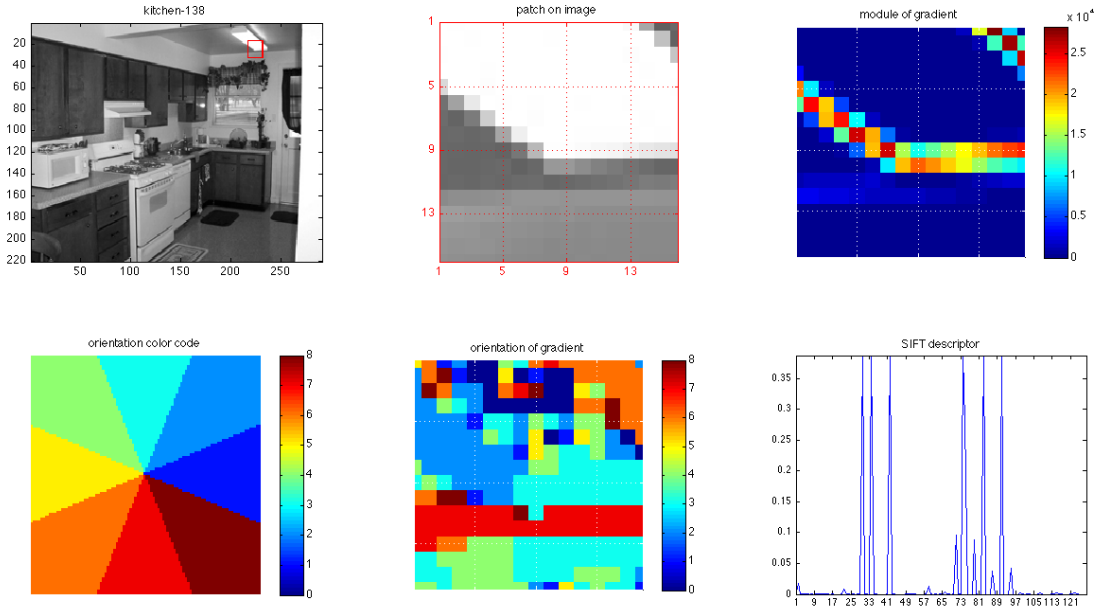


FIGURE 4 – Calcul et visualisation de descripteurs SIFT pour une image réelle

2 Calcul des descripteurs SIFT pour une image

Mettre en place une fonction `sifts = computeSIFTsImage(I)` qui va calculer l'ensemble des descripteurs SIFT d'une image I. Pour cela :
— Calculer le module I_g et l'orientation I_{or} du gradient en chaque pixel.
— Utiliser la fonction `denseSampling` pour extraire des patchs de l'image.
— Appeler la fonction `computeSIFT` pour calculer le descripteur SIFT en chacun de patchs.
Tester cette méthode dans un script `testExtractionSIFTsImage` qui appelle la méthode précédente pour une image donnée de la base 15-Scene.

Cas des régions homogènes

Dans une image où la région est totalement homogène (par exemple dans une région blanche ou noire de la figure 3), quelle va être la forme du descripteur SIFT ? Quelle va être sa norme ? Dans une région d'une image réelle ”grossièrement” homogène (par exemple dans une région blanche de la figure 4), quelle va être la forme du descripteur SIFT ? Quelle va être sa norme ? Pourquoi cela est-il problématique (penser à la normalisation ℓ_2 du descripteur) ?

Pour contrer ce problème, on propose d'avoir un critère pour estimer le contraste d'un patch, et de ne calculer le descripteur que pour les patchs où le contraste est ”suffisant”. On choisiera comme critère la norme du descripteur SIFT : si elle est inférieure à un seuil donné (avant normalisation), le descripteur SIFT est fixé au vecteur nul. Modifier votre fonction `computeSIFT` afin de mettre en place ce traitement.

Afin de trouver un bon seuil de binarisation, utiliser la fonction
`drawPatches(I,s,r,sifts)` qui affiche les patchs correspondant à des sifts non nuls ont été calculés en rouge, les autres en bleu. La figure 5 présente un exemple de résultat où les régions "grossièrement" homogènes ont été mise à 0, avec un seuil variable. Présenter des résultats de différentes images de la base avec votre seuil de binarisation.

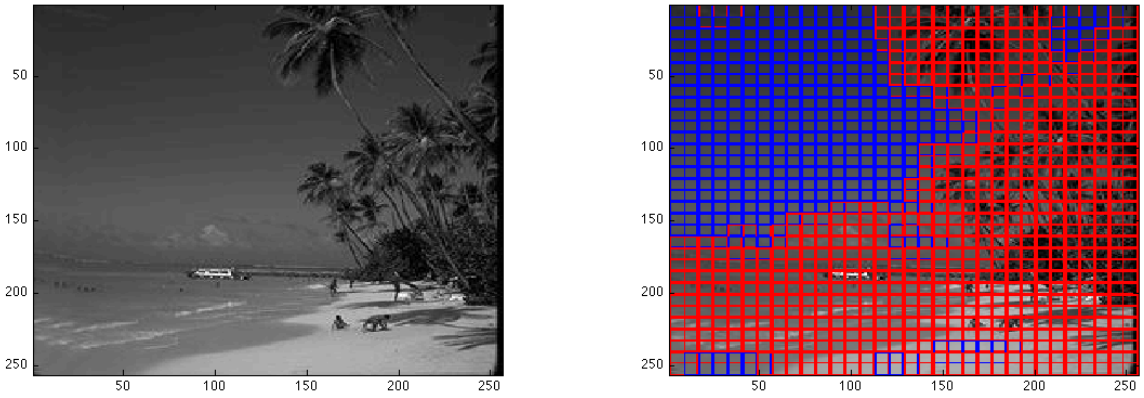


FIGURE 5 – Seuil de contraste pour ignorer le calcul du SIFT dans les régions homogènes

3 Calcul de l'ensemble des descripteurs de la base

Adapter le script `computeSIFTsBase` (fourni) pour calculer l'ensemble des descripteurs SIFT de la base. Le script va effectuer les opérations suivantes : parcourir l'ensemble des images de la base, calculer les SIFTs avec la fonction `computeSIFTsImage` précédente, stocker les descripteurs SIFT de chaque image dans un fichier (.mat) de même nom que l'image.

N.B. : pour limiter la taille de stockage, chaque descripteur sift est stocké sur un octet (uint8). Pour cela, on multiplie le descripteur (normé) par 255 avant de le stocker.

Comparaison par rapport au descripteur de Lowe [1]

Quelle est la différence principale dans le calcul du descripteur proposé ici par rapport à celui décrit dans l'article de D. Lowe [1] (voir section 5 du papier) ? Pour des tâches de classification d'image, cette différence introduite dans le descripteur calculé ici permet en fait d'augmenter les performances. Proposer une explication de ce phénomène.

Exercice 2 Génération de dictionnaire visuel

Quantification et K-Means

A partir du vecteur SIFT (dimension $d = 128$) extrait en chaque patch de l'image, on veut calculer un dictionnaire visuel. L'objectif est de pouvoir déterminer un ensemble de K centres (clusters) de l'espace minimisant la distorsion des données : on est donc ramené à un problème de quantification optimale.

Plus formellement, on considère :

— Un ensemble de N pixels décrits par leurs coordonnées $x_i \in \mathbf{R}^{128}$ dans l'expace des SIFT : $\{x_i\}$, $i \in \{1; N\}$

— Un ensemble de M clusters de l'expaces des SIFT $\{c_m\}$, $c_m \in \mathbf{R}^{128}$ et $m \in \{1; M\}$

Le problème se formule donc comme la détermination des K clusters $\widehat{c_m}$ optimaux minimisant la fonction $f(c_m)$ suivante :

$$\{\widehat{c_m}\} = \underset{\{c_m\}\in\mathbf{R}^d}{\arg\min}\, f(c_m) = \underset{\{c_m\}\in\mathbf{R}^d}{\arg\min} \sum_{m=1}^{M} \sum_{x_i \in c_m} ||x_i - c_m||_2^2 \tag{2}$$

La résolution exacte de l'équation 2, même avec M fixé, est de complexité prohibitive, $\mathcal{O}(N^{Md+1}log(N))$. On a donc couramment a des heuristiques pour résoudre le problème de manière approximée. l'algorithme du K-Means [2] est

l'une d'entre elles. L'algorithme 2 détaille son fonctionnement. À partir d'une initialisation aléatoire des centres parmi les points des données, l'algorithme alterne entre phase d'assignement, où les points x_i sont associés au cluster le plus proche, et une étape de mise à jour, où les centres des clusters sont ré-estimés.

Algorithme 1 Algorithme du K-Means

Entrées : $\{\mathbf{x_i}\} \in \mathbf{R}^d$, $i \in \{1; N\}$ (N points de départ) et $\mathbf{M}$ (nombre de centres voulu).
Sortie : $\{\widehat{\mathbf{c_m}}\} \in \mathbf{R}^d$: coordonnées des M centres optimaux dans $\mathbf{R}^d$.

1: $\{c_m^0\} \leftarrow \{x_i\}, (m \times i) \in \{1; M\} \times \{1; N\}$
// Initialisation aléatoire des M centres parmi les N points.
2: **répéter**
3: Assignement : $c_m^t = \{x_i \ : \ ||x_i - c_m||_2 < |x_i - c_{m'}||_2, m' \neq m\}$
// Détermination des points x_i plus proches du centre c_m que des autres centres $c_{m'}$.
4: Mise à jour : $c_m^t = \frac{1}{|c_m|} \sum_{x_i \in c_m} x_i$
// Le centre du cluster c_m est ré-estimé comme le barycentre des points $x_i \in c_m$.
5: **tant que** $\sum_{m=1}^{M} ||c_m^t - c_m^{t-1}||_2 > 0$ // Les centres calculés évoluent
6: **retourner** $\{c_m\} = \{\widehat{c_m}\}$

Étant une heuristique, l'algorithme du K-Means ne garantit pas d'aboutir à la solution optimale de l'équation 2. Il donne cependant en pratique des solutions satisfaisantes. Une stratégie couramment utilisée pour surmonter l'initialisation (aléatoire) des centres consiste à lancer plusieurs fois l'algorithme (et donc d'effectuer plusieurs initialisations différentes), et à garder le clustering (centres $\{\widehat{c_m}\}$) minimisant la fonction de distorsion $f(\widehat{c_m})$ de l'équation 2.

Question théorique

Si on suppose l'assignement effectué et qu'on se place au niveau d'un cluster, montrer que le barycentre des points appartenant au cluster minimise la distorsion des données :

$$\widehat{c_m} = \mathbb{E}(c_m) = \frac{1}{|c_m|} \sum_{x_i \in c_m} x_i = \underset{\{c_m\} \in \mathbf{R}^d}{\arg\min} \sum_{x_i \in c_m} ||x_i - c_m||_2^2 \tag{3}$$

Complexité algorithmique

Dans l'algorithme du K-Means, la fonction la plus coûteuse d'un point de vue temps de calcul est de loin la fonction d'assignement. Pour chaque point extrait de la base, il faut déterminer le cluster le plus proche. Si on utilise un programmation basée boucle, le temps de calcul devient prohibitif. Il faut donc passer par une programmation vectorielle, beaucoup plus efficace en matlab.

On demande de réléchir à la mise au point de la fonction d'assignement suivante :
`nc = assignementKMeans(listPts,centres)`, qui, à partir d'une liste de points (taille $N \times d$) et des centres (taille $M \times d$), renvoie la liste des clusters les plus proches pour chaque point de la base (taille $1 \times N$).

Proposer une solution pour que cette fonction n'utilise pas de boucles mais uniquement des opérations sur les vecteurs.

Indication : l'objectif va être de calculer la matrice de distance D de taille $M \times N$ entre l'ensemble des couples descripteurs/centres. Pour chaque colonne de de D, on calcule l'indice de valeur mininal, qui correspond au centre le plus proche du descripteur. La question est de savoir comment calculer D en ne faisant que des opérations vectoriellles (penser à la décomposition $||x_i - c_m||_2^2 =< x_i - c_m; x_i - c_m >$=?). Donner le code matlab de la fonction. **N.B.** : La fonction `assignementKMeans` pourra prendre en paramètre d'autres arguments nécessaires à la minimisation de la distance précédemment étudiée.

Travail demandé

On demande ici de mettre au point les fonctionnalités pour apprendre un dictionnaire visuel par un algorithme du K-Means, dans l'espace des descripteurs SIFT.

1. Mettre en place une fonction `[centres,erreur] = solutionKMeans(points,M)`, qui à partir d'une liste de points (matrice de taille $N \times d$) et du nombre M souhaité de centres, renvoie les coordonnées des M centres en utilisant l'algorithme du K-Means ainsi que l'erreur de quantification obtenue.
   — Cette fonction utilisera la fonction `centres = randomSeed(points,K)` fournie qui sélectionne aléatoirement K points parmi la liste de points initiaux.

— Tant que le centres ne seront pas stables, cette fonction appellera :
  — la fontion `nc = assignementKMeans`, dont la programmation vectorielle efficace aura au préalable été définie. Cette fonction prend en entrée une liste de points (taille $N \times d$) et des centres (taille $M \times d$), et renvoie la liste des indices des clusters les plus proches pour chaque descripteur. **N.B.** : La fonction `assignementKMeans` pourra prendre en paramètre d'autres arguments nécessaires à la minimisation de la distance précédemment étudiée.
  — une fontion `[newcenters , erreur , movecenters ] = miseAjourKMeans( listPts , centers , nc)`, qui permet de mettre à jour les centres et de renvoyer l'erreur de quantification ainsi que le déplacement des centres.

2. Écrire un script `testKMeans` pour créer le dictionnaire visuel de la base d'image. **On choisira d'apprendre un dictionnaire avec $\mathbf{M = 1000}$ mots visuels**. Le script `testKMeans` doit comporter les étapes suivantes :
   — Sélection aléatoire d'un certain nombre ($\sim$ 50000) de descripteurs SIFT des images de la base : fonction `[points,norms] = randomSampling(dir)` fournie. **N.B.** : cette fonction va uniquement échantillonner des descripteurs de norme non nuls (*i.e.* ceux correspondant aux régions non homogènes).
   — Appel de la fonction `solutionKMeans` précédentes. Optionnellement, on pourra calculer plusieurs (T) solutions du K-Means (correspondant à différentes initialisations aléatoires des centres), et sélectionner la solution minimisant l'erreur de distorsion des données.
   — Sauvegarde du dictionnaire visuel. **N.B.** : On ajoutera aux clusters déterminés par l'algorithme du K-Means le mot correspondant à une région homogène, *i.e.* le vecteur nul.

Références

[1] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60 :91–110, 2004.

[2] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.