Calcul de représentations image et apprentissage supervisé

Travaux sur Machines Encadrés

7-21 octobre 2015

Exercice 1 Calcul de représentation image : le modèle Bag of Words (BoW)

À partir des descripteurs locaux et du dictionnaire visuel calculés au premier TME, on va calculer une signature vectorielle pour représenter chaque image de la base.

On va utiliser le modèle sac de mots (Bag of Words, BoW). On considère un ensemble de N descripteurs locaux (SIFT) $\mathbf{X} = \{x_i\}_{i\in\{1;N\}}$, $x_i \in \mathbf{R}^{128}$, calculés dans une image I, couramment appelé BoF ("Bag of Features"). Soit $\mathbf{C} = \{c_m\}_{m\in\{1;M\}}$, $c_m \in \mathbf{R}^{128}$, un ensemble de M mots visuels calculés sur l'ensemble de la base par une étape de clustering. La formation du BoW à partir du BoF peut être décomposée en deux étapes, comme illustré à la figure 1 :

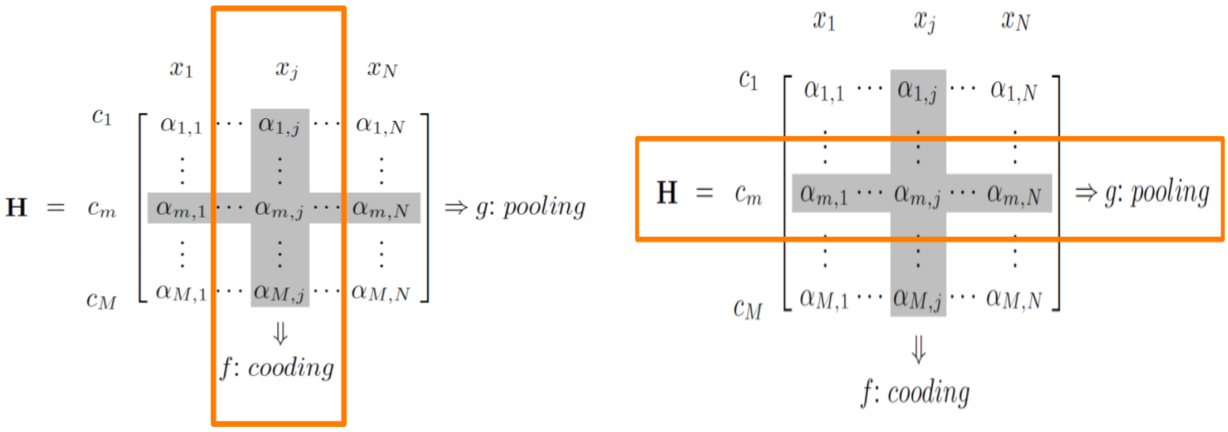


FIGURE 1 – Le modèle BoW : codage (coding) et agrégation (pooling)

— Une étape de codage, où chaque descripteur local x_i est projeté sur le dictionnaire visuel $\mathbf{C}$. Cette étape permet de comparer les descripteurs SIFT dans un référentiel commun.
— Une étape d'agrégation des projections, où une statistique des codes des différents descripteurs locaux est calculée, pour chaque mot visuel c_m. Cette étape à pour but de gagner en invariance : pour un mot visuel, les codes correspondant aux N descripteurs de l'image sont résumés par une grandeur scalaire.

La méthode historique du BoW consiste à

— Effectuer un codage au plus proche voisin : $f = f_Q$

$$f_Q(x_j) = \begin{cases} 1 & \text{if } m = \underset{k\in\{1;M\}}{\operatorname{argmin}} \|x_j - c_k\|^2 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

— g Calculer un pooling somme pour la fonction g :

$$z_m = \sum_{j=1}^{N} \alpha_{m,j} \tag{2}$$

Le BoW représente alors un histogramme d'occurence des différents mots visuels du dictionnaire dans une image donnée.

Préalable : visualisation du dictionnaire visuel

Pour visualiser le dictionnaire visuel, vous pouvez utiliser la fonction `[ patchmin ] = visuDico( nomDico , baseDir , baseDirDes)` fournie. Cette méthode va charger le dictionnaire précédemment appris par K-Means à partir de son nom, chercher le patch le plus proche de chaque mot visuel dans la base, visualiser l'image résultante à partir de la fonction drawDico(patchmin), et renvoie l'image résultante patchmin (image de taille $16 \times 16 \times M$). La figure 2 montre un exemple de dictionnaire appris sur la base 15-Scene. Commenter la forme de votre dictionnaire. Stocker l'image patchmin obtenue : elle sera utilisée ensuite pour la visualisation du BoW.

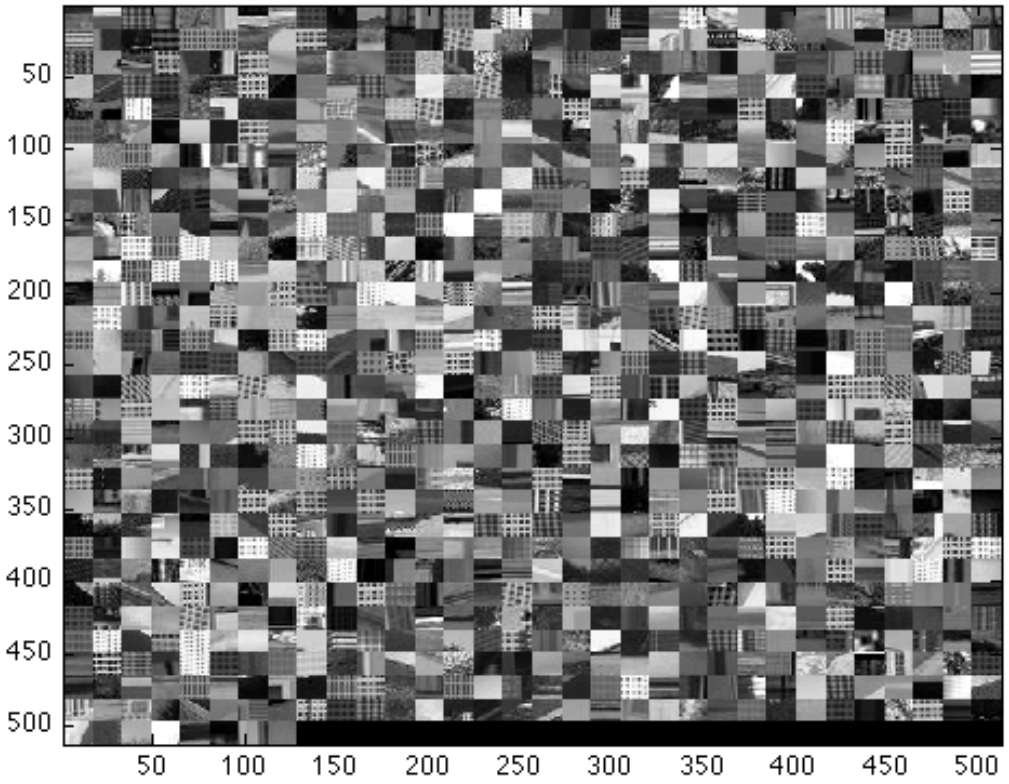


FIGURE 2 – Exemple de visualisation de dictionnaire visuel

N.B. : dans la fonction `visuDico`, la recherche n'est effectuée que pour une image sur 10 de chaque catégorie (variable pas). Si vous voulez effectuer une recherche exacte (mais plus lente), fixer $pas = 1$.

Calcul de BoW sur des images et visualisation des résultats

On demande ici :

1. D'écrire une méthde `[bow,nc] = computeBow(sifts, clusters, matNormClusters)`, qui va calculer un BoW à partir d'un ensemble de descripteurs SIFT et d'un dictionnaire visuel. On calculera un BoW simple (hard coding sum pooling). On passera la matrice de taille $M \times N$ contenant la norme (au carré) des différents centres, qui est constante puisque les centres sont fixés. la fonction va effectuer les opérations suivantes :
   — Calcul d'un vecteur `nc` contenant, pour chaque descripteur local, le cluster le plus proche du dictionnaire. Utiliser pour cela la fonction `assignementKMeans` mise en place au TME 1 (en prenant en paramètre la norme des centres qui est ici constante).
   N.B. : `nc` sera renvoyé par la fonction uniquement car il sera utilisé pour la visualisation du BoW (si on désire simplement calculer le BoW, le retour de `nc` peut être ignoré).
   — `nc` correspond à l'étape de codage (hard assignement). Afin d'agréger les codes, on effectuera ici un somme "pooling" afin de produire l'hitogramme de mots visuels

— Finalement, une normalisation ℓ_1 du vecteur sera effectuée : dans notre cas, ceci renviendra simplement à diviser chaque composante du BoW par le nombre de descripteurs dans l'image (on obtient ainsi un histrograme exprimé en pourcentage, invariant par rapport au nombre de descripteurs dans l'image).

2. De tester cette méthode dans un script `testBowImage` qui va
   — Ouvrir une image aléatoire de la base. Utiliser pour cela la fonction `[ I , nomim , sifts] = randomImageDes( baseDir , baseDirDes)` fournie : à partir du dossier des images et des descripteurs, cette fonction renvoie le nom d'une image aléatoire de la base, l'image ouverte `I`, et les descruipteurs SIFT correspondants.
   — Appeler la méthode `[bow,nc] = computeBow` précédente pour calculer le BoW de l'image.
   — Utiliser la méthode `visuBoW( I , patchmin , bow , nc, nomim)` fournie pour visualiser et interpréter le calcul du BoW. `patchmin` correspond à l'image des patchs du dictionnaire visuel. Un exemple est montré à la figure 3. Cette méthode présente :
   (a) L'image originale
   (b) Le calcul du BoW
   (c) La visualisation de la localisation spatiale des 8 mots du dictionnaire les plus fréquents dans l'image
   (d) L'apparence des 8 mots du dictionnaire les plus fréquents dans l'image.

Tester votre script `testBowImage` avec différentes images de la base. Analyser les résultats.

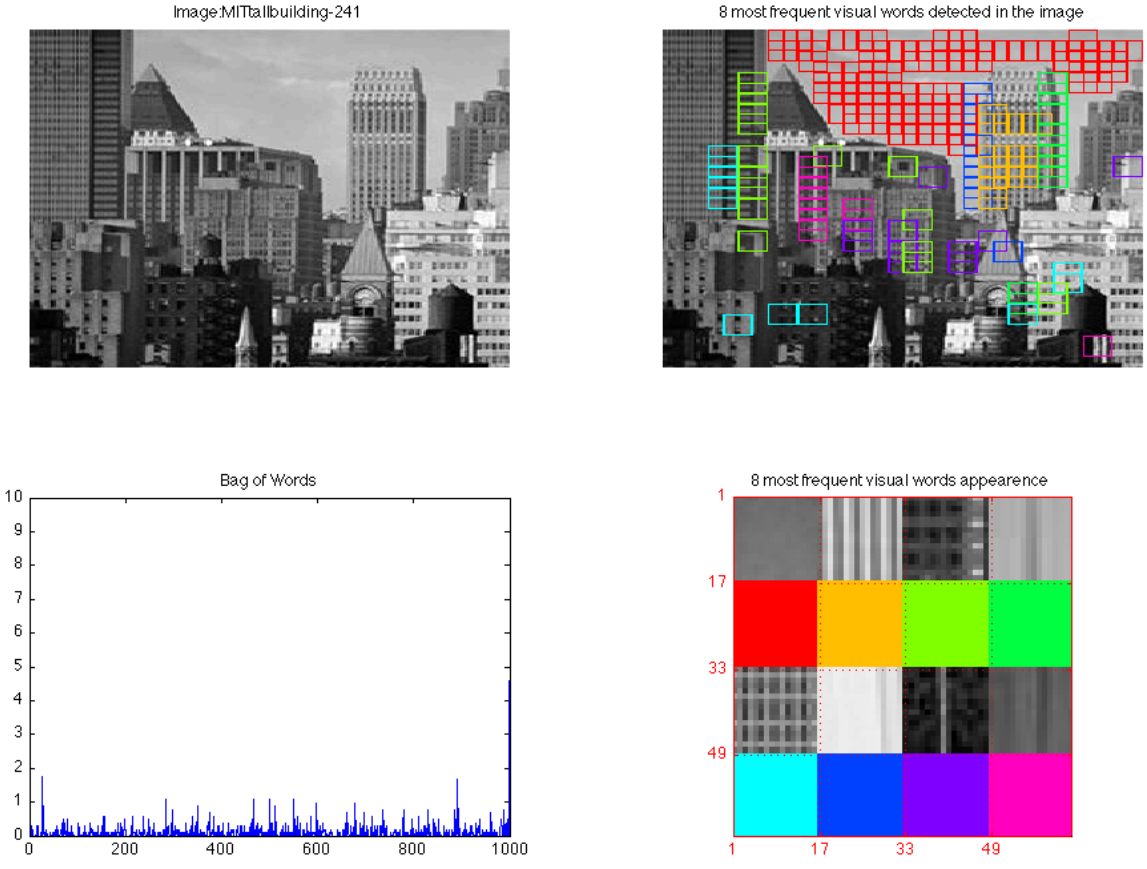


FIGURE 3 – Exemple de calcul et de visualisation du BoW

Calcul de l'ensemble des BoW sur la base

On demande ici de mettre en place un script `computeBowBase` qui va calculer le BoW (vecteur de taille $M = 1000$) pour chacune des 4485 images de la base. Stocker chaque BoW dans un fichier.

Exercice 2 Apprentissage supervisé pour la classification sémantique d'images

À partir des représentions BoW calculées pour chaque image de la base, on veut mettre en place un apprentissage supervisé dans l'espace des BoW ($\mathbb{R}^M$, $M = 1000$). L'objectif consiste ensuite à proposer un système capable de prédire la classe (parmi les 15 possibles) pour une image quelconque de la base qui n'aura pas été apprise.

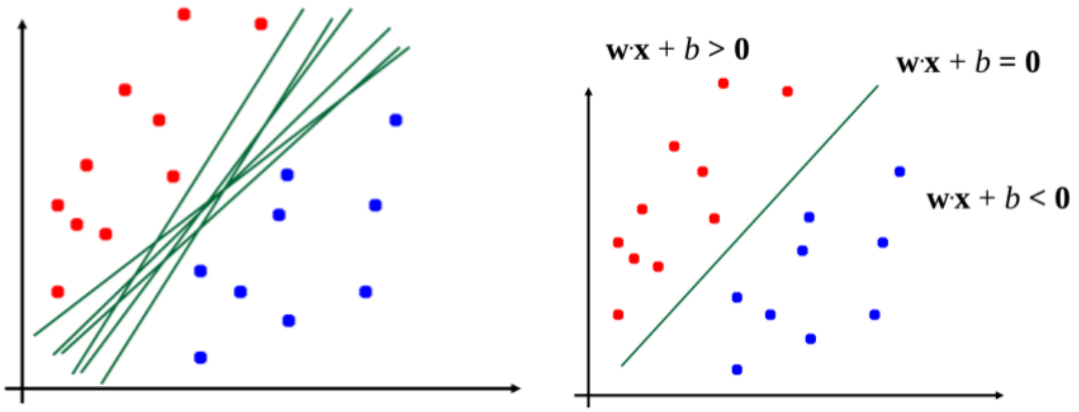
Pour l'apprentissage, on va utiliser 15 classifieurs binaires qui vont apprendre à classer correctement un ensemble annoté d'exemples $\oplus$ (les images de la catégorie recherchée, par exemple MITforest) et $\ominus$ (toutes les autres images).

Apprentissage de SVM binaires

On va entraîner un classifieur binaire par catégorie. Formellement, on considère un ensemble d'exemples d'apprentissage $z_i \in \mathbb{R}^M$, avec leurs labels associés $y_i = \pm 1$. Un classifieur SVM est une machine d'apprentissage linéaire, c'est à dire qu'on cherche un hyperplan dans $\mathbb{R}^M$ permettant de séparer correctement les données d'apprentissage. Un hyperplan peur etre représenté par le couple (w, b), où w est le vecteur normal à l'hyperplan et b le bias (distance de l'hyperplan par rapport à l'origine). Pour un exemple z à classer, l'évaluation de la fonction de classification s'écrit :

$$f(z) =< w; z > +b \tag{3}$$

Pour un problème linéairement séparable, il existe souvent de nombreux hyperplans séparateurs, comme illustré à la figure 4a). Afin de régulariser le problème, le classifieur SVM cherche l'hyperplan de "marge" maximale, c'est à dire celui qui maximise la distance des points les plus proches de la frontière. Un tel hyperplan est représenté à la figure 4b).



a) hyperplans séparateurs b) hyperplan SVM

FIGURE 4 – Apprentissage d'hyperplan séparateurs : Machines à vaste marges (SVM)

La formulation du problème d'optimisation SVM s'écrit donc :
$min\frac{1}{2}||w||^2$ sous les contraintes : $y_i \cdot (< w; z_i > +b) \geq 1$. Dans le cas non linéairement séparable, on introduit la possibilité de mal classer les données, avec une tolérance ξ_i par exemple z_i. Le problème devient :
$min\frac{1}{2}||w||^2$ sous les contraintes : $y_i \cdot (< w; z > +b) \geq 1 - \xi_i$. Ce problème d'optimisation sous contrainte est équivalement à minimiser la fonction $g(w, b)$ suivante :

$$g(w,b) = \frac{1}{2}||w||^2 + C \cdot \sum_{i=1}^{N} max[0, 1 - yi \cdot f(z_i)] = \frac{1}{2}||w||^2 + C \cdot \sum_{i=1}^{N} L[f(z_i), yi] \tag{4}$$

où $L(x)$ est la fonction "charnière" (hinge). Il existe de nombreux algorithmes (dans l'espace primal des paramètres ou dans l'espace dual des exemples) pour résoudre le problème. On utilisera la librairie LibSVM [2] pour résoudre le problème d'optimisation SVM.

Formation d'un classifieur multi-classe

À partir d'un classifieur SVM binaire appris par catégorie sémantique, on va produire un résultat de classification multi-classe. Pour cela, on va appliquer le classifieur binaire appris pour chaque catégorie sur chaque exemple de test

z_i, en utilisant l'équation 3. La classe dont le score $f(z_i)$ sera maximal définiera la classe prédite par le système. Cette stratégie pour produire un résultat de classification multi-classe à partir de l'apprentissage de classifieurs binaires est connue sous le nom de "un contre tous" (one against all).

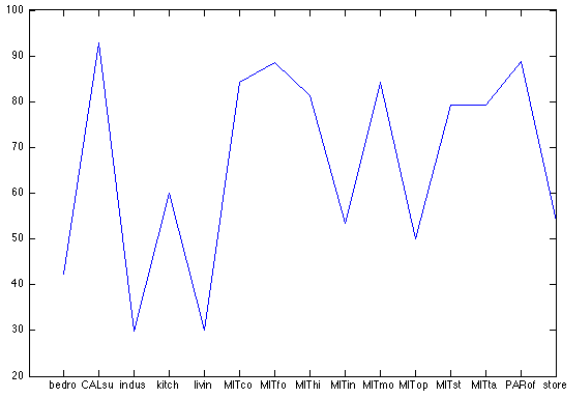
Travail Demandé

On demande de mettre en place un script `eval_15Scene` qui va effectuer l'apprentissage et le test du système sur la base 15-Scene, à partir des signatures vectorielles de type BoW précédemment extraites pour chaque image de la base. Le protocole d'évaluation classiquement utilisé dans la communauté pour évaluer la performances d'une méthode est le suivant :

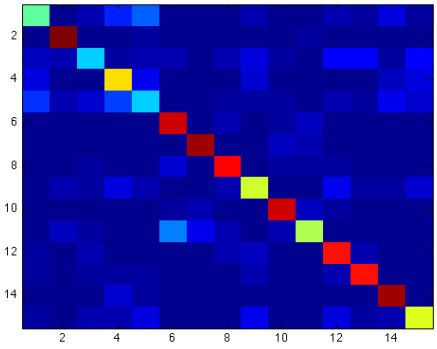
- Entrainement de chaque classifieur binaire sur 100 exemples par classe, tirés aléatoirement
- Test des performances sur les image restantes : pour chaque classe, on produit un résultat de classification multi-classe pour chaque exemple, et on calcule le taux de bonne classification
- La performance finale est la moyenne des taux de classification sur les 15 classes
- On renouvelle toutes les étapes précédentes T fois afin de mesurer la variablilité des performances en fonction de l'échantillonnage des données pour apprendre et tester. On calcule ainsi moyenne et écart-type des performances sur les T découpages (splits) des données en train/test, ce qui permet de mesurer le niveau de significance statistique entre les performances des différentes méthodes

Le script `eval_15Scene` devra comprendre les étapes suivantes, pour chacun des T splits :

1. Génération des données d'apprentissage et de test. Utiliser pour cela la la fonction `[ train , test ] = loadData( nTrain , imCat , pathBow, K)` fournie. La fonction retourne des matrices de taille 1500×1000 pour train, 2985×1000 pour test. `pathBow` est le dossier où les BoW ont été stockés, `K` la taille de chaque BoW (ici 1000), `nTrain` est le nombre d'exemples utilisés pour l'apprentissage de chaque catégorie (ici 100), `imCat` est le nombre total d'images par catégorie (utiliser la fonction `[ imCat,imCatTest ] = NbImCatAllTest( pathBow , nTrain)` fournie pour les calculer).
2. Apprentissage d'un classifieur binaire pour chacune des catégorie :
   - Utiliser la fonction `[ y, ytest ] = labelsTrainTest( nTrain,ntest,imCat,i )` pour générer un vecteur de labels (± 1), en apprentissage et en test, pour la $i^{ème}$ catégorie
   - Mettre en place une fonction `[ values ] = trainTest( train, test, y)` qui va apprendre apprendre un classifieur SVM binaire pour la $i^{ème}$ catégorie et va appliquer le classifeur appris sur les exemples de test.
     - Pour apprendre le classifieur, on utilisera la librairies LibSVM fournie. **N.B.** : il faudra compiler les mex-files qui seront ici utilisés, tapper make dans le dossier matlab.
     - La fonction pour apprendre le SVM est `model = svmtrain(y, train, '-c 1000 -t linear')` qui va apprendre un SVM à partir des données et labels d'apprentissage. l'option '-t linear' spécifie l'utilisation d'un classifieur (noyau) linéaire, '-c 1000' précise le compromis entre régularisation et classement des exemples d'apprentissage (plus C est grand, plus les données d'apprentissage sont forcées à être correctement étiquetées)
     - Utiliser ensuite la fonction `[w,b] = getPrimalSVMParameters(model)` qui renvoie les paramètres de l'hyperplan appris en fonction de la structure de données interne (`model`) de LibSVM.
     - Enfin, utiliser les paramètres de l'hyperplan (w, b) appris pour attribuer un score de classification pour chaque exemple de test, en utilisant l'équation 3.
3. Une fois l'ensemble des SVM binaires appris pour les 15 catégories, mettre en place une fonction `[ matConf , txCat ] = multiClassPrediction( predictclassifieurs , imCatTest)` qui va calculer le taux de reconnaissance dans chacune des 15 catégories `txCat` et la matrice de confusion `matConf`, c'est à dire la répartition pour chaque catégorie des catégories prédites par le système. Cette fonction prend en entrée la matrice `predictclassifieurs` de taille 15×2985 qui contient l'évaluation de chaque exemple de test par chaque classifieur, ainsi que `imCatTest` qui précise le nombre d'images en test pour chaque catégorie (calculé par la fonction `NbImCatAllTest` fournie). La figure 5 montre un exemple de taux de reconnaissance par classe et de matrice de confusion obtenue avec le setup actuel : extraction de SIFT dense (16×16, gap de 8), dictionnaire visuel appris par K-Means avec 1000 mots, BoW calculé avec hard assignement et sum pooling, classifieur linéaire.
4. Le score pour un split des données correspond à la moyenne des taux de reconnaissance `txCat` sur les 15 catégories. Pour donner un score sur les T splits on calculera la moyenne et l'écart-type de ces T valeurs.



Taux de reconnaissance par catégorie



Matrice de confusion

FIGURE 5 – Exemple de performances obtenue sur la base 15-Scene avec la représentation BoW

Carte de chaleur de classification

On propose d'analyser de l'impact de chaque descripteur local dans le processus de décision de l'image. L'objectif est de mettre en place une fonction permettant de calculer une "carte de chaleur de classification" :

1. Montrer que la contribution de chaque descripteur local x_j dans la fonction de classification s'écrit $p(x_j) = \frac{1}{N} w_{m^*}$, où N est le nombre de descripteurs locaux dans l'image et $m^* = \underset{k \in \{1;M\}}{\arg\min} \|x_j - c_k\|^2$
2. Mettre en place les fonctionnalités pour calculer la carte de chaleur de classification :
   - Écrire une fonction `hmap=compute_prediction_heatmap(I, nc, w)`, qui calcule une carte de chaleur de classification pour une image. On aura en entrée une image `I`, un classifieur binaire avec son vecteur de poids `w` associé (vecteur de taille K), et l'assignement `nc` chacun des N descripteurs script au mot visuel le plus proche.
   - Dans un script `test_heatmap_prediction`, apprendre un ensemble de classifieurs sur un split donné, et stocker les modèles de prédiction résultants (utiliser la fonction `loadData2`)
   - Dans le script `test_heatmap_prediction` :
     - Choisir une image de test, charger le classifier binaire correspondant à la classe de l'image, charger le fichier de descripteurs SIFTs correspondants ainsi que le dictionnaire visuel, et calculer l'assignement `nc` des SIFTs au mot visuel le plus proche.
     - Appeler la fonction `compute_prediction_heatmap` précédente pour calculer
     - Utiliser la fonction `visuHeatMap` pour visualiser la carte de saillance de prédiction.
     - Analyser les résultats de classification : type de régions servant à classifier, analyse des bonnes *vs* mauvaises prédictions. La figure 6 donne un exemple de résultat sur la classe MITmoutain :
       - Interpréter les régions contribuant positivement à classer l'image dans la bonne classe
       - Interpréter le score de régions homgènes

Bonus : analyse de resultats et limitations de la représentation utilisée

- Comparer les performances que vous obtenez à ceux reportés dans les articles récents [3, 1]. Quelles sont les différences entre la chaîne de traitement mise en place en TME et celle que les auteurs utilisent ? Analyser notamment les points suivants :
  1. Prise en compte de l'information spatiale dans la représentation BoW
  2. Différences dans les méthodes de coding/pooling pour la formation des BoW
  3. Normalisation des BoW (ℓ_1 *vs* ℓ_2)
- Améliorer la représentation BoW générée pour coller à la chaîne décrite en [3, 1]. Évaluer à nouveau les performances et commenter l'évolution des performances.

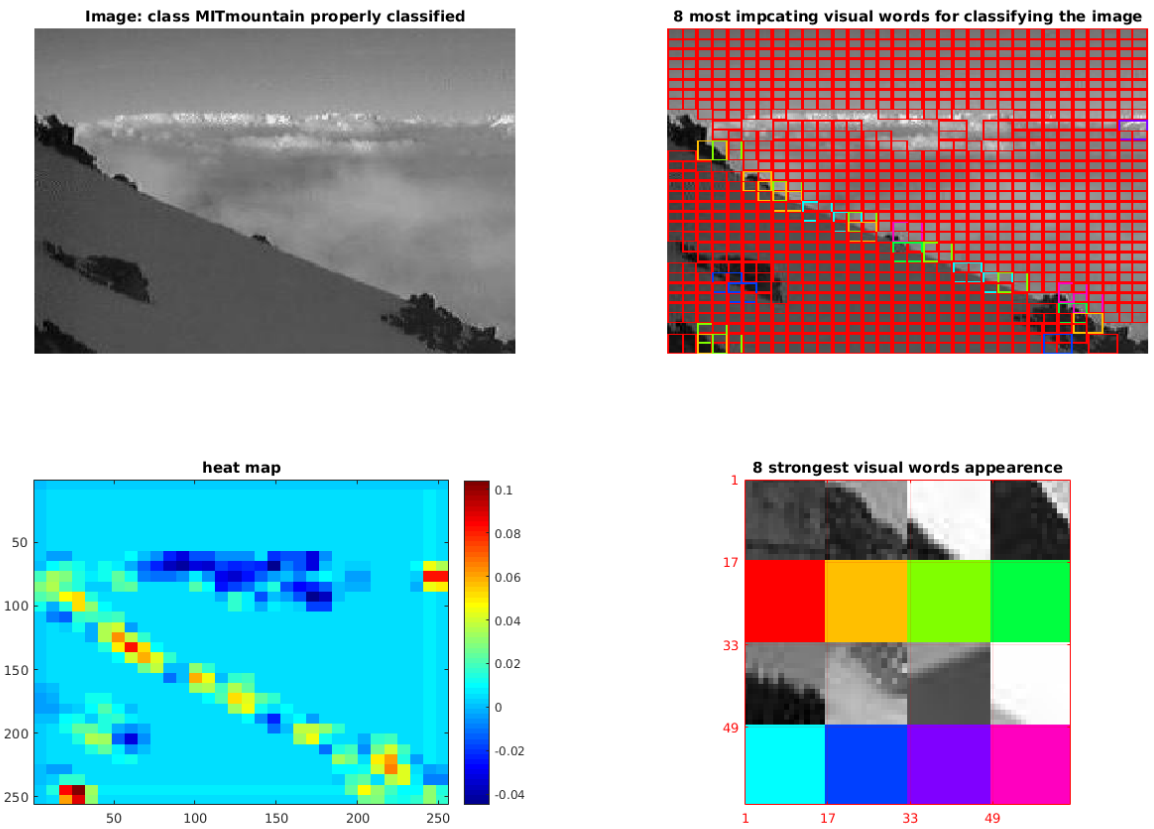


FIGURE 6 – Analyse de classification d'image : carte de chaleur localisée

Références

[1] Y. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning mid-level features for recognition. In *CVPR*, pages 2559–2566, 2010.

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM : A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2 :27 :1–27 :27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, pages 1794–1801, 2009.