Université de la Rochelle
Master 1 ICONE - Codage de l'information

TP - JPEG2000 et Codage Scalable

© Renaud Péteri

`Année universitaire 2019-2020`

`Vous ferez un rapport sur ce TP à déposer sous Moodle: vos réponses aux questions seront détaillées et vos codes Python commentées. Vous pourrez utiliser Pweave pour générer votre rapport (en pdf ou html)`

1 Préliminaires

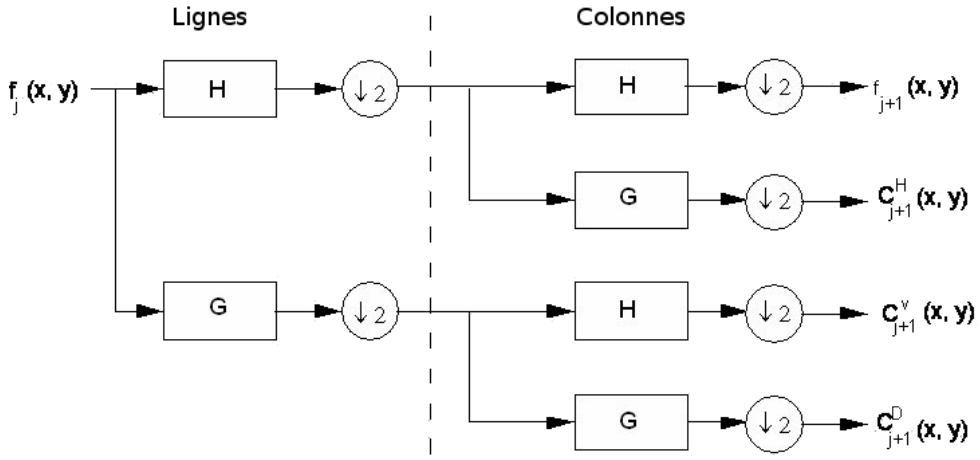
Les packages `scipy`, `numpy`, `matplotlib` et `scikit-image` sont nécessaires pour ce TP.
Au besoin, installez-les localement via : `pip3 install --user scikit-image`. Tester votre installation en lançant python (python3 du système) et en essayant d'importer `scikit-image`.

2 Une (brève) rencontre avec la transformée en ondelettes

Nous allons dans cet exercice manipuler la transformée en ondelettes, à la base du format JPEG 2000. La connaissance approfondie des ondelettes nécessite des détails théoriques en mathématiques et traitement du signal, mais nous en aurons ici une approche empirique, suffisante pour comprendre son principe.

Nous allons ici décomposer une image par ondelettes, puis la reconstruire à partir de ses coefficients d'ondelettes seuillés et voir l'influence dans la qualité visuelle de l'image reconstruite.
Dans une deuxième partie du TP, nous simulerons une transmission progressive de cette image sur un réseau. On rappelle le schéma d'analyse multirésolution d'une image, qui s'effectue par une suite de traitements numériques (banc de filtres). Ici, il s'agit d'une décomposition pour une seule échelle.

Pour une étape de multirésolution j, nous obtenons donc 4 images de dimensions divisées par 2 par rapport à l'image originale f_j : une image d'ondelettes C_{j+1}^H donnant les



Bancs de filtres 2D (analyse).

détails horizontaux, une image d'ondelettes donnant les détails verticaux C^V_{j+1}, une image d'ondelettes donnant les détails diagonaux C^D_{j+1}et une image d'approximation f_{j+1} (qui est l'image originale à une résolution réduite).

L'obtention de ces 4 images se fait en suivant le schémas ci-dessus : l'image est filtrée sur les lignes et les colonnes par des filtres H (passe-bas) ou G (passe-haut) suivant que l'on calcul les détails horizontaux, verticaux, diagonaux ou l'image d'approximation.

2.1 Décomposition multirésolution d'une image

Dans cette partie, nous allons effectuer une analyse multirésolution de l'image `technoforum.jpg`.

1. Tout d'abord, des fonctions utiles pour ce TP ont été regroupées dans un package Python `JP2000` disponible sous Moodle. Téléchargez et dézippez ce package dans votre répertoire de travail.
   Créez un nouveau fichier `TP_format.py` puis effectuez l'import des fonctions de ce package par : `import JP2000.Functions as jp2`.
   Vérifiez que ces fonctions sont désormais connues de votre programme principal (par complétion par exemple).

2. Après importations des modules nécessaires, chargez l'image `technoforum.jpg` dans la variable numpy `techno`.

```
import numpy as np
import matplotlib.pyplot as plt
...

techno= np.double(plt.imread("./technoforum.jpg"))
```

3. Recopier dans votre programme principal `TP_format.py` le code suivant, qui permet de charger les valeurs des filtres H et G dans les variables `LO_D` et `HI_D` :

```
# Affecte à la variable LO_D le filtre H de décomposition
LO_D=[1/np.sqrt(2),1/np.sqrt(2)]
```

```
# Affecte à la variable HI_D le filtre G de décomposition
HI_D=[-1/np.sqrt(2),1/np.sqrt(2)]
```

4. La fonction Python

   `jp2.decompose(filtre_ligne, filtre_colonne,im_in)`

   permet de renvoyer dans une variable le résultat de la décomposition d'une image `im_in` par un filtre sur les lignes (`filtre_ligne`) puis d'un filtre sur les colonne (`filtre_colonne`), suivi d'une réduction par 2 de la taille de l'image.
   Utiliser cette fonction pour calculer les images donnant les détails horizontaux (variable `h1`), verticaux (`v1`), diagonaux (`d1`) et l'image d'approximation (`a1`).

5. Importer tout d'abord `pyplot` : `import matplotlib.pyplot as plt`, puis visualiser le résultat de la décomposition pour une échelle avec la commande suivante (*np.abs*() donnant la valeur absolue) :

   ```
   plt.imshow(jp2.composite_image(a1,np.abs(v1),np.abs(h1),np.abs(d1)),cmap='gray')
   ```

6. Effectuer de même l'étape de multirésolution pour l'échelle suivante (images à créer : h2, v2, d2 et a2). *Attention à l'image à utiliser pour la décomposition à cette deuxième échelle ! ! !.*
   On pourra multiplier l'image a2 par 0.1 lors de l'affichage pour mieux voir les informations contenues dans les plans de détails.
7. Trouver un moyen pour visualiser toute la décomposition en une seule image (voir cours). Commentaires sur l'image obtenue ?

   `Appeler l'enseignant pour valider.`  1

2.2 Reconstruction de l'image à partir de son analyse multirésolution

Une fois la décompositon effectuée, seule la dernière approximation et les détails sont conservés (pour codage et transmission). Il s'agit ici de reconstruire l'image originale à partir de cette approximation et de ses détails.

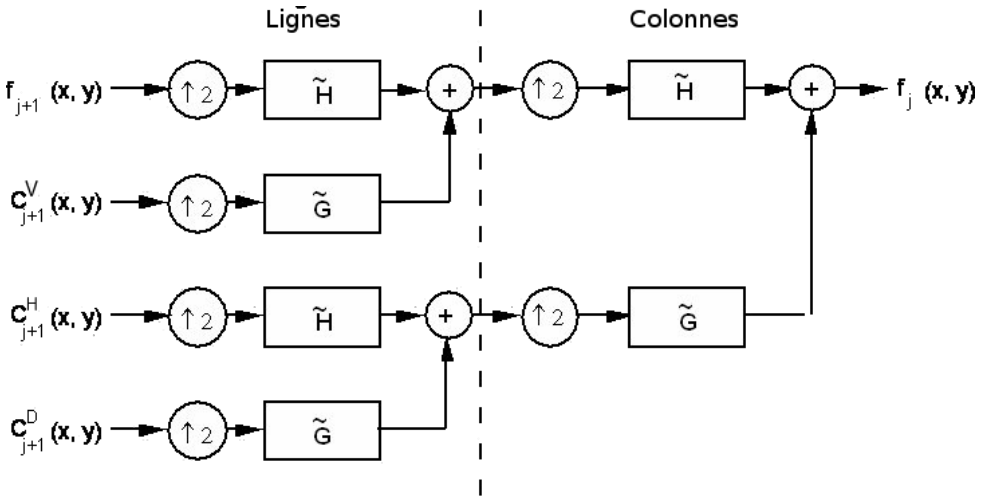
On effacera donc les variables a_1 et *techno* puisque l'on souhaite les reconstruire à partir des détails et de la dernière approximation a_2 .

La figure ci-dessous rappelle le schéma de reconstruction d'une image par un banc de filtres (ici pour une seule échelle). Les filtres $\tilde{H}$ et $\tilde{G}$ sont les filtres permettant de reconstruire le signal f_j.

1. Recopier dans votre programme principal `TP_format.py` le code suivant, qui permet de charger les valeurs des filtres de reconstruction Htilde et Gtilde dans les variables `LO_R` et `HI_R` :

   ```
   #Charge dans la variable LO_R le filtre H tilde de reconstruction
   LO_R=[1/np.sqrt(2),1/np.sqrt(2)]

   # Charge dans la variable HI_R le filtre G tilde de reconstruction
   HI_R=[1/np.sqrt(2),-1/np.sqrt(2)]
   ```



Bancs de filtres 2D (reconstruction).

2. La fonction Python :

   ```
   jp2.reconstruction(filtre_ligne, filtre_colonne,im_in)
   ```

   permet de renvoyer dans une variable le résultat de la reconstruction d'une image im_in par un filtre sur les lignes (filtre_ligne) puis d'un filtre sur les colonne (filtre_colonne) et de multiplier la taille par 2 de l'image.
   Utiliser cette fonction pour reconstruire les images donnant les détails horizontaux (h1_r), verticaux (v1_r), diagonaux (d1_r) et l'image d'approximation (a1_r) à partir de h2, v2, d2 et a2.

3. L'image reconstruite à l'échelle 1 a1_rec sera égale à la somme de h1_r, v1_r, d1_r et a1_r (une image peut se décomposer comme une somme de détails et une approximation). Visualiser le résultat de la reconstruction.
4. Effectuer les 2 étapes précédentes pour la résolution suivante et reconstruire l'image originale.

5. Vérifier que la reconstruction est quasi-parfaite en affichant la difference entre l'image reconstruite et l'image originale.

6. Pour s'en convaincre numériquement, calculer l'erreur totale de reconstruction (après `import numpy.linalg as nplin`) :

   ```
   err = nplin.norm(techno_rec-np.double(techno),2)
   print('Erreur de reconstruction SANS SEUILLAGE sur les coefficients:',err)
   ```

   L'erreur doit être proche de zéro aux erreurs numériques près...
7. Appliquer la décomposition puis la reconstruction sur l'image `Lena.jpg` et afficher dans un subplot 3×1 l'image originale, l'image de la décomposition et l'image reconstruite.

☑ 2 `Appeler l'enseignant pour valider.`

2.3 Seuillage des coefficients d'ondelettes

On va maintenant rendre plus "compacte" la réprésentation de l'image à partir de sa décomposition en ondelettes. Pour cela, comme évoqué en cours, on va seuiller les images

d'ondelettes (pour ne garder que l'information importante) et reconstruire l'image. Le fait que beaucoup de coefficients d'ondelettes non significatifs soient mis à 0 rendra plus efficace les étapes de codage effectuées par la suite (codage entropique).

1. Que réalise le code Python ci-dessous ?

```
thres=100
v2_s = np.where(np.abs(v2) > thres, v2, 0.0)
```

2. Reconstruire l'image *techno_seuil* en seuillant tous les coefficients d'ondelettes avec la même valeur et en reconstruisant l'image.
3. Comparer visuellement pour différentes valeurs de seuil la qualitée de l'image reconstruite (par exemple seuil = 20, 50,100 et 200).
4. Représenter pour chacune des valeurs de seuil l'énergie du signal gardée en fonction du nombre total de coefficients mis à zéro sur les plans d'ondelettes (courbes de compression).

   Indications : pour connaître le nombre de pixel à zéro d'une image de détails `image_details` (image d'un plan d'ondelette) : `image_details.size - np.count_nonzero(i`
   Pour calculer l'énergie d'un plan d'ondeletttes D : `(D**2).sum()`
5. Représenter, sur une même figure (fonction *subplot*), les images *techno* et *lena* avec leurs courbes de compression respectives. Essayer d'interpréter ces deux courbes en les comparant.

`Appeler l'enseignant pour valider.`  3

2.4 Reconstruction progressive de type SVC

On va simuler un codage scalable en reconstruisant l'image `Lena` à partir d'un nombre variable de coefficients d'ondelettes.

1. Pour l'image Lena, construire un tableau ordonné et décroissant de l'ensemble des coefficients d'ondelettes (image d'approximation exclue). On suppose que l'on souhaite garder les 100 coefficients d'ondelettes les plus grands pour reconstruire l'image. L'idée est de trouver la valeur du coefficient correspondant au 100ème coefficient d'ondelette le plus grand et de s'en servir de seuil pour tous les plans d'ondelettes. La reconstruction est similaire ensuite à ce qui a été fait dans l'exercice précédent.
   Indications :

```
 #Les coefficients d'ondelettes sont TOUS rangés dans une liste C_wave
cfs_wave =np.abs(C_wave)
cfs_wave.sort()
cfs_wave = cfs_wave[::-1]
```

2. Reconstruire l'image à partir de 1, 250, 500, 750 ... jusqu'à 100000 coefficients d'ondelettes en simulant une transmission progressive, avec une pause de 0.5 seconde entre chaque reconstruction (on pourra afficher en rouge la dernière zone de pixel reconstruite).
3. Calculer aussi pour chacune des reconstructions progressives le PSNR de l'image reconstruite. On simulera comme précédemment une transmission progressive en afficha pour chaque image reconstruite son PSNR et son erreur de reconstruction.

Pour rappel, le PSNR est calculé afin de quantifier la qualité de l'image dégradée f_b par rapport à l'originale f :

$$PSNR_{(\mathrm{dB})} = 20\log_{10}\left(\frac{d}{\sqrt{\frac{1}{N}\sum_{x,y}\left(f[x,y]-f_b[x,y]\right)^2}}\right) \quad (1)$$

avec (x, y) les coordonnées d'un un pixel et N le nombre de pixel de l'image f, et d la dynamique de l'image (ici $d = 255$ niveaux de gris).

☑ 4 `Appeler l'enseignant pour valider.`

2.5 Création d'une fonction générique de compression

Créez une unique fonction permettant la compressions d'une image, avec comme paramètres d'entrée l'image, les filtres dans un dictionnaire, le niveau de décomposition et le seuil. En sortie, l'image reconstruite, le PSNR et son entropie `H`.

2.6 Pour terminer...

Pour ceux qui ont fini, essayer d'appliquer votre algorithme à des images couleur (image `fox.jpg` : sous Moodle). On effectuera en première étape un changement de l'espace couleur de l'image d'origine (RVB) au modèle YUV (1 luminance, 2 chrominances).

☑ 5 `Appeler l'enseignant pour valider.`