INF4710
Introduction aux technologies multimédia

A2017 - Travail pratique #1
Compression sans perte

Objectifs :

- Permettre à l'étudiant de se familiariser à la manipulation de données et au traitement d'images en C++ avec OpenCV, ou bien avec le logiciel Matlab
- Permettre à l'étudiant de s'initier aux algorithmes de compression de données sans perte (codage prédictif et Huffman pour signaux 1D/2D)

Remise du travail :

- Avant le 15 septembre 2017, 15h00, sur Moodle – **aucun retard accepté**

Références :

- Voir les notes de cours sur Moodle (compression sans perte)

Documents à remettre :

- L'ensemble de votre code source (.m pour Matlab, .hpp/cpp pour C++, mais pas les deux!)
- Un rapport (**format .pdf, max 3 pages**) contenant :
  - un survol bref de votre travail : présentation codage prédictif + Huffman dans vos mots (<1 page)
  - une description brève des fonctions de prédiction que vous avez implémenté (<1 page)
  - un tableau de vos taux de compression pour toutes les paires d'images, avec discussion (~1 page)
  - les réponses aux questions posées à la fin de l'énoncé (<1 page)

Autres directives :

- Pensez à commenter l'ensemble de votre démarche directement dans votre code! Sinon, difficile d'attribuer des points lorsque ça ne fonctionne pas...
- Les TDs s'effectuent en équipe de **deux** (peu importe la section de laboratoire – À VOUS DE VOUS TROUVER UN COÉQUIPIER). Ne remettez par contre qu'une version du rapport/code!

Présentation

L'objectif de ce travail pratique est d'implémenter et de tester un algorithme de compression d'images sans perte basé sur le codage prédictif et sur le codage de Huffman. Ceci vous permettra de vous familiariser avec la manipulation de données provenant d'images, et d'acquérir un regard critique sur l'efficacité de différentes méthodes de compression.

Le travail peut être réalisé en C++ à l'aide d'OpenCV (*suggéré*), ou en Matlab. Les signatures des méthodes à implémenter ainsi qu'une partie de leur définition vous sont déjà fournies sur Moodle. Les signatures (en-têtes/déclarations) sont fixes et servent à simplifier la correction, alors ne les modifiez pas. Toutefois, le contenu des fonctions peut être modifié tel que vous le désirez. D'ailleurs, seul le contenu de ces fonctions sera corrigé; un « testbench » de validation (i.e. une autre application 'main') sera utilisé lors de la correction, alors assurez-vous de bien tout tester de votre côté! Par ailleurs, notez que les seules images qui seront testées lors de la correction sont celles déjà fournies avec l'énoncé, donc toutes des paires d'images en niveaux de gris (1 canal) ou RGB (3 canaux).

Il vous est suggéré de débuter par vous familiariser avec l'environnement de développement que vous choisirez; voir les références fournies ci-dessous. Vous devrez ensuite entreprendre l'implémentation de différentes fonctions de prédiction pour améliorer la compression de paires d'images. Notez que vous n'avez pas dans le cadre de ce TP à compléter l'encodage de Huffman, car vous ne l'aurez peut-être pas encore vu en classe — son implémentation est déjà fournie. Étudiez-là bien, ça risque de ressortir dans un examen!

Références

- Matlab
  - http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf
  - http://web.mit.edu/18.06/www/Spring09/matlab-cheatsheet.pdf
  - Affichage d'images : utilisez toujours '*imshow*'!
  - Remarque 1 : Matlab entrepose les données de ses matrices en format 'column-major'
  - Remarque 2 : [x=1,y=1] dans une image => coin haut-gauche, et nb. lignes := hauteur, nb. colonnes := largeur
  - Remarque 3 : Matlab n'entrelace pas les canaux de ses images (le canal est la 3^{e} dimension de la matrice)
- C++/OpenCV
  - http://opencv.org/
  - http://docs.opencv.org/trunk/db/deb/tutorial_display_image.html
  - http://docs.opencv.org/trunk/d6/d6d/tutorial_mat_the_basic_image_container.html
  - http://docs.opencv.org/trunk/db/da5/tutorial_how_to_scan_images.html
  - Remarque 1 : OpenCV entrepose les données de ses matrices en format 'row-major'
  - Remarque 2 : [x=0,y=0] dans une image => coin haut-gauche, et nb. lignes := hauteur, nb. colonnes := largeur
  - Remarque 3 : OpenCV entrelace les canaux de ses images en format BGR (on a donc 3 éléments par pixel)
- Codage prédictif
  - Notes de cours : compression images fixes sans perte, p.15—18
- Codage Huffman
  - Notes de cours : compression images fixes sans perte, p.20—31

Travail demandé

A - D'abord, vous devez compléter la fonction **« image2signal »** permettant de transformer une matrice 2D de un ou trois canaux en un signal 1D, qui lui sera par la suite compressé. Cette fonction reçoit comme paramètre la matrice à transformer ainsi qu'un identifiant indiquant quel type de prédiction sera nécessaire lors de la compression. Vous pourrez utiliser cet identifiant plus tard afin de modifier le comportement de formatage de votre fonction dans les cas spéciaux, ce qui peut vous aider à prédire de meilleurs symboles.

Pour l'instant, afin de tester vos habilités en Matlab/C++ et pour permettre à tous d'avoir une méthode de compression commune, on vous demande d'implémenter la transformation 2D→1D de base (le 'default case' de **« image2signal »**) de façon telle que le signal 1D obtenu contienne les valeurs R,G,B des pixels lus ligne par ligne. Un exemple pour une image en couleur de taille 2x3 (i.e. de trois colonnes et de deux lignes) est illustré ci-dessous :

C++/OpenCV (0-based indexing, row-major, interlaced) :

	Col # 0			Col # 1			Col # 2		
Ligne #0	Idx#0	Idx#1	Idx#2	Idx#3	Idx#4	Idx#5	Idx#6	Idx#7	Idx#8
Ligne #1	Idx#9	Idx#10	Idx#11	Idx#12	Idx#13	Idx#14	Idx#15	Idx#16	Idx#17

Signal voulu (selon les indices internes) : 2, 1, 0, 5, 4, 3, 8, 7, 6, 11, 10, 9, 14, 13, 12, 17, 16, 15

Matlab (1-based indexing, column-major) :

	Canal #1			Canal #2			Canal #3		
	Col. #1	Col. #2	Col. #3	Col. #1	Col. #2	Col. #3	Col. #1	Col. #2	Col. #3
Ligne #1	Idx#1	Idx#3	Idx#5	Idx#7	Idx#9	Idx#11	Idx#13	Idx#15	Idx#17
Ligne #2	Idx#2	Idx#4	Idx#6	Idx#8	Idx#10	Idx#12	Idx#14	Idx#16	Idx#18

Signal voulu (selon les indices internes) : 1, 7, 13, 3, 9, 15, 5, 11, 17, 2, 8, 14, 4, 10, 16, 6, 12, 18

Dans le cas des images qui n'ont qu'un seul canal (i.e. image en niveau de gris), vous n'avez qu'à lire les intensités directement ligne par ligne.

B - Par la suite, vous devrez similairement compléter la transformation inverse (signal 1D vers image 2D) du cas par défaut dans la fonction **« signal2image ».** Une fois cette étape complétée, vous pourrez exécuter le 'main' de votre projet pour tester vos deux implémentations. Si tout est bon, vous aurez alors à votre disposition une version de base du codage prédictif/Huffman, car toutes les autres fonctions sont déjà partiellement implémentées (ah, les chanceux!). Assurez-vous par contre de bien comprendre comment l'application fonctionne, ça pourrait ressortir dans un examen!

C - Complétez le 'main' de votre projet afin de lui faire calculer les taux de compression de votre algorithme de base sur les images de test (**voir les commentaires dans le code**). La formule à utiliser est la suivante :

Taux de Compr. = 1 - (Longueur du signal compressé / Longueur du signal original)

Vous pourrez ensuite observer l'efficacité de votre méthode de base sur les cas spéciaux de prédiction, et déduire votre prochain objectif : améliorer les taux de compression obtenus!

D - L'objectif ultime de ce TP est de proposer et d'implémenter différents « prédicteurs » dans la fonction « **computePrediction** » afin d'améliorer la compression des images dans les quatre cas spéciaux déjà identifiés dans le 'main' de votre projet. Pour y arriver, vous pouvez aussi modifier (au besoin) les transformations 2D→1D et 1D→2D dans les fonctions « **image2signal** » et « **signal2image** » afin de regrouper des symboles ayant des propriétés similaires.

Pour l'évaluation de votre travail, on s'attend à voir au moins un prédicteur spatial (1D ou 2D) pour traiter les images floues (*blur*) et bruitées (*noisy*). Pour ce qui est de la logique de prédiction elle-même, vous pouvez utiliser l'approche que vous désirez, mais il faut **absolument** que vos résultats finaux soient meilleurs avec vos prédicteurs spéciaux qu'avec le prédicteur naïf par défaut (l'algorithme de base). N'hésitez pas à explorer et utiliser des fonctionnalités plus poussées de OpenCV/Matlab dans votre implémentation!

Questions (réponses à joindre à la fin de votre rapport)

Encodage Prédictif :

1. L'encodage prédictif permet-il par lui-même de compresser le signal traité? Si oui, quelle est l'ampleur de sa contribution au taux de compression final? Si non, pourquoi est-il utilisé?

2. Suite à vos tests dans les cas spéciaux, la transformation 2D→1D des images avant la prédiction affecte-elle les taux de compression obtenus? Pourquoi?

Huffman :

3. Une réorganisation de la séquence de symboles fournis à Huffman peut-elle améliorer le taux compression final de l'algorithme?

4. Quel est l'avantage d'utiliser un encodage de Huffman au lieu d'un encodage LZ77 dans le cadre de ce TP?

Général :

5. L'organisation du signal 1D telle que demandée dans le cas de base (transformation 2D→1D ligne par ligne, section **A**) serait-elle idéale si utilisée avec codage par plage (RLE) ou LZ77?

Barême

- **Implémentation et fonctionnement : (10 pts)**
  - image2signal (cas de base) = 1.5 pts
  - signal2image (cas de base) = 1.5 pts
  - prédicteur images floues (doit être spatial) = 2 pts
  - prédicteur images noircies = 1 pt
  - prédicteur images bruitées (doit être spatial) = 2 pts
  - prédicteur images avec couleurs modifiées = 1 pt
  - calcul taux compression = 1 pt
- **Rapport : (5 pts)**
  - Présentation algos et implémentations prédicteurs = 1 pts
  - Validité des taux de compression & commentaires sur efficacité (cas par cas) = 3 pts
  - Lisibilité, propreté et complétude = 1 pts
- **Questions : (5 pts)**
  - Q1 = 1 pt
  - Q2 = 1 pt
  - Q3 = 1 pt
  - Q4 = 1 pt
  - Q5 = 1 pt

Total sur 20 pts