Script de Présentation

Slide 1 Page initiale

Bonjour [mesdames et messieurs] du jury, je suis ici aujourd'hui pour vous présenter ma soutenance de TIPE. Cette année, moi et deux camarades avons travaillé sur la compression de données, et plus particulièrement son application aux images.

Slide 2 Introduction

Tout d'abord, commençons par **définir** ce qu'est la compression. La compression est un **procédé transformant une suite de bits A en une suite de bits B plus courte pouvant restituer les mêmes informations, ou des informations voisines.**

La notion d'informations voisines intervient lorsque l'on évoque des **deux types de compressions différent** : la compression **avec perte** (que l'on appelle *lossy* en anglais) et la compression **sans perte** (*lossless*). Nous verrons au cours de cette présentation comment choisir entre ces deux types et comment on peut également les faire fonctionner ensemble.

Dans le thème de TIPE de cette année qu'est **la ville**, la compression est une notion très présente, puisque **la nécessité** constante d'échange d'informations fait qu'il y a beaucoup de données à traiter et **la compression** permet de **de stocker et traiter efficacement toutes ces données**.

Slide 3 Procédés de compression

Afin de compresser des données, il existe **tout un tas de différents procédés**. Dans la compression *lossless* on retrouve encore deux types, que sont les compressions **entropiques** et **algorithmiques**. La compression entropiques supposent des *a priori* sur ce que l'on compresse, comme des **tableaux de fréquence** ou des **tables de codage** comme on le verra plus tard, tandis que les compressions algorithmiques ne nécessitent que de connaître la méthode de compression utilisés afin de décompressé.

De manière générale, les compressions fonctionnent par réorganisation des données, et par l'application de transformées mathématiques permettant de stocker l'information de manière plus optimale.

C'est le cas de l'algorithme JPEG, le format d'image le plus utilisés et celui sur lequel nous avons basé nos recherches, qui utilisent beaucoup des procédés les plus populaires pour compressé des images. Nous verrons son implementation au fur et à mesure de la présentation.

Slide 4 Entropie et Codage Optimal (1)

Lorsqu'il est question de codage en compression, la théorie mathématique qui définit les fondements de ces codages est la théorie de l'information notamment fondé par Claude Shannon en 1948. Cette théorie probabiliste, qui quantifie l'information d'un ensemble de messages, se base sur la notion d'entropie, définit ainsi : [lire définition du diapo]

Slide 5 Entropie et Codage Optimal (2)

Voici quelques définitions importantes de la théorie de l'information, dont nous aurons besoin pour la suite : [lire et expliquer les définitions]

Rq : l'extension C^+ est l'application qui à Ω^* associe la concatenation des codes.

Slide 6 Entropie et Codage Optimal (3)

Une famille de code uniquement décodable tel que définit précédemment est celle des code préfixe : [lire définition]

[Présenter l'exemple] : Comme vous pouvez le voir, a gauche on retrouve un code non préfixe, puisque a, codé par 0, est préfixe de c, codé par 01. On a alors un code non-uniquement décodable, puisque par exemple, 01 code à la fois ab et c. En revanche l'exemple à droite est bien préfixe, et chaque code est bien unique.

Slide 7 Entropie et Codage Optimal (4)

L'inégalité de Kraft, tel que défini ici, permet d'avoir une condition nécessaire et suffisante afin d'avoir un code préfixe. En effet si D est la taille de l'alphabet du code, la somme des D^{-l_i} est inférieure à 1 si et seulement si le code est préfixe. Où les l_i sont les longueurs des codes.

Cette inégalité intervient notamment dans le théorème fondamentale à la compression : le théorème de codage de source, qui nous dit que plus la compression augmente, plus la longueur moyenne des mots du code tendent vers l'entropie, telle que définit précédemment. Ce théorème impose donc une limite théorique quant à l'optimalité des codage.

Slide 8 Codage de Huffman (1)

Présentons maintenant deux exemples de codage de source, et commençons par le codage de Huffman. Ce codage, optimal au niveau symbole et imposant un nombre entier de bit par symbole, permet de former des codes plus longs pour les symboles apparaissant peu. En effet montrons le avec l'exemple du codage du message : "LES VILLES". Dans ce tableau, on représente pour chaque symbole, son nombre d'apparition. On crée ensuite un **tas-min** depuis lequel on extrait les deux nœuds de poids minimums. On crée alors un nouveau n œud dont le poids est la somme des poids des deux fils, et on ajoute ce dernier au tas-min. On procède ainsi de suite afin de construire un arbre de codage : voici ici celui de "LES VILLES".

Slide 9 Codage de Huffman (2)

En étiquetant les arrêtes par des 0 à gauche et des 1 à droite, on obtient pour chaque symbole un code, voici donc la table de codage associé. Le code de Huffman de "LES VILLES" est alors celui-ci, qui pour l'humain peut paraître plus long et indigeste, mais qui pour l'ordinateur est beaucoup plus court et confortable.

Slide 10 Codage Arithmétique (3)

Le second codage sur lequel nous avons travaillé est le codage arithmétique, qui est quant à lui optimal au niveau bits, et qui code par morceaux et non par symbole comme le fait Huffman. En prenant dorénavant pour exemple le mot "VILLE", on calcul maintenant les probabilités d'apparition de chacun des symboles afin de découper l'intervalle $[0, 1[$ en morceaux de taille proportionnelle à la probabilité du symbole. Par un calcul successifs de bornes inférieurs et supérieurs dont je ne m'attarderais pas ici, on trouve finalement que $x = 0,068$ peut coder le mot "VILLE". Pour le décodage, on transforme x inversement à ce qu'on a fait dans le codage, afin de retrouver le mot initial, mais toujours en se servant de la table construite initialement.

Slide 11 La représentation d'Image

Après avoir vu les procédés de compression s'appliquant à tous types d'information, intéressons nous maintenant aux procédés s'appliquant particulièrement aux images. Pour cela, demandons nous d'abord comment les représentées efficacement.

La représentation d'une image utilisée par les ordinateurs est RGB, chaque pixel est associé à une composante rouge, verte, et bleue, comme vous pouvez le voir si dessous.

Slide 12 La représentation YCbCr

Mais cette représentation n'est pas forcément optimale pour l'œil humain, on préférera la représentation $YCbCr$. Pour passer du RGB, au $YCbCr$, on applique φ à chacun des pixels. Le détail de φ est donnée ici, mais n'est pas essentiel, on change juste comment on stock l'information de l'image. Voici un exemple avec toujours la même photo. (Je ne sais si c'est très perceptible mais) Pour l'œil humain, la composant Y prédomine, tandis que les chrominances contiennent moins d'information.

Slide 13 Sous-Échantillonage

Ce phénomène permet de procéder à un sous-échantillonnage, c'est-à-dire que l'on va moyenner les pixels de chrominances afin de gagner de la place sans pour autant changer la perception de l'œil. Ce procédé est le premier *lossy*. Vous pouvez ici voir différentes manières de "sous-échantillonner" une matrice d'image.

Slide 14 DCT (1)

Après avoir optimisé la représentation des couleurs d'une image, on peut optimisé la représentation des variations de ces couleurs. Pour cela on applique à notre matrice d'image la DCT (transformée en cosinus discrète). Cette transformé, prend en en entrée une image perçue comme un signal, et le transforme comme des fréquences. Pour le cas des images, on appliquera la 2D-DCT qui consiste juste à appliquer sur chaque ligne puis sur chaque colonne.

Slide 15 DCT (2)

Les images "naturelles", celle que l'on manipule tous les jours, témoigne d'une sorte de "continuité". Cette "continuité" fait qu'il y a peut de variations de hautes fréquences et donc l'information est majoritairement regroupé dans les coefficients de basses fréquences, ce qui, encore une fois, optimise le regroupement de l'information.

Slide 16 Quantification

La seconde étape *lossy* de l'algorithme JPEG est la quantification. Comme on l'a vu, tous les composantes de couleurs et toutes les fréquences ne possèdent pas la même quantité d'information, on peut donc réduire celle qui en contiennent le moins en divisant, coefficients par coefficients, les matrices d'images par des valeurs déterminés empiriquement qu'on appelle matrice de quantification. Cette étape fait donc gagner de l'espace en uniformisant les valeurs des coefficients pour ainsi créer des redondances. Les matrices de quantifications distinguent donc les fréquences et les composantes, on peut en plus optimiser en leur appliquant un facteur de qualité, augmentant ou réduisant la perte d'information.

Slide 17 Codage par Plages

Après toutes les étapes décrites précédemment, on peut observer beaucoup de redondances dans la matrice de l'image. Pour en tirer parti lors de la compression, on effectue un codage par plages, également appelé RLE (pour Run-Length Encoding en anglais). Plutôt que de transmettre le message ainsi, on le transmet par symbole suivi du nombre d'occurrence avant le suivant. Ce qui donne ce nouveau message. On passe de 15 à 8 caractères et on n'a perdu aucune information. Dans les images, on fait apparaître encore plus de redondances, en lisant la matrice en zigzag, comme montré ici.

Slide 18 Exemple de Compression

Après avoir présenté tous les différents procédés de compression intervenant dans le JPEG, comparons les influences de certains de ces procédés. Voici d'abord l'image de la ville de Lyon non-compressée, on peut voir que sa taille est de 10,3 Mo.

En utilisant une compression presque lossless, c'est à dire en quantifiant très peu les matrices et en n'effectuant on gagne l'image parait inchangé et l'image prend un peu plus de 4 fois moins de place.

En continuant de baisser la qualité globale de l'image on peut voir que l'image reste très perceptible et que la taille ne cesse de diminuer.

Avec cette dernière images, on commence à apercevoir à l'œil nu les artefacts du JPEG, mais l'image reste compréhensible.

Slide 19 Comparaison (1)

Enfin, nous avons procédés à des études de gain de temps et de place en fonction des différentes compression, et on peut remarquer que le JPEG est d'autant plus efficace que l'on baisse en qualité. De manière générale, le JPEG semble donc plus utile lorsque la perte en qualité n'est pas un problème pour l'application ensuite.

Slide 20 Comparaison (2)

Lorsqu'il est question du codage de Huffman, il est possible de les calculer lors de la compression et de les rendre optimal pour une image particulière, au prix du temps de calcul, on d'utiliser des tables standards,

optimisées statistiquement, résultant en gain de temps mais une perte de place. Comme le montre ce graphique, le gain en temps est inférieur au gain en espace selon les deux cas. De manière générale, il semble donc préférable d'utiliser des tables standards.

Slide 21 Effectué

En conclusion, voici la liste des objectifs qui ont été réalisé et ceux qui n'ont pas pu l'être. L'Implémentation du codage arithmétique avec les standards du JPEG nous a notamment posée beaucoup de difficulté.

Ma présentation est terminée, avez-vous des questions ?