Fiche de CODO
Fanny Riols

Les phases de la compression JPEG (en super détaillé)

Image répartie en matrice 8x8 → Passage de RGB à YCbCr → Downsample
|
V
Linéarisation ← Quantification ← DCT
|
V
Compression RLE et Huffman → Image compactée

I. Des carrés en 8x8

Subdiviser l'image en carrés de 8x8 : on ne travaille ensuite que sur ces blocs (on ne touche pas aux basses fréquences de l'image, car on veut supprimer les hautes fréquences).

II. Centrage

On soustrait 128 à chaque pixel, pour avoir une moyenne proche de 0.

III. RGB → YUV

Ce traitement n'utilise pas le codage des couleurs RVB, mais à la place un codage de couleur de type luminance/chrominance, celui YUV.

- ➢ En effet, le codage à base de YUV propose une meilleure compression et est plus adapté étant donné que l'oeil est plus sensible à la luminance/luminosité, qu'à la chrominance d'une image.
- ➢ Or la luminance est présente dans les trois couleurs rouge, vert, bleu.
    - Ainsi, on transforme les composantes RGB de l'image en :

- Une composante de luminance notée Y : image en niveau de gris.
- Deux composantes de chrominance notée U et V : chrominance bleue et rouge.

IV. Downsample

L'image est représenté selon 3 tableaux : YUV. L'oeil est moins sensible aux variations de chrominance : sur chaque carré 2x2 des matrices U et V on fait la moyenne des valeurs, puis on enlève une colonne sur deux & une ligne sur deux de U et de V.

➢ Taille des données divisée par 4 : moins de compression à faire

V. DCT

La DCT (Discrete Cosine Transform) est une transformée de Fourier dont on a remplacé les sinus par des cosinus en rendant la fonction paire.

➢ Fourier rajoute des hautes fréquences (que l'on ne veut pas) à cause des discontinuités, qui disparaissent en rendant la fonction paire.

On l'applique sur chaque carré.

Plus on s'éloigne de l'origine, plus les fréquences correspondantes sont élevées. Or la plupart des images sont composées d'informations de basses fréquences. Donc quand on s'éloigne de l'origine, on trouve des coefficients faibles et qui en plus sont peu importants pour la vision. Ainsi la force de la DCT est de "choisir" les éléments importants de l'image (ce qui semble difficile à faire sur une image non transformée).

<u>Principe de l'algo :</u>

On extrait les fréquences de notre bloc avec la transformée de Fourier spéciale.

➢ On rend notre image paire puis Fourier dessus.

On décrit ainsi chaque bloc en un spectre de fréquence et en amplitude.

➢ La fréquence indique l'importance et la rapidité d'un changement.

➢ L'amplitude correspond à l'écart associé à chaque changement de couleur.

VI. Quantification

C'est la principale étape de perte de données : on perd de la précision sur les fréquences.

On utilise une matrice de quantification (en fonction de la qualité voulue) pour réduire les fréquences et ne garder que les basses fréquences du bloc.

- ➢ On divise chaque terme de la matrice obtenue par la DCT par son terme correspondant de la matrice de quantification.
- ➢ Elle annule plusieurs coefficients en bas à droite du bloc (hautes fréquences).

L'image n'a pas encore été réduite, c'est pendant le codage que l'on va obtenir un gain de place.

VII. Linéarisation

Dans notre matrice quantifiée, il y a beaucoup de 0. On utilise une matrice zigzag pour extraire les coefficients des fréquences selon les harmoniques (les anti-diagonales), en s'arrêtant au dernier nombre non nul (avant la fin donc, puisqu'il y a des 0 en bas à droite du bloc). Ainsi, on a plus beaucoup de valeurs pour définir notre bloc de 8x8.

VIII. Codage

On utilise une table de Huffman définie dans la norme JPEG (pas de perte de place dans l'entête) pour coder les valeurs des fréquences quantifiées linéarisées (assez petites et redondantes grâce à la quantification). Cette étape va compresser le résultat final, et c'est le contenu des fichiers jpeg.

IX. Limites

Problème avec Fourier : on supprime l'erreur, ce qui fait que pour un fort contraste, on a des oscillations avant et après qui sont normalement compensées par l'erreur.

X. Quand l'utiliser ?

JPEG est très bon pour les photos en niveau de gris et en couleur, mais très mauvais pour le noir et blanc (exemple : un texte).

Algorithmes à savoir faire/comprendre

I. Huffman

On veut coder les caractères les plus fréquents sur le moins de bits possibles.
Algo :

- On fait une liste de noeuds d'arbre contenant les chars, valués par leur fréquence et triés par fréquence décroissante.
- On fusionne les deux derniers et on les insère dans la liste à nouveau.

II. RLE (Run Length Coding)

C'est une méthode de compression très simple. Son principe est de remplacer une suite de caractères identiques par le nombre de caractères identiques suivi du caractère. Par exemple, AAAAAAA sera *7A.

III. BZIP2

- On réarrange le fichier pour mettre les caractères identiques à côté.
- On fait un MTF (move to front) : on garde la première occurrence d'une suite et on met les autres à 0.
- On utilise Huffman.

IV. LZW

Au fur et à mesure qu'on lit, on monte un dictionnaire pour sauvegarder les patterns souvent croisés. Quand on tombe sur le premier pattern, on met un char spécial pour dire qu'on augmente le nombre de bits de chaque char de 1, et on met le code du pattern sur 9 bits.

V. MPEG

On peut compresser naïvement toutes les images en JPEG : MJPEG. Mais il est plus efficace de calculer la différence entre deux images qui comporte souvent beaucoup de

zéros (ressemblance). Et en cas de changement brutal de l'image, on renvoie une image JPEG complète.

- ➢ Problème : accumulation de l'erreur lors du décodage à cause de JPEG.
    - Solution : on décompresse au moment de la compression pour calculer l'erreur et on l'ajoute à l'image envoyée pour la compenser.

Gagner en efficacité : on essaye de calculer le mouvement de translation sur des blocs 16 * 16, ainsi on envoie une image noire et un vecteur de translation.

Problème avec MPEG : pour le montage vidéo on ne peut pas couper où on veut (si on coupe entre une image de base et une image de différences, on ne pourra pas reconstruire l'image si on a perdu du l'image de base) => MJPEG c'est cool des fois.

Comparaison des différents algorithmes

	Avec ou sans perte	Utilité	Avantages	Inconvénients
JPEG	Avec	Image en niveau de gris, en couleur	Taux de compression à 90%. Cool pour de très grandes images.	Si contraste trop grand : mosaïque car l'image est divisée en petits carrés 8x8 traités séparément, donc on voit les bords des différents carrés restitués.
Huffman	Sans	Texte et image (<= 256 couleurs)	40/50% de compression pour les textes (suffisant). Ressemble au principe du morse : lettre fréquente codé sur peu de place → procédé d'agrégation objectif.	Lire tout le fichier pour connaître les fréquences. Très sensible à la perte d'un bit. Taux de compression trop bas pour une image en 16 millions de couleurs.
LZW	Sans	Texte et image (<= 256 couleurs)	50% de compression. Rapide, asymétrique (algo décompression différent de celui de compression). Meilleur que Huffman quand le fichier n'est pas trop petit. Bien quand il y a peu de couleur. Good pour image de synthèse (PNG).	Peu efficace pour les images (utilisé pour les GIF <= 256 couleurs).
RLE	Sans	Texte et image (<= 256 couleurs)	40% de compression. Composée de répétitions de pixels, de couleur identique, codés chacun par un caractère. Rapide, peu de ressources.	Souvent utilisé en complément de jpeg. Inefficace pour des images en 16 millions de couleurs : trop de nuances donc pas de longues séquences de pixels identiques.

Questions en vrac

Un ensemble de données aléatoires est-il compressible ? Dans le cadre de la théorie de l'information, contient-il un maximum ou un minimum d'information ? Quelle est la valeur de son entropie ?

- Un ensemble de données aléatoires n'est pas compressible.
- Dans le cadre de la théorie de l'information, il contient un maximum d'informations.
- Un ensemble de données aléatoires a une entropie maximale.

Dans une compression non conservative, quel type d'informations accepte-t-on de perdre ?

- On garde le sens général des données.
- On perd ce que l'on ne perçoit pas, ou peu.

Partiel 2006

Soit I = (10 20 40 20)
(10 30 50 30)
(10 30 40 30)
(20 20 30 10)

et les pixels sont codés sur un octet et varient de 0 à 254, la valeur 255 étant réservée.

Quelle est la taille en bits de ces données ?
4 * 4 * 8 = 128 bits.

Calculez l'entropie d'ordre zéro, So, en bits/pixel.
So = - sigma(Pi log2(Pi))
P10 = 4/16
P20 = 4/16
P30 = 5/16
P40 = 2/16
P50 = 1/16
So = 2.15 bit/pixel

Quelle taille approximative obtiendrait-on à la suite d'un codage entropique ?
16 * 2.15 = 34.4 = 35 bits.

Compressez par l'algorithme LZW classique l'image 1 linéarisée (sans caractère de saut de ligne). Le dictionnaire de départ est constitué des 254 premiers nombres codés sur un octets, la valeur 255 est réservée pour le changement de taille des adresses de 8 à 9 bits. Calculez la taille des données ainsi compressées.

reçu	émis	dico	adresse
10	/	/	/
20	10	1020	256
40	20	2040	257
20	40	4020	258
10	20	2010	259
30	10	1030	260

50	30	3050	261
30	50	5030	262
10	30	3010	263
30	*	1030	@260
40	@260	103040	264
30	40	4030	265
20	30	3020	266
20	20	2020	267
30	20	2030	268
10	*	3010	@263
EOF	@263	3010EOF	269

Taille des données : 9 * 8 + 6 * 9 = 126 bits.

Simulation d'une compression JPEG en utilisant une transformée d'Hadamard à la place d'une DCT dans le but de simplifier les calculs. On donne la matrice d'Hadamard pour N=4 :

H = (1 1 1 1)
(1 1 -1 -1)
(1 -1 -1 1)
(1 -1 1 -1)

Calculez la transformée d'Hadamard H(I) (lignes puis colonnes) de l'image I. Sans oublier de diviser par le coefficient de normalisation (1/4), on arrondira les coefficients de manière à obtenir des entiers.

H(I) lignes :

I * H = (45/2 -15/2 -15/2 5/2) = A
(30 -10 -10 0)
(55/2 -15/2 -15/2 -5/2)
(20 0 -5 5)

H(I) colonnes :

H * A = (25 -6 -8 1)
(1 -2 -1 0)

(	-4	3	1	2)
(	0	-1	0	-1)

Soit Q1 = (8 6 4 2) et Q2 = (1 1 2 4)
(6 1 2 1) (1 2 4 6)
(4 2 1 1) (2 4 6 8)
(2 1 1 1) (4 6 8 10),
deux matrices de quantification. Laquelle vous semble la plus adaptée à la compression de H(I) ? Quantifiez H(I). On obtient QH(I).

Q2 a des gros chiffres en bas à droite. Elle semble donc être la plus adaptée car elle élimine les hautes fréquences tandis que Q1 élimine les basses fréquences.

Q2H(I) =	(	25/1	-6/1	-8/2	¼	)
	(	1	-1	-¼	0	)
	(	-4/2	¾	⅛	2/8	)
	(	0	-⅛	0	-1/10	)
=	(	25	-6	-4	0.25	)
	(	1	-1	-0.25	0	)
	(	-2	0.75	⅛	0.25	)
	(	0	-⅛	0	-0.1	)

Linéariser QH(I) suivant une séquence zigzag dont voici les indices : 0, 1, 4, 8, 5, 2, 3, 6, 9, 12, 13, 10, 7, 11, 14, 15.

QH(I) =	(	25	-6	-4	0	)	
	(	1	-1	0	0	)	→ le but est
	(	-2	0	0	0	)	d'avoir des 0
	(	0	0	0	0	)	

Partiel 2012

Soit le fichier F = {BCADADBCBCADBCAD}
Calculez l'entropie d'ordre 1 de ces données. À cet ordre, le fichier est-il compressible ?
Entropie : $F(S) = - \text{sigma}(P_i \log_b P_i)$
A = 4/16, B = 4/16, C = 4/16 et D = 4/16.
→ $F(S) = -¼ * \log_2(¼) * 4 = -1 * -2 = 2$
Le fichier n'est pas compressible car les caractères sont équiprobables.

Existe-il une entropie d'ordre supérieur qui révélerait une autre organisation des données ?
On retrouve la même organisation à l'ordre 2. En effet :
BC = 4/8 et AD = 4/8.
Néanmoins, à l'ordre 4, on obtient :
BCAD = ¾ et ADBC = ¼
Dans cet ordre là, le fichier est compressible.

* : caractère réservé sur 8 bits, prévenant le décompresseur qu'il doit ensuite lire les données sur 9 bits.
d, o, *, 256, 256, r, e, m, i, 260, 256, 262, 260, 264, o.
Le dictionnaire par défaut du décompresseur est constitué des 256 caractères ASCII. Le fichier se termine par un caractère de fin dont on ne tiendra pas compte.
Décompressez ces données.

Reçu	Ecrit	Dico	Adresse	phrase
d	/	/	/	/
o	d	do	256	d
*	/	/	/	/
256	o	od	257	do
256	256	dod	258	dodo
r	256	dor	259	dododo
e	r	re	260	dododore
m	e	em	261	dododorem

i	m	mi	262	dododorem
260	i	ir	263	dododoremi
256	260	red	264	dododoremire
262	256	dom	265	dododoremiredo
260	262	mir	266	dododoremiredomi
264	260	rer	267	dododoremiredomire
o	264	redo	268	dododoremiredomirered
EOF	o	/	/	dododoremiredomireredo

Quel est le taux de compression de cette séquence ?
Avant compression : 22 * 8 = 176 bits.
Après compression : 3 * 8 + 12 * 9 = 132 bits.
Gain de 44 bits.

Dans le cadre d'une application où aucun caractère réservé n'est possible (un flux binaire codé sur 8 bits par exemple), quelle solution suggérez-vous pour s'en passer ?

- On commence le codage sur 9 bits, en laissant le poids fort à 0.
- Quand on arrive à l'emplacement où il doit y avoir un marqueur, on passe le poids fort à 1.
- Et on continue sur 10 bits.