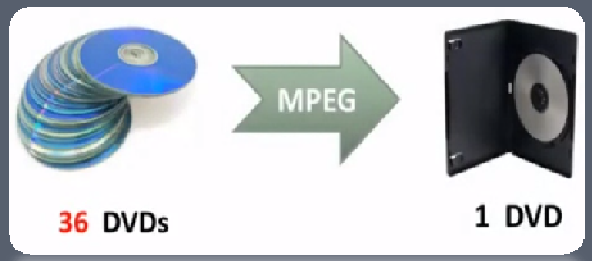


TRAITEMENT DES DONNÉES MULTIMÉDIA

CHAPITRE 2

COMPRESSION DES IMAGES ET VIDÉOS

ISET Kairouan

Bassem Seddik

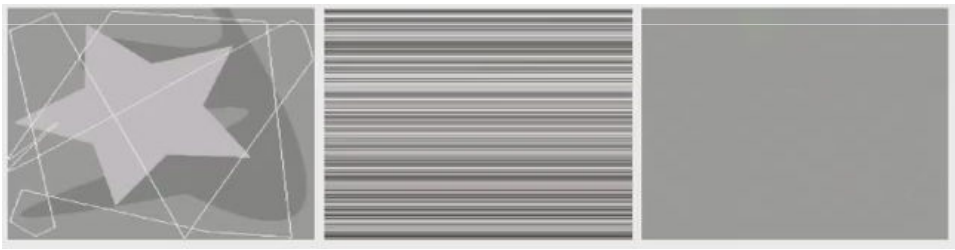
Pourquoi faire la compression?

- La taille de stockage

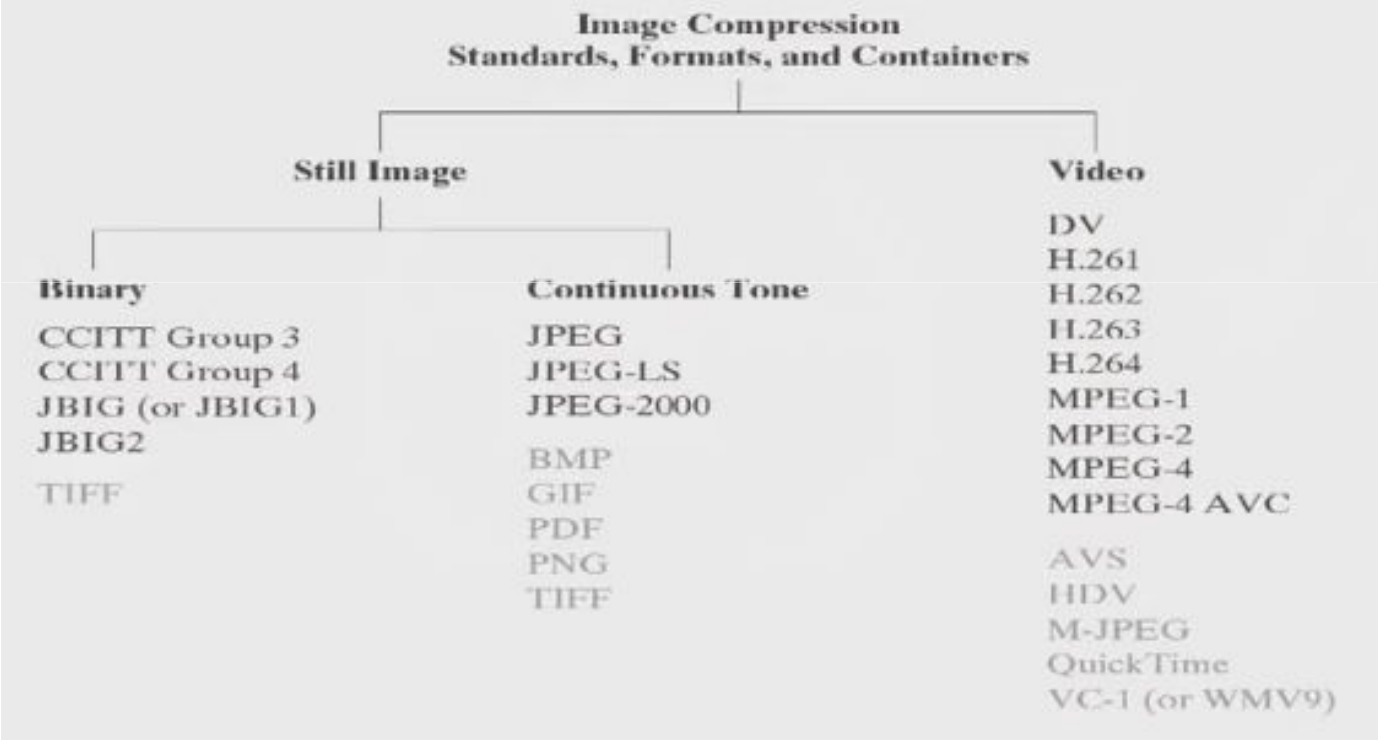
$$90^{min} \times 25^{frame/s} \times 576^{line/frame} \times 720^{pixel/line} \times 24^{bit/pixel}$$

$$= 168\ GB$$

- La présence de redondance
- Le nombre de couleurs limité
- Le gains des informations non exploitées



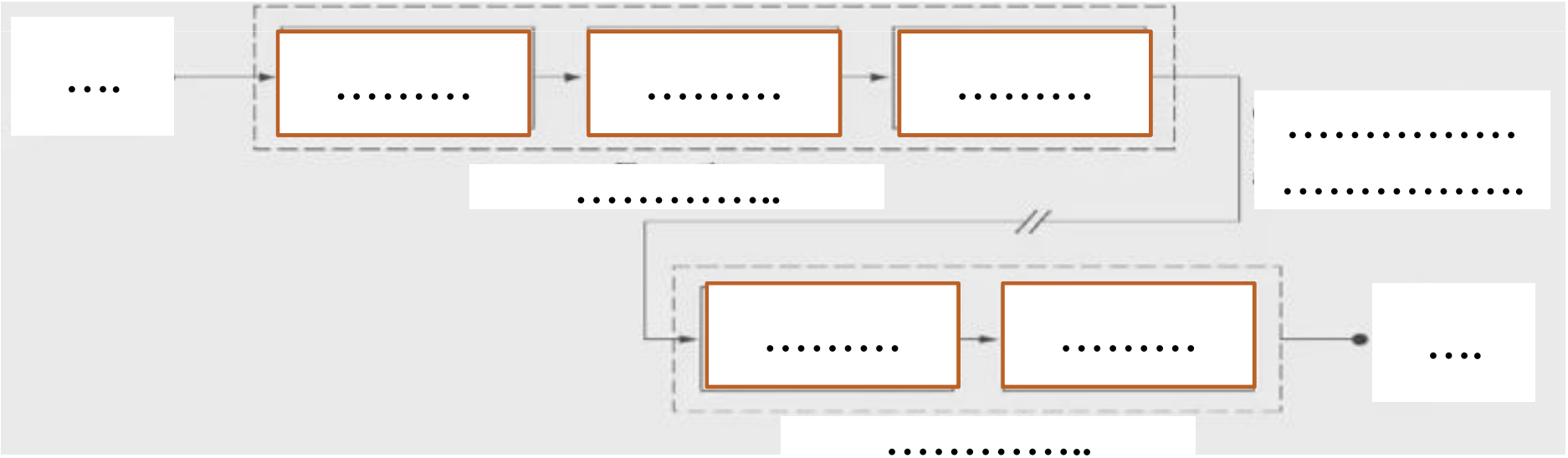
Création des standards de compression



- Images: JPEG-LS fidèle à la sources
- Vidéos: Les MPEG 1~4 sont les plus fameuses

Chaine de compression

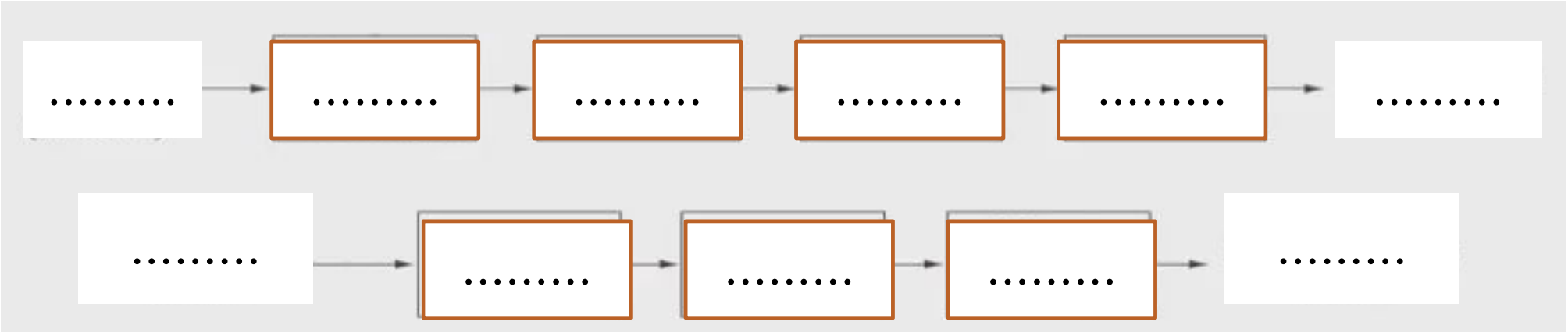
1. Mise dans un espace facile à compresser
2. Quantification → introduction de l'erreur (17 / 2) x2 ~16
3. Storage sous format limité par des symboles bien choisis



JPEG

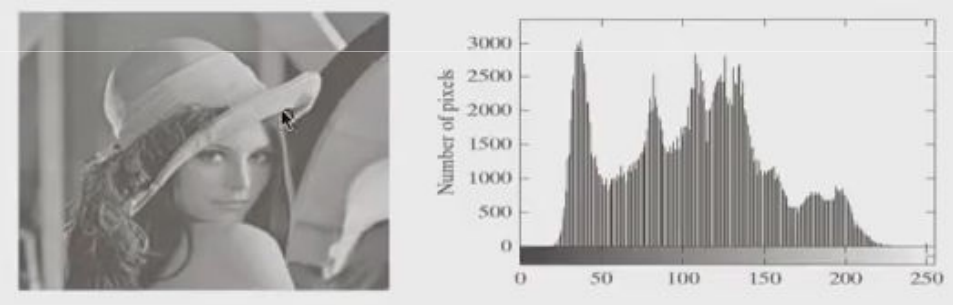
1. Grille 8 x 8
2. Application de la DCT
3. Codage Huffmann

- Décodage



Codage de Huffman

- Histogramme:
  - Combien une valeur apparait dans l'image
- Idée: Quelques pixels sont plus importants
  - Coder ceux qui apparaissent plus sur plus de codes

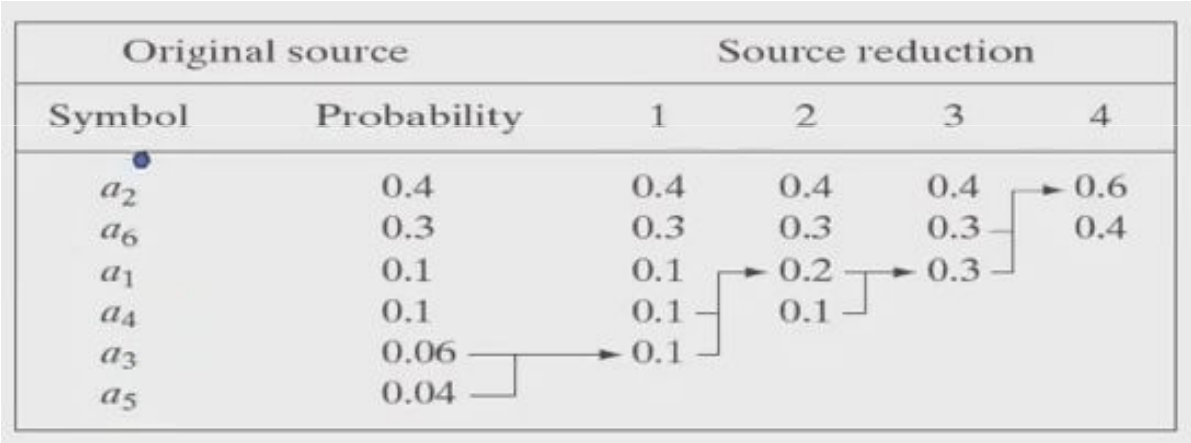


- Chaque pixels est représenté par sa donnée binaire
- Valeur totale : 1.81
- Par codage de Huffman

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	—	8	—	0

Codage de Hufman

- Mise en ordre des probabilités
- Sommes des valeur
- Mise en ordre encore
- Répéter …
- Choix si valeurs égales
- Arret si on 2 valeurs seulement



Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

- 0 nous ramène au groupe de 0.6
- 1 nous ramène au groupe de 0.4
- Division de la valeurs basse + attribution de code
- Obtention d'un code de longueurs différentes → gain d'espace

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

Exercice 1 : Codage de Huffman

- On considère un texte source formé à partir de 5 symboles distincts (a, b, c, d, r) avec les fréquences d'apparition suivantes :
  - f(a) = 0,43 ; f(b) =0,20 ; f(c) = 0,1 ; f(d) = 0,09 ; f(r) = 0,18.
  - a- Générer un arbre de Huffman binaire et proposer le codage correspondant en affectant les 0 aux probabilités les plus grandes.
  - b- Coder le texte suivant et calculer le gain de compression par rapport à un code binaire de longueur fixe et minimale : abracadabrabracadabra

Exercice 2 : Codage de Huffman

- Supposons que les caractères a, b, c, d, e et f aient des probabilités d'apparition respectivement égales à 0,07 ; 0,09 ; 0,12 ; 0,22 ; 0,23.
  - a- Quelle est la probabilité d'apparition du caractère f ?
  - b- Trouver le codage de Huffman pour ces 6 lettres en dessinant l'arbre binaire correspondant.
  - c- Quelle est la longueur moyenne du codage ?
  - d- Il faut au minimum 3 bits pour coder les 6 lettres. Pourquoi ne pas avoir choisi le code : code(a) = 0 ; code(b) = 1 ; code(c) =00 ; code(d) = 01 ; code(e) =10 ; code(f) = 11 ?

- Rq.: Code à préfixe libre:
- contre exemple:
  - a1 (1); a2 (0) ; a3 (01) → ambiguité
- Est-il le plus court des codages?
  - Oui (théorème)
- Calcul de l'entropie

$$H = -\sum_{symboles} p(s) \log_2 p(s)$$

  - Permet de savoir le taux de compression possible par notre encodeur

Blocs JPEG

- Codage 8x8 pixels
- Directe si en Niveaux de gris
- Les canaux Rouge, Vert, Bleu sont corrélés fortement
- → Passage Y (luminance) Cb Cr (couleurs) par le moyen d'une matrice 3x3 inversible
- Sur chaque sous image, on applique la Transformée de Cosinus Discrète (TCD)

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Transformée de Cosinus Discrète (DCT)

- Mean Square Error (MSE) ou Erreur carrée moyenne
- On cherche à dé-correler les composantes de l'image en temps réel
- La Transformation de Karhunen Loeve
  - Meilleurs résultat possibles avec 1seul px
  - Mais Matrice à calcul Lourd et imprévisible
- Solution: la DCT
  - Moins performante, mais
  - Efficace pour toutes les images
  - Transformation inversible identique

$$MSE = \sqrt{\left[1/nbpx. \sum_{pixels} (\hat{F} - F)^2 \right]}$$

$$T(u,v) = \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$F(u,v) = \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

- La DCT:
  - La partie réelle de la transformée de fourrier
- La même fonction pour la transformation et son inverse

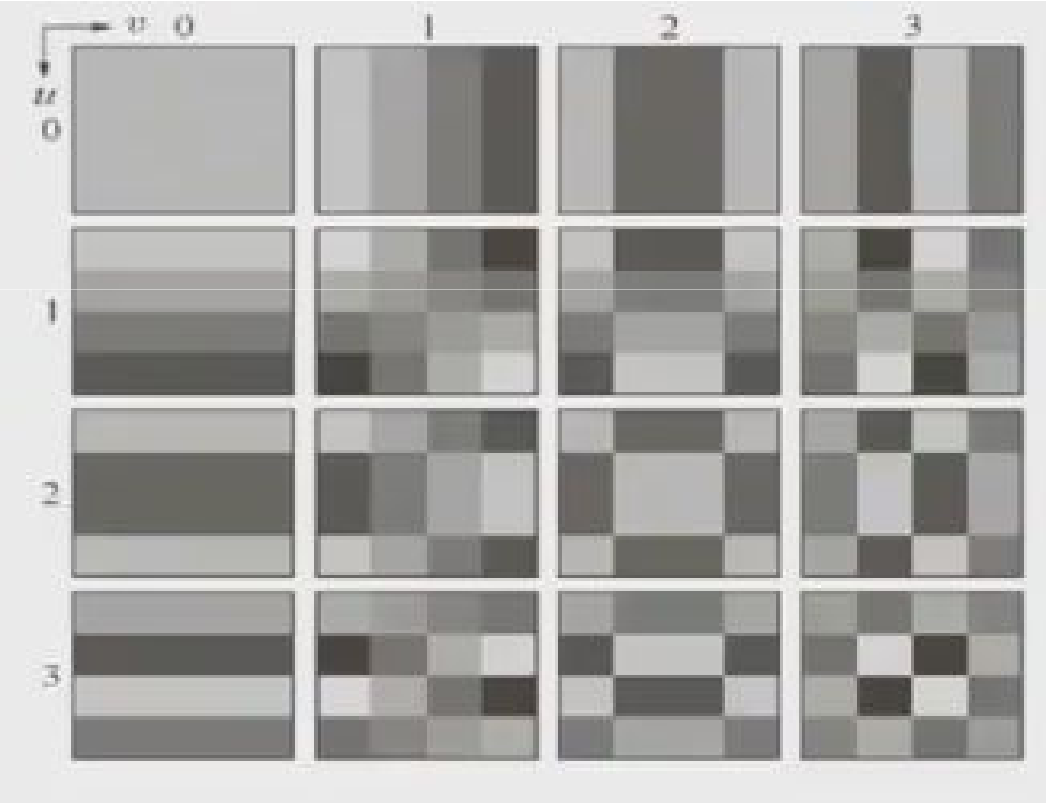
$$r(x,y,u,v) = s(x,y,u,v)$$

$$= \alpha(u)\alpha(v)\cos\left(\frac{(2x+1)u\pi}{2n}\right)\cos\left(\frac{(2y+1)v\pi}{2n}\right)$$

$$tq.: \alpha(u) = \begin{cases} \sqrt{1/n} \_\_ si\_u = 0 \\ \sqrt{2/n} \_\_ si\_u \neq 0 \end{cases}$$

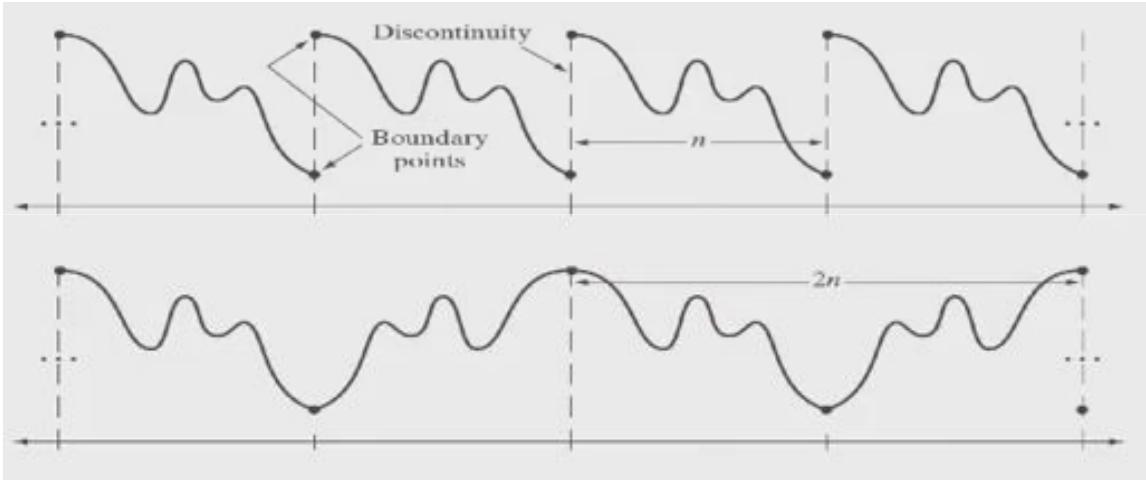
DCT: fonctionnement

- Exp.: Pour n = 4
- r et s sont tout simplement des fonctions de base de dé-corrélation
  - Combien on a de chaque composant de fréquence
    - T(3,3), T(1,2) , etc
- Chaque matrice nous donne une information sur la disposition des valeur dans l'image
  - Combien de chaque composant avons-nous?
- Pourquoi pas la TF ou autre?
  - Les pixels sont dépendants de leurs voisins (Images Markoviennes)
  - Dans ce cas, la DCT est quasi « Karhunen Loeve » !



Argument d'utilisation de la DCT

- La Transformée de Fourrier (TF) demande une contrainte de périodicité
  - Non logique pour les pixels lointains de 8px
- La DCT propose que le pixel juste proche est celui similaire (sorte de symétrie)



Choix de taille des blocs

- Nous avons pris 25% des coefficient de DCT et avons appliqué la reconstruction (tout le reste à 0 équivalent à du bruit)
- Meilleurs résultats avec les blocs 8x8
- On s'arréte à 8x8 (64px) pour des raisons de cout de calcul

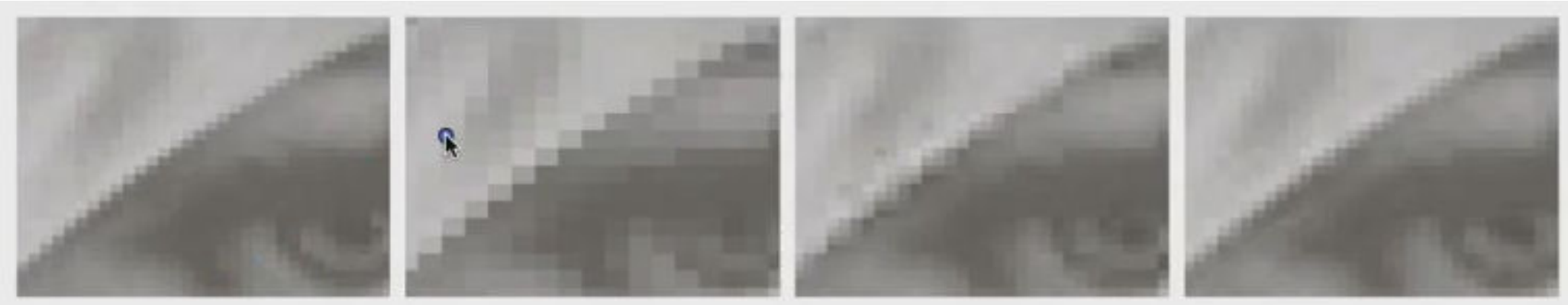
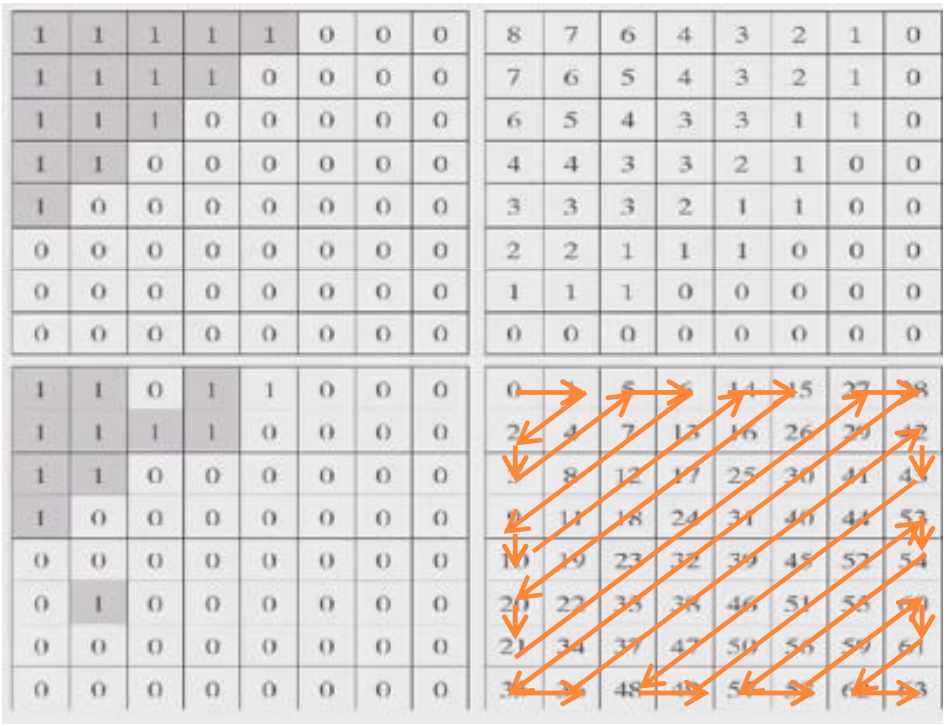


Image originale | DCT blocs 2x2 | DCT blocs 4x4 | DCT blocs 8x8

Quantification intelligente

- Avec les coefficients de la DCT:
- Prendre les coins selon un masque de sélection: le coin en haut droite par exp
- On peut aussi faire une quantification sur un nombre de bits bien choisit.
- **Quantification** pour avoir des coefficients qui apparaissent plus que d'autres
- Les coefficients de la quantification en zig-zag
- Isertion du signal « end » si tout le reste est zero
  - Gain énorme de données

$$\hat{F}(u,v) = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1}\hat{T}(x,y)s(x,y,u,v)$$



Quantification uniforme

- La quantification de plus en plus forte selon la position
- Favorise le codage de Huffman

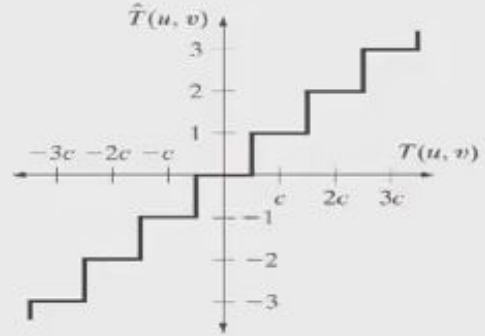
$$\hat{T}(0,0) = (T(0,0)/16) * 16$$

$$\Rightarrow 0..15 \approx 0$$

$$\Rightarrow 16..31 \approx 1$$

$$\Rightarrow 32..49 \approx 2$$

$$etc$$

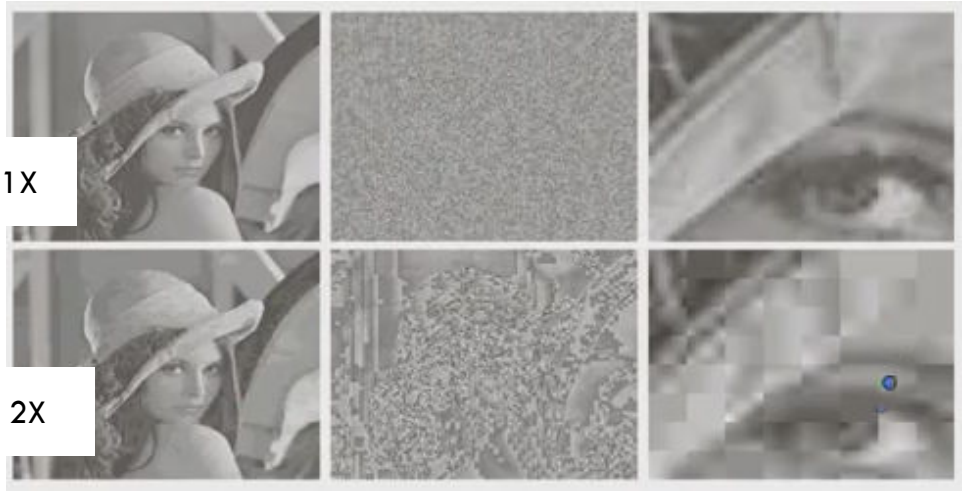


16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

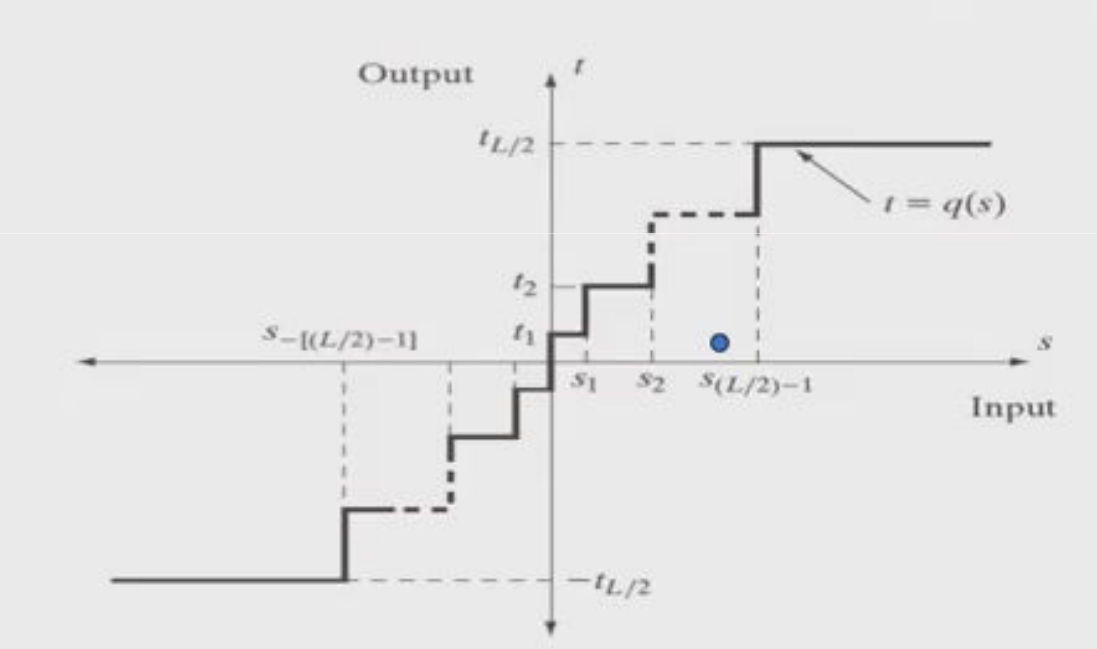
Qualité selon quantification

- Utilisation d'une matrice de quantification uniforme: un exemple:
- Plus le nombre par le quel on divise est grand
  - Plus on compresse
  - Plus on perd de qualité
  - Principe utilisé par Photoshop
- Si on quantifie par 100?
  - 0..99
    - ~0

52	55	61	66	70	61	64	73
63	59	66	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	63	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94



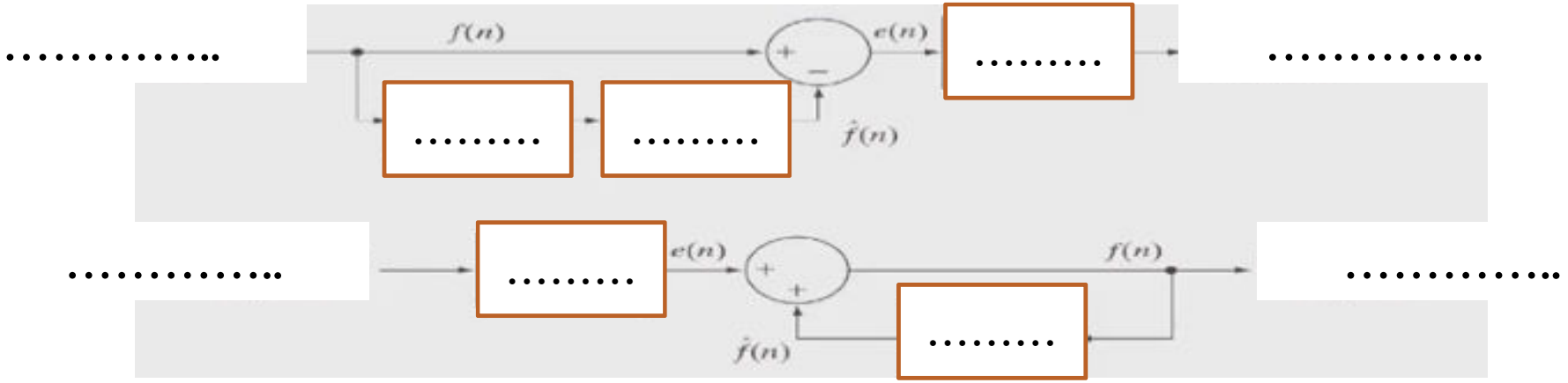
(NON JPEG) Quantification optimale



- Exp.: Max-Lloyd
  - Recherche du meilleur quantifieur possible
  - Quantification non uniforme
  - A condition de savoir la probabilité de distribution

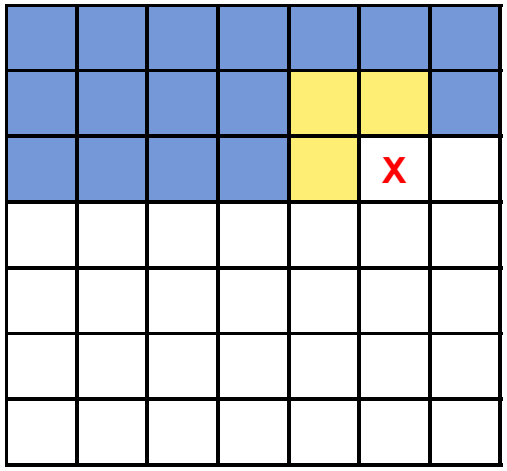
JPEG-LS

- Codage prédictif
- Compression prédictive sans pertes
- Prédire ce que j'ai à base du passé dans l'encodeur
- Avec une erreur rajoutée et encodée par l'encodeur (elle sera presque nulle)
- La même idée est faite dans le décodeur



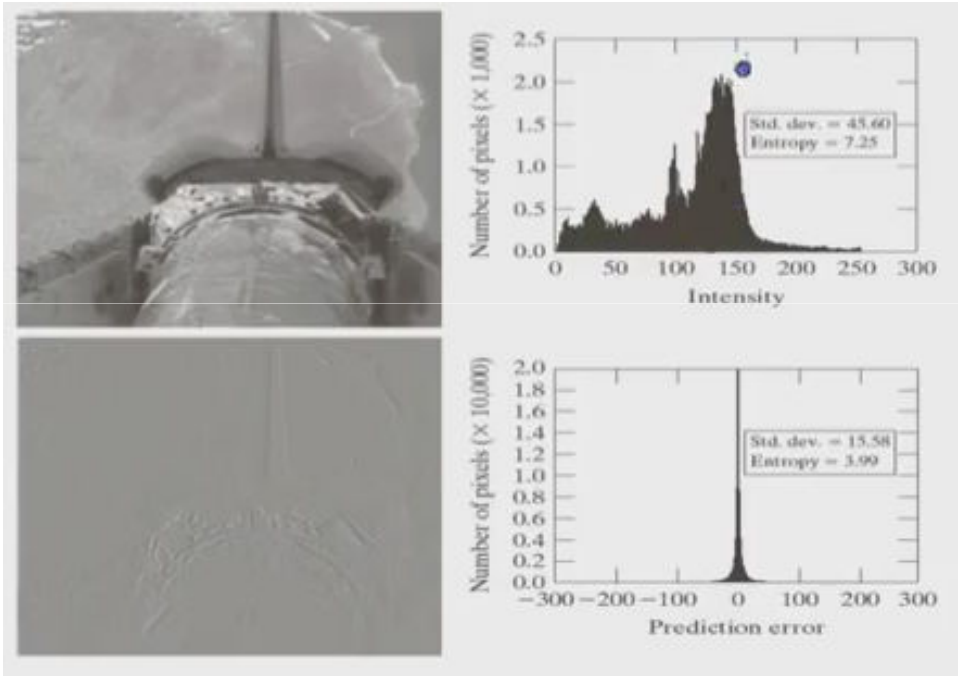
Exemple de prédicteurs

- Différence depuis le pixel précédent, celui en haut, en diagonale, …
- Utiliser les pixels voisins en haut (par exp)
- Moyenne des 2 pixels précédents
- L'erreur est concentrée autour de 0

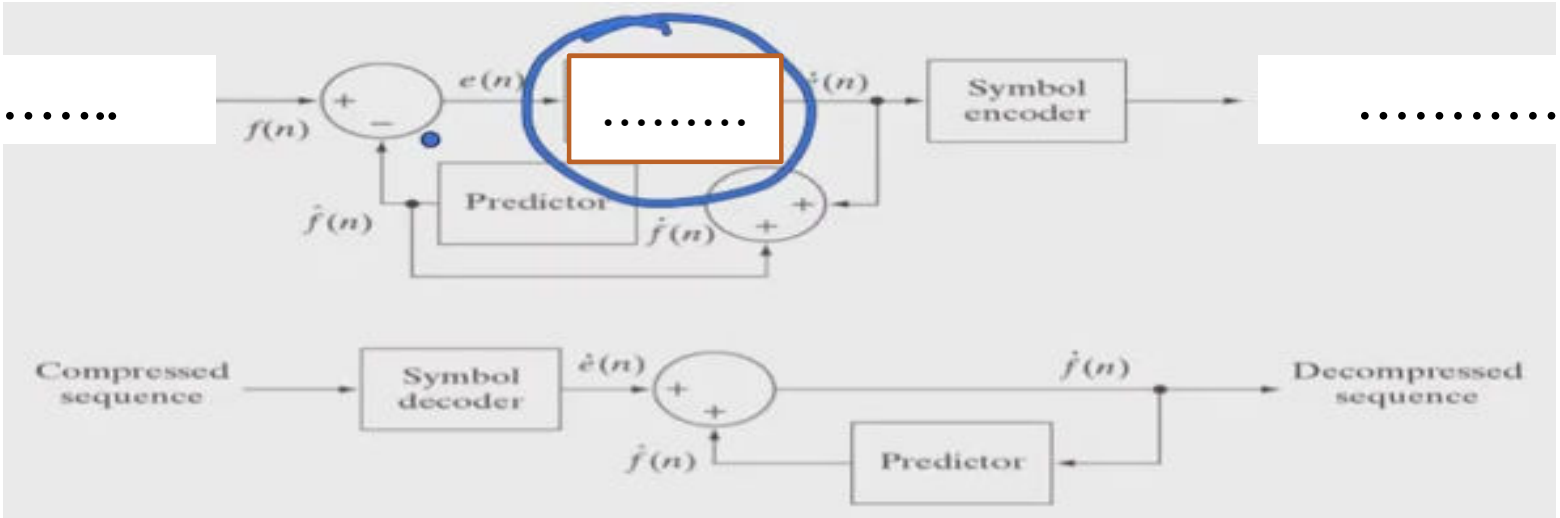


$$F(x-1,y)$$

$$e=F(x-1,y)-F(x,y)$$

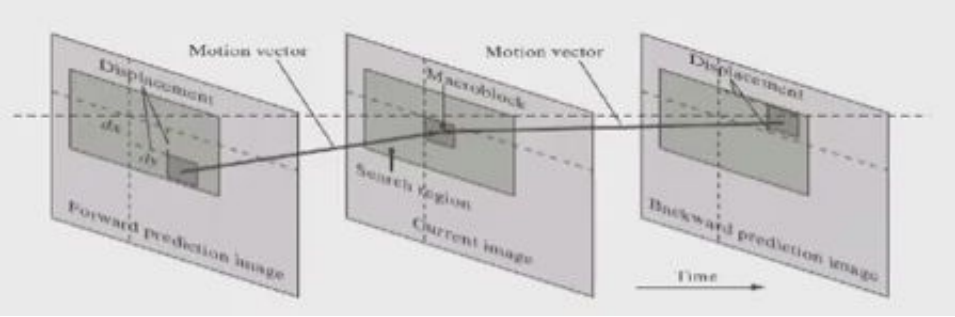


- On peut rajouter la quantification aux erreurs
  - Réduction des valeurs des erreurs



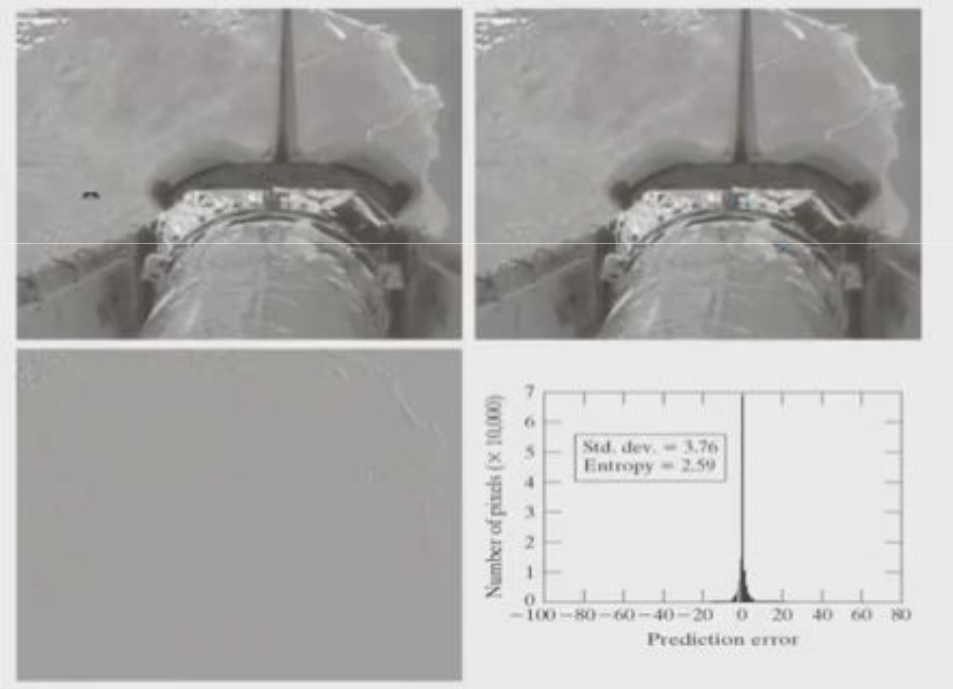
MPEG

- Standard en compression vidéo
- Compression temporelle
- On profite de l'information dans la frame précédente de la même région
- Réaliser la prévision dans le temps et encoder l'erreur de même façon que JPEG-LS

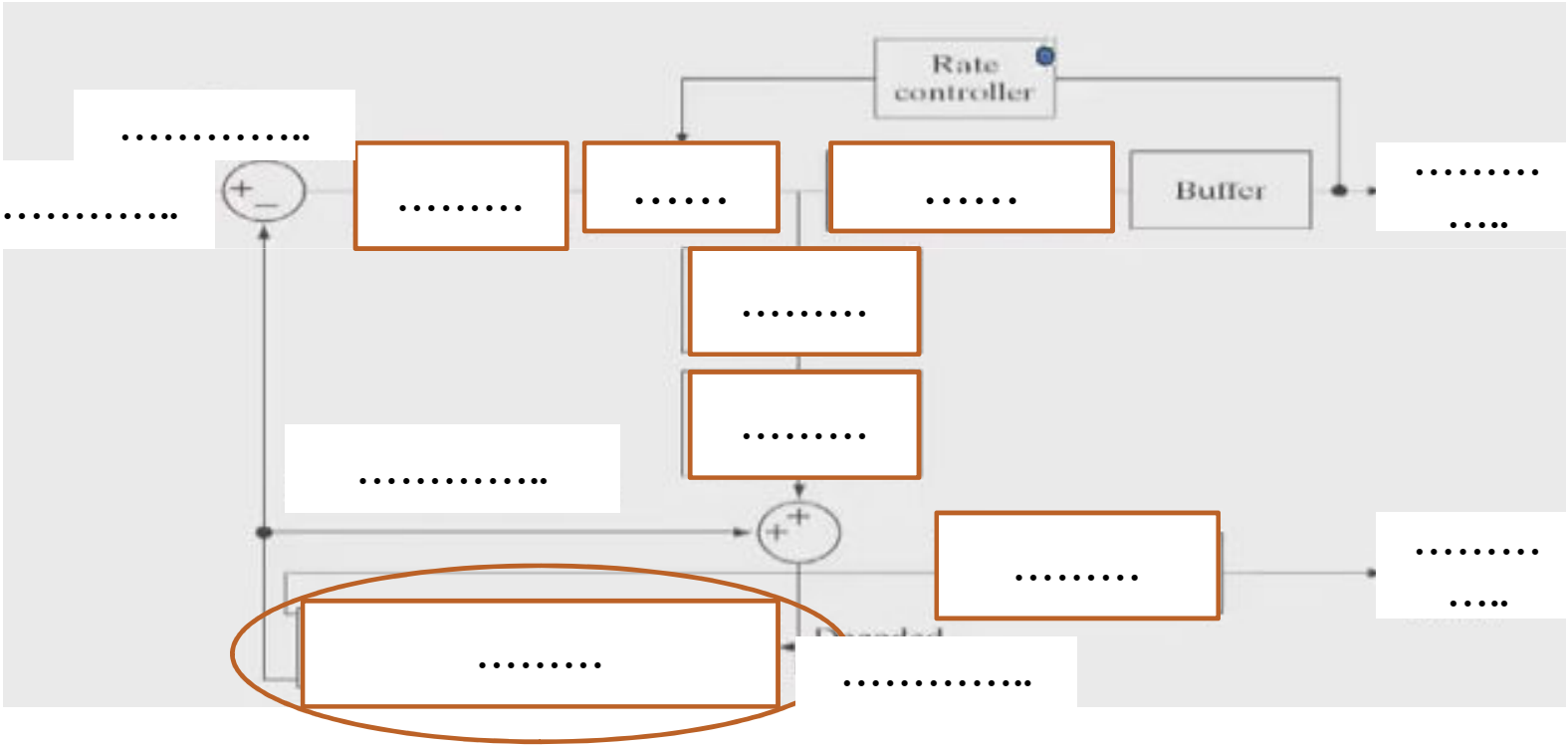


Exemple de prédiction

- La frame 2 est la même que celle précédente
- Erreur quasi nulle si on a pas beaucoup de mouvement



- Structure générale du codage MPEG
- En plus d'un estimateur de mouvement



Codage par plages

- Run Length Coding
- Efficace pour les images binaires
- On peut coder par deux valeurs uniques si on a un seul objet
  - Exp.: pour une image de 256 x 256 px
    - On a
      - 256;
      - …
      - 20, 50;
      - …