向量量化壓縮法

1

向量量化編碼 (Vector Quantization, VQ)

- 向量量化編碼法
  - 由Y. Linde, A. Buzo, and R. M. Gray 三位學者於1980年所提出
  - 將影像切割成一群大小是n×n的影像區塊
  - 利用具有代表性的少數向量來取代所有可能的影像區塊
- 範例
  - 給定八位元灰階影像區塊大小為4*4 pixels
  - 所有可能的影像區塊個數 256^{16}

2

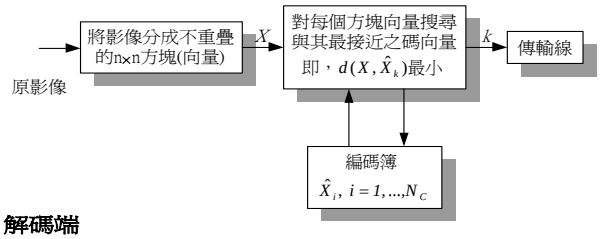
向量量化編碼 (Vector Quantization, VQ)

- 向量量化編碼法包含三個程序
  1. 編碼簿設計程序
     - 設計具有代表性的碼向量
     - 同時儲存於壓縮端與解壓縮端
  2. 影像壓縮程序
     - 為每一個影像區塊，找到編碼本中最近似的碼向量
     - 以碼向量的索引來編碼原本的影像區塊
  3. 影像解壓縮程序
     - 將個別索引對應的碼向量用來重建壓縮的影像區塊

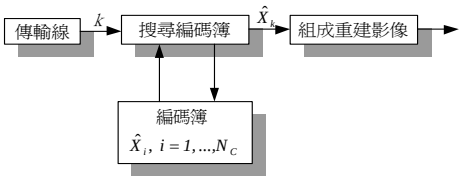
3

向量量化編碼 (Vector Quantization, VQ)

編碼端

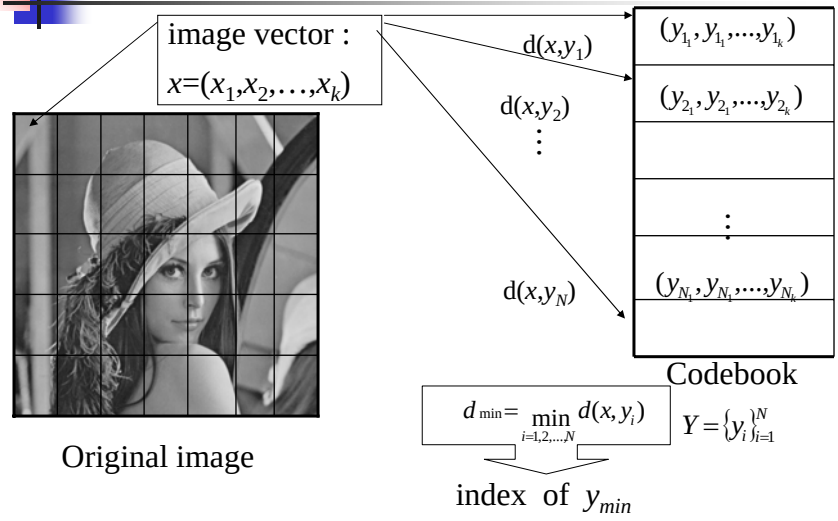


解碼端



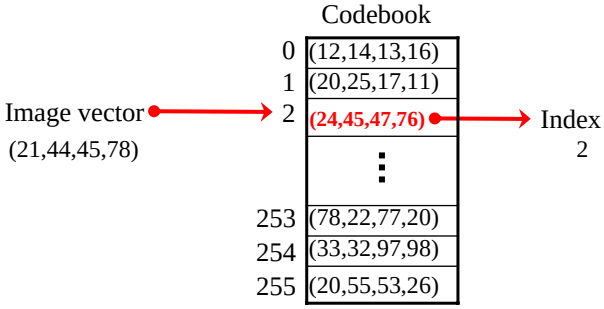
4

向量量化編碼之編碼程序

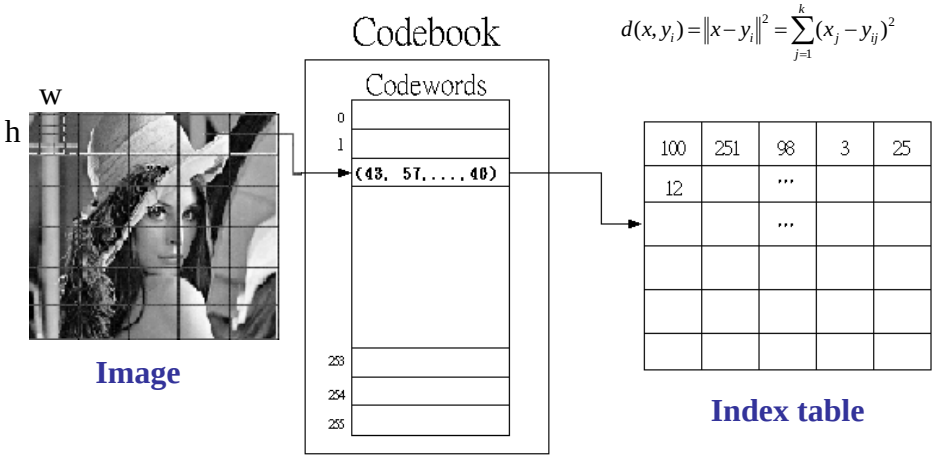


向量量化編碼之編碼程序

- Find Closest codeword

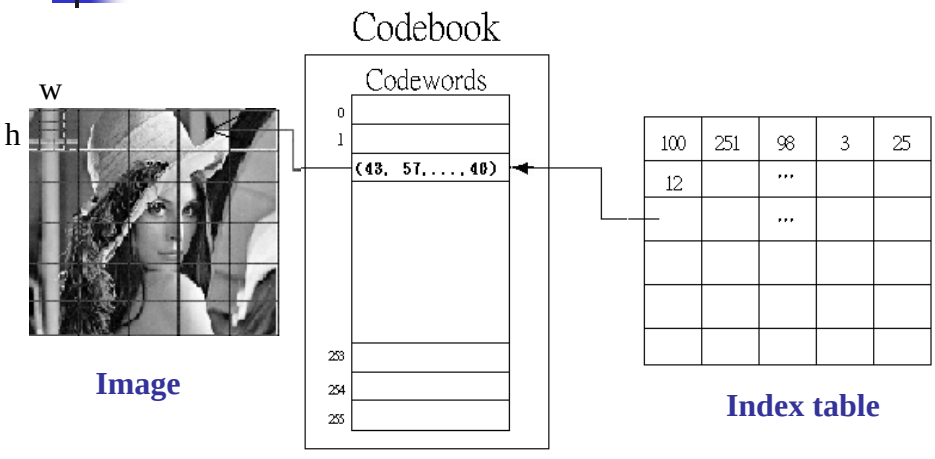


向量量化編碼之編碼程序



7

向量量化編碼之解碼程序



8

向量量化編碼 (Vector Quantization, VQ)

- 向量量化編碼法的優點
  - 具有簡單的解碼結構
    - 利用index 執行 table lookup （查表）的動作
  - 具有較低的位元率(bit rate)
    - 所需的位元率 $(\log_2 N_C)/k$ bits/pixel
    - 當使用256個碼向量並將影像區塊切割成 4 ×4,需要 8 bpp $(\log_2 256)/16$

9

向量量化編碼 (Vector Quantization, VQ)

- 向量量化編碼法的缺點
  - 編碼簿設計程序與影像壓縮程序需要大量時間成本
    - 找尋最近似碼向量需計算影像區塊與個別碼向量的距離
    - 從其中挑出距離最小的碼向量
    - 距離的計算相當耗時
  - 壓縮過的影像品質與編碼本的好壞息息相關
    - 如何設計具有代表性的編碼本？

10

11.1 編碼簿的產生

- 編碼簿的產生
  - 目的是產生一群具有代表性的碼向量 (codewords)來組成編碼簿 (codebook)
  - 相同的編碼簿會同時儲存在壓縮端與解壓縮端
  - 從文獻中我們知道 Linde-Buzo-Gray (LBG) 演算法是最常用來設計編碼簿的方法
  - 這個方法同時又被稱為k-means 演算法或是 Generalized Lloyd 演算法

11

Linde-Buzo-Gray (LBG) 演算法

LBG Algorithm

Let the codebook size be N_C and the training vectors be $\{ x(n) | n=1,\ldots,M\}$

Step 1: Let the initial codebook $C=\{ y(i) \mid i = 1,\ldots,N_C\}$ be randomly selected from $\{ x(j) | j = 1,\ldots,M\}$

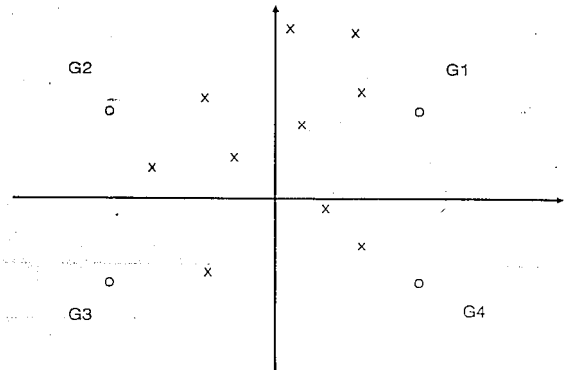
Step 2: Cluster the training vectors into N_C groups $G(i), i =1,\ldots,N_C$, where
$G(i)=\{x(j) \mid d(x(j), y(i)) < d(x(j), y(h)); h \neq i$ and $d(p, q)$ denotes the distance between p and $q\}$

Step 3: If the distortion decreases, then go to Step 4 ; otherwise stop

Step 4: New $y(i) = \frac{1}{|G(i)|} \sum_{x(j) \in G(j)} x(j)$, where $|G(i)|$ = the number of vector in $G(i)$; go to Step 2

12

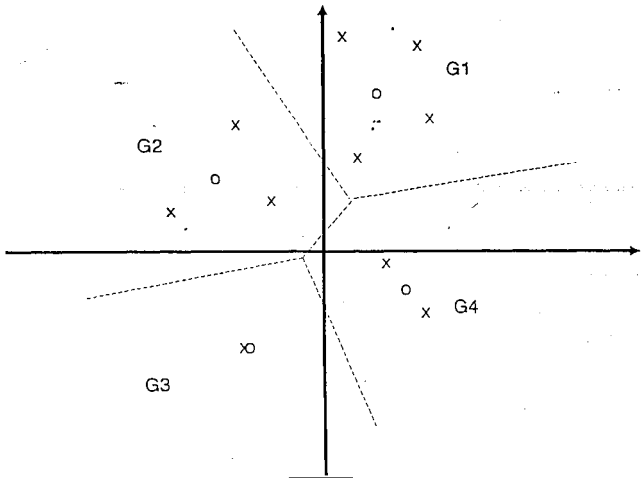
Linde-Buzo-Gray (LBG) 演算法



× = training codevectors
○ = codewords in the codebook
G*i* = region encoded into codeword *i*

13

Linde-Buzo-Gray (LBG) 演算法



14

Linde-Buzo-Gray (LBG) 演算法

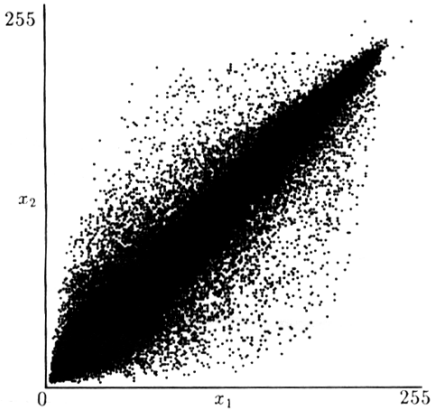
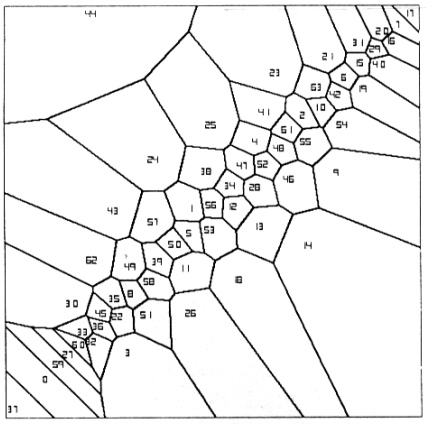


圖 *11.2* Lena 之*(a)*1 × 2 向量

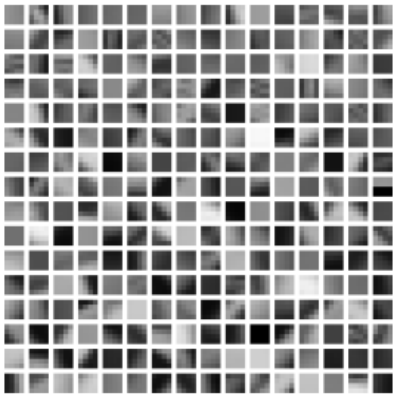


(b)N_c = 64 之決定範圍與碼向量。

15

Codebook Generation

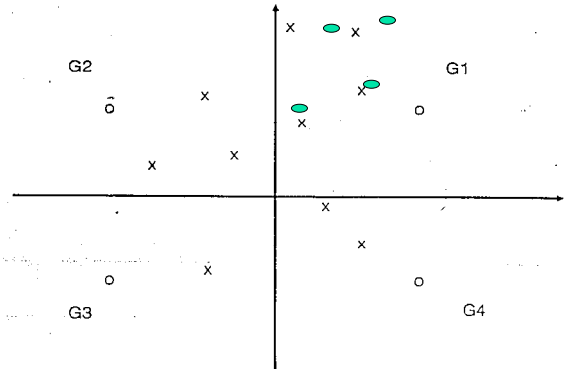
- A codebook example
  - Training set:
    - Lenna
    - Boots
  - codebook size 256
  - block size 4×4



16

編碼簿之起始

- 編碼簿的起始
  - LBG 演算法只能保證產生局部最佳化(local optimal)的編碼簿
  - 編碼簿的起始具有非常重要的影響



17

編碼簿之起始

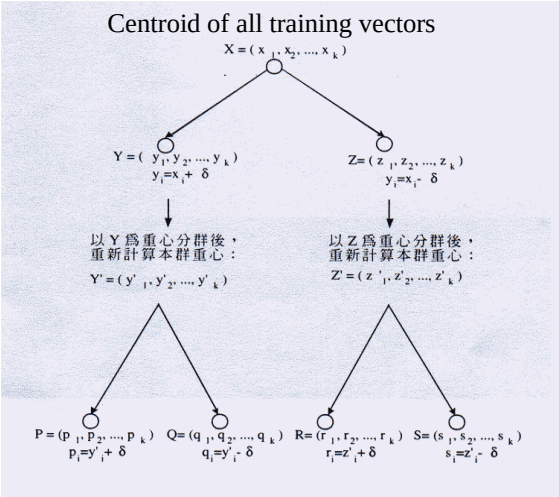
- 編碼簿的起始化
  - 亂碼 (random code)
    - 隨機挑選 *Nc* 個碼向量當作起始編碼簿
  - 分裂法 (split method)
    1. 先算出所有訓練向量的重心(平均值)
    2. 將目前的重心分裂成兩個新的向量
    3. 將新的向量當成叢聚的起始狀況來分類
    4. 將分群的重心算出
    5. 重複將2-4直到產生足夠數量的碼向量

18

編碼簿之起始-分裂法

δ = scale factor

Scale factor = 1.0



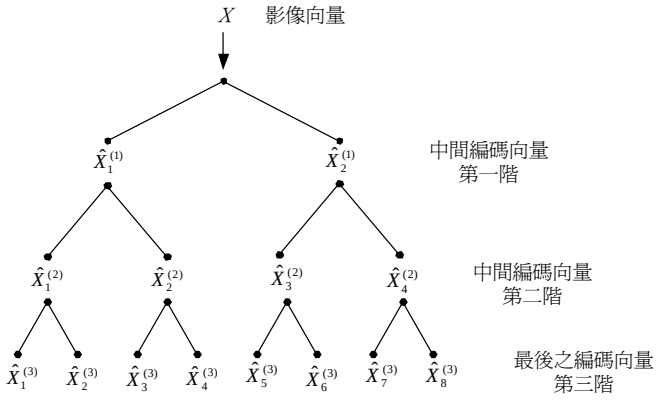
編碼簿之起始

- 編碼簿的起始化
  - 最近相鄰集結法 (pairwise nearest neighbor clustering)
    - 一開始將訓練集的每一個向量視為一個叢聚，假設共有N_t個叢聚。找出最接近的兩個向量將它們合併後，產生N_t-1個叢聚。
    - 重複這個步驟直到叢聚的數目等於編碼簿的大小Nc，所產生的這Nc個叢聚即可當成LBG演算法的起始編碼簿。

樹狀結構編碼簿設計

- 樹狀結構編碼簿(tree-structured codebook)
  - 利用樹狀結構來組織編碼簿
  - 能夠加速搜尋最鄰近碼向量的過程
    - 傳統編碼簿需要搜尋 N 個碼向量後才能決定哪一個碼向量與給定的輸入影像區塊 X 最近似
    - 樹狀結構編碼簿只需要搜尋 $\log_m N$ 個碼向量就能決定最鄰近的碼向量，其中 m 為樹狀結構的分支數
  - 需要較多的儲存空間
    - 除了樹葉節點(實際上的用來編碼的碼向量)要儲存之外，必須儲存內部節點
  - 利用樹狀結構編碼簿所得到的影像品質在相同位元率(bit rate)的情況下，較傳統向量量化編碼簿的影像品質來的差。

樹狀結構編碼簿設計



樹狀結構編碼簿設計

- 樹狀結構編碼簿設計策略
  - 不同的節點具有不同數目的分支
    - 變尖樹 (tapered tree)：節點的分支數越下層越多
    - 減枝樹 (pruned tree)：由一顆大樹開始，逐步砍掉部分分支

技　術	每個節點之分枝	節點數	計算量	記憶空間
二元樹	2	6	$O(12n)$	126n
四元樹	4	3	$O(12n)$	84n
八元樹	8	2	$O(16n)$	72n
完全搜尋	64	1	$O(64n)$	64n

表11.1 $N_c = 64$時不同的VQ樹之比較。

23

編碼簿的設計:乘碼

- 對於一個固定位元率 R= ($\log_2 N/k$) ，VQ的效率會隨著維度 k (dimension)的增加而改進；但是隨著維度的增加，編碼簿的大小以及搜尋的複雜度會快速的成長。
- 解決方案
  - 使用相乘結構(product structure)的編碼簿
  - 基本概念：如果一個向量可以用獨立的特性來描述，例如說方向與長度，那麼我們就可以針對每一個特性分別為它設計一個編碼簿。
    - 假設我們分別用兩個編碼簿來編碼向量的方向與長度，且編碼簿的大小分別為N_1與N_2：
      - 能夠使用的碼向量個數為 $N_1 \times N_2$
      - 儲存空間與計算量為 $N_1 + N_2$

24

平均值/餘值VQ (M/RVQ)

- 平均值/餘值VQ (M/RVQ)
  - 利用一些有限的資料來預測原影像的值，然後將預測值與原影像的值相減得到餘值影像。
    - 用來作預測的資料是採用一般的純量量化器(scalar quantizer)作編碼
    - 剩餘值則是利用向量量化器做編碼
  - 動機
    - 許多影像區塊都是在不同的平均值附近展現類似的方差
    - 只要將平均值去掉，基本上我們要用來表示餘值向量的碼向量就會比較少

25

平均值/餘值VQ (M/RVQ)

- 平均值/餘值VQ (M/RVQ)
  - 做法
    - 預測影像區塊的像素值是將該影像區塊的平均值重複拷貝
    - 餘值影像區塊的計算是將原本影像區塊的個別像素值減去預測影像區塊的像素值

26

平均值/餘值VQ (M/RVQ)

第一步：將原影像切割成不重疊的方塊，其大小爲 n(一般$n=4\times4=16$)，以形成影像向量；計算每一個方塊的平均值；

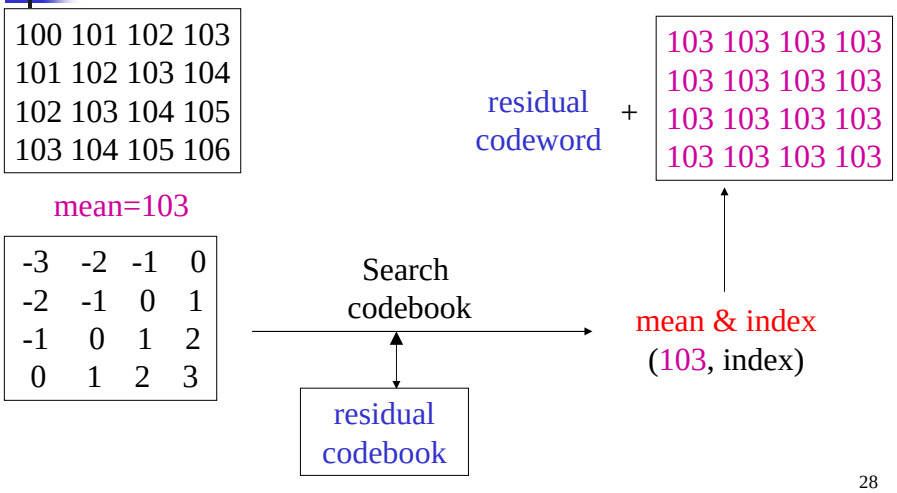
第二步：以純量量化器編碼平均值(一般使用 8 個位元)並送出給接收端；也可以用傳統的壓縮法，如DPCM，做平均值的編碼以更進一步降低位元率。

第三步：將該方塊之原影像向量減去量化後之平均值得到平均約等於0的餘值；

第四步：以VQ做餘值向量之量化，然後將最接近之餘值碼向量的指標送給接收端；

第五步：將平均值加回解碼所得之餘值向量得到重建方塊；

平均值/餘值VQ (M/RVQ)



內插法/餘值VQ(I/RVQ)

- 內插法/餘值VQ(I/RVQ)
  - 藉由原影像之次取樣(subsampling)與內插(interpolating)得到預測影像，然後利用原影像減去預測影像得到餘值影像。
    - 利用次取樣而非平均值
    - 使用內插法而非簡單的重複拷貝

29

內插法/餘值VQ(I/RVQ)

第一步：以 ℓ : 1 的比例（一般 $\ell = 8$）爲原影像做次取樣得到 $N/\ell \times N/\ell$ 的次取樣影像（原影像解析度爲 $N \times N$）；每一個次取樣值以純量量化器予以量化並送出給接收端；

第二步：傳送端與接收端都藉由將量化後之次取樣影像以內插法擴張成 $N \times N$ 的預測影像；將原影像減去預測影像得到傳送端的餘值影像；

第三步：將餘值影像切割成不重疊的方塊以形成向量（一般大小爲 $n = 4 \times 4 = 16$）；

第四步：使用VQ爲餘值向量做量化，並送出最接近的餘值碼向量之指標給接收端；

第五步：將解碼所得之餘值碼向量加回次取樣／內插後之預測影像得到重建影像。

30

11.5 內插法/餘值VQ(I/RVQ)

120	120	124	124
122	122	126	126
124	124	128	128
126	126	130	130

次取樣

$l = 2$

120	124
124	128

原始影像

120	124
124	128

內插

120	122	124	126
123	124	126	126
124	126	128	130
126	128	130	126

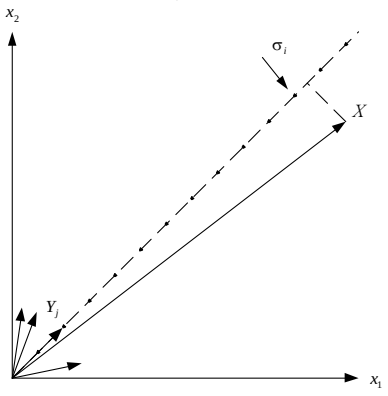
預測影像

31

11.6 增益/形狀VQ (G/SVQ)

- 增益/形狀VQ (G/SVQ)
  - 使用兩個編碼簿分別用以編碼向量的形狀(shape)和增益(gain)
    - 向量的增益(gain)指的是這個向量的能量或方差(variance)
    - 向量的形狀(shape)指的是去掉增益後的正常化向量(normalized vector)

$$x = \sigma_i \times Y_j$$



32

11.6 增益/形狀VQ (G/SVQ)

第一步：將原影像切割成不重疊的方塊形成影像向量，令其大小爲 n（一般取 $n = 4 \times 4 = 16$）；

第二步：從形狀編碼簿中找出和影像向量內積 *(inner product)* 所得值最大的單位能量形狀碼向量，Y_j;

第三步：給定所選擇的形狀碼向量 Y_j 後，找出純量的增益值 σ_i，其中純量 σ_i 使得重建向量 $\hat{X}=\sigma_i Y_j$, 與原向量 X 之間的失眞最小；圖 11.4 所示爲二維向量且只有四個形狀向量的情形；

第四步：送出形狀碼向量的指標及增益的碼給接收端；

第五步：將解碼所得之形狀碼向量與增益值相乘，即 $\hat{X}=\sigma_i Y_j$ 得到重建向量；

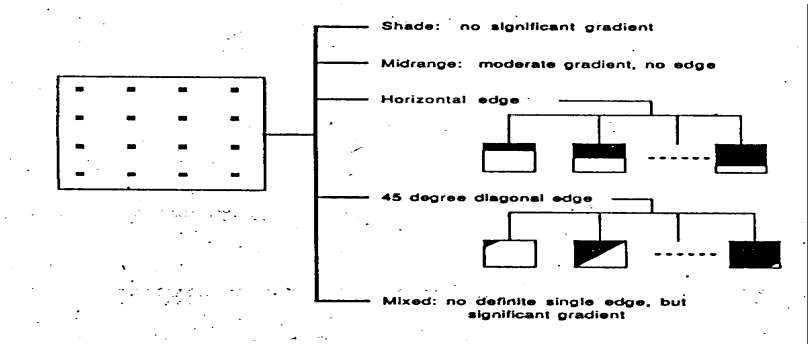
33

11.7 分類VQ (CVQ)

- 分類VQ(CVQ)
  - 採用多個編碼簿，每一個編碼簿專門用以編碼具有某一類特殊性質的影像區塊
  - 區塊特性的分類可由區塊分類器(block classifier)來辨識
  - 針對不同特性的影像區塊使用許多個小型的編碼簿，每一個小型的編碼簿專門針對某一類影像區塊來設計，可以達到和使用單一一個大型編碼簿相當的影像品質，但搜尋時間卻減少很多。

34

11.7 分類VQ (CVQ)



11.7 分類VQ (CVQ)

第一步：將原影像切割成大小爲 n（通常 $n = 4 \times 4 = 16$）且不相重疊的方塊，每一個方塊都經過一個以邊的方向來做分類的分類器將其歸類爲 M 類當中的一類；這些類別可能包括暗影像（*shade block*, 指沒什麼明顯梯度的方塊），中度範圍方塊（*midrange block*, 指具有中等梯度，但無明顯邊的方塊），水平邊方塊，垂直邊方塊，45° 或 135° 邊方塊，以及混合型方塊（有邊，但方向不清楚）；

第二步：每一個分類後的方塊向量各以其所屬之編碼簿做編碼；（編碼簿的大小可以各個不同，分別爲 $N_i, i = 1, 2, \cdots, M$，而且每一個方塊在選擇最接近的向量時也可以採用不同的失眞估算函數。整個碼向量數目爲 $N_c = \sum_{i}^{M} N_i$。

第三步：將所選出的碼向量指標送給接收端，接收端則用這些指標解碼；

11.8 有限狀態VQ (FSVQ)

- 有限狀態VQ (FSVQ)
  - 具有記憶特性的VQ，可以用一個有限狀態機器(finite state machine)來描述，其中每一個狀態代表一個分開的VQ編碼簿。
  - 使用好幾個小型的編碼簿，藉由一個下一狀態函數(next-state function) 來決定使用哪一個編碼簿
    - 下一狀態函數是一個以目前狀態(即編碼簿)以及目前輸出碼向量為輸入；而輸出值為另一個狀態的函數
    - $F(state_{i-1}, index)=state_i$

11.8 有限狀態VQ (FSVQ)

- 有限狀態VQ (FSVQ)
  - 動機
    - 由於相鄰的影像區塊通常很相似，因此可以藉由這種相關性或累贅，在知道前面影像區塊的結果後，幫我們選擇一個合適的編碼簿。

11.8 有限狀態VQ (FSVQ)

第一步：將原影像切割成大小爲 n(通常 $n = 4 \times 4 = 16$) 而且不相重疊的方塊。這些方塊排順序成爲一串影像向量，$X_i, i = 0, 1, \cdots, (N \times N) / n-1$ ；

第二步：給定一個起始狀態 s_0 及其連帶之編碼簿 c_{s_0}，我們首先爲第一個影像向量，X_0 ，編碼，找出 c_{s_0} 中和它最接近的碼向量，$\hat{X}_0$ ，送出 $\hat{X}_0$ 的指標給接收端；

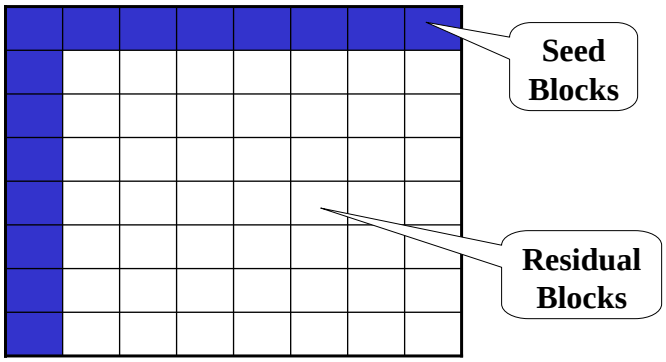
第三步：以前一個狀態 s_0 、及前一個狀態的輸出碼向量 $\hat{X}_0$ 做爲下一狀態函數 $f(\cdot)$ 的輸入，求出下一個狀態 s_1 ，即 $s_1 = f(s_0, \hat{X}_0)$； 使用下一個狀態 s_1 的編碼簿 C_{s_1} 爲下一個影像向量 X_1 做編碼； 假設從 C_{s_1} 中所找得最接近的碼向量爲 $\hat{X}_1$ ，則送出 $\hat{X}_1$ 在 C_{s_1} 中的指標給接收端；

第四步：以同樣的程序爲其餘的影像向量做編碼 (即，求新的狀態 $s_{n+1} = f(s_n, \hat{X}_n)$ ，然後從 $C_{s_{n+1}}$ 中找出與 X_{n+1} 最接近的碼向量 $\hat{X}_{n+1}$ 並送出其指標給接收端)。

39

11.8 有限狀態VQ (FSVQ)

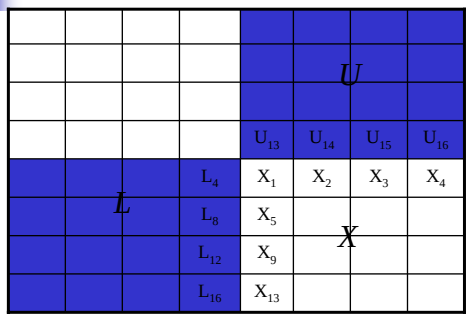
- Side Match Vector Quantization (SMVQ)



40

11.8 Side Match Vector Quantization (SMVQ)

Codebook $Y = \{y_i\}_{i=1}^{N}$



$(y_{1_1}, y_{1_1}, \ldots, y_{1_k})$
$(y_{2_1}, y_{2_1}, \ldots, y_{2_k})$
⋮
$(y_{N_1}, y_{N_1}, \ldots, y_{N_k})$

Side Match Distortion =

$$\{ \quad (U_{13} - y_{i1})^2 + (U_{14} - y_{i2})^2 + (U_{15} - y_{i3})^2 + (U_{16} - y_{i4})^2$$
$$+ (L_4 - y_{i1})^2 \; + (L_8 - y_{i5})^2 + (L_{12} - Y_{i9})^2 \; + (L_{16} - y_{i13})^2 \quad \}$$

41

11.8 Side Match Vector Quantization (SMVQ)

- Encoder
  - Select N_f smallest side match distortion from the super codebook to form the state codebook.
  - Search the state codebook to find minimum distortion codeword.
  - Encode the input vector with index in the state codebook.
- Decoder
  - Search out the same state codebook.
  - Use the index to recover the image.

42

11.8 Variable Rate SMVQ

- The state codebook size is not fixed.
  - var(U) <= TH_c and var(L) <= TH_c ➔ size = N_s
  - var(U) > TH_c and var(L) <= TH_c ➔ size = N_s
  - var(U) <= TH_c and var(L) > TH_c ➔ size = $2N_s$
  - var(U) > TH_c and var(L) > TH_c ➔ size = $4N_s$

43

Results

		LENA		BOOTS	
技術	位元率 位元/像素	RMSE (0-255)	SNR (dB)	RMSE (0-255)	SNR (dB)
M/RTVQ (含)					
Level 0	0.50	11.52	26.90	19.59	22.29
Level 1	0.69	8.62	29.42	14.70	24.79
Level 2	0.87	6.60	31.74	11.46	26.95
Level 3	1.06	5.25	33.72	9.29	28.77
Level 4	1.25	4.27	35.51	7.40	30.75
Level 5	1.44	3.24	37.85	4.86	34.39
M/RTVQ (不含)					
Level 0	0.50	11.52	26.90	19.59	22.29
Level 1	0.69	8.75	29.29	14.72	24.77
Level 2	0.87	6.64	31.68	11.61	26.83
Level 3	1.06	5.37	33.53	9.61	28.47
Level 4	1.25	4.63	34.82	8.45	29.59
Level 5	1.44	4.24	35.58	7.85	30.24

44

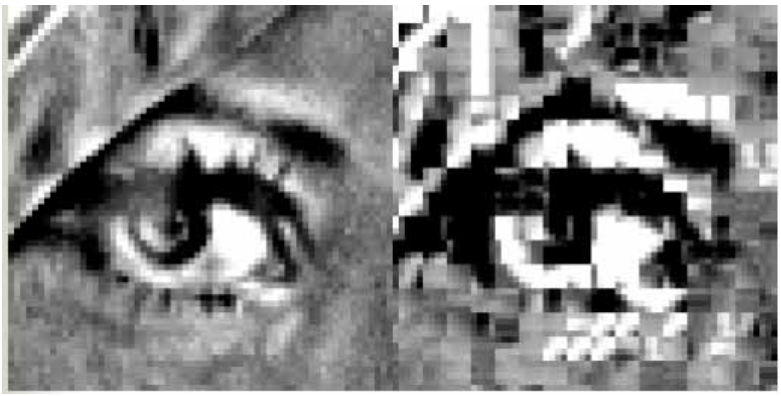
Results

		LENA		BOOTS	
技術	位元率 位元/像素	RMSE (0-255)	SNR (dB)	RMSE (0-255)	SNR (dB)
I/RTVQ (含)					
Level 0	0.13	16.21	23.94	27.62	19.31
Level 1	0.31	10.68	27.56	18.18	22.94
Level 2	0.50	8.05	30.02	14.09	25.15
Level 3	0.69	6.41	31.99	11.39	27.00
Level 4	0.87	5.19	33.83	9.06	28.99
Level 5	1.06	3.69	36.18	5.99	32.58
I/RTVQ (不含)					
Level 0	0.13	16.21	23.94	27.62	19.31
Level 1	0.31	10.75	27.50	18.09	22.98
Level 2	0.50	8.11	29.95	14.14	25.12
Level 3	0.69	6.53	31.83	11.69	26.78
Level 4	0.87	5.65	33.09	10.25	27.92
Level 5	1.06	5.23	33.77	9.57	28.52

45

Results

- I/RTVQ at 0.5 bpp for residual image



46