



National University of Sciences and Technology (NUST)

School of Electrical Engineering and Computer Science
(SEECS)

Digital Image Processing

Image Compression

- Storage needed for a two-hour standard television movie (Color)
 - Image size = 720 x 480 pixels
 - Frame rate = 30 fps (frame per seconds)

$$30 \frac{\text{frames}}{\text{sec}} \times (720 \times 480) \frac{\text{pixels}}{\text{frame}} \times 3 \frac{\text{bytes}}{\text{pixel}} = 31,104,000 \text{ bytes/sec}$$

For 2 hour movie

$$31,104,000 \frac{\text{bytes}}{\text{sec}} \times (60^2) \frac{\text{sec}}{\text{hr}} \times 2 \text{ hrs} = 2.24 \times 10^{11} \text{ bytes} = 224 \text{ GB}$$

Principal objective

To minimize the number of bits required to represent an image

Applications

Transmission: Broadcast TV, remote sensing via satellite, military communications via aircraft, radar and sonar, teleconferencing, computer communications, ...

Storage: Educational and business documents, medical images (CT, MRI and digital radiology), motion pictures, satellite images, weather maps, geological surveys, ...

- Image data compression methods fall into two common categories:
- Information preserving compression
 - Especially for image archiving (storage of legal or medical records)
 - Compress and decompress images **without losing information**
- Lossy image compression
 - Provide higher levels of data reduction
 - **Result in a less than perfect reproduction of the original image**

Data vs. Information

- **Data** are the means to convey **information**; various amounts of data may be used to represent the same amount of information
- Part of data may provide no relevant information: **data redundancy**
- **Probability and Information**
- **Compression**
 - Reducing the amount of data to represent a given quantity of information



Relative Data Redundancy

- Let b and b' refer to amounts of data in two data sets that carry the same information

$$\text{Compression Ratio } (C) = \frac{b}{b'}$$

$$\text{Relative data redundancy } (R) = 1 - \frac{1}{C}$$

of the first dataset b

- if $b = b'$, $C = 1$ and $R = 0$, relative to the second data set, the first set contains no redundant data
- if $b \gg b'$, $C \rightarrow \infty$ and $R \rightarrow 1$, relative to the second data set, the first set contains highly redundant data
- if $b \ll b'$, $C \rightarrow 0$ and $R \rightarrow -\infty$, relative to the second data set, the first set is highly compressed

$C = 10$ means 90% of the data in the first data set is redundant

- Image compression techniques can be designed for reducing or eliminating the **data redundancy**
- **Three basic data redundancies**
 - Spatial and Temporal redundancy
 - Coding redundancy
 - Irrelevant information

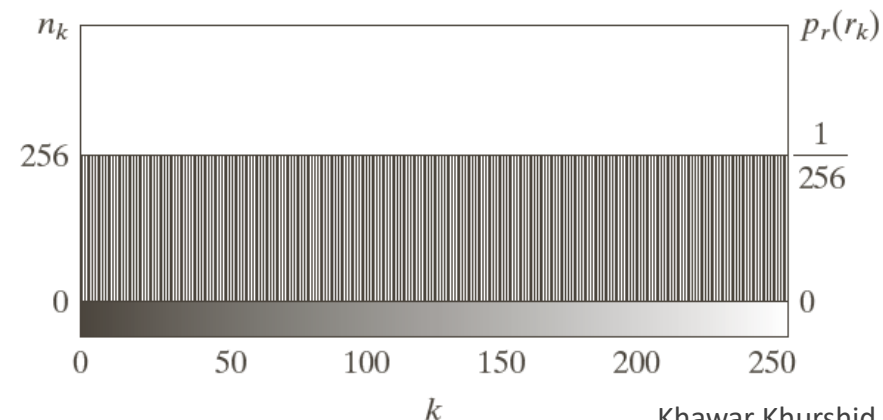
Spatial Redundancy

- Image features
 - All 256 gray levels are equally probable → uniform histogram (variable length coding can not be applied)
 - The gray levels of each line are selected randomly so pixels are independent of one another in vertical direction
 - Pixels along each line are identical, they are completely dependent on one another in horizontal direction

Spatial redundancy



**A computer generated
(synthetic) 8-bit image
 $M = N = 256$**



- The spatial redundancy can be eliminated by using *run-length pairs (a mapping scheme)*
- **Run length pairs** has two parts
 - Start of new intensity
 - Number of consecutive pixels having that intensity
- **Example** (consider the image shown in previous slide)
 - Each 256 pixel line of the original image is replaced by a single 8-bit intensity value
 - Length of consecutive pixels having the same intensity = 256
 - Compression Ratio =

$$\frac{256 \times 256 \times 8}{[256 + 256] \times 8} = 128$$

- A natural m-bit coding method assigns m-bit to each gray level without considering the probability that gray level occurs
→ very likely to contain **coding redundancy**
- **Basic concept?**
 - Utilize the probability of occurrence of each gray level (**histogram**) to determine length of code representing that particular gray level:
variable-length coding
 - Assign shorter code words to the gray levels that occur most frequently or vice versa

Let $0 \leq r_k \leq 1$: Gray levels (discrete random variable)

$p_r(r_k)$: Propability of occurrence of r_k

n_k : Frequency of gray level r_k

n : Total number of pixels in the image

L : Total number of gray level

$l(r_k)$: Number of bits used to represent r_k

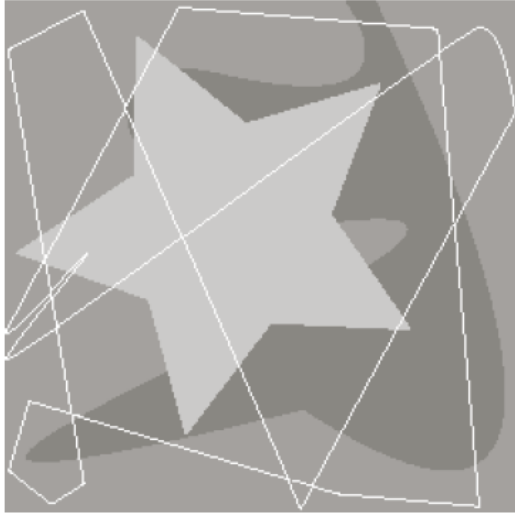
L_{avg} : Average length of code words assigned to gray levels

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k) \text{ where } p_r(r_k) = \frac{n_k}{n}, k = 0, 1, 2, \dots, L-1$$

Hence, the total number of bits required to code and $M \times N$ pixel image is MNL_{avg}

For a natural m-bit coding $L_{avg} = m$

Coding Redundancy - Example



A computer generated (synthetic) 8-bit image
 $M = N = 256$

- **Code 1:** Natural code ($m = 8$) is used,
 $L_{\text{avg}} = 8 \text{ bits}$

- **Code 2:** Variable length code
 $L_{\text{avg}} = (0.25)2 + 0.47(1) + 0.25(3) + 0.03(3)$
 $= 1.81 \text{ bits}$

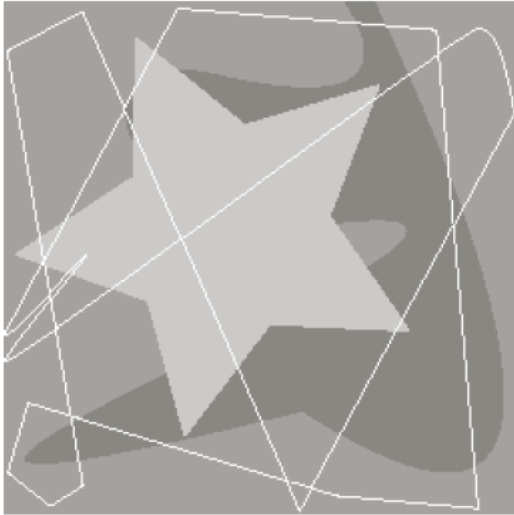
- Compression Ratio =

$$\frac{256 \times 256 \times 8}{256 \times 256 \times 1.81} = 4.42$$

- $R = 1 - 1/4.42 = 0.774$

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
$r_k \text{ for } k \neq 87, 128, 186, 255$	0	—	8	—	0

Coding Redundancy – Example



A computer generated
(synthetic) 8-bit image
 $M = N = 256$

- **Code 1:** Natural code ($m = 8$) is used,
 $L_{\text{avg}} = 8 \text{ bits}$
- **Code 2:** Variable length code
 $L_{\text{avg}} = (0.25)2 + 0.47(1) + 0.25(3) + 0.03(3)$
 $= 1.81 \text{ bits}$
- Compression Ratio =
$$\frac{256 \times 256 \times 8}{256 \times 256 \times 1.81} = 4.42$$
- $R = 1 - 1/4.42 = 0.774$

77.4% data in the image is redundant

Irrelevant Information

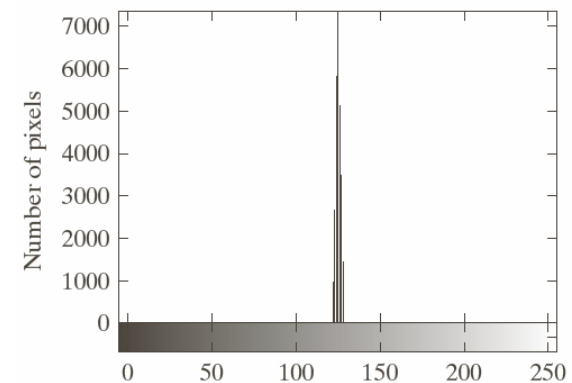
- The eye does not respond with equal sensitivity to all visual information
- Certain information has less relative importance than other information in normal visual processing
- The elimination of visually redundant data results in a loss of quantitative information → **lossy data compression method**
- **Quantization**



A computer generated (synthetic) 8-bit image

$$M = N = 256$$

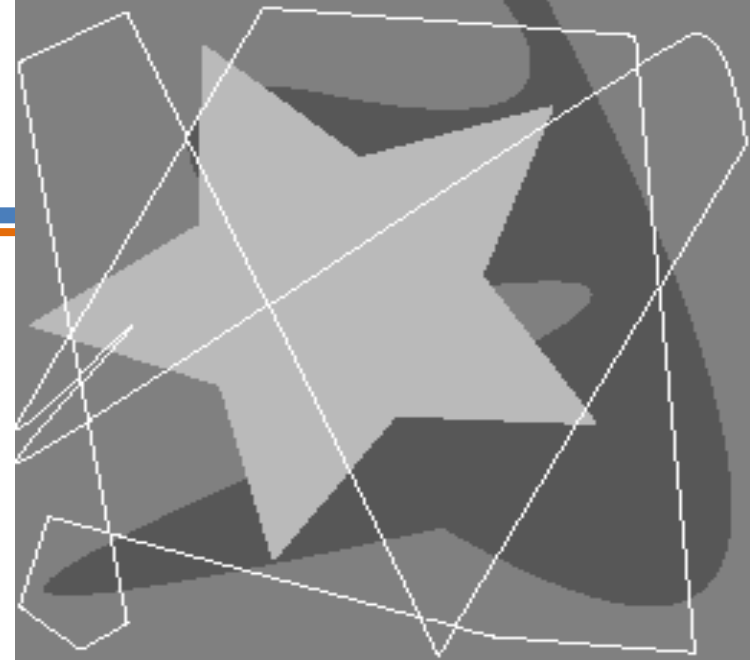
This image appears homogeneous so we can use its mean value to encode this image



Histogram of the image

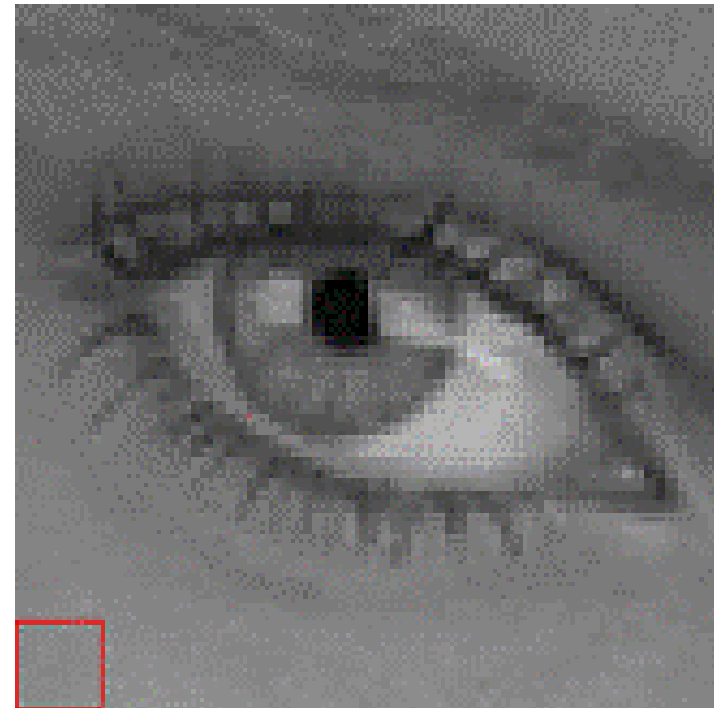
Redundancy

- Coding redundancy
 - Due to different occurrence rates
- Inter-pixel Redundancy
 - Spatial Redundancy
- Psycho-visual redundancy
 - Eye Response – Irrelevant Information



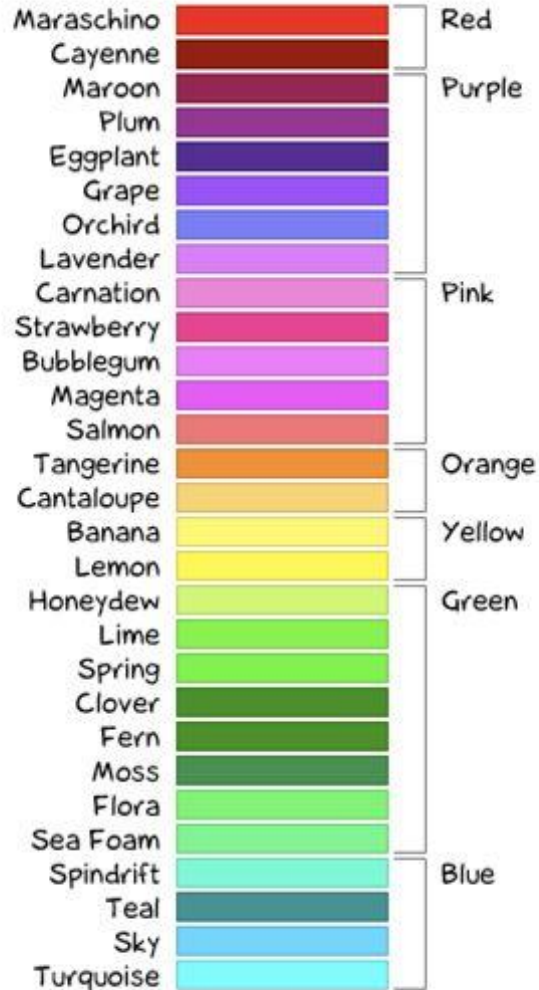
Original 39010 Colors Image

Quantized 16 Colors Image



Redundancy

Color names if
you're a girl...



Color names if
you're a guy...

Table 6.1 Variable-Length Coding Example

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_0 = 0$	0.19	000	3	11	2
$r_1 = 1/7$	0.25	001	3	01	2
$r_2 = 2/7$	0.21	010	3	10	2
$r_3 = 3/7$	0.16	011	3	001	3
$r_4 = 4/7$	0.08	100	3	0001	4
$r_5 = 5/7$	0.06	101	3	00001	5
$r_6 = 6/7$	0.03	110	3	000001	6
$r_7 = 1$	0.02	111	3	000000	6

- Compression Ratio?
- Relative Redundancy?

- Quantify the nature and extent of information loss
- Level of information loss can be expressed as a function of the original (input) and compressed-decompressed (output) image

Given an $M \times N$ image $f(x, y)$, its compressed-then-decompressed image $\hat{f}(x, y)$, then the error between corresponding values is given by

$$e(x, y) = \hat{f}(x, y) - f(x, y)$$

Total Error:

$$\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]$$

- Normally the objective fidelity criterion parameters are as follows:

Root mean square error:

$$e_{rms} = \left[\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]^2 \right]^{\frac{1}{2}}$$

Mean-square signal-to-noise ratio

$$SNR_{ms} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}(x, y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]^2}$$

Image Compression Models

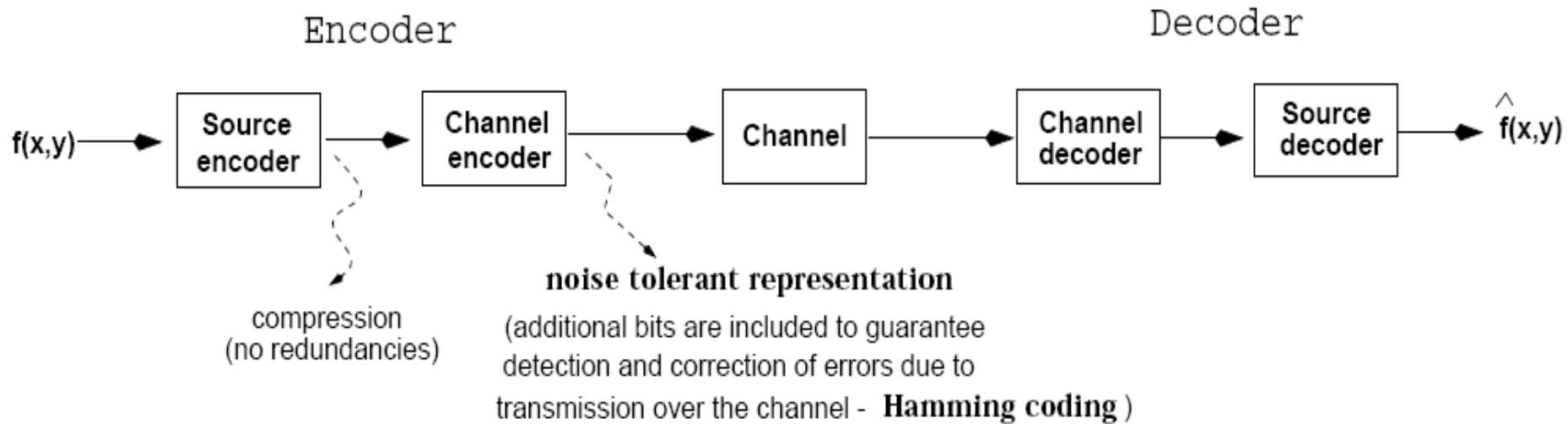
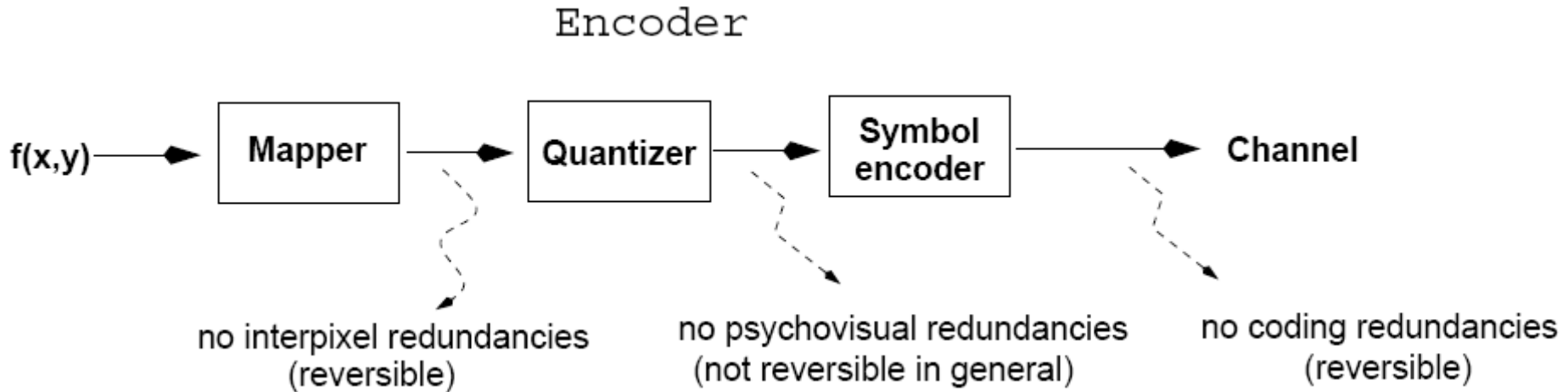


Image Compression Models



- **Mapper**: transforms the input data into a format that facilitates reduction of inter pixel redundancies.
- **Quantizer**: Compression of Lossy nature
- **Symbol encoder**: assigns the shortest code to the most frequently occurring output values.

Image Compression Models



- The inverse operations are performed.
- Quantization is irreversible in general.

- Run length coding
- Huffman coding
- Symbol-Based coding
- Bit-Plane coding
- Transform Coding
- Arithmetic coding
- LZW coding
- Predictive Coding

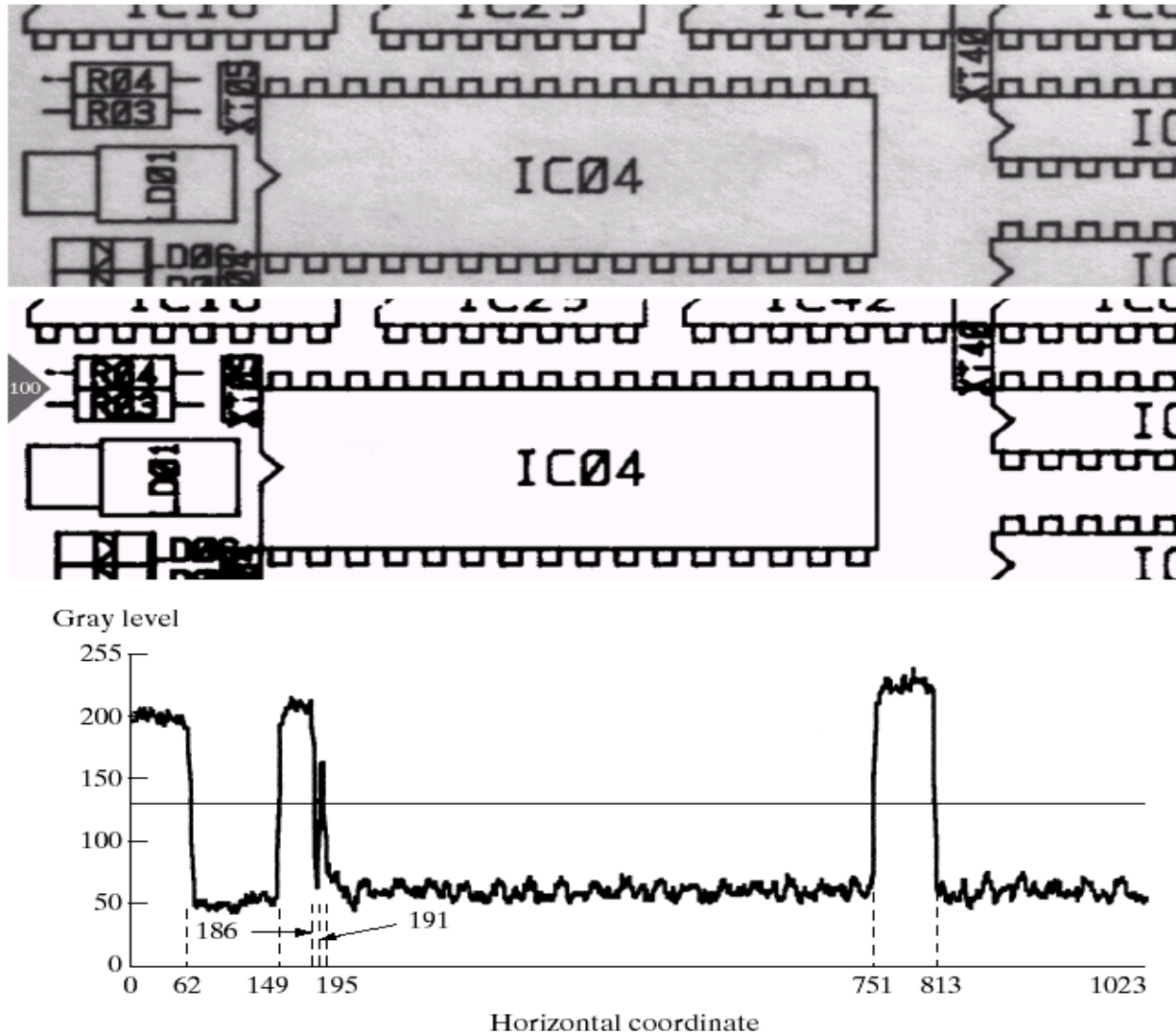
Run Length Coding

- Example:

0000011000001000000000101100000 (30 bits in a row)

- Assume each row begins with a white run (bit '1'):
 - *0525181125: length of the first (white) run is 0*

Run Length Coding



Initial 62
pixels are 1,
Next 87 pixels
are zero & so
on

- Variable length code
- Error-free compression technique
- Reduce only coding redundancy by minimizing the L_{avg} and assign shorter code words to the most probable gray levels

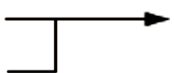
Huffman Coding

- Arrange the symbol probabilities p_i in a decreasing order; consider them (p_i) as leaf nodes of a tree
- While there are more than two nodes:
 - *Merge the two nodes with smallest probability to form a new node whose probability is the sum of the two merged nodes*
 - *Arrange the combined node according to its probability in the tree*
 - *Repeat until only two nodes are left*

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

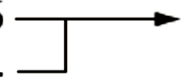
Huffman Coding

- Arrange the symbol probabilities p_i in a decreasing order; consider them (p_i) as leaf nodes of a tree
- While there are more than two nodes:
 - Merge the two nodes with smallest probability to form a new node whose probability is the sum of the two merged nodes
 - Arrange the combined node according to its probability in the tree
 - Repeat until only two nodes are left

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

Huffman Coding

- Arrange the symbol probabilities p_i in a decreasing order; consider them (p_i) as leaf nodes of a tree
- While there are more than two nodes:
 - Merge the two nodes with smallest probability to form a new node whose probability is the sum of the two merged nodes
 - Arrange the combined node according to its probability in the tree
 - Repeat until only two nodes are left

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

Huffman Coding

- Arrange the symbol probabilities p_i in a decreasing order; consider them (p_i) as leaf nodes of a tree
- While there are more than two nodes:
 - Merge the two nodes with smallest probability to form a new node whose probability is the sum of the two merged nodes
 - Arrange the combined node according to its probability in the tree
 - Repeat until only two nodes are left

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

Huffman Coding

- Arrange the symbol probabilities p_i in a decreasing order; consider them (p_i) as leaf nodes of a tree
- While there are more than two nodes:
 - Merge the two nodes with smallest probability to form a new node whose probability is the sum of the two merged nodes
 - Arrange the combined node according to its probability in the tree
 - Repeat until only two nodes are left

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	
a_6	0.3	0.3	0.3	0.3	
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				

Huffman Coding

- Arrange the symbol probabilities p_i in a decreasing order; consider them (p_i) as leaf nodes of a tree
- While there are more than two nodes:
 - Merge the two nodes with smallest probability to form a new node whose probability is the sum of the two merged nodes
 - Arrange the combined node according to its probability in the tree
 - Repeat until only two nodes are left

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

Huffman Coding

- Starting from the top, arbitrarily assign 1 and 0 to each pair of branches merging into a node
- Continue sequentially from the root node to the leaf node where the symbol is located to complete the coding

Original source			Source reduction			
Symbol	Probability	Code	1	2	3	4
a_2	0.4					0.6 0
a_6	0.3					0.4 1
a_1	0.1					
a_4	0.1					
a_3	0.06					
a_5	0.04					

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) = 2.2 \text{ bits/pixel}$$

Huffman Coding

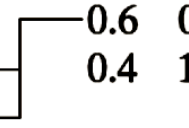
- Starting from the top, arbitrarily assign 1 and 0 to each pair of branches merging into a node
- Continue sequentially from the root node to the leaf node where the symbol is located to complete the coding

Original source			Source reduction			
Symbol	Probability	Code	1	2	3	4
a_2	0.4				0.4	1
a_6	0.3				0.3	00
a_1	0.1				0.3	01
a_4	0.1					
a_3	0.06					
a_5	0.04					

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) = 2.2 \text{ bits/pixel}$$

Huffman Coding

- Starting from the top, arbitrarily assign 1 and 0 to each pair of branches merging into a node
- Continue sequentially from the root node to the leaf node where the symbol is located to complete the coding

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

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) = 2.2 \text{ bits/pixel}$$

Huffman Coding

- Starting from the top, arbitrarily assign 1 and 0 to each pair of branches merging into a node
- Continue sequentially from the root node to the leaf node where the symbol is located to complete the coding

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

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) = 2.2 \text{ bits/pixel}$$

Huffman Coding

- Starting from the top, arbitrarily assign 1 and 0 to each pair of branches merging into a node
- Continue sequentially from the root node to the leaf node where the symbol is located to complete the coding

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				

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) = 2.2 \text{ bits/pixel}$$

- Consider the following encoded strings of code symbols

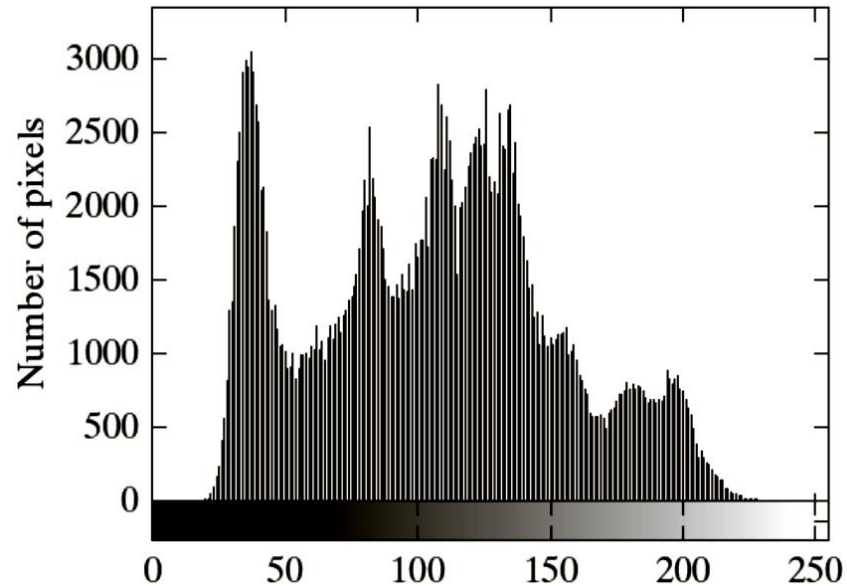
010100111100

- The sequence can be decoded by just examining the string from left to right

010100111100

$a_3 a_1 a_2 a_2 a_6$

Huffman Coding - Example



- Huffman code length = 7.428 bits/pixel
- Compression Ratio (C) = $8/7.428 = 1.077$
- Relative Redundancy (R) = $1 - 1/1.077 = 0.0715 = 7.15\%$

LZW – Lempel Ziv Welch

- Construct a “dictionary” containing the source symbols
- Examine the image’s pixel
- The gray sequences that are not in the dictionary are placed in algorithmically determined locations
- Coding dictionary or code book is created while the data are being encoded

LZW Encoding Algorithm

1. Initialize the dictionary to contain all blocks of length one
2. Search for the longest block **W** which has appeared in the dictionary.
3. Encode **W** by its index in the dictionary.
4. Add **W** followed by the first symbol of the next block to the dictionary.
5. Go to Step 2.

LZW Encoding – Images

- Images are scanned, A **codebook** or **dictionary** containing the source symbols to be coded is constructed on the fly
- For 8-bit monochrome images, first 256 words of the dictionary are assigned to intensities 0,1,2,3,...,255.

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

LZW Encoding – Images

- During the encoding process, intensity sequences that are not in the dictionary are placed in algorithmically determined locations e.g. a sequence of two-white pixels (255-255) may be assigned the location 256 in the dictionary

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

LZW Encoding – Images

- The next time that two consecutive white pixels are encountered, code word 256 (the address of the location containing 255-255) is used to represent them

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

LZW Encoding – Images

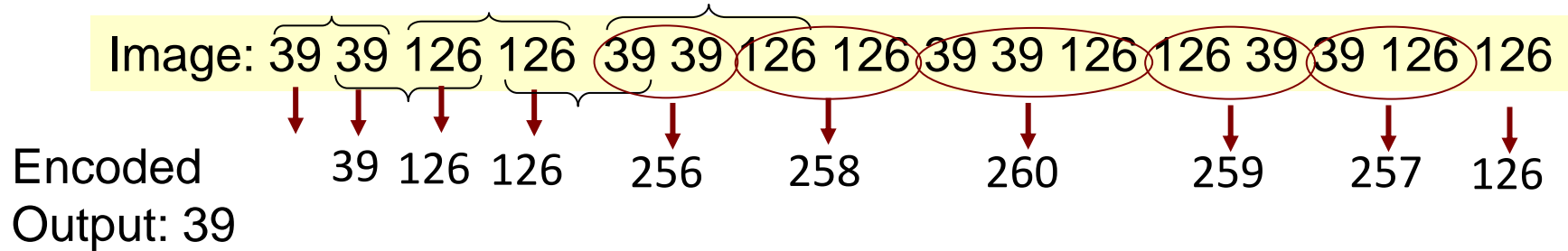
39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

- Image is scanned from left to right and from top to bottom OR consider it as a vector

Image: 39 39 126 126 39 39 126 126 39 39 126 126 39 39 126 126

Dictionary Location	Entry
0	0
1	1
...	...
255	255
256	-
257	-
...	-

LZW Encoding – Images



Dictionary Location	Entry
0	0
1	1
...	...
255	255
256	39 – 39
257	39 – 126
258	126 – 126
259	126 – 39
260	39 – 39 – 126
261	126 – 126 – 39
262	39 – 39 – 126 – 126
263	126 – 39 – 39
264	39 – 126 – 126

You do not need to send the entire table with the image. At the receiving end, you can create the table on the fly based on the encoded values received.

e.g. When two 39's are read, they are found to be a unique combination, therefore this combination is placed in the table at location 256.

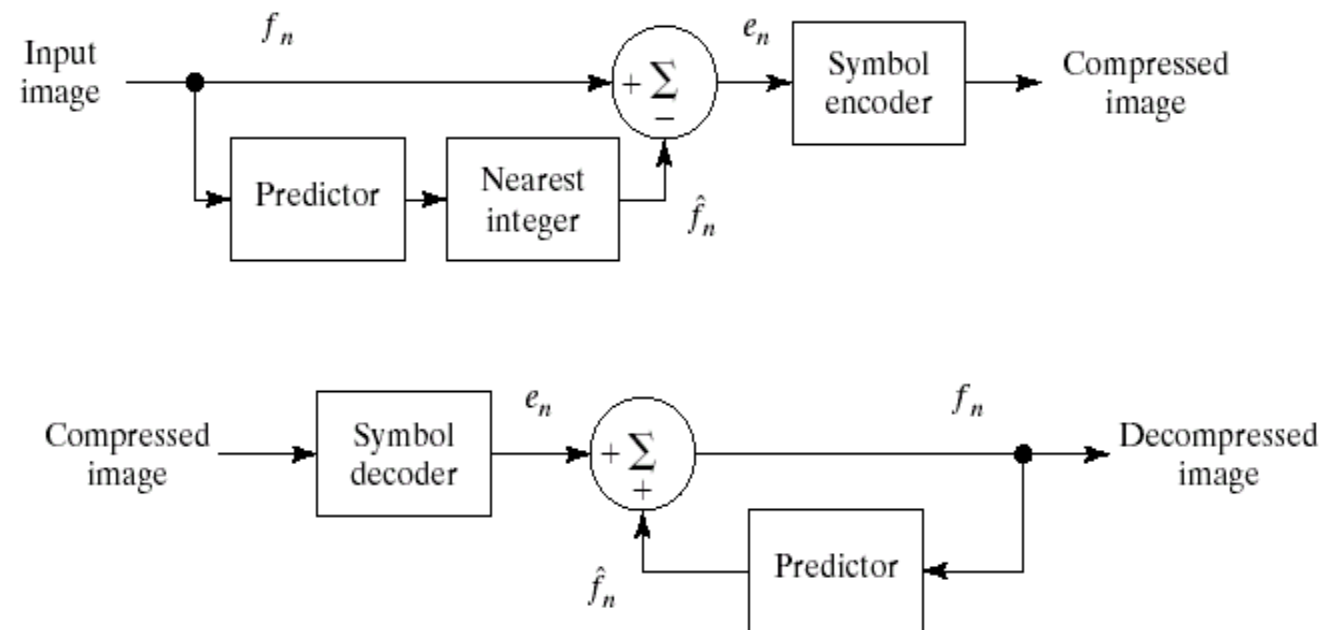
Similarly when 39-126 are decoded, this combination is placed at location 257, and so on.

When a value of 256 is read in the encoded stream, the receiving decoder has already placed the required combination at that location. So it will simply pick it from the table being created.

Note: We do need to tell the receiver that 0-255 are valid gray levels, and from 256 onwards are the table entries of different combinations.

Lossless Predictive Coding

- Eliminating Inter-Pixel Redundancy by prediction
- Predictor generates the anticipated value of each sample based on a specified number of past inputs
- Only the difference (or error) is transmitted
- Assuming good prediction; bits required for transmission of e_n can be significantly reduced via VLC



Simple possible predictors

$$\hat{f}_n = f_{n-1}$$

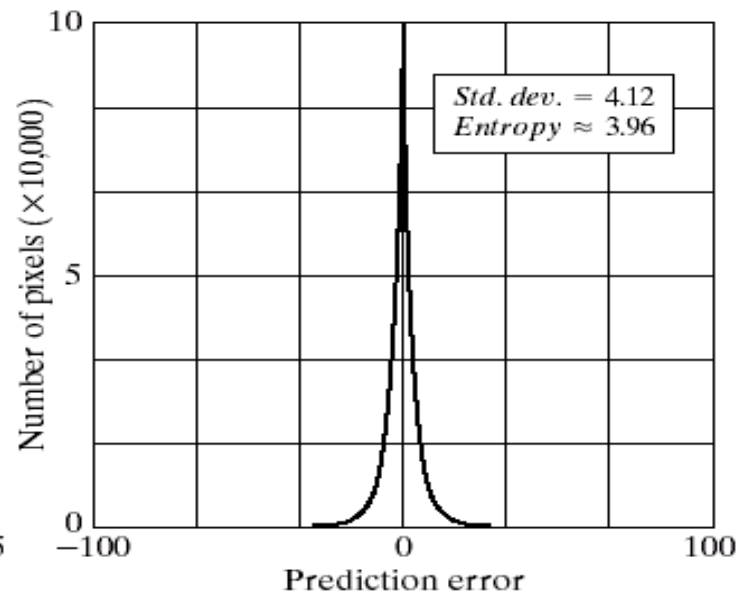
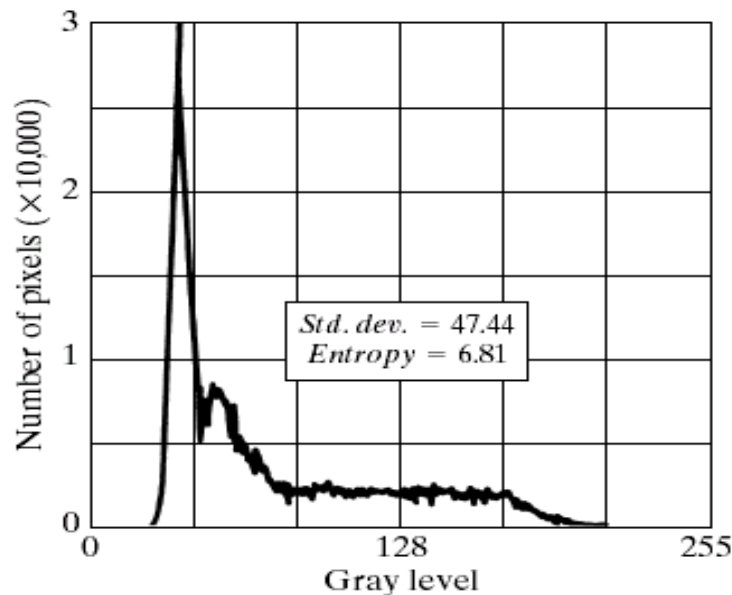
$$\hat{f}_n = \text{round} \left[\sum_{i=1}^m \alpha_i f_{n-i} \right]$$

$$e_n = f_n - \hat{f}_n$$

Lossless Predictive Coding



Since, the range of Prediction Error is much less as compared to original image; VLC based schemes can be used in the encoder !!



Arithmetic coding

- Variable length code
- Error-free compression technique
- A sequence of source symbols is assigned a single arithmetic code word
 - one-to-one correspondence between source symbol and code word does not exist

Arithmetic coding

- The code word defines an interval of real numbers in the range 0 and 1
- Each symbol of the message reduces the size of the interval in accordance with its probability of occurrence

```
Set low to 0.0 Set high to 1.0
While there are still input symbols do
    Get an input symbol code_
    range = high - low.
    high = low + range*high_range(symbol)
    low = low + range*low_range(symbol)
End of While
```

Arithmetic coding

Symbol	Probability	Range
a	.2	[0,0.2)
e	.3	[0.2, 0.5)
i	.1	[0.5, 0.6)
o	.2	[0.6, 0.8)
u	.1	[0.8, 0.9)
!	.1	[0.9, 1)

Set low to 0.0 Set high to 1.0
While there are still input symbols do
 Get an input symbol code_
 range = high - low.
 high = low + range*high_symbol
 low = low + range*low_symbol
End of While

Symbol	Probability	Range
a	.2	[0,0.2)
e	.3	[0.2, 0.5)
i	.1	[0.5, 0.6)
o	.2	[0.6, 0.8)
u	.1	[0.8, 0.9)
!	.1	[0.9, 1)

After seeing

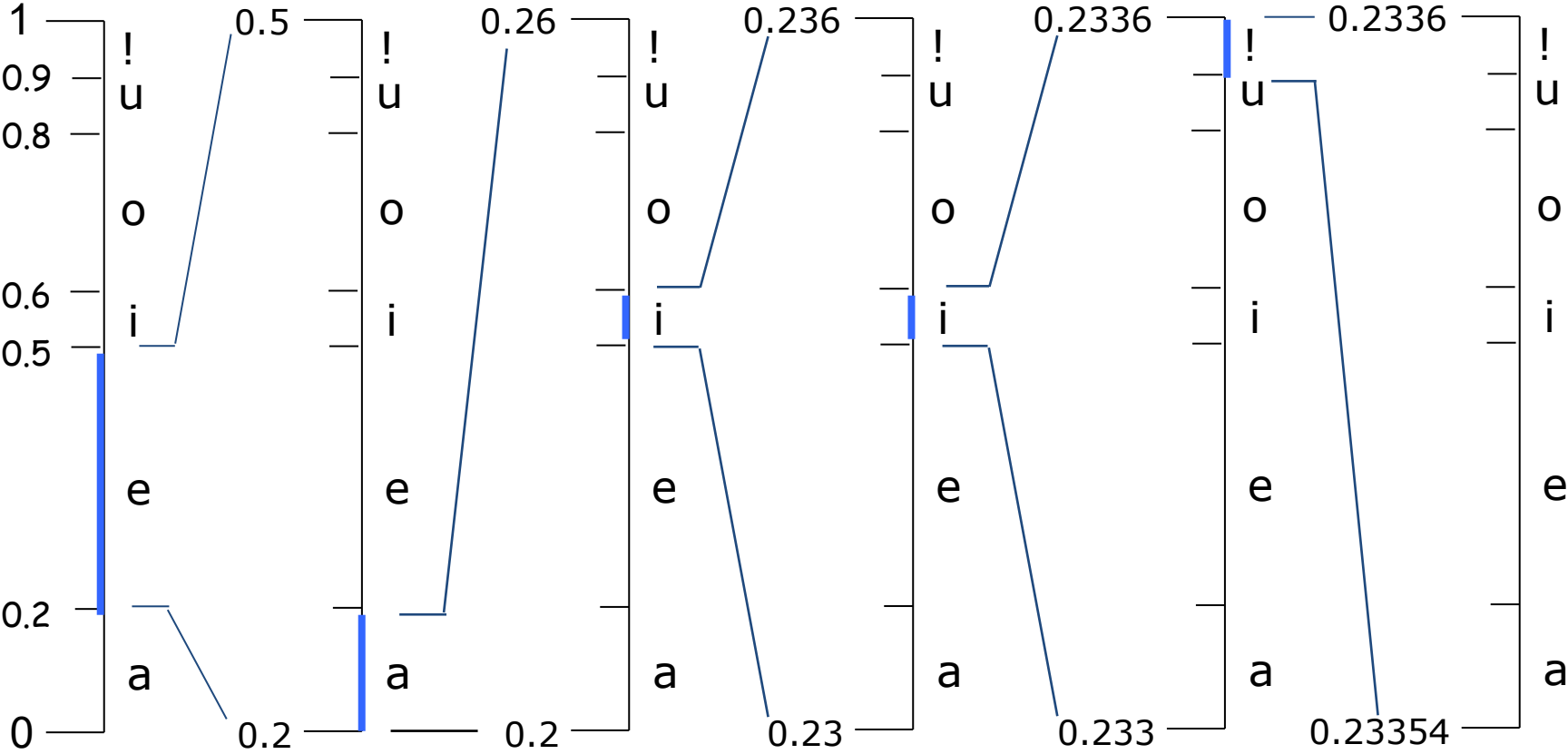
e

a

i

i

!



Arithmetic Decoding

Get encoded number

Do

*find symbol whose range straddles the encoded number
output the symbol*

range = high_symbol – low_symbol

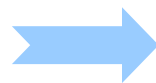
encoded number = encoded number - low_symbol

encoded number = encoded number / range

until no more symbols

Symbol	Probability	Range
a	.2	[0,0.2)
e	.3	[0.2, 0.5)
i	.1	[0.5, 0.6)
o	.2	[0.6, 0.8)
u	.1	[0.8, 0.9)
!	.1	[0.9, 1)

- The decoder gets the final number :
0.23354
- The number lies entirely within the space the model allocate for e




first character was **e**

Get encoded number

Do

find symbol whose range straddles the encoded number
output the symbol
range = symbol high value - symbol low value
subtract symbol low value from encoded number
divide encoded number by range
until no more symbols

Symbol	Probability	Range
a	.2	[0,0.2)
e	.3	[0.2, 0.5)
i	.1	[0.5, 0.6)
o	.2	[0.6, 0.8)
u	.1	[0.8, 0.9)
!	.1	[0.9, 1)

- The decoder gets the final number :
0.23354
- The number lies entirely within the space the model allocate for **e**
- Apply decoding  first character was **e**
 - Range = $0.5 - 0.2 = 0.3$
 - Encoded number = $0.23354 - 0.2 = 0.03354$
 - New number = $0.03354 / 0.3 = 0.1118$
- The range lies entirely within the space the model allocate for **a**

 second character was **a**

....

Get encoded number

Do

find symbol whose range straddles the encoded number

output the symbol

range = symbol high value - symbol low value

subtract symbol low value from encoded number

divide encoded number by range

until no more symbols

Symbol	Probability	Range
a	.2	[0,0.2)
e	.3	[0.2, 0.5)
i	.1	[0.5, 0.6)
o	.2	[0.6, 0.8)
u	.1	[0.8, 0.9)
!	.1	[0.9, 1)

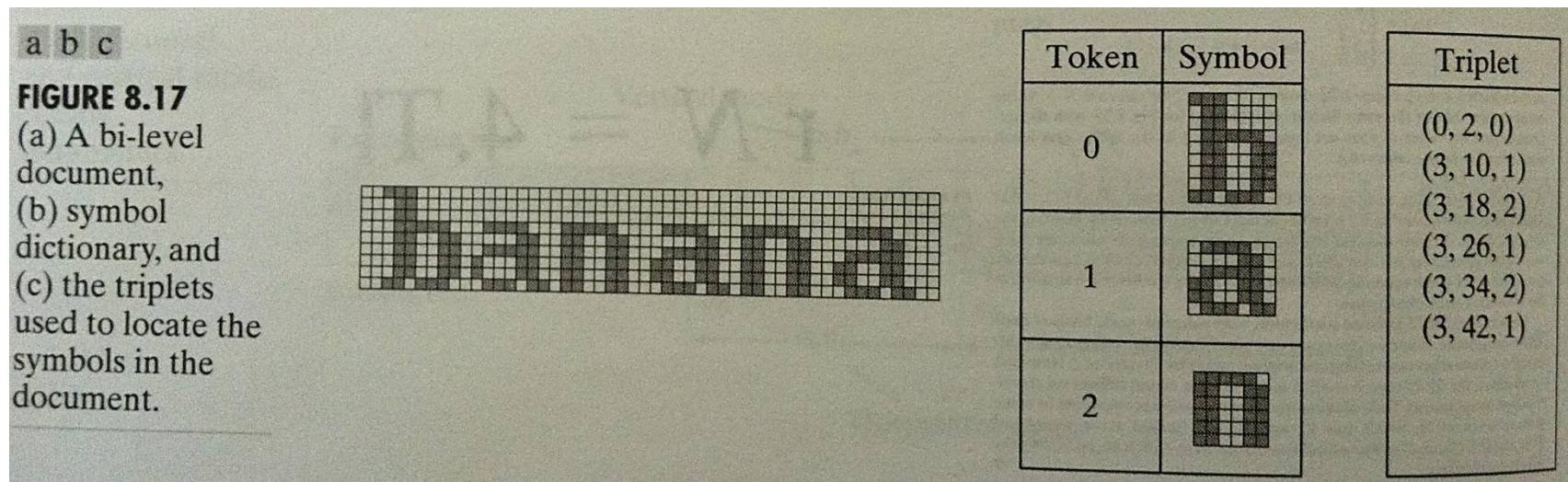
Encoded Symbol	Output Symbol	Low	High	Range
0.23354	e	0.2	0.5	0.3
0.1118	a	0	0.2	0.2
...

Arithmetic coding

- Draw the encoding sequence using arithmetic coding for the following symbols. Clearly write all the calculated values and draw the figure for each iteration of the encoding process, starting with iteration zero.
- symbols = {a, b, c, d}
- corresponding probabilities = {0.2, 0.2, 0.4, 0.2}
- The sequence to encode is (a, b, c, c, d).

Some Other Compression Methods

- Bit Plane Coding
- Symbol Based Coding
- Transform Based Coding – RGB to YCbCr



End
Image Compression