Aston University

CS3330: Image and Video Processing

Unit 10: Assessed Lab 3

**Instructions:**

Submission is required for this unit. Please fill in your answers and submit the completed document as “Lab Assessment 3” in the “Lab Assessment Submission” folder of the “Labs” section of the Blackboard module.

You have until 13:50 to complete this assessment. It is your responsibility to ensure that your work is submitted on time and 2 marks will be subtracted from your total for every part minute over this deadline that you submit. After submission, please ensure that your work has been uploaded by choosing the “my submissions” option from the top of the assignment submission page and then selecting your file’s name next to your submission. If the document that you submitted is displayed, then your assignment has been submitted correctly.

**Task 1 (6 marks):**

Write a Matlab script which uses Huffman coding to compress the following signal:

**0,1,1,0,0,3,3,4,1,1,0,0,1**

When called your script should not produce any output and it should only add two new variables to your workspace: the variable containing the **encoded signal** and the variable containing the **Huffman dictionary**. Copy the contents of your script into the box below.

|  |
| --- |
| signal = {0,1,1,0,0,3,3,4,1,1,0,0,1};  symbols = {0,1,3,4};  probabilities = [5,5,2,1]/13;  dict = huffmandict(symbols,probabilities);  encSignal = huffmanenco(signal,dict);  clear signal symbols probabilities; |

**Task 2 (4 marks):**

Write a Matlab function **detect\_points** which takes a grayscale image matrix as input and detects points in the image. Make use of the prctile function to set a threshold at which 5% of the pixels are classified as points.

The function should return a matrix which is the same size as the original image, containing 1 (alternatively, true) where a point has been detected and 0 (alternatively, false) elsewhere.

Copy the contents of your function into the answer box below, indicating the name of the file.

|  |
| --- |
| function [ absolute\_value ] = detect\_points( img )  %DETECT\_POINTS Summary of this function goes here  % Detailed explanation goes here  point\_mask = [-1 -1 -1;-1 8 -1;-1 -1 -1];  point\_filter = filter2(point\_mask, img);  absolute\_value = abs(point\_filter);  threshold\_point = prctile(absolute\_value,95);  absolute\_value(absolute\_value>=threshold\_point) = 1;  absolute\_value(absolute\_value<threshold\_point) = 0;  end |

**Task 3 (10 marks):**

Write two functions:

* **quantize\_block** which should take two inputs: a matrix of DCT coefficients and a quantization matrix. It should use the quantization matrix to quantize the coefficients and return the result.
* **inverse\_quantize\_block** which should take two inputs: a matrix of quantized DCT coefficients and a quantization matrix. It should use the two to perform inverse quantization, as described in the unit 8 lecture notes.

To test your function, download **lab\_10\_data.mat** from Blackboard. Loading the matrix should result in two variables appearing in your global workspace:

* **dct\_coefficients**, an 8 by 8 matrix of DCT coefficients
* **quantization\_matrix**, an 8 by 8 quantization matrix .

If you test your **quantize\_block** function with these inputs, you should get the following result:

If you test your **inverse\_quantize\_block** function with the result of the previous step and the quantization matrix, you should get the following result:

Copy the contents of your function files into the answer box below indicating, for each, the name of the file.

|  |
| --- |
| function [ quantize ] = quantize\_block( dct\_coefficients, quantization\_matrix )  %QUANTIZE\_BLOCK Summary of this function goes here  % Detailed explanation goes here  quantize = dct\_coefficients./quantization\_matrix;  end  function [ inverse\_quantize ] = inverse\_quantize\_block( quantized\_dct\_coefficients, quantization\_matrix )  %INVERSE\_QUANTIZE\_BLOCK Summary of this function goes here  % Detailed explanation goes here  inverse\_quantize = quantized\_dct\_coefficients.\*quantization\_matrix;    end |

Write a third function **calculate\_quantization\_error**, which should take two arguments: a matrix of DCT coefficients and a quantization matrix. Using the two functions you have created previously it should calculate the error resulting from (lossy) quantization as follows:

1. quantize the DCT coefficients using the quantization matrix,
2. perform inverse quantization on the result, using the same quantization matrix as in the previous step,
3. subtract (elementwise) the result of step 2. from the original quantization matrix,
4. sum up the elements resulting from step 3.

If you test your **calculate\_quantization\_error** function with the inputs loaded from **lab\_10\_data.mat**, your result should be **311.5440**.

Copy the contents of your function file into the answer box below.

|  |
| --- |
| function [ final\_answer ] = calculate\_quantization\_error( dct\_coefficients, quantization\_matrix )  %CALCULATE\_QUANTIZATION\_ERROR Summary of this function goes here  % Detailed explanation goes here  quantize = dct\_coefficients./quantization\_matrix;  inverse\_quantize = quantize.\*quantization\_matrix;  new\_quantize = quantization\_matrix-inverse\_quantize;  final\_answer = sum(new\_quantize(:));  end |