Battula Venkata Sai Ankit
2019AAPS0331H

ECE F344 Information Theory and Coding
Assignment 1

Entropy of an Image

Theory and Definition:

The entropy of an image can be determined approximately from the histogram of the image. The normalised histogram of a grayscale image shows the different grey level probabilities in the image. Entropy can be calculated by continuous summation of the values of each probability times log2(probability) at that pixel in the whole of histogram.

Commands Used:

1. imread: Used to read an image from a given file location path.
2. imhist : imhist calculates the histogram of the grayscale image and stores the histogram counts in counts matrix and the bin Locations in binLocations matrix.
3. numel : returns the number of elements in the matrix (here used to calculate the size of the image)

Approach:

1. Image is initially read using imread function and stored in an array and numel is used to calculate the size of that array representing the total size of the image.
2. We then compute the histogram of the image and store the counts matrix.
3. We then normalise the by dividing the matrix with a scalar i.e, size of the image.
4. We then compute the entropy of the image by continuously summating the values of each probability times log2(probability) at that pixel in the whole of histogram

Battula Venkata Sai Ankit
2019AAPS0331H

Code:

```
% Clearing Workspace before starting the run
clc; clear;
% Input a grayscale image
image = imread('cameraman.tif');
% imhist calculates the histogram of the grayscale image and stores the
% histogram counts in counts matrix and the bin Locations in binLocations
% matrix
[counts, binLocations] = imhist(image);
% To normalize the histogram counts, and get the probabilites of each pixel
% we are dividing the counts matrix with a scalar [ numel(image) => total
% number of pixels in the whole image ]
normalisedHistogramCounts = counts./numel(image);
% Probabilty matrix is the the normalised Histogram Matrix but considering
% only cases when it is greater than 0, so that the logarithm doesn't
% throw any error
probabilityMatrix = normalisedHistogramCounts(normalisedHistogramCounts>0);
% Entropy is the - (sum of p * log2(p))
H = sum(-probabilityMatrix.*log2(probabilityMatrix));
fprintf("The entropy of the image is %f \n", H);
```

Results:

Input :

A grayscale image

Output :

The entropy of the image is 7.009716

Conclusion:

This is useful for automatic image focussing because once the image focussing changes the entropy changes, we can therefore perceive the change in entropy and change focus accordingly.

Battula Venkata Sai Ankit
2019AAPS0331H

Shannon Fano Coding

Theory and Definition:

Shannon-Fano Coding is a source coding technique used for constructing a prefix code based on a set of symbols and their probabilities. It produces fairly efficient variable-length encodings; when the two smaller sets produced by a partitioning are in fact of equal probability.

Commands Used:

1. sort: This command is used to sort the given matrix and with the extra argument ‘descend’ the matrix is sorted in descending order.
2. strcat: This command is used to concatenate the strings in it’s input arguments horizontally producing a single string.
3. strlength : This command is used to calculate the size of a string in it’s input arguments.

Approach:

1. Messages (here probabilities) are first written in descending order (sorted in descending order). Here the initial matrix is also saved so that we don’t lose the access to map the final code words.
2. Divide the set into two most equiprobable subsets.
3. Message in the first set is given a bit ‘0’ and the message in second set is given a bit ‘1’.
4. The above procedure is repeated for each set separately and continued until no further division is possible.
5. Finally we get the code word for the respective symbol by backtracking the bits and storing them into a matrix which is returned at last.

Battula Venkata Sai Ankit
2019AAPS0331H

Code:

Encoder Code

```
% Clear the workspace before starting the program
clc; clear;
% Probability array
p = [0.13 0.10 0.36 0.12 0.14 0.09 0.02 0.04];
code_words = ShannonFano(p);
n = length(p);
h_cap = 0;
h = 0;
for i = 1:n
   h = h + (p(i) * log2(1/p(i)));
   h_cap = h_cap + (p(i) * strlength(code_words{1,i}));
end
eff = (h / h_cap) * 100;
fprintf('Efficiency with Shannon Encoding is : %f %%\n', eff);
```

Shannon Fano Code

```
function code_words = ShannonFano(p1)
   p1 = p1/sum(p1);

   if(length(p1)>2)
       [pdes,idx] = sort(p1,'descend');
       qsum = (2*cumsum(pdes))-1;
       [~,idx1] = min(abs(qsum));
       if((idx1>1)&&(idx1<length(pdes)-1))
           q1 = pdes(1:idx1);
           q2 = pdes((idx1+1):length(pdes));
           old_code1 = ShannonFano(q1);
           old_code2 = ShannonFano(q2);
           new_code = [strcat('0',old_code1) strcat('1',old_code2)];
       elseif(idx1==1)
           q1 = pdes(1);
           q2 = pdes(2:length(pdes));
           old_code1 = ShannonFano(q1);
           old_code2 = ShannonFano(q2);
           new_code = [old_code1  strcat('1',old_code2)];
       else
           q1 = pdes(1:((length(pdes)-1)));
           q2 = pdes(length(pdes));
           old_code1 = ShannonFano(q1);
           old_code2 = ShannonFano(q2);
           new_code = [strcat('1',old_code1)  old_code2];
       end
       code_words(idx) = new_code;
   elseif(length(p1)==2)
       code_words = {'0', '1'};
   else
       code_words = {'0'};
   end
end
```

Battula Venkata Sai Ankit
2019AAPS0331H

Results:

Input

```
p = [0.13 0.10 0.36 0.12 0.14 0.09 0.02 0.04];
```

Output

Efficiency with Shannon Encoding is : 96.712722 %

Conclusion:

Shannon Fano Coding is never used practically because it is not so computationally feasible to always have equiprobable message scenario, also the average word length is much higher as compared to Huffman Source coding implementation.

Battula Venkata Sai Ankit
2019AAPS0331H

Huffman Coding

Theory and Definition:

Huffman Coding is a source coding technique used for constructing a prefix code based on a set of symbols and their probabilities. It provides good construction of prefix code even for general source code distribution and not a symmetric one.

Commands Used:

1. sort: This command is used to sort the given matrix and with the extra argument 'descend' the matrix is sorted in descending order.
2. strcat: This command is used to concatenate the strings in it's input arguments horizontally producing a single string.
3. strlength : This command is used to calculate the size of a string in it's input arguments.

Approach:

1. Messages (here probabilities) are arranged in descending order of probability.
2. The last two symbols after sorting are assigned '0' and '1'.
3. Combine the last two symbols and move the combined symbols as high as possible (combining is done by adding the probabilities)
4. Repeat the above steps until we are left with the final list of source symbols of only two for which '0' and '1' are assigned.
5. We then backtrack the bits to find the codeword of that particular symbol.

Battula Venkata Sai Ankit
2019AAPS0331H

Code:

Encoder Code

```
% Clear the whole workspace before running
clc; clear;
% Probability matrix
p = [0.13 0.10 0.36 0.12 0.14 0.09 0.02 0.04];
code_words = huffman(p);
n = length(p);
h_cap = 0;
h = 0;
% Loop to calculate the entropy and average word length
for i = 1:n
   h = h + (p(i) * log2(1/(p(i))));
   h_cap = h_cap + (p(i) * strlength(code_words{1,i}));
end
eff = (h / h_cap) * 100;
fprintf('Efficiency with Huffman Encoding is : %f %%\n', eff);
```

Huffman Code

```
function code_words = huffman(p)
   p = p/sum(p);
   if(length(p)>2)
        [pdes,idx] = sort(p,'descend');
        q1 = pdes(1:(length(pdes)-1));
        q1(length(pdes)-1) = pdes(length(pdes)-1)+pdes(length(pdes));
        old_code = huffman(q1);
        new_code = [old_code(1:(length(old_code)-1))
strcat(old_code{length(old_code)},'0')
strcat(old_code{length(old_code)},'1')];
        code_words(idx) = new_code;
   elseif(length(p)==2)
        code_words = {'0', '1'};
   end
end
```

Results:

Input

```
p = [0.13 0.10 0.36 0.12 0.14 0.09 0.02 0.04];
```

Output

```
Efficiency with Huffman Encoding is : 97.070918 %
```

Conclusion:

Huffman generates a uniquely decodable code and smallest average codeword length. Because of non-symmetric source code, there might be an ambiguity which is completely eradicated with the help of Huffman Coding. Shannon Fano coding is computationally much simpler than the Shannon Fano coding too, making it the easiest algorithm to implement.