Table of Contents

Video Technology Labb 3

This report shows how to implement differential-encoding and top it with huffman-coding in matlab.

```
%%Reads a video and imports it into the "mov" variable.
vidObj = VideoReader('video.mp4');
width = 10;
height = 10;
framesToRead = 3; %vidObj.NumberOfFrames;

movBW(1: framesToRead) = struct('data', zeros(height, width ));
movDiffEncoded(1: framesToRead) = struct('data', zeros( height * width , 1, 'doubl
movDiffDecoded(1: framesToRead) = struct('data', zeros( height * width , 'uint8'))

%Read every frame and convert to BW in same procedure. Add to movBW-struct.
for i = 1: framesToRead
    frame = read(vidObj, i);
    frame = imresize(frame, [height width]);

    frameYUV = frameRGB2YUV(frame);
    frameBW = frameYUV(:,:,1);      %The Black and White frame.
    movBW(i).data = frameBW;

end
```

Entropy before diff-encoding

```
disp(entropy(movBW(1).data));
disp(entropy(movBW(framesToRead).data));

for i = 1: framesToRead    %Diff-encoding
    movDiffEncoded(i).data = diffEncoder(movBW(i).data);
end
```

```
    5.7825

    5.7339
```

Entropy after differential-encoding

```
disp(entropy(movDiffEncoded(1).data));
disp(entropy(movDiffEncoded(framesToRead).data));
```

```
% Now we create a huffman-codebook
codebook = huffmanCodebook(movDiffEncoded(1).data);

%And then encode the allready diff-encoded movie with huffman aswell.
for i = 1: framesToRead
    movDiffHuffman{i} = huffmanEncoder(movDiffEncoded(i).data, codebook);
end

%and then decode it
for i = 1: framesToRead
    movDiffEncoded(i).data = huffmanDecoder(movDiffHuffman{i}, codebook, height, w
end

for i = 1: framesToRead %And diff-decode
    movDecoded(i).data = diffDecoder(movDiffEncoded(i).data, width);
end

if isequal(movBW, movDecoded)
    disp('Theyre equal');
else
    disp('Theyre not equal');
end
```

```
    0.9815

    0.9765

Theyre equal
```

Published with MATLAB® R2014b

huffmanCodebook

```
function [ CodeBook ] = huffmanCodebook( imgBW )

% This function generates Huffman dictionary (aka codebook) from a BW image
% Input: imgBW, which is a black and white image (UINT8)
% Output: CodeBook, which is a 'table' for converting 256 grayscale levels to
% the corresponding codewords
% Note: Some of the codewords might be 'null' due to the absence of certain pixel
% You can add the input/output arguments if needed.
```

Entropy and probability distribution

```
[height,width] = size(imgBW);
if (width>1)
    imgBW = reshape(imgBW, [1,(height*width)]);
end

% Calculate probability for each symbol
% freq = tabulate(imgBW);
% P = freq(:,3)./100;
% symbols = freq(:,1);

%
[freq,symbols] = histcounts(imgBW,-255:256);
symbols(end) = [];

P = freq./sum(freq);


%%%%%%% GÖR TEMPBOOK TILL EN KOLUMN, SORTERA TEMPBOOK OCH SYMBOLERNA
%%%%%%% MED HJÄLP AV SANNOLIKHETERNA OCH LÄGG IHOP
```

Create codebook

```
tempbook = cell(length(symbols),2); % List of codewords, same length as number of
for i=1:length(symbols)
    tempbook{i,1} = symbols(i);
end

mEvents = num2cell(1:length(P)); % Changing list for events, merged and single
mProb = P; % Changing list for probabilities, merged and single

% Loop until there is only 1 merged value left
while(length(mEvents) > 1)

    % Sort the list of merged/single probabilities
    % and retrieve both a list of probabilites and index values of the
    % events
    [sProb, sInd] = sort(mProb);

    % Get the events with the smallest probabilites from merged list
    % events are in a cell array with all the sub-events in the cell
```

```
        smallestE1 = mEvents{sInd(1)};
        smallestE2 = mEvents{sInd(2)};

        % Add a 0 to the codeword of the smallest probability
        % and all the sub-probabilities "under" the chosen node
        for i = 1:length(smallestE1)
            tempbook{smallestE1(i), 2} = ['0', tempbook{smallestE1(i),2}];
        end

        % Add a 1 to the codeword of the second smallest probability
        % and all the sub-probabilities "under" the chosen node
        for i = 1:length(smallestE2)
            tempbook{smallestE2(i), 2} = ['1', tempbook{smallestE2(i),2}];
        end

        % Add the events to the merged event cell array, as a cell of
        % both the events to be able to add code to them
        mEvents{end+1} = [smallestE1, smallestE2];

        % Add the sum of both the probabilites to the merged probabilities to
        % be able to sort them into the "tree"
        mProb(end+1) = sProb(1)+sProb(2);

        % Remove the single events and probabilities as they now exists as
        % merged cells
        mEvents(sInd(1:2)) = [];
        mProb(sInd(1:2)) = [];

    end

    CodeBook = tempbook;
```

Published with MATLAB® R2014b

```
function [ binaryVector ] = huffmanEncoder( imgBW, codeBook )

%This function encodes a black n white image into a single binary vector
% Input: 'bwImg' is black image, e.g. a Y frame of a YUV image
%        'CodeBook', is the codebook/dictionary generated by huffmanCodebook

% Output: encoded bwImg, a binary vector formed by concatenated huffman codewords

[height, width] = size(imgBW);

placeholder = cell(height, width);

for i = 1: length(codeBook)
    codeWord = codeBook(i, 2);          %The codeWord we are gonna substitute with
    logical = imgBW == codeBook{i,1};   %Logical matrix that only targets the pixe
    placeholder(logical) = codeWord;    %Replace with codeWord
end

binaryVector = reshape(placeholder, 1, []); %Reshape to vector
binaryVector = cell2mat(binaryVector);      %Get rid of type 'cell'.
```

Published with MATLAB® R2014b

huffmanDecoder

```
function [ decodedImage ] = huffmanDecoder( binaryVector, CodeBook, width, height)

% This function decodes a single binary vector into a black n white image
% Input: 'binaryVector' is a binary vector formed by concatenated huffman
%       codewords representing, e.g. a Y frame.
%         'codebook' is the generated by huffmanCodebook
%         'width' is the width of the image to be decoded
%         'height' is the height of the image to be decoded
% Output: decodedImage, Black and White, type uint8
```

Code

```
% Pre-allocate imagematrix, as a vector
tempimagevector = zeros(height*width, 1);

% Assign variables for faster computing
codebooklength = length(CodeBook);
binvectorlength = length(binaryVector);


% An index for the imagevector, to assign values into a pre-allocated
% vector for faster computing
pixelpos = 1;

% Loop until all elements are checked
i = 1;
while(i <= binvectorlength)

    % Reset the found variable
    found = 0;

    % Reset the codeword that is checked
    codeword = [];

    % Loop the binaries and add the next binary to the checked codeword if
    % the current doesn't exist in the codebook
    while(found == 0 && i <= binvectorlength)

        % Add the current binary to the checked codeword
        codeword(end+1) = binaryVector(i);

        % Loop the codebook to check the current codeword
        % if it exists, for-loop breaks and found-switch is true which
        % breaks the second while loop, that resets the codeword checked.
        for n=1:codebooklength
            if(isequal(codeword,CodeBook{n,2}))
                tempimagevector(pixelpos,1) = CodeBook{n,1};
                found = 1;
                pixelpos = pixelpos + 1;
                break;
            end
```

```
        end

        % Go to next index even if the codeword has been found or not
        i = i+1;

    end
end

% Reshape the vector to the right size as given by the input argument
decodedImage = reshape(tempimagevector, [height,width]);

end
```

Published with MATLAB® R2014b

diffEncoder

This function encodes a black and white image into a single vector using differential coding

```
function [ encodedVector ] = diffEncoder( BWImg )
% Input:
%   BWImg: black and white image i.e. a Y frame
%
% Output:
%   encodedVector: the BWImg encoded into a single vector through
%       differential-encoding.


[rowSize, colSize] = size(BWImg);       % We need to know how many columns there i
encodedImg = zeros(rowSize, colSize);   % Placeholder for the encoded-img.

BWImg = double(BWImg);                  %Original type is uin8. uin8 cant be negat

for colIndex = 1: colSize               %For every column..

    if colIndex == 1                    %If first column, save as reference-value
        encodedImg(:, 1) = BWImg(:,1);

    else                                %Calculate difference between indexed colu
        encodedImg(:, colIndex) = BWImg(:, colIndex) - BWImg(:, colIndex - 1);

    end
end

encodedVector = reshape(encodedImg, [], 1); %Reshape the matrix C x R into a singl
```

Published with MATLAB® R2014b

diffDecoder

decodes the previously encoded image.

```
function [ decodedImg ] = diffDecoder( encodedVector, columns )
%   encodedVector : nx1 vector with the image differential-encoded
%   columns : the amount of columns in the original image.
% ouput:
%   decodedImg: a NxR image decoded from the encodedVector.


% Reshape the vector into a matrix, so we can do column-wise operations.
encodedImg = reshape(encodedVector, [], columns);

decodedImg = zeros(size(encodedImg));   %Placeholder

for colIndex = 1: columns               %Loop over all columns.

    if colIndex == 1                    %Reference value at the start of the rows,
        decodedImg(:, 1) = encodedImg(:,1);

    else                                %Take the last value and add the differenc
        decodedImg(:, colIndex) = decodedImg(:, colIndex - 1) + encodedImg(:, colI

    end
end

decodedImg = uint8(decodedImg);         %the encodedVector was type 'double' to su
end
```

Published with MATLAB® R2014b