Table of Contents

1. Select and show a random frame from your video

You can also load a pre-stored grayscale frame in but it needs to maintain a good resolution!

```
patchDim = 8;
frame = imread('michelleBW.jpg');
[originalHeight, originalWidth] = size(frame);
```

2. Segment the frame into image patches using patchImage function

```
[patches, height, width] = patchImage(frame, patchDim, patchDim);
```

3. For each patch use [matDCTCoeff] = dctCoeffi(imagePatch)

to get the cosine square matrix and denote the patch using a vector of DCT coefficients.

```
baseVectorMatrix = dctCoeffi(patches(:,:,1));

for i = 1: length(patches)
    patch = reshape(patches(:,:,i), 1, []) - 128;             %Reshapes patch to a
    patchesEncoded(:,:,i) = patch * baseVectorMatrix';
end
```

4. For each patch, reconstruct it using the first 16/32 coefficients

```
for i = 1: length(patches)
    patchRow = patchesEncoded(:,:,i);
    patchesDecoded(:,:,i) = reshape( patchRow * baseVectorMatrix, patchDim, patchD
end
```

5. Reconstruct the frame and calculate the quality loss in Mean Squared Error

task 1. show the reconstructed frame using 16/32 coefficients respectively task 2. calculate loss

```
X = reshape(patchesDecoded, patchDim, []);
X = reshape( X, [ patchDim width height./patchDim ] );
X = permute( X, [ 1 3 2 ] );
X = reshape( X, [ height width ] );
X = uint8(X);

X = X(1:originalHeight, 1: originalWidth);       %Make sure X has same dimensions
disp('Mean squared error is:');
disp(meanSquaredError(frame, X));
imshow(X);
```

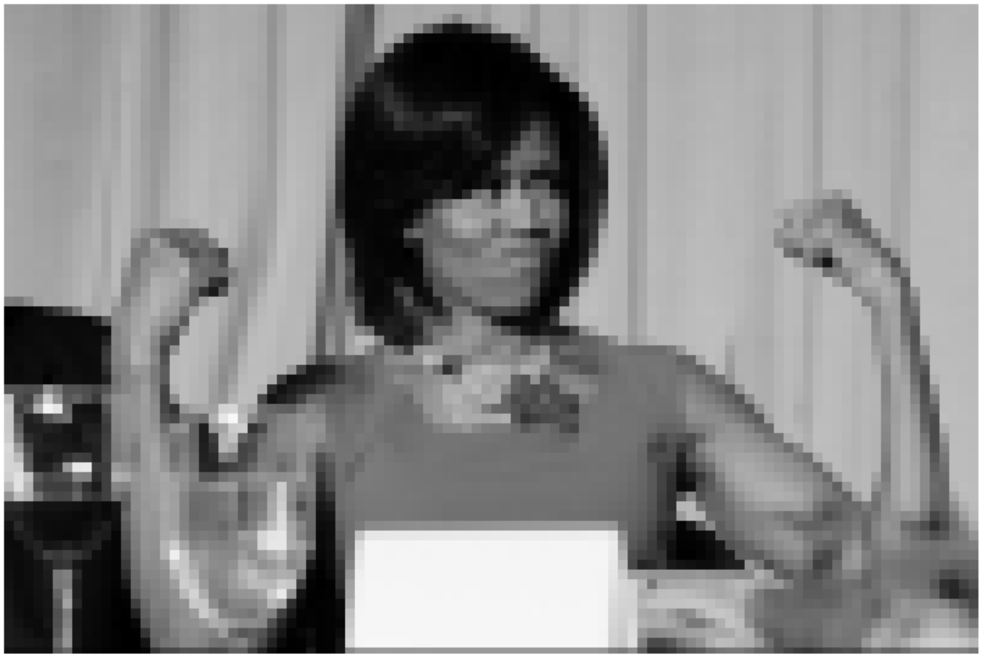
```
Mean squared error is:
    0.0820
```

6. Apply quantization to DCT coefficients and study the quanlity loss in Mean Squared Error

task 1. reconstructed frame using quantized 16/32 coefficients respectively (command floor can be used for quantization) task 2. calculate loss

```
% We will quantize by setting only keeping the top-left value of every
% patch. This is also called the constant component and defines the
% constant hue of the patch.

patchesEncoded(1,2:end,:) = 0;

for i = 1: length(patches)
    patchRow = patchesEncoded(:,:,i);
    patchesDecoded(:,:,i) = reshape( patchRow * baseVectorMatrix, patchDim, patchD
end


X = reshape(patchesDecoded, patchDim, []);
X = reshape( X, [ patchDim width height./patchDim ] );
X = permute( X, [ 1 3 2 ] );
X = reshape( X, [ height width ] );
X = uint8(X);

[height, width] = size(frame);
X = X(1:height, 1:width);       %Make sure X has same dimensions as frame.
disp('Mean squared error is:');
disp(meanSquaredError(frame, X));
imshow(X);
```

```
Mean squared error is:
  285.6206
```



Published with MATLAB® R2014b

```
function [ imagePatches, height, width ] = patchImage( aBWFrame, patchHeight, patc
```

Introduction of patchImage

This function segments an input image/frame into a number of non-overlapping patches as the output.

```
Input:
1. aBWFrame, is a black and white image or frame to be segmented
2. patchWidth and patchHeight define the width and height of the segmented image p
Output:
imagePatches is a multi-dimiension matrix (a stack of segmented patches)
whose size == patchHeight * patchWidth * numPatch
numPatch is the total number of image patches can be segmented from aBWFrame by th
Width/Height, dimensions of the the frame with padded zeros.
```

```
%   Note that! patchImage function should be able to handle the cases when
%   width/height of the input frame is not exactly integer times of
%   patchWidth/patchHeight. This can be done by padding zeros to the
%   aBWFrame OR simply ignore the remainders:
%   i.e. If frame size == 10*10 and both patchWidth/patchHeight == 3
%        you can pad zeros to enlarge it to 12*12 and get 16 patches,
%        OR get 9 patches by ignoring a row and a column in the frame.
%   You can decide the zero-padding method yourself.


%This function padds zeros around the frame to not clip any data.

[height, width] = size(aBWFrame);
neededHeight = ceil(height/patchHeight);
neededWidth = ceil(width/patchWidth);

%Padd zeros:
aBWFrame(height: neededHeight * patchHeight, width:neededWidth * patchWidth) = 0;

rows = neededHeight;
columns = neededWidth;

placeholder = zeros(patchHeight, patchWidth, rows*columns);

index = 1;

for row = 1: rows
    rowEnd = row * patchHeight;
    rowStart = rowEnd - patchHeight + 1;


    for col = 1: columns
        columnEnd = col * patchWidth;
        columnStart = columnEnd - patchWidth +1;

        placeholder(:, :, index) = aBWFrame(rowStart:rowEnd, columnStart:columnEnd
        index = index + 1;

    end
end

imagePatches = placeholder;
[height, width] = size(aBWFrame);

end
```

Published with MATLAB® R2014b

```
function [ matDCTCoeff ] = dctCoeffi( imagePatch  )

% Input imagePatch , an image patch
% Output matDCTCoeff, a N*N matrix of DCT coefficients

% Obviously, it is a completed function.

N = numel(imagePatch);
n = 0 : N-1;

wei = [ 1/sqrt(N), ones(1,N-1) * sqrt(2/N)];
matDCTCoeff = zeros(N);

for kPtr = 1: N
    k = kPtr -1;
    matDCTCoeff(kPtr,:) = cos(pi/N * (n+0.5)*k).* wei(kPtr);
end

end
```

Published with MATLAB® R2014b

```
function [ error ] = meanSquaredError( before, after )
%MEANSQUAREDERROR Calculates mean squared error between to matrixes.
% In: before = matrix before. After = matrix after.
% Out : error : value of MSE

before = double(before); %Needed to not clip data when counting.
after = double(after);

errorMatrix = (before - after).^2;
error = mean(mean(errorMatrix));

end
```

Published with MATLAB® R2014b