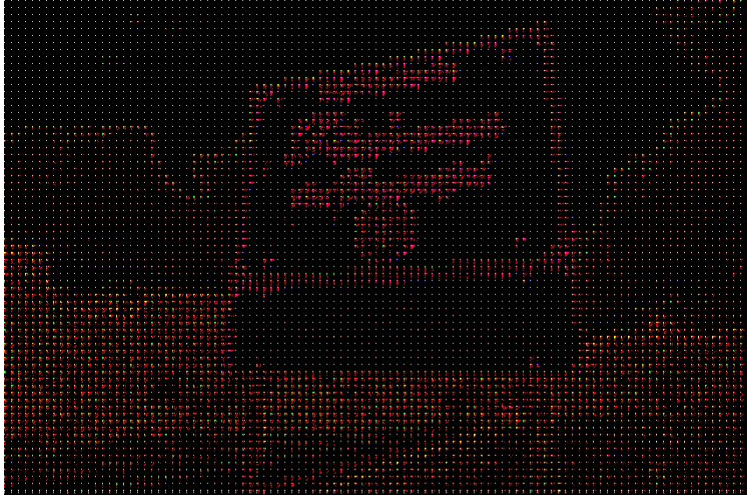
Group Project: JPEG Compression

TJ Couch, Austin Vickers, Matthew Robertson

Findings:

-


Welcome to
Alabama
the Beautiful



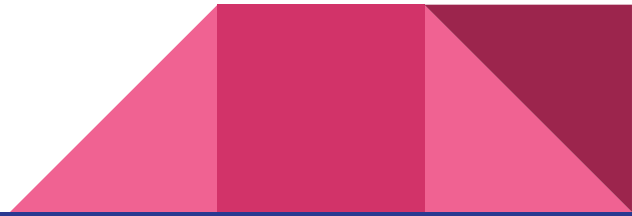

Welcome to
Alabama
the Beautiful

Findings:

- File sizes around 1/10 original size
- Compression not noticeable unless zoomed in
- Pixel-wise error is not particularly large in most places
- Pixel-to-Signal Noise ratios very large
    - 11.6639 for image 1

Challenges Faced

- Rewrote Chroma Subsampling function because original was incorrect
- Parallelizing work was nearly impossible since everything builds off of previous image stages
- Precisely specifying loop bounds on DCT functions proved difficult

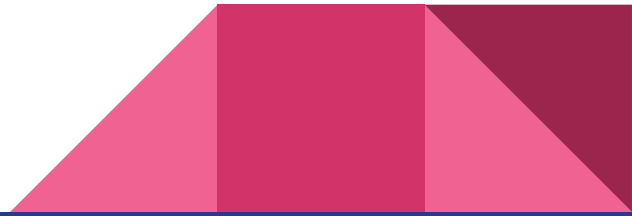


Work Distribution

TJ Couch: Team Leader, Steps 1 to 3

Austin Vickers: Steps 4 and 5, Group Report, Submission

Matthew Robertson: Presentation



CompressInput

- Prompts user for image, then compresses and decompresses the image at the path.

```
% compressInput
% prompts the user for an image, then compresses and decompresses the image at the supplied path
%
% TJ Couch, Matthew Robertson, Austin Vickers
% JPEG Compression Project
% CS 443 Multimedia
% 4/8/19

close all;

%get the name of the file to compress
fileName = input('Enter the path to the image to compress: ', 's');

if (~exist(fileName, 'file'))%if file doesn't exist, say so. Replace exists with isfile in MATLAB newer than R2017a
    fprintf('File %s does not exist!\n', fileName);
else
    %display the current file to keep track
    fprintf('File: %s\n', fileName);

    %read rgb image
    sourceImg = imread(fileName);

    %compress image
    compressedImg = jpegCompress(sourceImg);

    %show image
    %imshow(compressedImg);

    %write compressed image to its own file
    writeName = sprintf('compressed-%s%s', fileName(1:end-4), '.png');
    imwrite(compressedImg, writeName);
```

```
    %decompress image
    decompressedImg = jpegDecompress(compressedImg);

    %show image and values
    imshow(decompressedImg);
    decompressedImg(1:8, 1:8, :)

    %calculate pixel wise error
    errorImg = pixelWiseError(sourceImg, decompressedImg);
    imagesc(errorImg)

    %compute PSNR
    PSNRresult = PSNR(sourceImg, decompressedImg)

    %write decompressed image to its own file
    writeName = sprintf('decompressed-image%d.png', i);
    imwrite(decompressedImg, writeName);

    pause
end
```

CompressImage

- Converts images in directory from RGB to JPEG compressed, then back.

```
% compressImages
% converts the three images in this directory from rgb to JPEG compressed and back again
%
% TJ Couch, Matthew Robertson, Austin Vickers
% JPEG Compression Project
% CS 443 Multimedia
% 4/8/19

close all;

%compress each of 3 images
for i = 1:3
    fileName = sprintf('image%d.jpg', i);

    %display the current file to keep track
    fprintf('File: %s\n', fileName);

    %read rgb image
    sourceImg = imread(fileName);

    %compress image
    compressedImg = jpegCompress(sourceImg);

    %show image values step I
    sourceImg(1:8, 1:8, :)
    compressedImg(1:8, 1:8, :)

    %write compressed image to its own file
    writeName = sprintf('compressed-image%d.png', i);
    imwrite(compressedImg, writeName);
```

```
    %decompress image
    decompressedImg = jpegDecompress(compressedImg);

    %show image and values
    imshow(decompressedImg);
    decompressedImg(1:8, 1:8, :)

    %calculate pixel wise error
    errorImg = pixelWiseError(sourceImg, decompressedImg);
    imagesc(errorImg)

    %compute PSNR
    PSNRresult = PSNR(sourceImg, decompressedImg)

    %write decompressed image to its own file
    writeName = sprintf('decompressed-image%d.png', i);
    imwrite(decompressedImg, writeName);

    pause
end
```

jpegCompress

- Compresses image matrix to JPEG compression standards

```
% jpegCompress(image) - compresses the image matrix according to JPEG compressior
% image: the rgb image matrix to compress
% Note: the image must be in rgb format
% returns compressed image in jpeg compressed format
%
% TJ Couch, Matthew Robertson, Austin Vickers
% JPEG Compression Project
% CS 443 Multimedia
% 4/8/19

function f = jpegCompress(sourceImg)

%block size 8
blockSize = 8;

%convert image to yCbCr
yCbCrImg = rgb2ycbcr(sourceImg);

%chroma subsample the image 4:2:0
subsampledImg = chromaSub(yCbCrImg, 2, 0);

%2D DCT with block size 8
blockSize = 8;
dctImg = dct2D(subsampledImg, blockSize);

%quantize
quantizedImg = quantize(dctImg, blockSize);

f = quantizedImg;
```

chromaSub

```
% chromaSub.m - applies chroma subsampling to the image
% chromaSub(image, cols1, cols2)
% sourceImg: the rgb image matrix to convert
% cols1: the number of columns to sample from the first row
% cols2: the number of columns to sample from the second row
% Note: the image must be in YCbCr format
% returns YCbCr subsampled image
%
% TJ Couch, Matthew Robertson, Austin Vickers
% JPEG Compression Project
% CS 443 Multimedia
% 4/8/19

function f = chromaSub(sourceImg, cols1, cols2)

%get image size
imSize = size(sourceImg);

%convert to YCbCr
yCbCrImg = sourceImg;

%create blank image matrices
yccSubImg = uint8(zeros(imSize));%subsampled image

for i = 1:2:imSize(1)%every two rows
    for j = 1:imSize(2)%every column
        %apply 4:cols1:cols2 subsampling
        %copy Cb and Cr as the pattern specifies by row and by column
        %1st and 2nd row - get first in group of four columns

        %1st row
        %copy y
        yccSubImg(i, j, 1) = yCbCrImg(i, j, 1);
        %subsample Cb and Cr
        cb = yCbCrImg(i, floor((j - 1) / (4 / cols1)) * (4 / cols1) + 1, 2);
        cr = yCbCrImg(i, floor((j - 1) / (4 / cols1)) * (4 / cols1) + 1, 3);
        yccSubImg(i, j, 2) = cb;
        yccSubImg(i, j, 3) = cr;

        %2nd row
        if (i + 1 <= imSize(1))
            %copy y
            yccSubImg(i + 1, j, 1) = yCbCrImg(i + 1, j, 1);
            %subsample Cb and Cr
            %make new cb and cr if it has its own values to get, otherwise use the row 1 values
            if (cols2 > 0)
                cb = yCbCrImg(i + 1, floor((j - 1) / (4 / cols2)) * (4 / cols2) + 1, 2);
                cr = yCbCrImg(i + 1, floor((j - 1) / (4 / cols2)) * (4 / cols2) + 1, 3);
            end
            yccSubImg(i + 1, j, 2) = cb;
            yccSubImg(i + 1, j, 3) = cr;
        end
    end
end

%create subsampled rgb image
f = yccSubImg;
```

DCT 2D

```
% dct2D.m - turns YCbCr image into frequency domain using 2D DCT according to JPEG standard
% dct2D(subsampledImg, blockSize)
% subsampledImg: the quantize image matrix to dequantize
% blockSize: the size of the chunks to use
% Note: the image must be in YCbCr format
% returns DCT image
%
% TJ Couch, Matthew Robertson, Austin Vickers
% JPEG Compression Project
% CS 443 Multimedia
% 4/8/19

%Written by TJ Couch
function f = dct2D(subsampledImg, blockSize)

%get image size
imSize = size(subsampledImg);

dctImg = zeros(imSize);

%for each block
for originRow = 1:blockSize:imSize(1) %block origin point (0, 0) for row
    for originCol = 1:blockSize:imSize(2) %block origin point (0, 0) for column
        %max row for block, generally 0-7. Cut off for end of image
        blockTopRow = blockSize - 1;
        if (originRow + blockTopRow > imSize(1))
            blockTopRow = imSize(1) - originRow;
        end
        %max column for block, generally 0-7. Cut off for end of image
        blockTopCol = blockSize - 1;
end
%max column for block, generally 0-7. Cut off for end of image
blockTopCol = blockSize - 1;
if (originCol + blockTopCol > imSize(2))
    blockTopCol = imSize(2) - originCol;
end

%for each u and v
for u = 0:blockTopRow
    %get c value for u
    cu = 1;
    if (u == 0)
        cu = 0.70710678118;%sqrt(2) / 2 but fast
    end
    for v = 0:blockTopCol
        %get c value for v
        cv = 1;
        if (v == 0)
            cv = 0.70710678118;%sqrt(2) / 2 but fast
        end

        cosSumY = 0;%all the stuff inside the i and j loops
        cosSumCb = 0;%all the stuff inside the i and j loops
        cosSumCr = 0;%all the stuff inside the i and j loops
        %for each pixel in the block
        for i = 0:blockTopRow
            for j = 0:blockTopCol
                dI = double(i);
                dJ = double(j);
                cosSumY = 0;%all the stuff inside the i and j loops
                cosSumCb = 0;%all the stuff inside the i and j loops
                cosSumCr = 0;%all the stuff inside the i and j loops
                %for each pixel in the block
                for i = 0:blockTopRow
                    for j = 0:blockTopCol
                        dI = double(i);
                        dJ = double(j);
                        %cos * cos * f(i, j)
                        cosSumY = cosSumY + cos((2 * dI + 1) * u * pi / 16) * cos((2 * dJ + 1) * v * pi / 16) * int32(subsampledImg(originRow + i, ori
                        cosSumCb = cosSumCb + cos((2 * dI + 1) * u * pi / 16) * cos((2 * dJ + 1) * v * pi / 16) * int32(subsampledImg(originRow + i, c
                        cosSumCr = cosSumCr + cos((2 * dI + 1) * u * pi / 16) * cos((2 * dJ + 1) * v * pi / 16) * int32(subsampledImg(originRow + i, c
                    end
                end

                %calculate final F(u, v)
                dctImg(originRow + u, originCol + v, 1) = cu * cv / 4 * cosSumY;
                dctImg(originRow + u, originCol + v, 2) = cu * cv / 4 * cosSumCb;
                dctImg(originRow + u, originCol + v, 3) = cu * cv / 4 * cosSumCr;
            end
        end
    end
end

%return dct image
f = dctImg;
```

Quantize

- Quantizes DCT images to JPEG standard
  dctImg: DCT image to quantize
  BlockSize: Size of chunks to use
  MUST be in YCbCr format
  Return quantized DCT image

```
%Written by TJ Couch
function f = quantize(dctImg, blockSize)

%get image size
imSize = size(dctImg);

%luminance quantization table
lumQuant = [16  11  10  16  24  40  51  61;
            12  12  14  19  26  58  60  55;
            14  13  16  24  40  57  69  56;
            14  17  22  29  51  87  80  62;
            18  22  37  56  68 109 103  77;
            24  35  55  64  81 104 113  92;
            49  64  78  87 103 121 120 101;
            72  92  95  98 112 100 103  99];

%chrominance quantization table
chromQuant = [17 18 24 47 99 99 99 99;
              18 21 26 66 99 99 99 99;
              24 26 56 99 99 99 99 99;
              47 66 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;];

quantizedImg = zeros(imSize);

for i = 1:imSize(1)
    for j = 1:imSize(2)
        %get the image position wrapped to the block size for the quantization tables
        quantRow = mod(i - 1, blockSize) + 1;
        quantCol = mod(j - 1, blockSize) + 1;

        %divide the dct values by the quantization table values piece-wise
        quantizedImg(i, j, 1) = round(dctImg(i, j, 1) / lumQuant(quantRow, quantCol));
        quantizedImg(i, j, 2) = round(dctImg(i, j, 2) / chromQuant(quantRow, quantCol));
        quantizedImg(i, j, 3) = round(dctImg(i, j, 3) / chromQuant(quantRow, quantCol));
    end
end

%return quantized dct image
f = quantizedImg;
```

jpegDecompress

- Decompress the image matrix according to JPEG decompression

```
% jpegDecompress(image) - decompresses the image matrix according to JPEG decompression
% image: the jpeg compressed image matrix to decompress
% returns decompressed image in rgb format
%
% TJ Couch, Matthew Robertson, Austin Vickers
% JPEG Compression Project
% CS 443 Multimedia
% 4/8/19

%Written by TJ Couch
function f = jpegDecompress(quantizedImg)

%block size 8
blockSize = 8;

%dequantize
dctImg = dequantize(quantizedImg, blockSize);

%2D IDCT
yCbCrImg = idct2D(dctImg, blockSize);

%convert image to yCbCr
rgbImg = ycbcr2rgb(yCbCrImg);

f = rgbImg;
```

Dequantize

- Dequantizes the quantized image to JPEG standards.

```
% dequantize.m - dequantizes the quantized image according to JPEG standard
% quantize(quantizedImg, blockSize)
% quantizedImg: the quantize image matrix to dequantize
% blockSize: the size of the chunks to use
% Note: the image must be in YCbCr format
% returns dequantized DCT image
%
% TJ Couch, Matthew Robertson, Austin Vickers
% JPEG Compression Project
% CS 443 Multimedia
% 4/8/19

%Written by TJ Couch
function f = dequantize(quantizedImg, blockSize)

%get image size
imSize = size(quantizedImg);

%luminance quantization table
lumQuant = [16  11  10  16  24  40  51  61;
            12  12  14  19  26  58  60  55;
            14  13  16  24  40  57  69  56;
            14  17  22  29  51  87  80  62;
            18  22  37  56  68 109 103  77;
            24  35  55  64  81 104 113  92;
            49  64  78  87 103 121 120 101;
            72  92  95  98 112 100 103  99];

%chrominance quantization table
chromQuant = [17 18 24 47 99 99 99 99;
              18 21 26 66 99 99 99 99;
```

```
%chrominance quantization table
chromQuant = [17 18 24 47 99 99 99 99;
              18 21 26 66 99 99 99 99;
              24 26 56 99 99 99 99 99;
              47 66 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;
              99 99 99 99 99 99 99 99;];

dctImg = zeros(imSize);

for i = 1:imSize(1)
    for j = 1:imSize(2)
        %get the image position wrapped to the block size for the quantization tables
        quantRow = mod(i - 1, blockSize) + 1;
        quantCol = mod(j - 1, blockSize) + 1;

        %multiply the dct values by the quantization table values piece-wise
        dctImg(i, j, 1) = quantizedImg(i, j, 1) * lumQuant(quantRow, quantCol);
        dctImg(i, j, 2) = quantizedImg(i, j, 2) * chromQuant(quantRow, quantCol);
        dctImg(i, j, 3) = quantizedImg(i, j, 3) * chromQuant(quantRow, quantCol);
    end
end

%return dequantized dct image
f = dctImg;
```

IDCT 2D

- Turns DCT image into time domain using 2D IDCT according to JPEG standard

```
% idct2D.m - turns DCT image into time domain using 2D IDCT according to JPEG standard
% idct2D(dctImg, blockSize)
% dctImg: the DCT image matrix to IDCT
% blockSize: the size of the chunks to use
% Note: the image must be in DCT format
% returns YCbCr image
%
% TJ Couch, Matthew Robertson, Austin Vickers
% JPEG Compression Project
% CS 443 Multimedia
% 4/8/19

%Written by TJ Couch
function f = idct2D(dctImg, blockSize)

%get image size
imSize = size(dctImg);

yCbCrImg = zeros(imSize);

%for each block
for originRow = 1:blockSize:imSize(1) %block origin point (0, 0) for row
    for originCol = 1:blockSize:imSize(2) %block origin point (0, 0) for column
        %max row for block, generally 0-7. Cut off for end of image
        blockTopRow = blockSize - 1;
        if (originRow + blockTopRow > imSize(1))
            blockTopRow = imSize(1) - originRow;
        end
        %max column for block, generally 0-7. Cut off for end of image
        blockTopCol = blockSize - 1;
        if (originCol + blockTopCol > imSize(2))
            blockTopCol = imSize(2) - originCol;
        end

        %for each pixel in the block
        for i = 0:blockTopRow
            for j = 0:blockTopCol

                cosSumY = 0;%all the stuff inside the u and v loops
                cosSumCb = 0;%all the stuff inside the u and v loops
                cosSumCr = 0;%all the stuff inside the u and v loops

                %for each u and v
                for u = 0:blockTopRow
                    %get c value for u
                    cu = 1;
                    if (u == 0)
                        cu = 0.70710678118;%sqrt(2) / 2 but fast
                    end
                    for v = 0:blockTopCol
                        %get c value for v
                        cv = 1;
                        if (v == 0)
                            cv = 0.70710678118;%sqrt(2) / 2 but fast
                        end

                        dI = double(i);
                        dJ = double(j);
                        % cu * cv / 4 * cos * cos * F(u, v)
                        cosSumY = cosSumY + cu * cv / 4 * cos((2 * dI + 1) * u * pi / 16) * cos((2 * dJ + 1) * v * pi / 16) * int32(dctImg(originRow + u, origi
                        cosSumCb = cosSumCb + cu * cv / 4 * cos((2 * dI + 1) * u * pi / 16) * cos((2 * dJ + 1) * v * pi / 16) * int32(dctImg(originRow + u, ori
                        cosSumCr = cosSumCr + cu * cv / 4 * cos((2 * dI + 1) * u * pi / 16) * cos((2 * dJ + 1) * v * pi / 16) * int32(dctImg(originRow + u, ori
                    end
                end

                %calculate final f~(i, j)
                yCbCrImg(originRow + i, originCol + j, 1) = cosSumY;
                yCbCrImg(originRow + i, originCol + j, 2) = cosSumCb;
                yCbCrImg(originRow + i, originCol + j, 3) = cosSumCr;
            end
        end
    end
end

%return ycbcr image
```

PSNR

- Performs the Peak Signal-to-Noise Ratio (PSNR) formula for error computation.

```
function f = PSNR(origFrame, newFrame)

%src and final image should be the same size
imsize = size(origFrame);

M = imsize(1);
N = imsize(2);

origFrame = im2double(origFrame);
newFrame = im2double(newFrame);

sum = 0;
diff = 0;
for x = 1:M
    for y = 1:N
        diff = origFrame(x,y,1) - newFrame(x,y,1);
        diff = diff + origFrame(x,y,2) - newFrame(x,y,2);
        diff = diff + origFrame(x,y,3) - newFrame(x,y,3);

        sum = sum + diff^2;
    end
end

MSE = (1/M*N)*sum;
%%typecast(MSE, 'double');

result = 20*log10(255/sqrt(MSE));

f = result;
```

pixelWiseError

- Performs the pixel-wise Error between the original frame and output frame.

```
function f = pixelWiseError(sourceImg, finalImg)

%src and final image should be the same size
imsize = size(sourceImg);

%create blank image for the difference
errorImg = uint8(zeros(imsize));

%create an x by y array to add all the differences together
totalError = uint8(zeros(imsize(1),imsize(2)));

for i = 1:imsize(1)
    for j = 1:imsize(2)
        errorImg(i,j,1) = sourceImg(i,j,1) - finalImg(i,j,1);
        errorImg(i,j,2) = sourceImg(i,j,2) - finalImg(i,j,2);
        errorImg(i,j,3) = sourceImg(i,j,3) - finalImg(i,j,3);

        totalError(i,j) = errorImg(i,j,1) + errorImg(i,j,2) + errorImg(i,j,3);
    end
end

f = totalError;
```