NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Advanced Topics in Digital Image Processing

Session 2 – JPEG Compression

Objective:

The objective of this class is to give students contact with JPEG image codification algorithm and to visualize the consequences of image compression, namely, how it affects image quality and compression artefacts.

Preparation:

1 – Copy to your python project directory the file, ***JPEGCompression.py***, available in the moodle, which contains the following functions:

- **blockProcessing**- where the JPEG compression / decompression functions should be implemented;

- **GetQuantificationMatrix** - Returns an 8 x 8 quantization matrix to the luminance or chrominance channels and to which a divisive factor can be applied to increase the compression ratio (100 (less compression)-0 (higher compression)).

JPEG Compression and Decompression

2 – With the image supplied (usb_32x32.png) or other with a width multiple of 8 pixels, apply the following JPEG codification steps (recommended OpenCV functions below):

1. Transform the colour space from BGR to YCbCr and split channels

```
cv.cvtColor( … , cv.COLOR_BGR2YCrCb)
imgChannelY,imgChannelCb,imgChannelCr = cv.split( … )
```

2. Downsampling of Cb and Cr components (optional)

```
imgChannelC = cv.resize( … ,None,fx=0.5, fy=1, interpolation=cv.INTER_AREA )
```

3. Analyse the image in 8x8 pixel blocks (be aware that Y component is a full size image and downsampled Chrominance components have half the width)
   a) Block splitting (copy a region of interest to a new matrix)

```
imgChannelBlock = imgChannelY[startX : endX, startY : endY]
```

Compression

   b) Convert block to *float* format and subtract the DC component (128) (*np.float32*)
   c) Apply the Discrete Cosine Transform (DCT) (*cv.dct*)
   d) Coefficients Quantization (*np.divide*)
   e) Coefficients rounding *(np.round)*

Decompression

f) Coefficients recovering (*np.multiply*)

g) Apply the Discrete Cosine Inverse Transform

```
result = cv.dct(result, None, flags=cv.DCT_INVERSE)
```

h) Add DC component (128), clip to 0..255 and convert to byte (*np.clip* and *np.uint8*)

i) Copy the block to the fullsize image (use the same procedure as in a))

4. Resize Chrominance images to fullsize

```
imgChannelC = cv.resize( … ,None,fx=2, fy=1, interpolation=cv.INTER_AREA )
```

5. Merge the Y, Cb and Cr components and convert to BGR

```
result = cv.merge((imgChannelY,imgChannelCb,imgChannelCr))
result = cv.cvtColor(result, cv.COLOR_YCrCb2BGR)
```

3 – Compare the differences between the original image and the recovered one, using function showSideBySideImages that uses pyplots from Matplotlib. (ex. compFactor = 30)

4 – Try other compression factors for the quantization matrix and comment on the recovered image.

5 – (optional) Codify (in paper or excel) the luminance block (2,0) using JPEG coding rules, namely Huffman for the DC component and the *Run Length Encoding* (RLE) for the AC component.