REPORT

COL783 – ASSIGNMENT 1

RAVINDRA KUDCHADKAR **ABHINAV GUPTA**

2018TT10941 **2018TT10868**

PART 1:

I) *Popularity Algorithm*: Made a histogram using the input image and then chose the top k colors (the k colors with the highest frequency). Then the image is quantized by using these k colors by mapping the color of the pixels in the image to the nearest color. This algorithm usually cannot quantize well for a small k or when the colors range in the image is very wide. It doesn't choose those colors that are in the sparse regions of the color-space.

```
Usage:
python popularity.py --input INPUT_IMAGE_PATH --output
OUTPUT_PATH --colors NO_OF_COLORS
```

II) *Median Cut Algorithm*: Used the color channel with the highest variance to split the whole bin of colors into 2 bins at the median of this color channel. Recursively performed this procedure on each bin obtained until $\log_2(k)$ number of bins were formed. The centroid of each of these bins was then used to define the colormap. Then the image is quantized by using these k colors by mapping the color of the pixels in the image to the nearest color.

```
Usage:
python popularity.py --input INPUT_IMAGE_PATH --output
OUTPUT_PATH --colors NO_OF_COLORS
```

III) *Floyd Steinberg dithering*: Used to quantize and dither at the same time in a single top to bottom pass. Dithering is basically the propagation of the quantization error of a pixel to the neighbors of the pixel. This method gives a reasonably good output given that the appropriate colors

```
Usage:
python dithering.py --input INPUT_IMAGE_PATH --output
OUTPUT_PATH --quantizer [median_cut or popularity] --colors
NO_OF_COLORS
```

Observations:

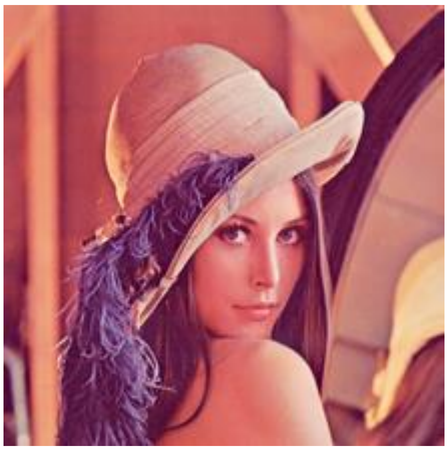
Fig 1: Original Image of shape 220 x 220 x 3

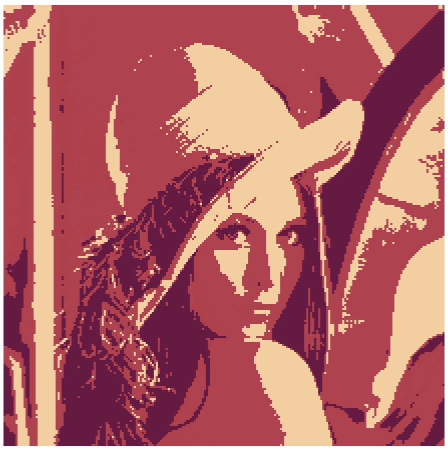
Fig 2: Popularity with 4 colors

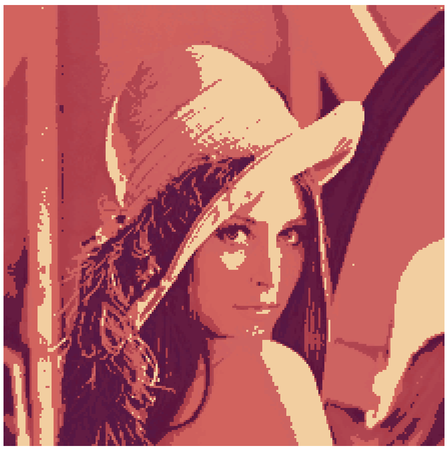
Fig 3: Popularity with 32 colors

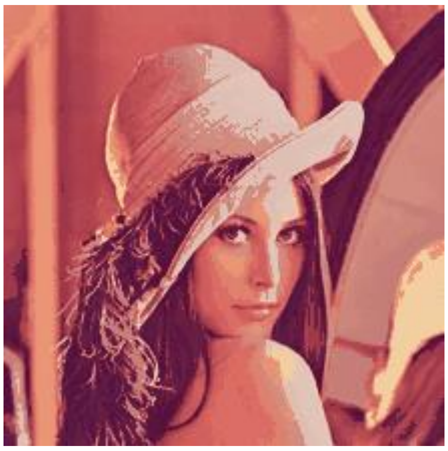
Fig 4: Popularity with 256 colors

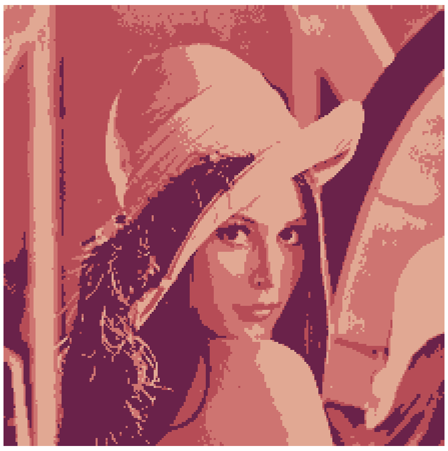
Fig 5: Median Cut with 4 colors

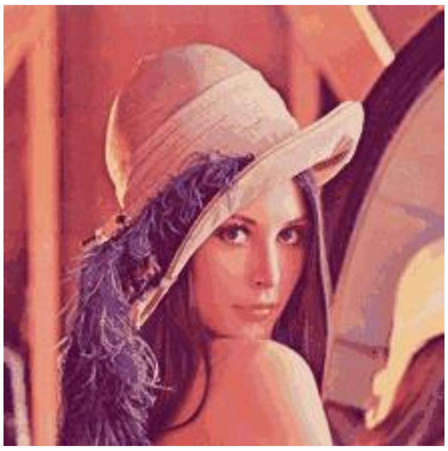
Fig 6: Median Cut with 32 colors

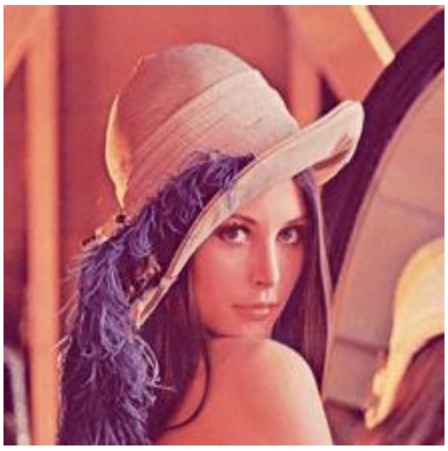
Fig 7: Median Cut with 256 colors

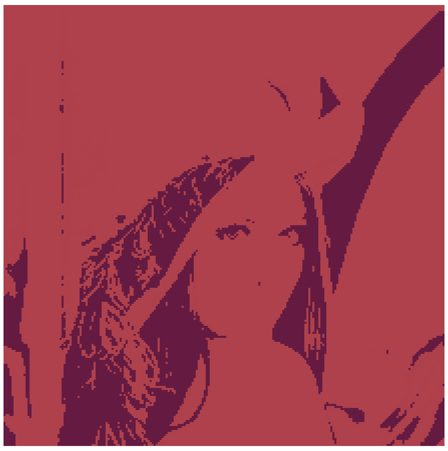
Fig 8: Dithering with Popularity (4 colors)

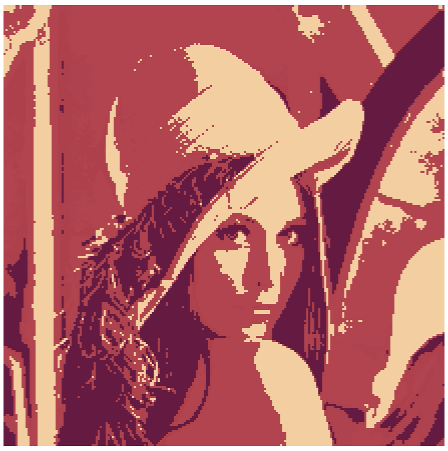
Fig 9: Dithering with Popularity (32 colors)

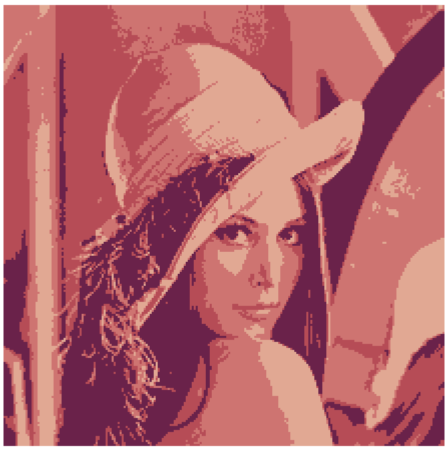
Fig 10: Dithering with Median Cut (4 colors)

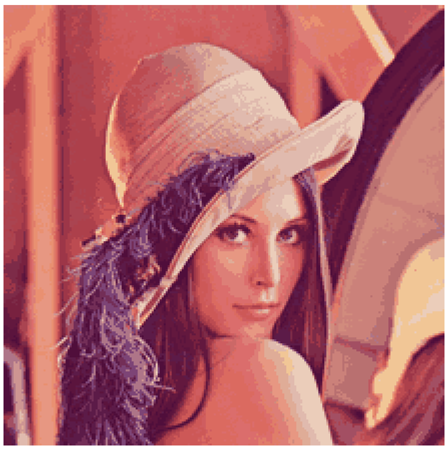
Fig 11: Dithering with Median Cut (32 colors)

Inference

1) We can see that median cut performs better than popularity. Even so, median cut with 32 colors outperforms popularity at 256 colors.
2) Median cut is able to find better colors to quantize with.
3) Dithering works best with median cut even at 4 colors due to the above reason.

PART 2:

I) *Global Color Transfer*: Took the colored source and converted it to LAB format in OpenCV. Then split this image's channels. Input the target image. Performed luminance remapping of the source's L channel w.r.t. the target image by equating their mean and standard deviation, then rescaling accordingly. Took jittered sampling (with 256x256 blocks) of the source image, and used these samplings to create A and B channels for the target image (by matching the luminance value from the samplings. Merged these channels and converted back to BGR format.

```
Usage:
python global.py --source SOURCE_IMAGE_PATH --target TARGET_IMAGE_PATH --output OUTPUT_PATH
```

II) *Swatch Color Transfer*: Took the colored source and converted it to LAB format in OpenCV. Then split this image's channels. Input the target image. Created an interface to drag-and-drop swatches from the source and target images. Performed luminance remapping just on the swatches and then transferred color to the target swatch using global transfer from source swatch. Took jittered sampling of the newly formed colored swatch and saved it corresponding to a key. Looped over multiple swatches. Used texture mapping with Error formula to colorize entire image using the swatches. Used a window mask sized n*n (n is odd) around the pixel to be colored for better matching

```
Usage:
python swatch.py --source SOURCE_IMAGE_PATH --target TARGET_IMAGE_PATH --output OUTPUT_PATH --window_size SIZE_OF_MASK_N
```

Observations:

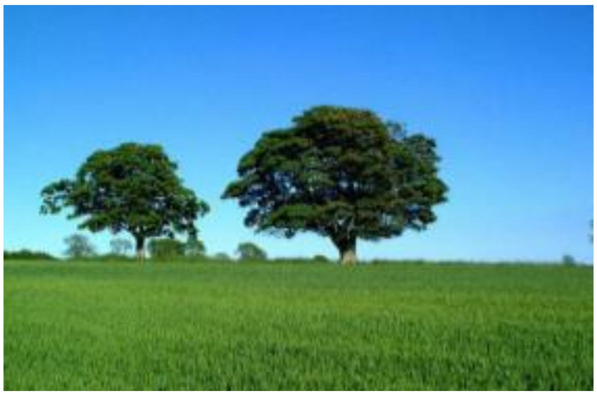
Fig 1: Source Image (colored)

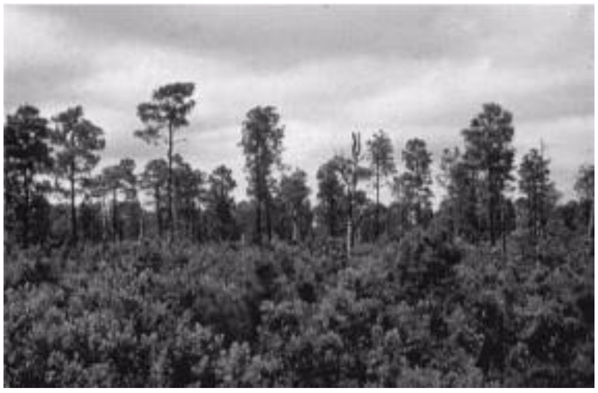
Fig 2: Target Image (gray scale)

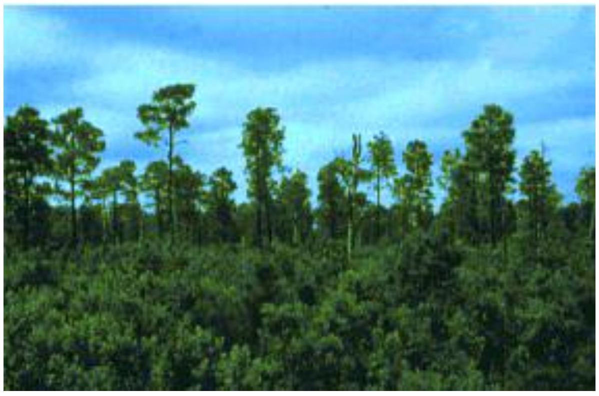
Fig 3: Global Color Transfer

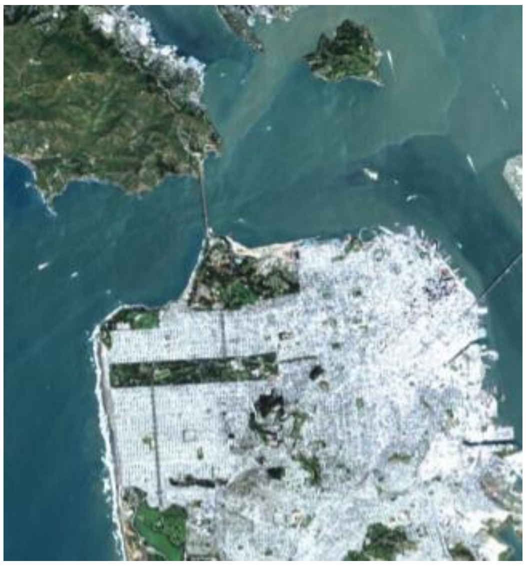
Fig 4: Source Image (colored)

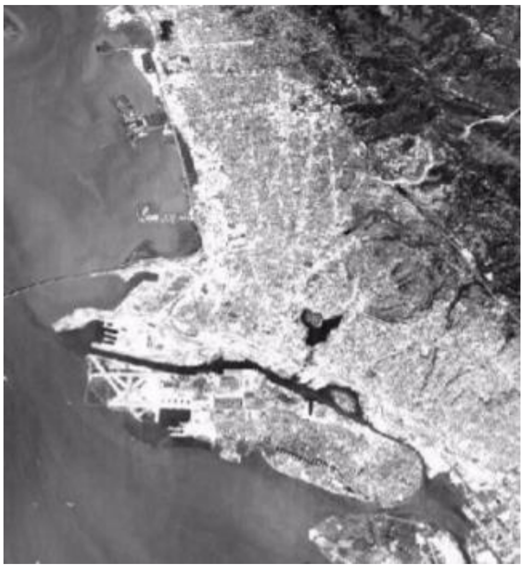
Fig 5: Target Image (gray scale)

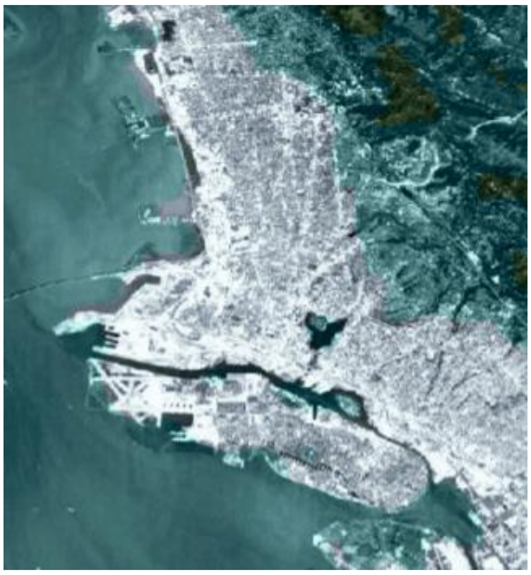
Fig 6: Swatch Color Transfer with Mask-Size 7

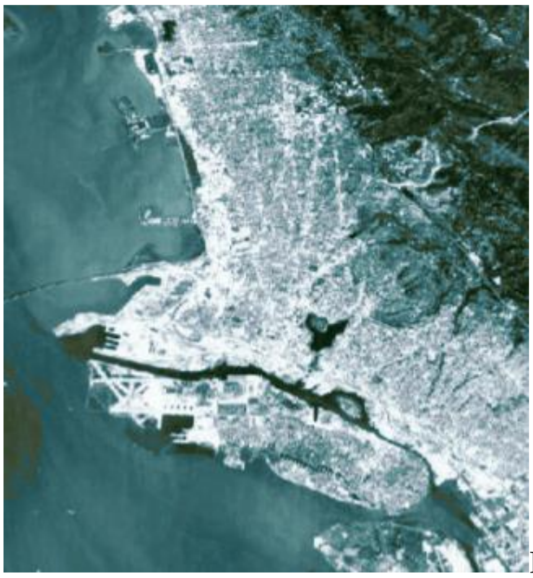
Fig 7: Swatch Color Transfer with Mask-Size 23

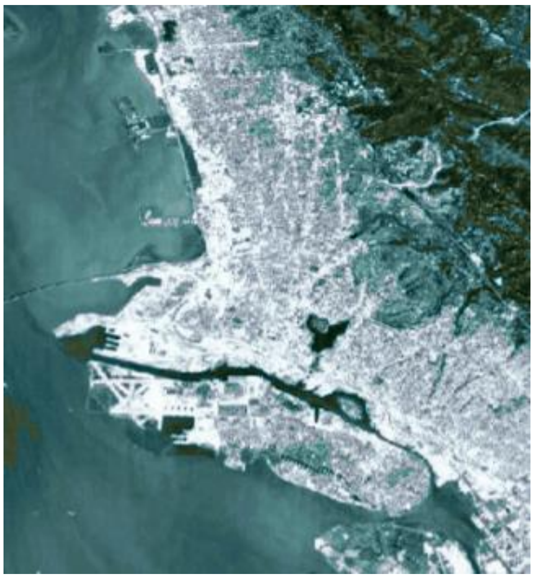
Fig 8: Swatch Color Transfer with Mask-Size 45

Inference

I) Global transfer performs better than swatches for images with more number of regional differences in terms of color and luminance.

II) Swatch transfer performs better when the luminance overlapping between regions is large.

III) Generally, the color transfer becomes observably better with increasing mask size during texture matching in swatch method; however the trend is reversed in image where more concentration of luminance overlapping occurs.

IV) Time taken by Swatch method increases with increase in number of swatches.

V) Global process is significantly faster than swatch process in terms of back end processing.