Exercise Session 6

Image Processing

Rolf Ingold, Anna Scius-Bertrand, Najoua Rahal
DIVA Group, University of Fribourg, Switzerland

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Reminder: assignment 4 et 5

- Tonight: assignment 4, local operators (convolution)

- Next week: assignment 5, global operators (Fourrier et Hadamard transform)
  - Deadline: November 2nd, end of the day

- Any questions?

Assignment 3: Histogram equalization

- Separate the three rgb channels: r, g, b = image.split()
- Merge channel : image = Image.merge("RGB”, (r, g, b))
- HSL algorithm: do not forget to normalize channels
- Store the result in an array.
- Histogram equalization on RGB channels create a color distortion. On a HSL image the colors are brighter and the contrast more visible.

Separate an RGB image into the three channels of the HSL color space (H, S, L)

Exercise 1b)

In [3]:

```python
def rgb_to_hsl(rgb_data):
    # New reference, with floats instead of ints
    new_hsl = rgb_data.astype(np.dtype('float64'))

    for i, row in enumerate(rgb_data):
        for j, column in enumerate(row):
            # Normalize
            pixel = column / 255
            c_max = np.max(pixel)
            c_min = np.min(pixel)
            delta = c_max - c_min
            argmax = np.argmax(pixel)
            red, green, blue = pixel

            # Compute hue
            if delta == 0:
                hue = 0
            elif argmax == 0:
                hue = 60 * (((green - blue) / delta) % 6)
            elif argmax == 1:
                hue = 60 * (((blue - red) / delta) + 2)
            else:
                hue = 60 * (((red - green) / delta) + 4)

            # Compute lightness
            lightness = (c_max + c_min) / 2

            # Compute saturation
            if delta == 0:
                saturation = 0
            else:
                saturation = delta / (1 - np.abs(2 * lightness - 1))

            new_hsl[i][j] = np.array([hue, saturation, lightness])

    return new_hsl
```

Reconstruct an RGB image from the H, S, L channels

Exercise 1c)

In [4]:

```python
def hsl_to_rgb(hsl_data):
    new_rgb = hsl_data.astype(np.dtype('uint8'))

    for i, row in enumerate(hsl_data):
        for j, column in enumerate(row):
            hue, saturation, lightness = column

            # Compute the constant need in the calculation for RGB
            c_const = (1 - np.abs(2 * lightness - 1)) * saturation
            x_const = c_const * (1 - np.abs((hue / 60) % 2 - 1))
            m_const = lightness - c_const / 2

            # Assign RGB values depending on the hue
            if 0 <= hue and hue < 60:
                red = c_const
                green = x_const
                blue = 0
            elif 60 <= hue and hue < 120:
                red = x_const
                green = c_const
                blue = 0
            elif 120 <= hue and hue < 180:
                red = 0
                green = c_const
                blue = x_const
            elif 180 <= hue and hue < 240:
                red = 0
                green = x_const
                blue = c_const
            elif 240 <= hue and hue < 300:
                red = x_const
                green = 0
                blue = c_const
            elif 300 <= hue and hue < 360:
                red = c_const
                green = 0
                blue = x_const

            pixel = np.array([red, green, blue])
            pixel = (pixel + m_const) * 255
            new_rgb[i][j] = pixel.astype(np.dtype('uint8'))

    return new_rgb
```

Exercise 2a)

In [7]:

```python
def equal_histo(channel):
    max_value = int(np.ceil(np.max(channel)))
    equal_channel = copy.deepcopy(channel)

    # Compute the histogram
    unique, counts = np.unique(channel, return_counts=True)
    value_to_counts = dict(zip(unique, counts))
    # This will ensure to also count possible values that could not appear in the image
    histogram = []
    for i in range(max_value+1):
        try:
            histogram.append(value_to_counts[i])
        except KeyError:
            histogram.append(0)

    # +1 in slicing otherwise it is excluded
    cumul_histogram = np.array([np.sum(histogram[:i+1]) for i in range(len(histogram))])
    cumul_min = np.min(cumul_histogram)
    cumul_max = np.max(cumul_histogram)
    assert cumul_max == len(channel) * len(channel[0]), f"Wrong numbers, {cumul_max} VS {len(channel) * len(channel[0])}"

    for i, row in enumerate(channel):
        for j, column in enumerate(row):
            new_value = np.round((cumul_histogram[column] - cumul_min) / (cumul_max - cumul_min) * max_value)
            equal_channel[i][j] = new_value

    return equal_channel
```

Exercise 2b)

In [8]:

```python
images_path = Path('Images_greyscale')
output_path = Path('output')
output_path.mkdir(exist_ok=True)
for img_path in images_path.iterdir():
    img_data = np.asarray(Image.open(img_path))
    new_data = equal_histo(img_data)
    new_img = Image.fromarray(new_data)
    new_img.save(output_path / f"{img_path.stem}_equal.png")
```

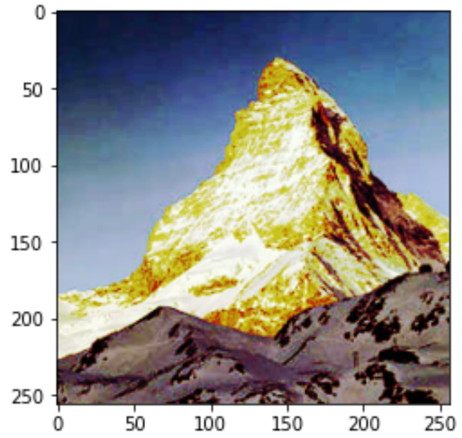
UNI
FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Color Histogram Equalization

Exercise 3a)

In [9]:

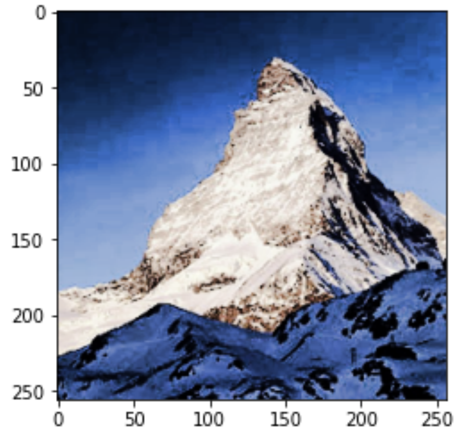
```
rgb_path = Path('Cervin.png')
rgb_img = Image.open(rgb_path)
rgb_data = np.asarray(rgb_img)

new_channels = [equal_histo(rgb_data[:,:,i]) for i in range(3)]
new_rgb_data = np.dstack(tuple(new_channels))
new_rgb_image = Image.fromarray(new_rgb_data)
plt.imshow(new_rgb_image)
new_rgb_image.save(output_path / f"{rgb_path.stem}_rgb_equal.png")
```



Exercise 3b)

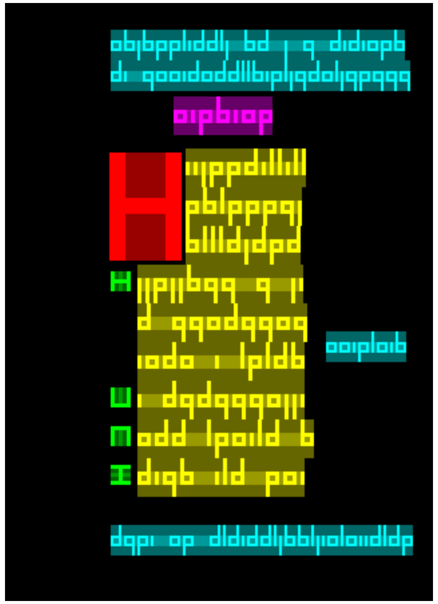
In [10]:

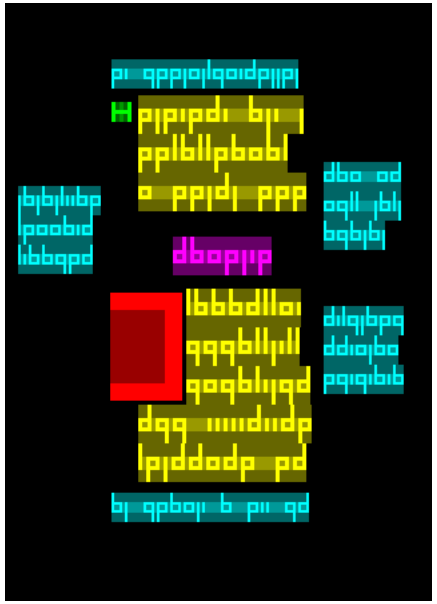
```
rgb_path = Path('Cervin.png')
rgb_img = Image.open(rgb_path)
rgb_data = np.asarray(rgb_img)
hsl_img = rgb_to_hsl(rgb_data)

# Discretize the L channel from 0 to 256
max_l = np.max(hsl_img[:,:,2])
norm_l_channel = np.array(hsl_img[:,:,2] * 256 / max_l, dtype=np.dtype('uint8'))
new_l_channel = equal_histo(norm_l_channel)

# Return to the interval [0,1[ for L
new_l_channel = np.array(new_l_channel * max_l / 256, dtype=np.dtype('float32'))
new_channels_hsl = [hsl_img[:,:,0], hsl_img[:,:,1], new_l_channel]

# Stack new channels and convert back to RGB
new_hsl_data = np.dstack(tuple(new_channels_hsl))
new_rgb_data = hsl_to_rgb(new_hsl_data)
new_rgb_image = Image.fromarray(new_rgb_data)
plt.imshow(new_rgb_image)
new_rgb_image.save(output_path / f"{rgb_path.stem}_hsl_equal.png")
```

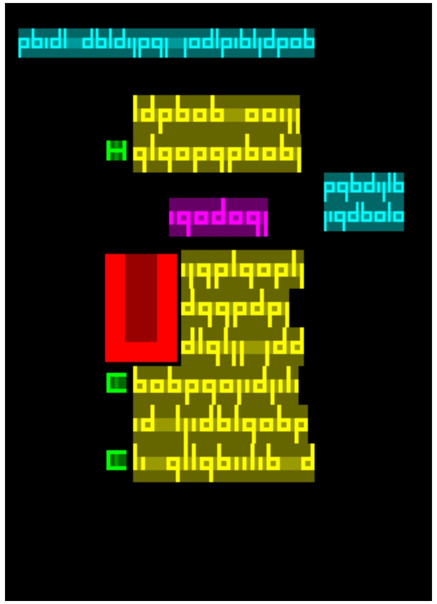


Assignment 6: Image compression

- Goal: Realize a lossless image compression and compare your result with GIF compression for the sample image provided on Ilias (`https://ilias.unibe.ch/goto_ilias3_unibe_fold_2572742.html`).
- Analyze the three images present on ILIAS to decide which compression strategy is the most promising one. Below, we suggest two such strategies, namely 2D run-length encoding and quadtree representation. Choose one of them, or yet another strategy of your own design. Regardless of the compression method you choose, write an algorithm to decompress the image back to the original image







2D run-length encoding

- The idea is to represent the image as a sequence of horizontal bands that are composed of several identical rows of pixels.
- Within each band, the repeated row of pixels is represented by means of runs of pixels with the same color.
- Finally, entropic encoding is applied to reduce the number of bits needed to encode the different run lengths.
- (a) Determine the width w, height h, and number of distinct colors c of the input image.
- (b) Represent the image as a sequence of bands $(h_1, r_1), (h_2, r_2), \ldots$, where $h_i \in [1, h]$ the height of the band, r_i the repeated row, and $\sum h_i = h$.
- (c) Represent each row r_i as a sequence of runs $(c_1, w_1), (c_2, w_2), \ldots$, where $c_i \in [1, c]$ the color of the run, $w_i \in [1, w]$ the length of the run, and $\sum w_i = w$ (see lecture slides).
- (d) Instead of using a fixed number of bits to represent the run lengths w_i, use entropic coding to represent the most frequent lengths with only few bits and less frequent lengths with more bits, for example by means of Huffman codes (see lecture slides).

Quadtree representation

- The idea is to subdivide the image into quadratic cells and then represent each cell by means of a quadtree.
- (a) Determine the width w, height h, and number of distinct colors c in the input image.
- (b) Cover the image with a grid of quadratic cells, for example cells of 64 × 64 pixels.
- (c) For each cell, build a quadtree, which contains non-terminal nodes for heterogeneous regions with different colors, and terminal nodes for homogeneous regions with the same color $c_i \in [1, c]$ (see lecture slides).
- (d) Represent each quadtree as a sequence of symbols for non-terminal and terminal nodes, for example using a depth-first traversal (see lecture slide

Compression result

- (a) For the chosen compression strategy, calculate the number of bits needed to represent the sample image after compression.
- (b) Implement the decompression algorithm to recover the original image

- Hand-in
- Submit on ILIAS one and only one folder containing:
  - The response to question 3a).
  - A text file, with your name, github link and a brief explanation of your algorithm

Questions?