Computer Vision 1: Assignment 1
(Due date: 25.10.2021)

Submission instructions:

- For each programming task, submit a `.py` source code file. Do not include any images or data files you used.
- For each pen & paper task, submit a `.pdf` file. Type your solution in LaTeX, Word, or another text editor of your choice and convert it to PDF. Do not submit photographs or scans of handwritten solutions!
- In all submissions, include at the top names of all students in the group.
- Choose one person in your group that submits the solution for your entire group.

General hints for programming tasks:

- Use `imageio` for loading and saving images. See `https://imageio.readthedocs.io/en/stable/`
- Prefer to use `numpy` for array manipulation and basic image arithmetic.
- Use `scikit-image` for image processing tasks. See `https://scikit-image.org/`
- Many image processing functions and methods referred to throughout the exercises are from `scikit-image`.

Task 1: Appearance based classification (programming)

In this task, we will train and test an image classifier based on the nearest neighbour method. To train the classifier, proceed as follows:

- Download the CIFAR-10 dataset (Python version) at `https://www.cs.toronto.edu/~kriz/cifar.html`.
- Read the first data batch in file `data_batch_1` as instructed on the webpage.
- For each of the classes "automobile", "deer", and "ship" (labels 1, 4, and 8, respectively), extract the 30 first images.
- Calculate and store the grayscale histograms of all of these images. First convert the images to grayscale by the averaging method. That is, for every pixel, the grayscale value is calculated as $(R+G+B)/3$. Then calculate the histogram using a **fixed** set of 51 bins covering the range $0 - 255$, each bin has length 5. **Do not take the automatically chosen bins by numpy.histogram**, rather create the vector for bins yourself and use it for every image. Think about why this is necessary. Hint: what happens if bins are chosen automatically and you compare histograms as in the test phase below?

Then, test the classifier as follows:

- Read the test batch in file `test_batch`. For the same classes as above, extract the 10 first images.
- Calculate the histograms of all the images in the same way as above, using the same bins vector.

- Classify each image in the test set by finding its nearest neighbour in the training set. For each test image:

  1. Calculate the Euclidean (L_2) distances of the histogram of the test image and the histogram of *every* training image. If you have n training images, you should calculate n distances for every test image.
  2. Classify the test image by finding the minimum of the distances. Once you find the minimum distance, the predicted class of the test image is the class of the nearest training image.

- Calculate the classification accuracy as the ratio of the number of correctly classified testing images and the total number of testing images.

- Change your code by modifying the bins vector to have a different number of bins, for examples 2, 10, or 255, rerun your code and record how the classification accuracy changes.

Hints:

- When you have loaded the data (in the dictionary called `train`), this is how you can access the color channels of the `i`'th image.

```
raw_data = train["data"][i,:]
red = raw_data[:1024].reshape((32,32))
green = raw_data[1024:2048].reshape((32,32))
blue = raw_data[2048:].reshape((32,32))
```

- Be careful with the data types. Most images you load will have `uint8` data type, which will overflow if you sum them. Before arithmetic operations, always prefer to convert the image array to a floating point data type.

Task 2: Computing surround-center contrasts using integral images (programming)

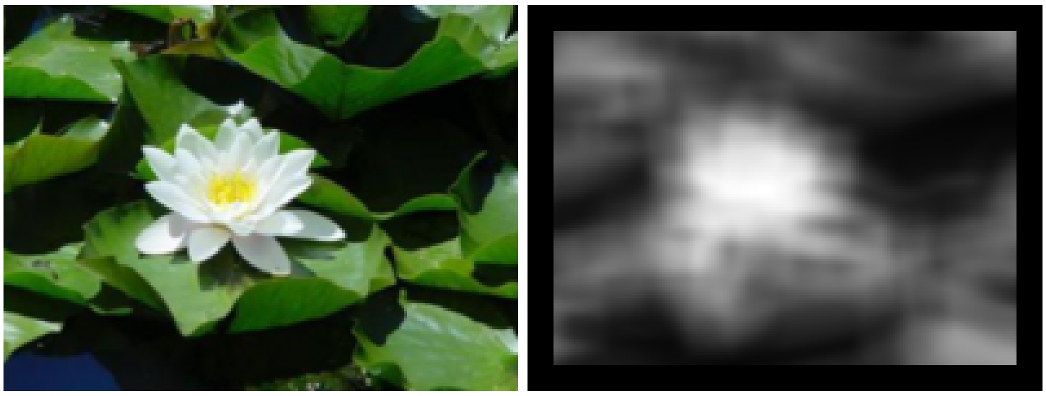


Figure 1: An image (left) and its surround-center contrast map (right).

A saliency map highlights what visually stands out in an image and grabs our attention (e.g. the white flower amidst the green leaves in Fig. 1). One step in computing a saliency map involves calculating center-surround and surround-center contrasts using integral images. An example of a surround-center contrast map is shown in Fig. 1 (right).

Replicate calculation of the surround-center contrast map by following the following steps. You will need several methods from `scikit-image`.

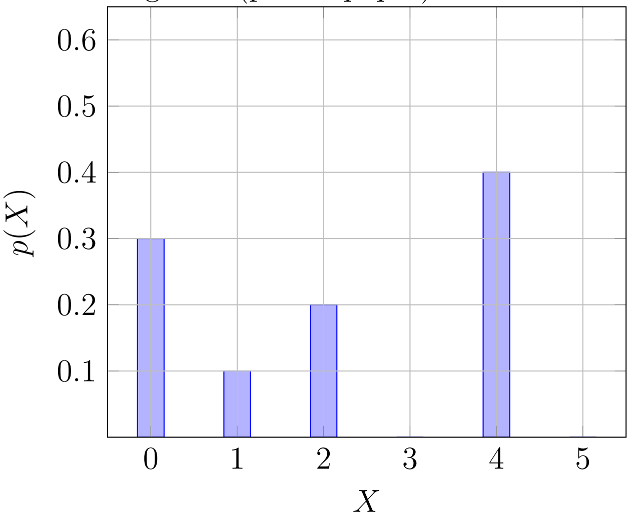
- Read the image "visual_attention_ds.png" in Moodle. Using `rgba2rgb` and `rgb2gray` convert it to grayscale. Compute the integral image using `integral_image`.
- Use sizes 11×11 and 21×21 for the center and surround windows, respectively. For each valid pixel around which both the center and surround windows fit, compute the center average c and surround average s from the integral image using `integrate`.
- Subtract the center average from the surround average to get a contrast value: $s - c$
- Place each contrast value in its corresponding position in a new array.
- Display the contrast map created. Your result should match Fig. 1 (right).

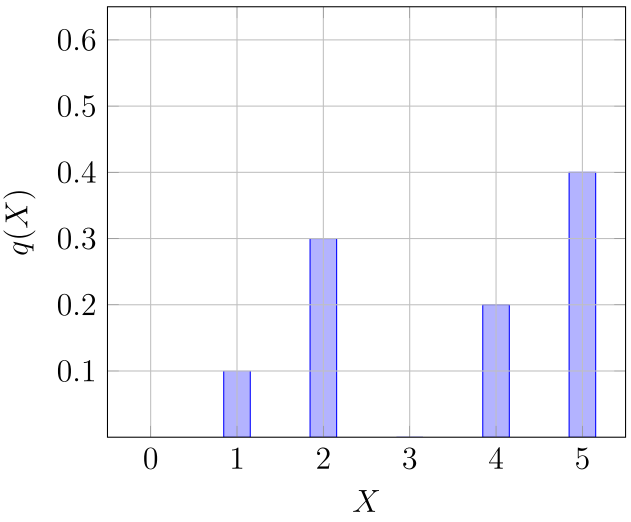
Note how the contrast map highlights the location of the flower. Now change the window sizes as follows and display the new contrast map:

- 3×3 for the center, and 7×7 for the surround window
- 31×31 for the center window and 51×51 for the surround window.

How does the resulting contrast map change? Write brief answers as a comment in your code.

Task 3: Histograms (pen & paper).





a) Consider the normalized histogram indicated by $p(X)$ in the figure above on the left.
   - Find the expected value of X, i.e., $\mathbb{E}[X]$.
   - Write down a table that describes the cumulative histogram of X. The table should have two rows, one for the bins and the other for the cumulative histogram value.

b) Calculate the L_1 (Manhattan) distance between the normalized histograms $p(X)$ (above left) and $q(X)$ (above right).