Exercises for
Introduction to Machine Learning
Linda-Sophie Schneider, Johannes Enk, Fengyu Han, Gia Long Nguyen, Lukas Huettner, Yangbin Peng
Task 1: deadline: 01. June 2025

Exercise 01: Thresholding and Histograms

Exercise 1 Otsu's Thresholding

Many thresholding algorithms exist for image binarization where the goal of this process is to split an image into two parts. For example in document processing, the document has to be split into page and writings. One widely used approach is Otsu's thresholding, which was also discussed in the lecture. In this task, you have to implement Otsu's thresholding according to the lecture slides. Use the code skeleton provided. You are only allowed to use the predefined imported modules (in this case - numpy only). The function should work by running the given *main.py* module.

The steps of Otsu's are as follows:

(a) Create Histogram from image with 256 bins

(b) For every bin assume that this bin is the threshold for binarization and calculate for each threshold:

   (a) Calculate p_0, p_1

   (b) Calculate μ_0, μ_1

   (c) Calculate between class variance $\sigma^2_{inter} = p_0\, p_1\, (\mu_1 - \mu_0)^2$

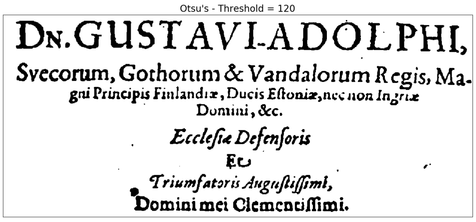
(c) The threshold (= bin) with the highest between class variance is the threshold

(d) Binarize the image (0,255)

Output Example:

Original

Otsu's - Threshold = 120



Exercise 2 Histogram Equalization

You made a picture of your wonderful cat. Unfortunately, the stupid camera can not make good pictures of your cat, as your cat has only few contrast in her awesome fluffy fur. So you decide to enhance the histogram of the picture. In introduction to machine learning you learned that histogram equalization is a good choice, so you choose to implement it:

(a) Load `hello.png` into a numpy array, using, for example, PIL or opencv

(b) Compute the intensity histogram of your cat image, for pixel values between 0 and 255

(c) Compute its cumulative distribution function C

(d) Change the gray value of each pixel:

$$pixelvalue_{new} = g(pixelvalue_{old}) = \lfloor \frac{C(pixelvalue_{old}) - C_{min}}{1 - C_{min}} \cdot 255 \rfloor \quad (1)$$

(e) Save the result as `kitty.png`

(f) During the submission, explain why the background looks like this, i.e., with clear intensity boundaries

Important: Do not use any function that computes histograms automatically, histogram equalization, or cumulative sums. Any computation on the pixel values, or any modification of their values, must be completely done with your own code.[1]

Note: If the histogram is computed correctly, then the sum of its 90 first values will be equal to 249.

Note: If the cumulative distribution is computed correctly, then the sum of its 90 first values will start with the following: 0.001974977

Note: Your result should look as the picture below. Did you know that this cute cat is extremely intelligent and can open doors?

Histogram equalized image:

[1] We would really like to make sure that you all know how to produce histograms, compute cumulative distributions, and do an histogram equalization.

Exercise 3 Image Noise

When working with a limited number of training images, adding random transformations such as image noise can effectively increase your dataset. This exercise focuses on implementing four types of noise using *np.random*:

(a) **Gaussian Noise**: Gaussian noise follows a normal distribution characterized by two parameters:

- *Mean*: The average value around which noise is centered.
- *Variance*: Determines the spread or intensity of the noise around the mean.

(b) **Poisson Noise**: Controlled by the parameter λ, representing the average number of events occurring within a fixed interval. Each pixel intensity is randomly varied according to probabilities calculated from λ.

(c) **Salt-and-Pepper Noise**: This noise simulates defective pixels, randomly setting some pixels either to 0 (pepper) or 255 (salt). Implement this using two thresholds:

- Threshold for *salt* (white pixels).
- Threshold for *pepper* (black pixels).

(d) **Uniform Noise**: Draw random values uniformly from a specified interval. You must define this interval clearly and generate random pixel values accordingly.

Note: After creating the noise and adding it to your image, don't forget to clip your pixel intensity values to ensure they remain within the valid range.