Lab 10

Basic Image Processing
Fall 2018

Markov Random Fields in Image Segmentation

- Segmentation as pixel labeling
- Probabilistic approach
  - Segmentation as MAP estimation
  - Markov Random Field (MRF)
  - Gibbs distribution & Energy function
- Classical energy minimization
  - Simulated Annealing
  - Markov Chain Monte Carlo (MCMC) sampling
- Example MRF model & Demo
- Parameter estimation (EM)

MRF slides adopted © Zoltan Kato, University of Szeged, http://www.inf.u-szeged.hu/~kato/

Spaces & dimensions

It is important to understand all the spaces and their dimensionalities in this task.

Consider an S space: $S \subset \mathbb{R}^n$. In this space, every $s \in S$ is a pixel.

Each pixel of the image has a feature vector f_s and for the whole image we have a set of features: $F = \{f_s : s \in S\}$

We define a set of labels too: Λ . In this set, every $\lambda \in \Lambda$ is a label (e.g. ‘class1’,’class2’,...)

To each pixel of the image we assign a label $\omega_s \in \Lambda$ and for the whole image we have a set of labels: $\omega = \{\omega_s : s \in S\}$

The Ω set is the set of all possible ω labelings. For an $N \times M$ image there are $|\Omega| = |\Lambda|^{NM}$ possible good labelings

Problem formulation

So which one is the right segmentation? To find an answer to this question, we define a probability measure on the set of all possible labelings:

$$P(\omega|f)$$

This probability measure gives the likelihood of ω being a good labelling given f.

We look for a labeling $\hat{\omega}$ for which $P(\hat{\omega}|f)$ is maximal. Using the Bayes theorem:

$$P(\omega|f) = \frac{P(f|\omega)P(\omega)}{P(f)} \propto P(f|\omega)P(\omega)$$

where $P(f)$ does not depend on the chosen labeling.

Now we just have to define $P(f|\omega)$ and $P(\omega)$.

Energy-minimization problem

So our problem is

$$\hat{\omega} = \arg\max_{\omega} \Big[P(\omega|f) \Big] = \arg\max_{\omega} \Big[P(f|\omega) P(\omega) \Big]$$

Let's transform this to an energy-minimization problem (details in lecture slides):

$$\hat{\omega} = \arg\min_{\omega} \Big[\underbrace{-\log\big(P(f|\omega)\big)}_{\text{posterior}} \underbrace{-\log\big(P(\omega)\big)}_{\text{prior}} \Big]$$

Posterior

In the energy-minimization problem the posterior (**feature-dependent**) term is

$$-\log\left(P(f|\omega)\right) = \sum_{s\in S} -\log\left(P(f_s|\omega_s)\right)$$

where every label λ is represented by a Gaussian distribution $\mathcal{N}(\mu_\lambda, \sigma_\lambda)$

Therefore, the posterior probability for a pixel is

$$-\log\left(P(f_s|\omega_s)\right) = \log\left(\sqrt{2\pi}\,\sigma_{\omega_s}\right) + \frac{1}{2}\left(\frac{f_s - \mu_{\omega_s}}{\sigma_{\omega_s}}\right)^2$$

Prior

In the energy-minimization problem the prior (**neighbor-dependent**) term is

$$-\log\big(P(\omega)\big) = \sum_{s,r\in S} V_2(\omega_s, \omega_r)$$

where

$$V_2(\omega_s, \omega_r) = \begin{cases} +\beta & \text{if } \omega_s, \omega_r \text{ are neighbors and } \omega_s = \omega_r \\ -\beta & \text{if } \omega_s, \omega_r \text{ are neighbors and } \omega_s \neq \omega_r \\ 0 & \text{if } \omega_s, \omega_r \text{ are not neighbors} \end{cases}$$

The final form of the Energy function

$$\hat{\omega} = \arg\min_{\omega} \Bigg[\underbrace{\sum_{s \in S} \left[\log\left(\sqrt{2\pi}\,\sigma_{\omega_s}\right) + \frac{1}{2}\left(\frac{f_s - \mu_{\omega_s}}{\sigma_{\omega_s}}\right)^2 \right]}_{\text{posterior}} + \underbrace{\sum_{s,r \in S} \left[V_2(\omega_s, \omega_r) \right]}_{\text{prior}} \Bigg]$$

where

$$V_2(\omega_s, \omega_r) = \begin{cases} +\beta & \text{if } \omega_s, \omega_r \text{ are neighbors and } \omega_s = \omega_r \\ -\beta & \text{if } \omega_s, \omega_r \text{ are neighbors and } \omega_s \neq \omega_r \\ 0 & \text{if } \omega_s, \omega_r \text{ are not neighbors} \end{cases}$$

The segmentation algorithm

Now we know the equation of the best labeling. How to compute it, how to get a good labeling? This is an optimization problem!

After defining the parameters (i.e. the number of classes and some others too) the MRF segmentation algorithm has 3 main steps:

1. Define regions
For every class label we should give a sample containing some feature vectors from that class.

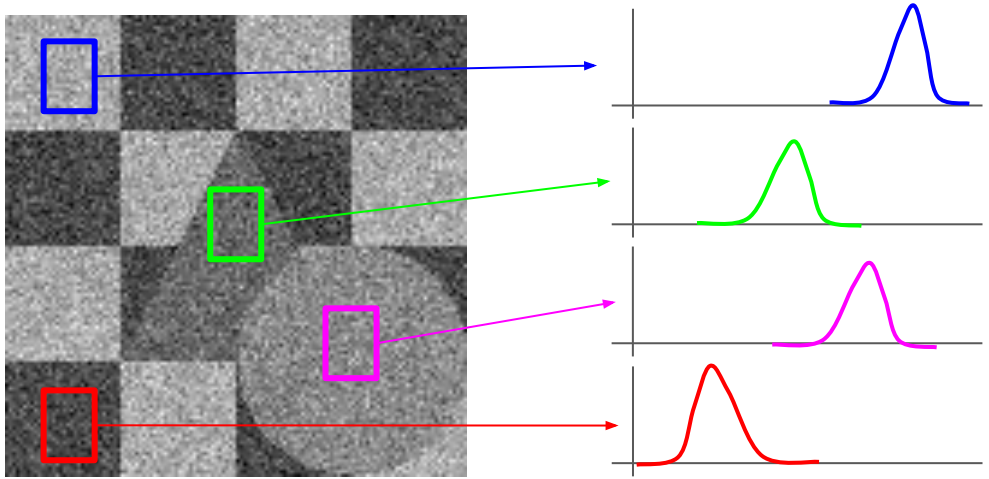
2. Initialize labeling
For every feature vector we assign a label. This assignment can be random or it can be based on some knowledge, e.g. per-pixel Maximum a Posteriori (MAP)

3. Run optimization
Optimization is an iterative process in which we reduce the energy function.

1. Define regions

In this step we select some parts of the input image and use these parts to estimate the properties of each class.

E.g. on a grayscale image the feature of a pixel is the pixel intensity. The class will be described using a Gaussian that models the mean intensity and standard deviation of the pixel intensities in the selected region:



$$\text{class}\lambda : \mathcal{N}(\mu_\lambda, \sigma_\lambda)$$

where the mean and std are the empirical mean and std:

$$\mu_\lambda = \frac{1}{|S_\lambda|} \sum_{s \in S_\lambda} f_s$$

$$\sigma_\lambda = \sqrt{\frac{1}{|S_\lambda|} \sum_{s \in S_\lambda} (f_s - \mu_\lambda)^2}$$

2. Initialize labeling

Random

The random initialization is pretty straightforward, assign a random label to every pixel (preferably using uniform distribution).

Maximum a Posteriori (MAP)

For each pixel, find the most likely class label based on the formula

$$\omega_s^{(0)} = \arg\min_{\lambda \in \Lambda} \left[\log\left(\sqrt{2\pi}\,\sigma_\lambda\right) + \frac{1}{2}\left(\frac{f_s - \mu_\lambda}{\sigma_\lambda}\right)^2 \right]$$

Please note that this is the posterior part of the energy function. In practice the MAP initialization gives a good estimation, however it can be very noisy due to the fact that MAP does not care about the neighboring pixels.

3. Run optimization

ICM

A gradient descent method, well described in the lecture slides. (This method is already implemented in today's code).

Modified Metropolis Dynamics (MMD)

An initial temperature (T_0) and a temperature decay (c) is given. In every iteration a trail perturbation (η) is constructed which only differs in one label. Compute the difference of the energy functions with the original and the perturbed labeling. ($\Delta U = U(\eta) - U(\omega)$) The new labeling is given by the *Metropolis criteria*:

$$\omega = \begin{cases} \eta & \text{if } \Delta U < 0 \\ \eta & \text{if } \Delta U > 0 \text{ and } \xi < \exp(-\Delta U / T) \\ \omega & \text{otherwise} \end{cases}$$

where ξ is a uniform random number [0,1).

Also, T is updated: $T = c \cdot T$

Now please

download the ‘Lab 10’ code package

from the

submission system

Some comments about the provided code

Today's code package contains an input folder, function files, a script file and a .mat file describing a colormap.

Today you'll complete the partially implemented functions.

After each exercise you can test your code by running the **script** `MRF_script` with the appropriate parameters. Before each run, please set the parameters (in the first lines of the script) as described on the slides.

THESE METHODS USE RANDOM INITIALIZATIONS AND PERTURBATIONS!
EVERY RUN WILL YIELD A DIFFERENT RESULT!
THE FIGURES IN THIS DOCUMENT ARE EXAMPLES!

Exercise 1

Complete the function `gmrf_defineRegions` in which implement the body of the second loop:

- The corners of the rectangle are stored in the object `rects{ind}`
- Corp the relevant part of the input image (the rectangle defined by the `y1`, `y2`, `x1`, `x2` fields of the rectangle object). The input image is `params.InputImage`.
- Compute the mean and standard deviation of the pixel intensities of the cropped region (use the `mean` and `std` functions). Save them into the MRF model variable: `mrf.gauss_params{ind}.mean` and `mrf.gauss_params{ind}.dev` fields.
- Compute the normal probability density function of the input image. Use the function `normpdf`. Arguments are the input image (in double format), and the previously computed mean and standard deviation. The $-\log()$ of the normal PDF must be saved into `mrf.logProbs{ind}`.

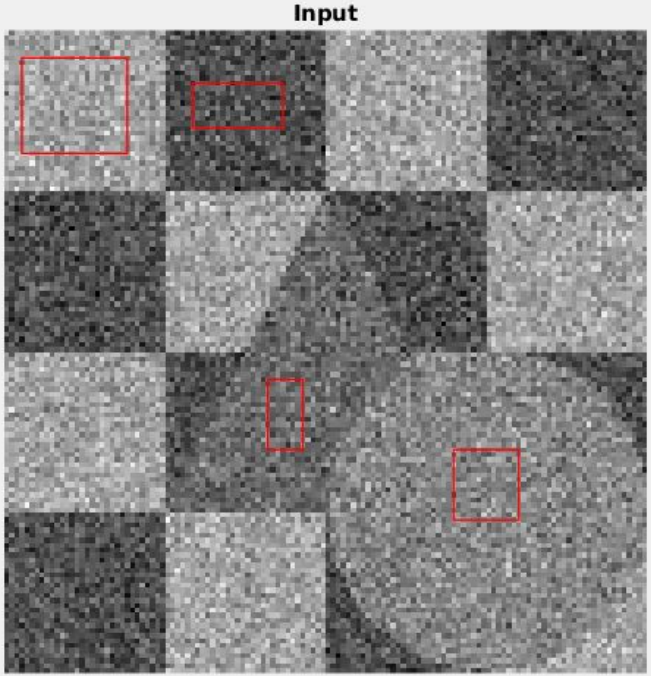
Test 1

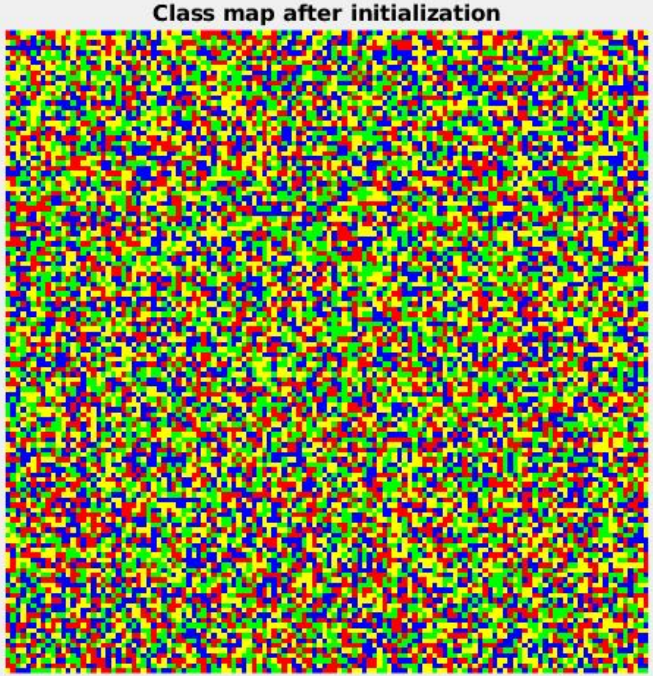
Run the script `MRF_script` after setting the following parameters:

- `params.InputImgPath = 'input/trin3.bmp';`
- `params.NumOfClasses = 4;`
- `params.InitMethod   = 'RAND';`
- `params.OptiMethod   = 'ICM';`

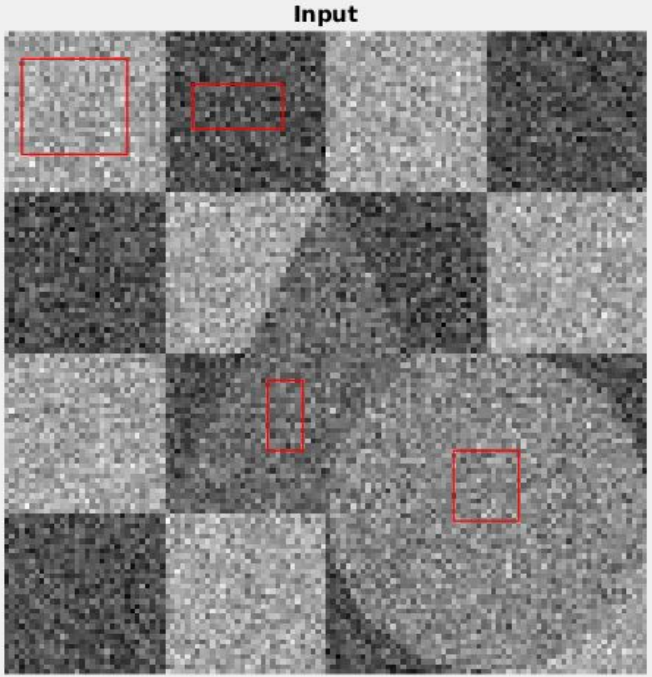
The script will raise a figure. Use ginput to define 4 × 2 corner points: take a sample from every class by drawing a rectangle.

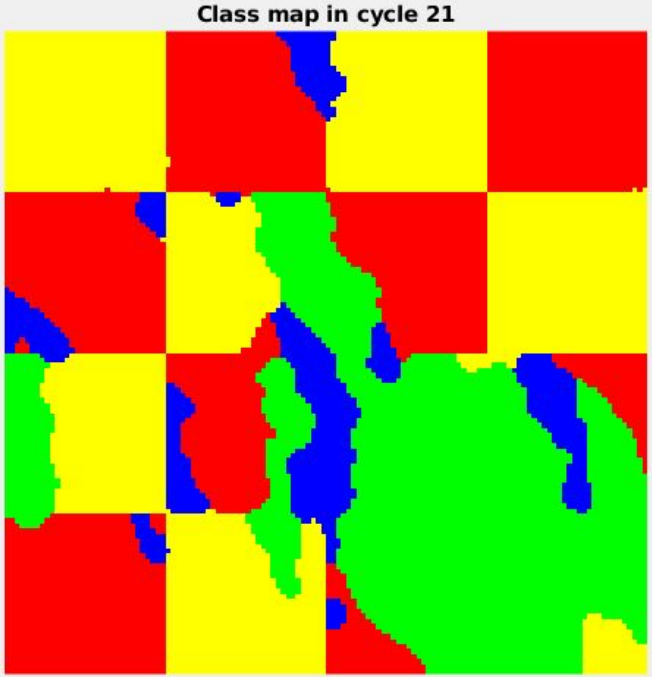
After drawing all the rectangles, press ENTER to run the optimization.





Right after initialization





After the optimization process finished

Exercise 2

Complete the function `gmrf_initClassMaskMAP` in which implement the body of the nested loop:

- For every pixel `(y,x)` find the class index for which `mrf.logProbs{cind}(y,x)` is minimal.
- The minimal class index should be called `minind` and it will be stored in the `mrf.classmask` field at the location `(y,x)`. (This is already implemented.)

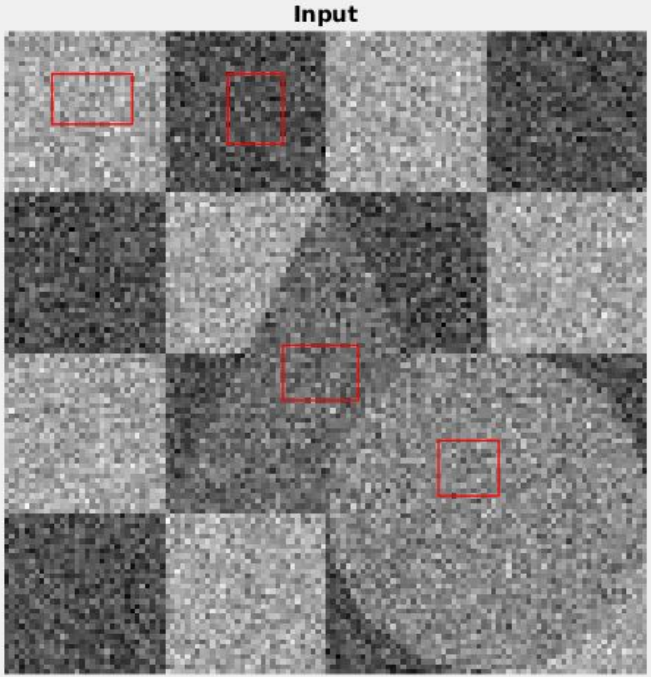
Test 2

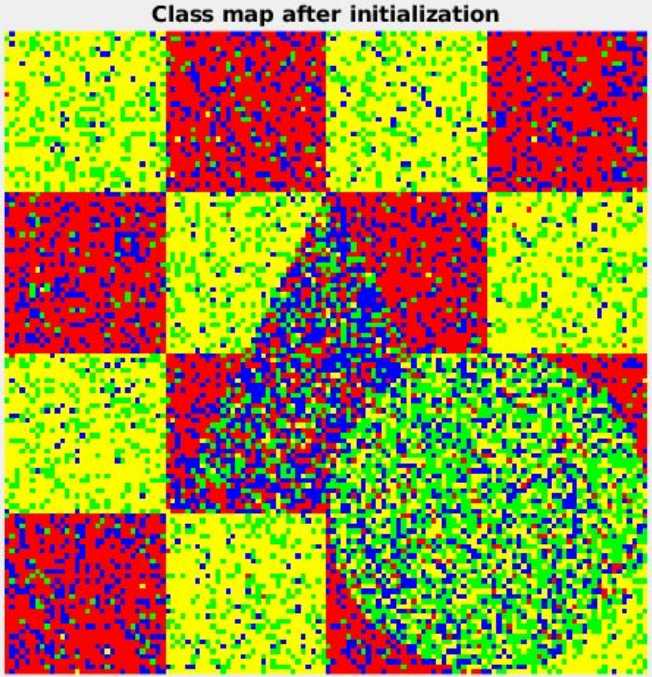
Run the script `MRF_script` after setting the following parameters:

- `params.InputImgPath = 'input/trin3.bmp';`
- `params.NumOfClasses = 4;`
- `params.InitMethod   = 'MAP';`
- `params.OptiMethod   = 'ICM';`

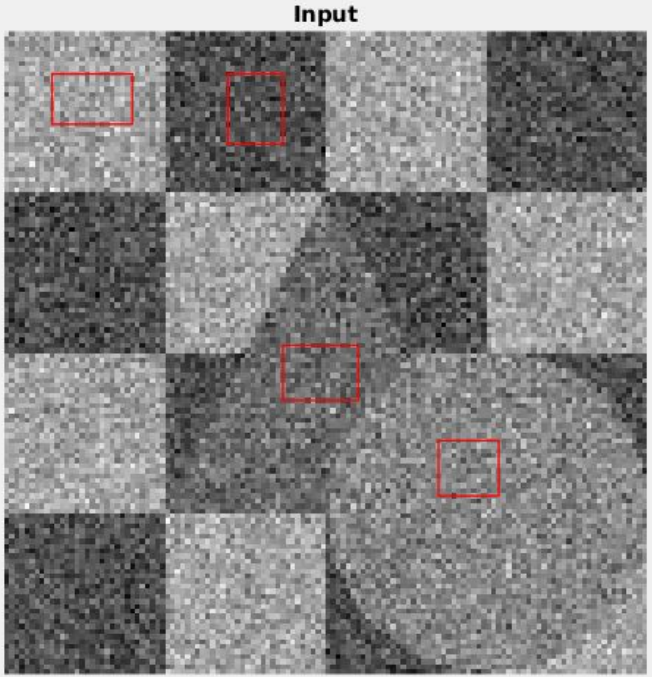
The script will raise a figure. Use ginput to define 4 × 2 corner points: take a sample from every class by drawing a rectangle.

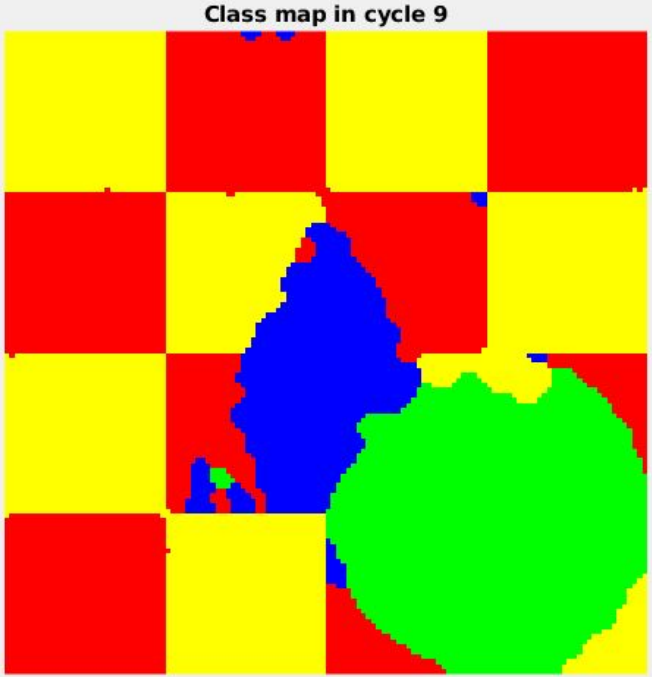
After drawing all the rectangles, press ENTER to run the optimization.





Right after initialization





After the optimization process finished

Exercise 3

Complete the function `gmrf_doMMD` in which implement the body of the while loop:

- Set `summa_deltaE` to zero.
- Increment the `cycle` counter.
- For each pixel:
  - Get the current class label at location `(y,x)`
  - Get the class label of the 8 (or less) neighboring pixel.
  - Get the actual posterior probability (it is `mrf.logProbs{C}(y, x)` where `C` is the actual class label.
  - Compute the actual prior probability (+beta times the matching neighboring labels, -beta times the different neighboring labels)

Description continues on the next slide...

Exercise 3

- Randomly pick a class label. This should be different from the current label of the pixel at `(y,x)`.
- Compute the new posterior and prior probabilities using the new, randomly picked class label.
- Compute the energy change (`dU = U_new - U_act`)
- If this gain is less than 0 or the gain is higher than 0 but a random float from the [0,1) interval is smaller than `exp(-dU/t)`, then update:
  - `summa_deltaE`, increase its value by `abs(dU)`
  - `mrf.classmask(y, x)`, store the randomly picked label into the class mask.
- Update the temperature (`T`), multiply it by `c` and store the new value.

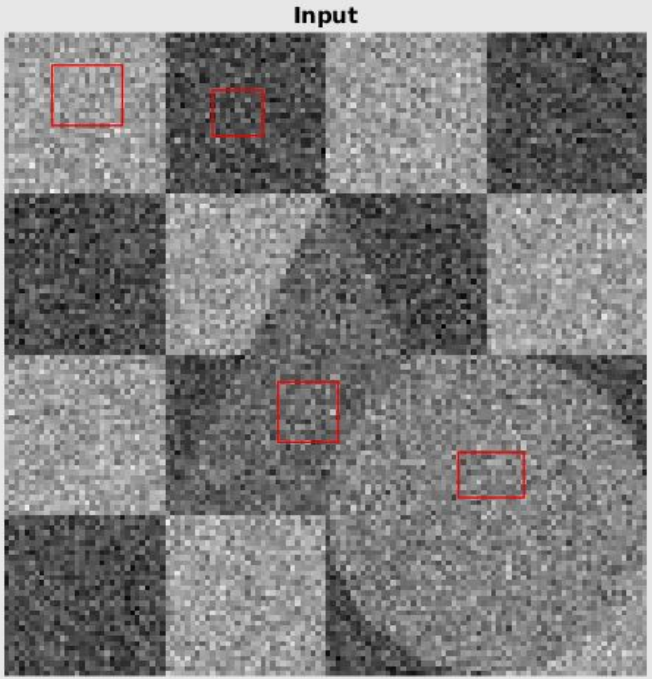
Test 3

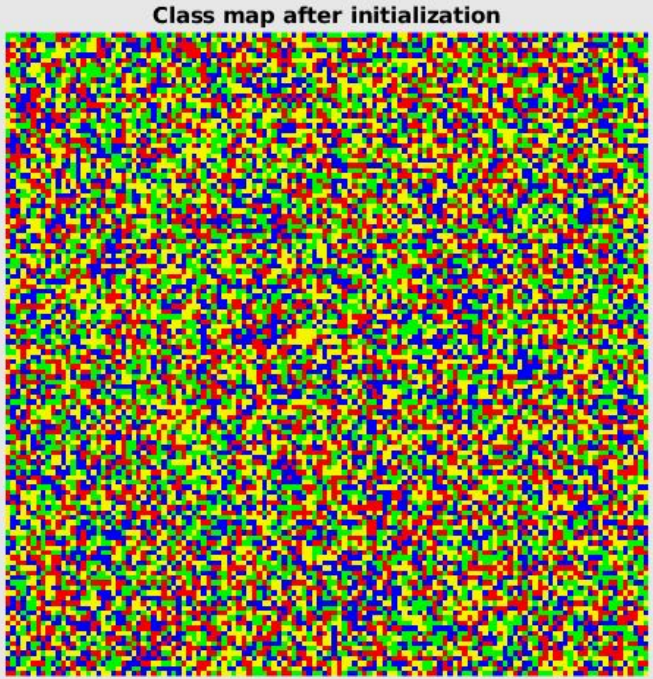
Run the script `MRF_script` after setting the following parameters:

- `params.InputImgPath = 'input/trin3.bmp';`
- `params.NumOfClasses = 4;`
- `params.InitMethod   = 'RAND';`
- `params.OptiMethod   = 'MMD';`

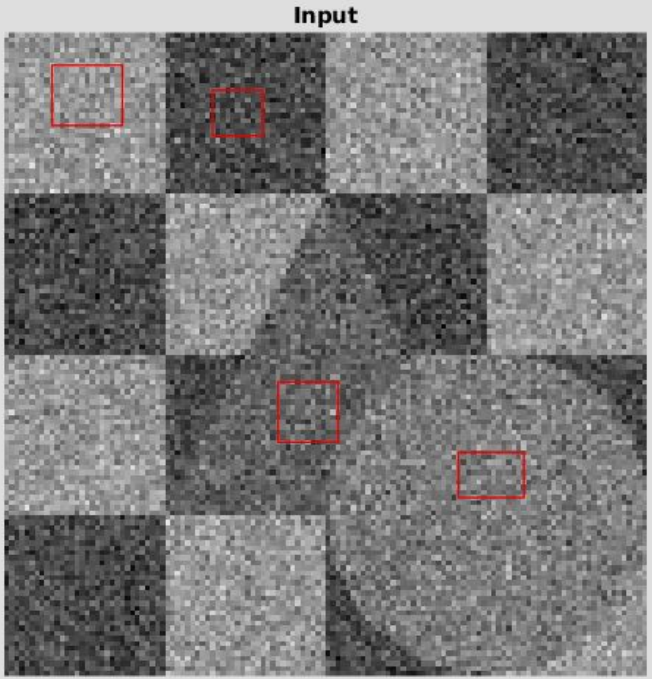
The script will raise a figure. Use ginput to define 4 × 2 corner points: take a sample from every class by drawing a rectangle.

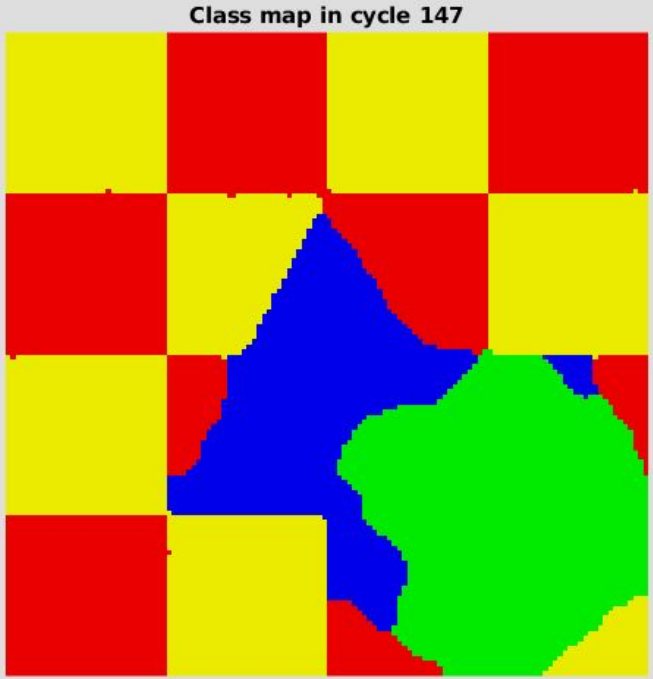
After drawing all the rectangles, press ENTER to run the optimization.





Right after initialization





After the optimization process finished

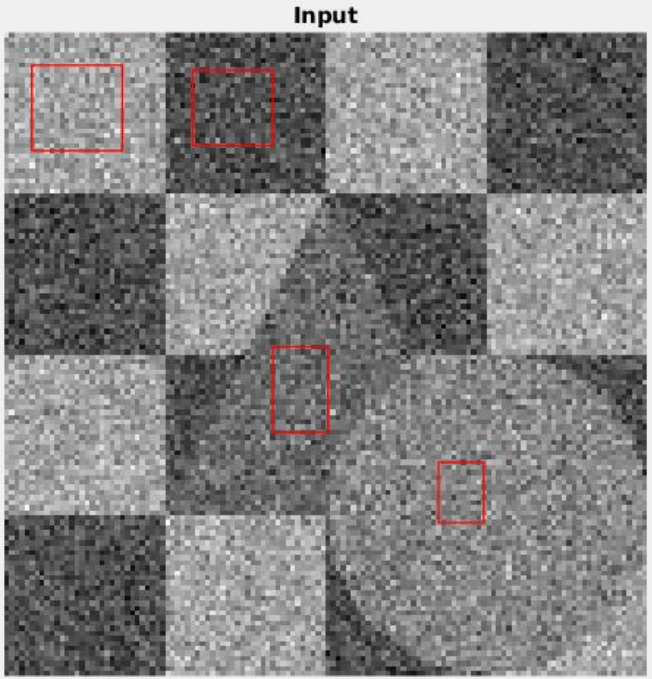
Test 4

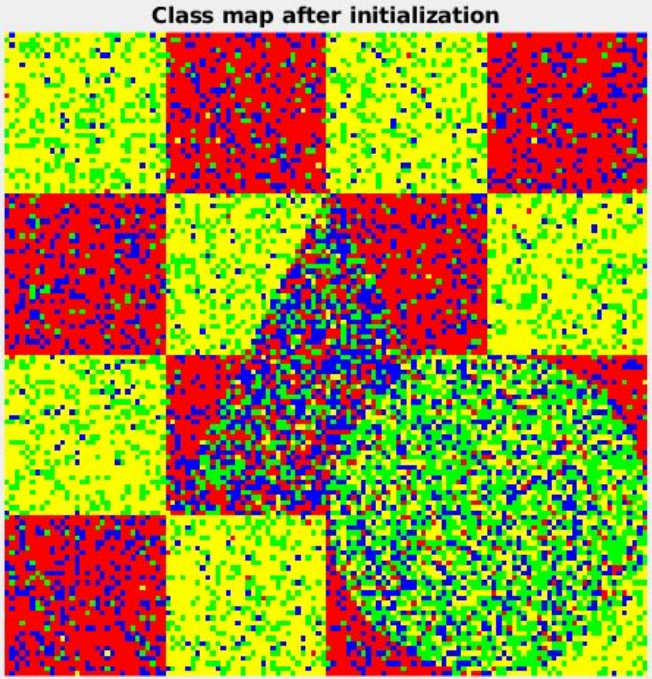
Run the script `MRF_script` after setting the following parameters:

- `params.InputImgPath = 'input/trin3.bmp';`
- `params.NumOfClasses = 4;`
- `params.InitMethod   = 'MAP';`
- `params.OptiMethod   = 'MMD';`

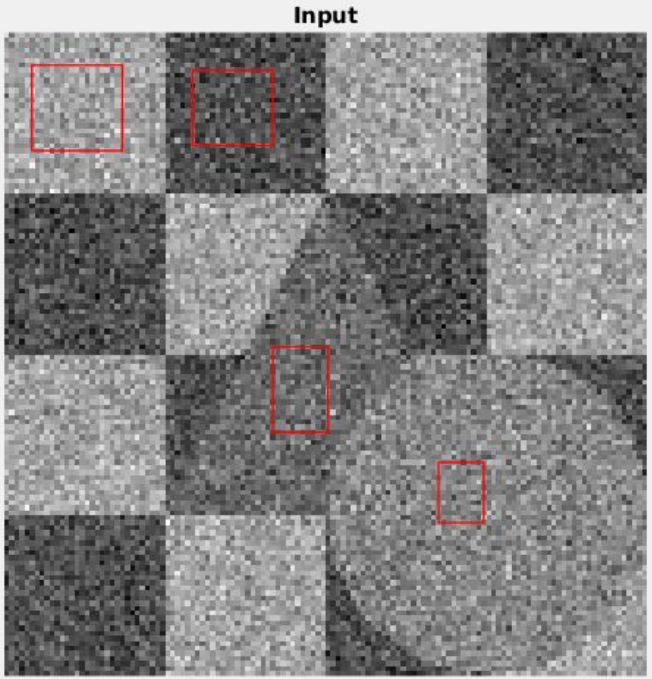
The script will raise a figure. Use ginput to define 4 × 2 corner points: take a sample from every class by drawing a rectangle.

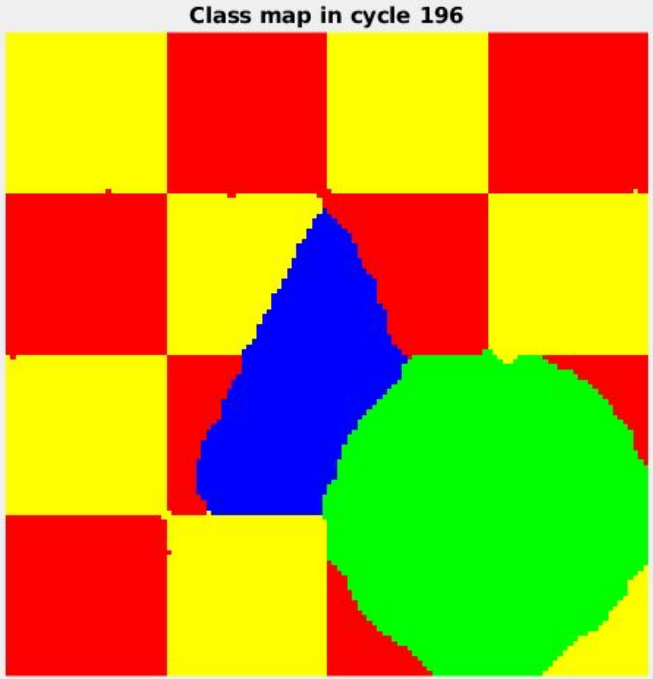
After drawing all the rectangles, press ENTER to run the optimization.





Right after initialization





After the optimization process finished

THE END