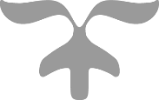
19MAT301 : MATHEMATICS FOR INTELLIGENT SYSTEMS 5



**MARKOV RANDOM fields**

FOR IMAGE SEGMENTATION & DENOISING



TEAM 13

CB.EN.U4AIE19016 – **ASSWIN C R**

CB.EN.U4AIE19024 – **DHARSHAN KUMAR K S**

CB.EN.U4AIE19028 – **AVINASH DORA**

CB.EN.U4AIE19060 – **SIVA PRAKASH R**

GUIDED BY

**DR. SOMAN K P**



BTECH CSE-AI

CEN DEPARTMENT

AMRITA SCHOOL OF ENGINEERING

**ACKNOWLEDGEMENT**

This project would not have been possible without the support and guidance of Dr. Soman K P, who has provided us with sufficient knowledge and guidelines.

We would also like to thank our Computer Science and Engineering (Artificial Intelligence) – CEN department for giving us this opportunity to improve our skills.

Furthermore, we would like to thank the Amrita Vishwa Vidyapeetham management for providing ample resources to avail our project needs and a platform for online lectures during this lockdown.

Our thanks and appreciations also go to colleagues in developing the project and people who have willingly helped us out with their abilities unconditionally.

**TABLE OF CONTENTS**

**1.** [**ABSTRACT**](#_Toc43146547) **4**

**2. RESEARCH ARTICLES****5**

**3. INTRODUCTION****6**

**4.** [**IMAGE SEGMENTATION**](#_Toc43146551) **7**

**5.** [**MARKOV RANDOM FEILDS**](#_Toc43146551) **9**

5.1 [PROPERTIES OF MRF](#_Toc43146552) 10

5.2 [GLOBAL & LOCAL MARKOV PROPERTY](#_Toc43146552) 11

5.3 [ISING MODEL](#_Toc43146552) 14

5.4 [POTTS MODEL](#_Toc43146552) 16

5.5 [SPATIAL LATTICE SCHEMES](#_Toc43146552) 16

5.6 [MAP](#_Toc43146552) 17

5.7 [HAMMERSLY-CLIFFORD THEOREM](#_Toc43146552) 18

5.8 [PAIR-WISE MRF](#_Toc43146552) 19

**6.** [**MONOGRID SEGMENTATION MODEL**](#_Toc43146551) **22**

6.1 [SEGMENTATION MODEL](#_Toc43146552) 22

6.2 [CODE](#_Toc43146552) 26

**7.** [**IMAGE DENOISING**](#_Toc43146551) **35**

7.1 INTRODUCTION35

7.2 METHODS36

7.3 GIBBS SAMPLING37

7.4 ENERGY41

7.5 LOSS FUNCTION42

7.6 CODE44

**8.** [**REFERENCES**](#_Toc43146551) **48**

**ABSTRACT**

This is a report based on end semester 5 project, Image Denoising and Segmentation using Markov Random Fields, under the subject “Mathematics for Intelligent Systems 5” – 19AIE301.

This report gives an introduction to Fundamentals of Markovian modeling in **Image segmentation** as well as a brief overview of recent advances in the field. Segmentation is considered in a common framework, termed image labeling, where the problem is concentrated to assigning labels to pixels. In a probabilistic approach, label dependencies are modeled by Markov random fields (MRF) and an optimal labeling is determined by Bayesian estimation, in particular maximum a posteriori (MAP) estimation. The main advantage of MRF models is that prior information can be imposed locally through clique potentials. The primary goal is to demonstrate the basic steps to construct an easily applicable MRF segmentation model and further develop its multiscale and hierarchical implementations as well as their combination in a multilayer model. MRF models usually yield a non-convex energy function. The minimization of this function is crucial in order to find the most likely segmentation according to the MRF model. Besides classical optimization algorithms, like simulated annealing or deterministic relaxation, we also present recently introduced graph cutbased algorithms

In this project we apply Gibbs Sampling based on different Markov Random Fields (MRF) structures to solve the image denoising problem. Compared with methods like gradient ascent, one important advantage that Gibbs Sampling has is that it provides balances between exploration and exploitation. This project also discusses about the behaviours of different MRF structures.

**RESEARCH ARTICLES**

Image Segmentation is important for classifying different objects in a picture. So, there are numerous research papers available which was written by many scholars.

Few of those notable research papers are mentioned below:

1. Markov Random Fields in Image Segmentation.

Date published: 5th October 2012

DOI: 10.1561/2000000035

Authored by: Zoltan Kato, Josiane Zerubia,

Published by: University of Szeged, Hungary, [kato@inf.u-szeged.hu](mailto:kato@inf.u-szeged.hu)

Objective of paper :MRF in Image Segmentation

1. MRF and CRF based Image Denoising and Segmentation

Date published: 2014

Authored by: Wei Zhang, Min Li

Published by: 2014 International Conference on Digital Home

Objective of paper: MRF and CRF for Image Denoising and Image Segmentation.

1. MRF and Gibbs Sampling for Image Denoising

Date published: August 2013

Authored by: Chang Yue

Objective of paper: MRF for denoising using Gibbs sampling

**INTRODUCTION**

An image processing system comprises a sensing device (usually a camera) and computer algorithms to interpret the image. The term image (more precisely, monochrome image) refers to a **two-dimensional light intensity function** whose value at any point is proportional to the brightness (gray-level) of the image at that point . A digital image is a discretized image both in spatial coordinates and in brightness. It is usually represented as a two-dimensional matrix, the elements of such a digital array are called **pixels**. The digitized image is the starting point of any kind of computer analysis. In some applications, the sensing device may be more specific responding to other forms of light: infrared imaging, photon emission tomography, radar imaging, ultrasonic imaging, etc.

Most of these problems can be formulated in a general framework, called image labeling, where we associate a label to each pixel from a finite set. The meaning of this label depends on the problem that we are trying to solve. For **image restoration**, it means a gray-level; for **edge detection**, it means the presence or the direction of an edge; for **image segmentation**, it means a region; etc. The problem here is how to choose a label for a pixel, which is optimal in a certain sense. Herein, we deal with a statistical approach of labeling. In real scenes, neighboring pixels usually have similar features (intensity, color, texture, etc). In a probabilistic framework, such regularities are well expressed mathematically by Markov random fields.

**IMAGE SEGMENTATION**

The primary goal of any segmentation algorithm is to divide the domain R of the input image into the disjoint parts Ri such that they belong to distinct objects in the scene.

The solution of this problem sometimes requires high level knowledge about the shape and appearance of the objects under investigation.

In many applications, however, such information is not available or impractical to use. Hence low-level features of the surface patches are used for the segmentation process.

One broadly used class of models is the so called **cartoon** model, which has been extensively studied from both probabilistic and variational viewpoints. The model assumes that the real world scene consists of a set of regions whose observed **low-level features** change slowly, but across the boundary between them, these features change abruptly. What we want to infer is a cartoon ω consisting of a simplified, abstract version of the input image I: regions Ri have a constant value (called a label in our context) and the discontinuities between them form a curve Γ — the contour. The pair (ω,Γ) specifies a **segmentation**. Region based methods are mainly focused on ω while edge based methods try to determine Γ directly. Taking the probabilistic approach, one usually wants to come up with a probability measure on the set Ω of all possible segmentations of I and then select the one with the highest probability.

We shall assume that we have a set of observed (Y ) and hidden (X) random variables. In our context, any observed value y ∈ Y represents the low-level features used for partitioning the image, and the hidden entity x ∈ X represents the segmentation itself. First, we have to quantify how well any occurrence of x fits y. This is expressed by the probability distribution P(y|x) the imaging model. Second, we define a set of properties that any segmentation x must possess regardless the image data. These are described by P(x), the prior, which tells us how well any occurrence x satisfies these properties. Factoring these distributions and applying the Bayes theorem gives us the posterior distribution P(x|y) ∝ P(y|x)P(x). Note that the constant factor 1/P(y) has been dropped as we are only interested in \_x which maximizes the posterior, that is, the maximum a posteriori (MAP) estimate of the hidden field X. The models of the above distributions also depend on certain parameters that we denote by Θ. Supervised segmentation assumes that these parameters are either known or a set of joint realizations of the hidden field X and observations Y (called a training set) is available . This is known in statistics as the complete data problem which is generally easier to solve than the incomplete case . Although the prior knowledge of the parameters is a strong assumption, supervised methods are still useful alternatives when working in a controlled environment. Many industrial applications, like quality inspection of agricultural products , fall into this category. In the unsupervised ase, however, we know neither Θ nor X. This is called the incomplete data problem where both Θ and X have to be inferred from the only observable entity Y . Hence our MAP estimation problem becomes (x, Θ) = argmax x,ΘP(x,Θ|y). Expectation Maximization (EM) and its variants (Stochastic EM , Gibbsian EM ), as well as Iterated.

Conditional Expectation (ICE) ,are widely used to solve such problems. It is important to note, however, that these methods calculate a local maximum. Due to the difficulty of estimating the number of pixel classes (or clusters), unsupervised algorithms often suppose that this parameter is known a priori .When the number of pixel classes is also being estimated, the unsupervised segmentation problem may be treated as a model selection problem over a combined model space

**MARKOV RANDOM FIELDS**

In the early 20th century, mostly inspired by the Ising model , a new type of stochastic process appeared in the theory of probability, called Markov random field (MRF). MRF’s rapidly became a broadly used tool in a variety of problems, not only in statistical mechanics.

A **potential**  is a non-negative function of the variable x. A **joint potential**  is a non negative function of the **set** of variables

For a set of variables  a Markov Random Field is defined as a product of potentials over the (maximal) cliques  of the Undirected graph G.

For a set of variables a Markov Random Field is defined as a product of potentials over the (maximal) cliques of the Undirected graph G.

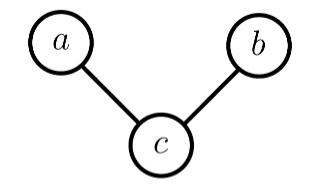
 (1)

* - Cliques are subset of Variables.( for eg the 1st clique can contain x1 ,2nd as x1, x2 etc) depending upon the graphs.
* - 1/Z is the normalizer, known as the **partition function**. To make the LHS , P(X) a proper probability distribution.

Special cases for Cique size two : **Pair-wise Markov Random Field**

If all potentials are strictly positive: **Gibbs Distribution**

**Properties of MRF**



**Fig 1**

* - The above figure contains 3 random variables(a,b,c)

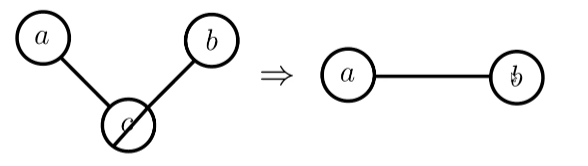
- here , 2 clique are maximal cliques [ (a,c) and (a,b) ]





- from eq 2, Z is partition function which sums to 1., in order to do , we take the above expression and sum it all over the combinations of a,b,c .

**Property 1 : Marginalizing:**

****

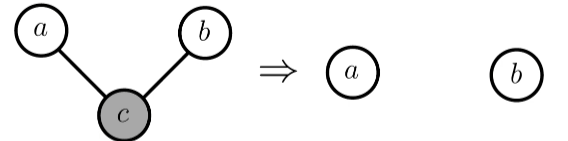
**Fig 2**

* - From the above figure 2 Marginalizing over c makes a and b dependent ( earlier independent) and becomes directly connected.
* 
* - Probabilty of a,b joint distribution is not equal to p(a).p(b)

**Proof:**





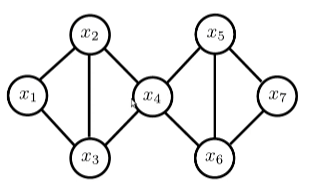
**Property 2 : Conditioning**

**Fig 3**

* -From Fig 3 Conditioning on c makes a and b independent
* - above expression can be proved by following the same approach as Property 1.



# **Global Markov Property**



A

B

**Fig 3.1**

**Sepration:**

A subset S (from fig 3.1 x4 represents S) separates A from B if every path from a member A to any member passes through S.

**Global Markov Property**

For disjoint sets of variables (A,B,S) where S seperates A from B, we have AB| S

-which means if we conditioning x4, then x2 becomes conditionally independent of x5 or x1|| x5 or x3|| x5 etc.

- join of x1,x2,x3 || x5,x6

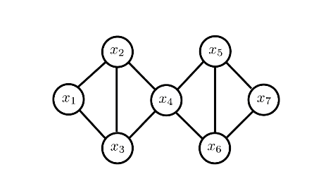
# **Local Markov Property.**

**Local Markov Property:**

When Conditioned on its neighbors, x becomes independent of the remaining varibales of the graph.



* - for eg x1 conditioned on other random variables x2,x3,x4,x5,x6,x7 is equal to x1 conditioned on x2 and x3, because x2 and x3 shield x1 from remaining variables.( separation property).
* - Set of neighbouring nodes ne(x) in eq 3 is called as **Markov Blanket.**



**Fig 4**

**Example from fig 4**

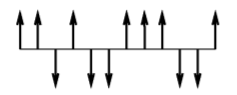


# **ISING Model :**

Consider a sequence, 0,1,2,...,n on the line. At each point, there is a small spin which is either up or down at any given moment . Now, we define a probability measure on the set Ω of all possible configurations ω = (ω0,ω1,...,ωn). In this context, each spin is a function given by equation 4.



An energy U(ω) is assigned to each configuration (eq 5):



1-D Ising Model

In the first sum, Ising made a simplifying assumption that only interactions of points with one unit apart need to be taken into account. This term represents the energy caused by the spin-interactions. The constant J is a property of the material. If J > 0, the interactions tend to keep neighboring spins in the same directions (attractive case). If J < 0, neighboring spins with opposite orientation are favored (repulsive case). The second term represents the influence of an external magnetic field of intensity H and m > 0 is a property of the material. The probability on Ω is then given by;



where T is the temperature and k is a universal constant. The normalizing constant (also called partition function) Z is defined by



The probability defined in Equation is called a Gibbs distribution. One could extend the model to two dimensions in a natural way. The spins are arranged on a lattice, they are represented by two coordinates and a point have 4 neighbors unless it is on the boundary. In the two dimensional case, the limiting measure P is unstable, there is a phase transition.

**Potts Model**

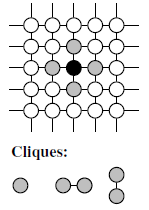
Another important extension of the Ising model to more than two states per points is the Potts model .The problem is to regard the Ising model as a system of interacting spins that can be either parallel or antiparallel. More generally, we consider a system of spins, each spin pointing one of the q equally spaced directions. These vectors are the linear combinations of q unit vectors pointing in the q symmetric directions of a hypertetrahedron in q − 1 dimensions. For q = 2,3, 4, examples are shown in Figure 1.3. The energy function of the Potts model can be written as



where J(Θ) is 2π periodic and Θij is the angle between two neighboring spins in i and j. The q = 2 case is equivalent to the Ising model.

**Spatial Lattice Schemes**

In this section, we deal with a particular subclass of MRFs which are the most commonly used schemes in image processing. In this case,



we consider S as a lattice L so that ∀s ∈ S : s = (i, j) and define the so-called nth order homogeneous neighborhood systems as



Obviously, sites near the boundary have fewer neighbors than interior ones (free boundary condition). Furthermore, G0 ≡ S and for all n ≥ 0 : Gn ⊂ Gn+1. Figure shows a first-order neighborhood corresponding to n = 1. The cliques are {(i, j)},{(i, j), (i, j + 1)},{(i, j), (i + 1,j)}. In practice, more than two order systems are rarely used since the energy function would be too complicated requiring a lot of computation. Although not as widespread as orthogonal lattice schemes, hexagonal lattices as well as MRFs on graphs have also been studied in the literature.

**Maximum A Posteriori (MAP)**

The MAP estimator is the most frequently used estimator in image processing. Its cost function is defined by



where δ(ω’,ω) from equarion 9 is the Kronecker delta. Clearly, this function has the same cost for all configurations different from ω’. From Equations the MAP estimator of the label field is given by



This estimator provides for a given observation f, the modes of the posterior distribution, that is the most likely labelings given the observation f. Equation is a combinatorial optimization problem which requires special algorithms such as Simulated Annealing .

**Hammersley-Clifford Theorem**

The Hammersley-Clifford Theorem states that a random field is a MRF if and only if P(w) follows a Gibbs distribution.

X is a MRF with respect to the neighborhood system G if and only if π(ω) = P(X = ω) is a Gibbs distribution with a nearest neighbor Gibbs potential V , that is



The main benefit of this equivalence is that it provides us a simple way to specify MRFs, namely specifying potentials instead of local characteristics (see Definition 1.3), which is usually very difficult.

**Pairwise MRF using Ising Model Approach:**

Pairwise MRF only considers pairwise neighborhood,which means the local interactions of the nodes in MRF is defined by pairwise potential Ψ(xi, xj), where xi and xj areneighboring nodes. The joint distribution of X and Y is given

by:



Ising model is an example that arose from statistical physics. Each node in the MRF only has two states. In particular, xi ∈ {0.255} . According to the Ising model, the pairwise potential function can be written as



where δ is the delta impulse function.

For the bi-level segmentation task, the unary potential function can be defined as follows



where μx and σx are the mean and standard deviation of the Gaussian distribution of Y for background and foreground respectively.

The Segmentation problems can be regarded as an optimization function. We want to find x∗ that maximizes the objective function



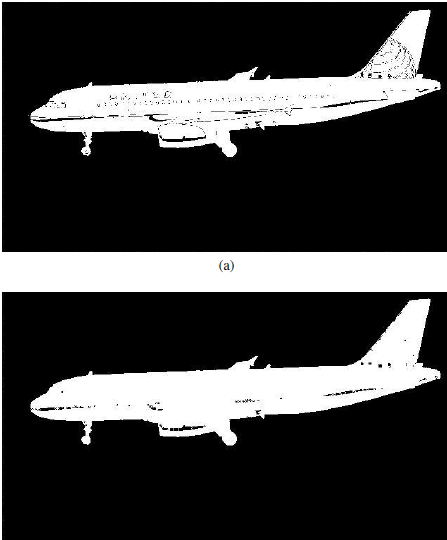


where β is the weight for pairwise potential and Nxi is the neighbor of xi.

Given this approximation, we can maximize each pixel individually, i.e., finding xi by maximizing p(xi|y, xNxi),where



To perform the image segmentation task, we randomly initialize the labels. Fig. shows the results generated by a relatively small and big β.



(a) β = 0.1 and (b) β = 10.

From the images, we can see that if the weight is small,there are some details on the body of the airplane that are treated as the background. This is because the intensity of these details is close to the background. The method to fix this is to enlarge the weight to make it emphasis more on the pairwise potential functions. In this way, the pixels tend to have the same value with their neighbors. However, if we apply a very large weight, every pixel in the image tends to have the same value, which disagrees with the purpose of image segmentation. The weight is a trade-off between the unary potential and the pairwise potential, depending on different application

**MONOGRID SEGMENTATION MODEL**

**A Classical Monogrid Segmentation Model**

Now we will show how to construct a simple Markovian image segmentation model. Our goal is to demonstrate the basic steps to construct an easily applicable MRF model for non-textured images and further develop its multi-scale and hierarchical implementations as well as their combination in a multilayer model. Let us suppose that the observations consist of the gray-levels. A very general problem is to find the labeling ˆω which maximizes the a posteriori probability P(ω | F). Note that ˆω is nothing else than the segmentation of the input image F. Obviously, the actual segmentation ˆω is determined by the probability measure P(ω | F). In other words, our segmentation model is expressed by the posterior probability P(ω | F), and then the optimal segmentation is simply found as the most likely labeling according to the probability distribution P(ω | F). Using the results reported in Section 2.1.1, ˆω is simply the MAP estimate of the label field. Therefore the main question is how to define P(ω | F). Bayes theorem tells us that



It is then easy to see that the global labeling, which we are trying to

find, is given by:



It is obvious from this expression that the a posteriori probability also derives from a MRF. The energies of cliques of order 1 (also called singletons) directly reflect the probabilistic modeling of labels without taking into account context, which would be used for labeling the pixels independently. Let us assume that P(fs | ωs) is Gaussian, the class λ ∈ Λ = {0,1, . . .,L − 1} is represented by its mean value μλ and its deviation σλ. Furthermore, we will adopt a smoothing prior which prefers homogeneous regions. We thus get the following energy function



where 

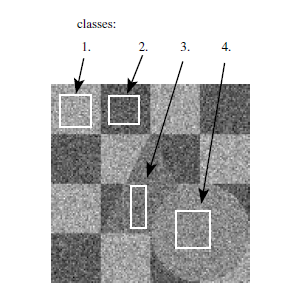
where β > 0 is a model parameter controlling the homogeneity of the regions. In fact, our prior terms corresponds to the Potts model in statistical physics . As β increases, the resulting regions become more homogeneous. Clearly, we have 2L + 1 parameters. They are denoted by the vector Θ:



If the parameters are supposed to be known, we say that the segmentation process is supervised. If they are unknown (and hence they have to be estimated simultaneously during the segmentation), the segmentation process is called unsupervised.

For supervised segmentation, we are given a set of training data

(small sub-images), each of them representing a class.



According to the law of large numbers, we can approximate the statistics

of the classes (mean and variance) by the empirical mean and

empirical variance:

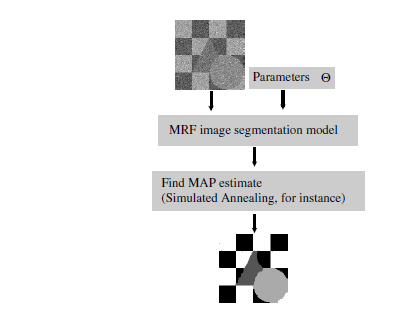


where Sλ is the set of pixels included in the training set of class λ. The

parameter β is initialized in an ad-hoc way (by trial and error). Typical

values are between 0.5 and 1.

In Figure, we give an overview of a supervised segmentation process. We have two inputs: the image itself and the parameters Θ. They yield an energy function as defined in Equations. To find the MAP estimate, an algorithm is needed to minimize this energy function. In Section 3, we discuss a variety of such algorithms. Here, we have used the Gibbs Sampler to get the minimum. The resulting image is just the labeling with minimum energy.



**Code:**

from PIL import Image

import numpy as np

import pandas as pd

import os, os.path

from scipy import misc

import glob

import sys

from matplotlib.pyplot import imshow

import imageio

import scipy.stats

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from scipy import optimize

import random

import warnings

warnings.filterwarnings('ignore')

# np.seterr(all='raise');

path = "test2-mini.jpg"

arr = misc.imread(path, flatten=True)

print ("initial image")

imshow(arr, cmap='gray');



def read\_image\_h(path):

img = Image.open(path)

hsv\_arr = matplotlib.colors.rgb\_to\_hsv(img)

hsv\_arr = np.asarray(hsv\_arr)

h\_arr = hsv\_arr[:,:,0]

# plt.imshow(h\_arr, cmap="gray")

# plt.show()

return 256\*h\_arr

initial\_probability = {"Sky.jpg": 0.30,"Road.jpg": 0.20, "Grass.jpg":0.50}

number\_of\_pixels = arr.size

class\_info = []

paths= ["Sky.jpg", "Road.jpg", "Grass.jpg"]

for path in paths:

tmp\_arr = read\_image\_h(path)

class\_mean = np.mean(tmp\_arr)

class\_var = np.var(tmp\_arr)

class\_freq = len(tmp\_arr)

class\_probabilty = class\_freq/number\_of\_pixels

class\_info.append([initial\_probability[path], class\_mean, class\_var])

print ("class\_info")

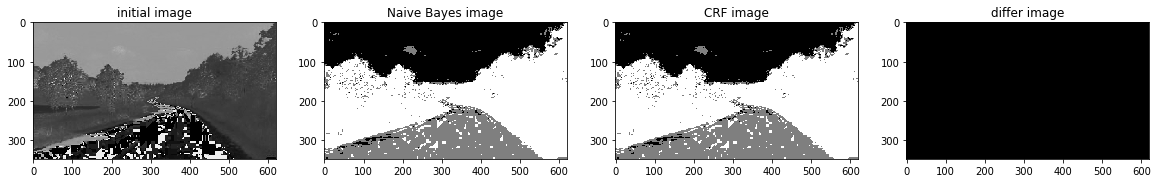
print (class\_info)

plt.figure(figsize=(16, 18), dpi=80, facecolor='w', edgecolor='k')

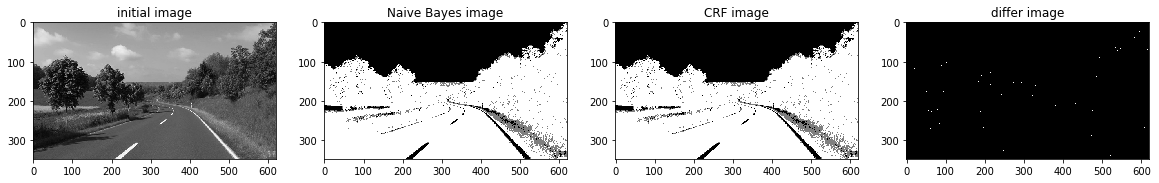
arr\_h = read\_image\_h("test2-mini.jpg")

plt.figure(figsize=(16, 18), dpi=80, facecolor='w', edgecolor='k')

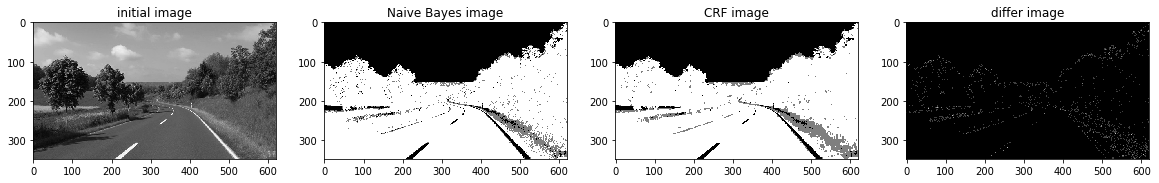
a\_complete\_set\_for\_part\_2(arr\_h,class\_info, max\_iter=1e2, betha=1e4)



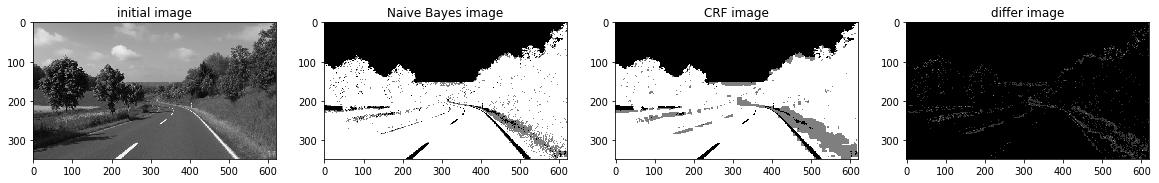
a\_complete\_set\_for\_part\_2(arr,class\_info, max\_iter=1e4, betha=1e6)



a\_complete\_set\_for\_part\_2(arr,class\_info, max\_iter=1e6, betha=1e6)



a\_complete\_set\_for\_part\_2(arr,class\_info, max\_iter=1e7, betha=1e6)



**HSV color space:**

 We want to use HSV color space for training our data.

import matplotlib

path = "test2-mini.jpg"

img = Image.open(path)

hsv\_arr = matplotlib.colors.rgb\_to\_hsv(img)

hsv\_arr = np.asarray(hsv\_arr)

h\_arr = hsv\_arr[:,:,0]

plt.imshow(h\_arr, cmap="gray")

plt.show()



def read\_image\_h(path):

img = Image.open(path)

hsv\_arr = matplotlib.colors.rgb\_to\_hsv(img)

hsv\_arr = np.asarray(hsv\_arr)

h\_arr = hsv\_arr[:,:,0]

# plt.imshow(h\_arr, cmap="gray")

# plt.show()

return 256\*h\_arr

initial\_probability = {"Sky.jpg": 0.30,"Road.jpg": 0.20, "Grass.jpg":0.50}

number\_of\_pixels = arr.size

class\_info = []

paths= ["Sky.jpg", "Road.jpg", "Grass.jpg"]

for path in paths:

tmp\_arr = read\_image\_h(path)

class\_mean = np.mean(tmp\_arr)

class\_var = np.var(tmp\_arr)

class\_freq = len(tmp\_arr)

class\_probabilty = class\_freq/number\_of\_pixels

class\_info.append([initial\_probability[path], class\_mean, class\_var])

print ("class\_info")

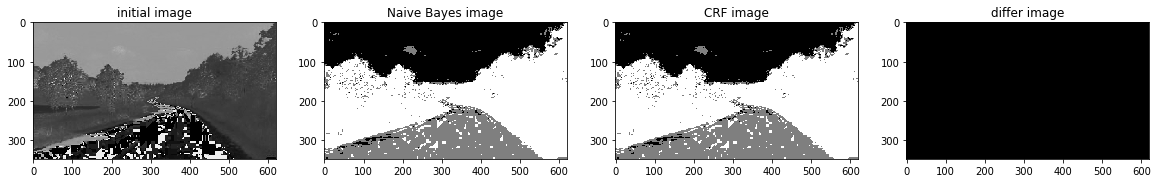
print (class\_info)

plt.figure(figsize=(16, 18), dpi=80, facecolor='w', edgecolor='k')

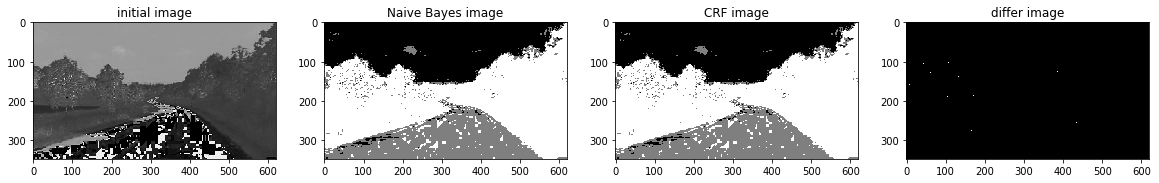
arr\_h = read\_image\_h("test2-mini.jpg")

plt.figure(figsize=(16, 18), dpi=80, facecolor='w', edgecolor='k')

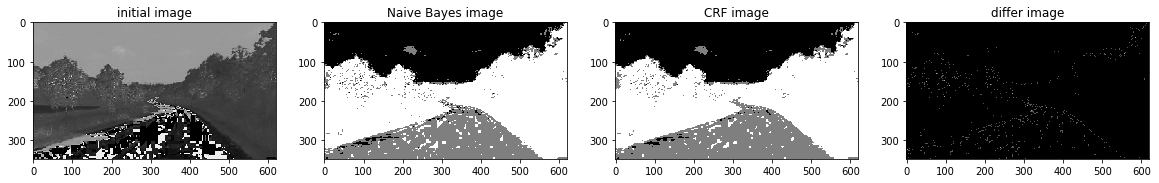
a\_complete\_set\_for\_part\_2(arr\_h,class\_info, max\_iter=1e2, betha=1e4)



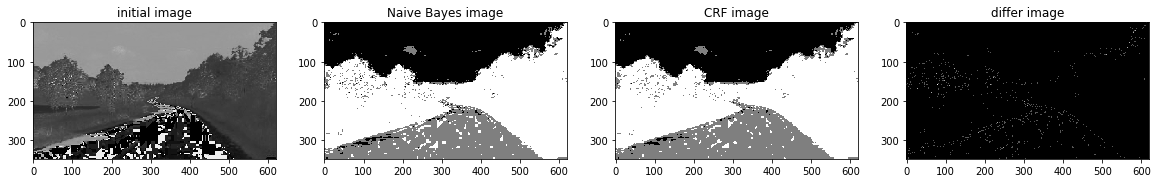
a\_complete\_set\_for\_part\_2(arr\_h,class\_info, max\_iter=1e4, betha=1e4)



a\_complete\_set\_for\_part\_2(arr\_h,class\_info, max\_iter=1e6, betha=1e4)



a\_complete\_set\_for\_part\_2(arr\_h,class\_info, max\_iter=1e6, betha=1e6)



### RGB color space

In this part, we used RGB color format in training since there is some information that can be captured by pixels colors. We used RGB values in potential function.

def get\_class\_info\_color(img\_path, color\_index,

paths=["Sky.jpg", "Road.jpg", "Grass.jpg"],

initial\_probability={"Sky.jpg": 0.30,"Road.jpg": 0.20, "Grass.jpg":0.50}):

arr\_general = misc.imread(img\_path)

arr = arr\_general[:,:,color\_index]

number\_of\_pixels = arr.size

class\_info = []

paths= ["Sky.jpg", "Road.jpg", "Grass.jpg"]

for path in paths:

tmp\_arr = misc.imread(path)

tmp\_arr = tmp\_arr[:,:,color\_index]

class\_mean = np.mean(tmp\_arr)

class\_var = np.var(tmp\_arr)

class\_freq = len(tmp\_arr)

# class\_probabilty = class\_freq/number\_of\_pixels

class\_info.append([initial\_probability[path], class\_mean, class\_var])

return class\_info

def naive\_bayes\_predict\_3\_color (arr, class\_infos, fixed\_pixels\_index=[], correct\_arr = []):

predict\_array = np.zeros((len(arr), len(arr[0])), dtype=float)

class\_color = [0,127,255]

for i in range(0, len(arr)):

for j in range(0, len(arr[0])):

if (len(fixed\_pixels\_index)>0 and len(correct\_arr)>0 and fixed\_pixels\_index[i][j]==1):

predict\_array[i][j]=correct\_arr[i][j]

continue

max\_probabilty = 0

best\_class = -1

for cls\_index in range(0, len(class\_color)):

cls\_posterior = class\_infos[0][cls\_index][0]

for c in range(0, 3):#for RGB

val = arr[i][j][c]

class\_info = class\_infos[c]

mean = class\_info[cls\_index][1]

var = class\_info[cls\_index][2]

pos =pdf\_of\_normal(val, mean, var)

cls\_posterior \*= pos

if (cls\_posterior > max\_probabilty):

max\_probabilty = cls\_posterior

best\_class = cls\_index

predict\_array[i][j] = class\_color[best\_class]

return predict\_array

def initial\_energy\_function\_colored(initial\_w, pixels, betha, cls\_infos, neighbors\_indices):

w = initial\_w

energy = 0.0

rows = len(w)

cols = len(w[0])

for i in range(0, len(w)):

for j in range(0, len(w[0])):

for c in [0,1,2]:

cls\_info = cls\_infos[c]

mean = cls\_info[int (w[i][j])][1]

var = cls\_info[int (w[i][j])][2]

pixel\_value = pixels[i][j][c]

energy += np.log(np.sqrt(2\*np.pi\*var))

energy += ((pixel\_value-mean)\*\*2)/(2\*var)

for a,b in neighbors\_indices:

a +=i

b +=j

if 0<=a<rows and 0<=b<cols:

energy += betha \* differnce(w[i][j], w[a][b])

return energy

def delta\_enegry\_colored(w, index, betha, new\_value, neighbors\_indices, pixels, cls\_infos):

initial\_energy = 0

(i,j) = index

rows = len(w)

cols = len(w[0])

for c in [0,1,2]:

cls\_info = cls\_infos[c]

mean = cls\_info[int(w[i][j])][1]

var = cls\_info[int(w[i][j])][2]

pixel\_value = pixels[i][j][c]

initial\_energy += np.log(np.sqrt(2\*np.pi\*var))

initial\_energy += ((pixel\_value-mean)\*\*2)/(2\*var)

for a,b in neighbors\_indices:

a +=i

b +=j

if 0<=a<rows and 0<=b<cols:

initial\_energy += betha \* differnce(w[i][j], w[a][b])

new\_energy = 0

for c in [0,1,2]:

cls\_info = cls\_infos[c]

mean = cls\_info[new\_value][1]

var = cls\_info[new\_value][2]

pixel\_value = pixels[i][j][c]

new\_energy += np.log(np.sqrt(2\*np.pi\*var))

new\_energy += ((pixel\_value-mean)\*\*2)/(2\*var)

# print("/////// \n first enegry", new\_energy)

for a,b in neighbors\_indices:

a +=i

b +=j

if 0<=a<rows and 0<=b<cols:

new\_energy += betha \* differnce(new\_value, w[a][b])

# print ("END energy", new\_energy)

return new\_energy - initial\_energy

def simulated\_annealing\_colored(init\_w, class\_labels, temprature\_function,

pixels, betha, cls\_infos, neighbors\_indices, max\_iteration=10000,

initial\_temp = 1000, known\_index=[], correct\_arr = [], temprature\_function\_constant=None ):

partial\_prediction=False

if (len(known\_index)>0 and len(correct\_arr)>0):

partial\_prediction=True

w = np.array(init\_w)

changed\_array = np.zeros((len(w), len(w[0])))

iteration =0

x = len(w)

y = len(w[0])

current\_energy = initial\_energy\_function\_colored(w, pixels, betha, cls\_infos, neighbors\_indices)

current\_tmp = initial\_temp

while (iteration<max\_iteration):

if (partial\_prediction):

is\_found=False

while (is\_found==False):

i = random.randint(0, x-1)

j = random.randint(0, y-1)

if (known\_index[i][j]==0):

is\_found=True

else:

i = random.randint(0, x-1)

j = random.randint(0, y-1)

l = list(class\_labels)

l.remove(w[i][j])

r = random.randint(0, len(l)-1)

new\_value = l[r]

delta = delta\_enegry\_colored(w, (i,j), betha, new\_value, neighbors\_indices, pixels, cls\_infos)

r = random.uniform(0, 1)

if (delta<=0):

w[i][j]=new\_value

current\_energy+=delta

changed\_array[i][j]+=1

# print ("CHANGED better")

else:

try:

if (-delta / current\_tmp < -600):

k=0

else:

k = np.exp(-delta / current\_tmp)

except:

k=0

if r < k:

# print("CHANGED worse")

w[i][j] = new\_value

current\_energy += delta

changed\_array[i][j] += 1

if (temprature\_function\_constant!=None):

current\_tmp = temprature\_function(iteration, current\_tmp, initial\_temp, constant =temprature\_function\_constant)

else:

current\_tmp = temprature\_function(iteration, current\_tmp, initial\_temp)

iteration+=1

return w, changed\_array

def a\_complete\_set\_for\_part\_2\_3\_color (max\_iter=1000000, var = 10000,

betha = 100,

neighbor\_indices = [[0,1],[0,-1],[1,0],[-1,0]],

class\_labels = [0,1,2],

class\_color = [0,127,255],

schedule= exponential\_schedule,

temprature\_function\_constant=None,

image\_path = "test2-mini.jpg"):

fig, (ax1, ax2, ax3, ax4) = plt.subplots(1,4)

# fig.suptitle('Comparision', fontsize=20)

arr = misc.imread(image\_path)

ax1.set\_title("initial image")

ax1.imshow(arr)

# cls\_info = naive\_bayes\_learning(arr, noisy\_arr, labels)

cls\_infos = []

for c in [0,1,2]:

tmp\_info =get\_class\_info\_color(image\_path,c)

cls\_infos.append(tmp\_info)

initial\_arr = naive\_bayes\_predict\_3\_color(arr, cls\_infos)

ax2.set\_title('Naive Bayes image')

ax2.imshow(initial\_arr, cmap='gray')

convert\_to\_class\_labels(initial\_arr)

w, test\_array = simulated\_annealing\_colored(initial\_arr, class\_labels, schedule,

arr, betha, cls\_infos, neighbor\_indices, max\_iteration=max\_iter)

for i in range (0, len(w)):

for j in range(0, len(w[0])):

w[i][j] = class\_color[int (w[i][j])]

ax3.set\_title('CRF image')

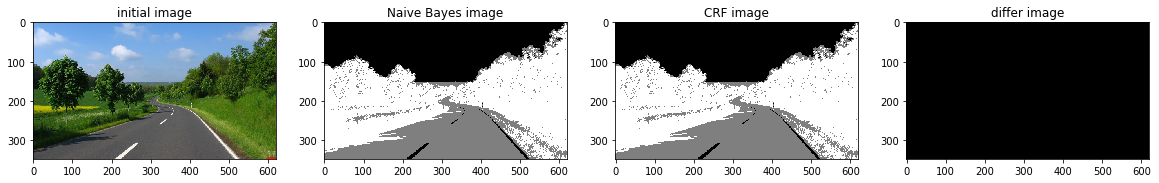
ax3.imshow(w, cmap='gray')

plt.rcParams["figure.figsize"] = (20,3)

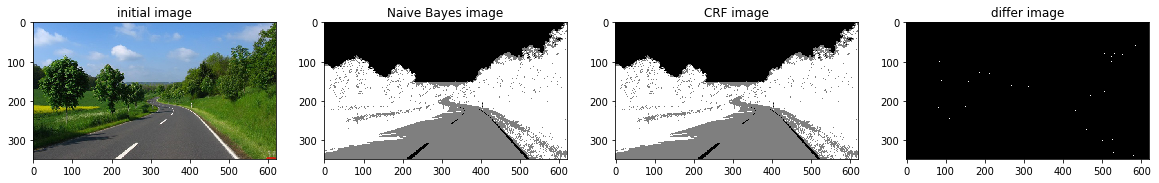
ax4.set\_title('differ image')

ax4.imshow(test\_array, cmap='gray')

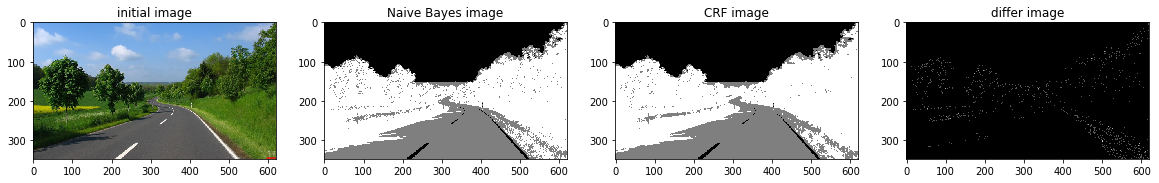
a\_complete\_set\_for\_part\_2\_3\_color(max\_iter=1e2, betha=1e4)



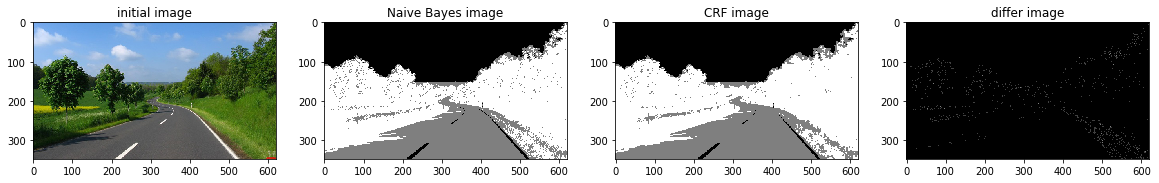
a\_complete\_set\_for\_part\_2\_3\_color(max\_iter=1e4, betha=1e4)



a\_complete\_set\_for\_part\_2\_3\_color(max\_iter=1e6, betha=1e6)

 a\_complete\_set\_for\_part\_2\_3\_color(max\_iter=1e6, betha=1e6,

schedule=linear\_multiplicative\_cooling\_schedule, temprature\_function\_constant=0.5)



**Conclusion**

Grayscale image format didn't have sufficient information for MRF models in this task.  
The value H in HSV image format had better information for segmentation using MRF models. And the result was better. The RGB format also had good information for segmenting the image. Because these three segments have different colors. So if a MRF model considers colors of the image for classification, then the result is going to be better compared to Grayscale images.

**IMAGE DENOISING**

# **Introduction:**

Image denoising has been a popular field of research for decades, and lots of methods have been developed. A typical approach would be going through an image pixel by pixel, and, at each pixel, by applying some Math models, set that pixel’s value based on the evidence of its neighbors’ values. Gaussian smoothing is the result of blurring an image by a Gaussian function, typically to reduce image noise. Mathematically, applying a Gaussian blur to an image is the same as convolving the image with a Gaussian function. Gradient ascent is another widely used method, where it iteratively assigns pixels values such that they bring improvement on the pre-defined function (usually called loss function). It will converge to a suboptimal state, and different initializations give different results in general. The drawback of these methods is that they are too deterministic: assign pixel value based on the current state with a probability of 1. And, methods like gradient ascent need careful initialization

# Gibbs Sampling is a Markov chain Monte Carlo (MCMC) algorithm for obtaining a sequence of observations which are approximated from a specified multivariate probability distribution. It provides exploration by sampling a pixel’s value based on probabilities computed from a MRF model. It is proved that after enough iteration passes, Gibbs Sampling will converge to the stationary distribution of the Markov Chain, no matter what the initialization is.

# **Methods:**

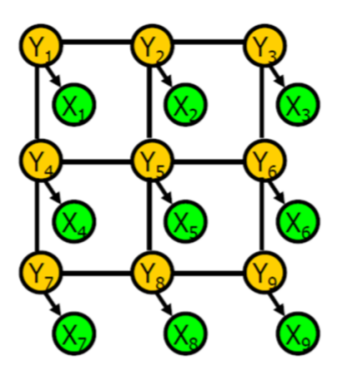
# Given a noisy image X, where X(i, j) (also denoted as xi,j ) is the pixel value at row i and column j. Assume the original image is Y . Denoising can be treated as a probabilistic inference, where we perform maximum a posteriori (MAP) estimation by maximizing the a posteriori distribution p(y|x). By Bayes Theorem,

p(y|x) =.

By taking logarithm on both sides, we get

X is given, so MAP estimation corresponds to minimizing,

A classic structure used in image denoising domain is the pairwise MRF, shown in Figure,

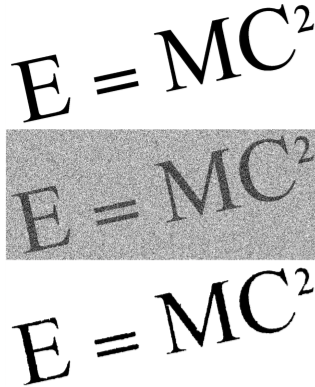


The yellow nodes (original image Y) are what we want to find, but we only have information about the green nodes (noisy image X). Each node y is only connected with its corresponding output x and four direct neighbors (up, down, left, right). Therefore, given a pixel y its 5 neighbors, we can determine the probability distribution of y without looking into other pixels.

**Gibbs Sampling :**

It is a Markov chain Monte Carlo (MCMC) algorithm that samples each random variable of a graphical model, one at a time. In our case, we sample one value of a single pixel yi,j at a time, while keeping everything else fixed. Assume the input image has a size of M × N.

**Gibbs Sampling for Binary Images:**

****

The original image contains a clean formula. Then by generating a noisy image such that each pixel from the original has a 20% chance of been flipped. The sampling probability of each pixel is set using the Ising model. After only a few iterations, the output which recovered most part of the original image with an error rate of 0.8%.

# **MCMC and Gibbs Sampling**

Markov chain Monte Carlo (MCMC) is a sampling method used to approximate the posterior distribution of a parameter of interest by randomly sampling from a probability distribution and constructing a Markov chain.

Gibbs Sampling is a MCMC algorithm that generates a Markov chain of samples, each of which is calculated with its direct neighbors. For example, in a Bayes Network, each sample is only dependent on its parents, co-parents, and children nodes; in Markov Random Field, each sample is associated with its Markov Blanket. This independency attribute simplifies the problem in that to get a sample value for state s, we only need the conditional probability P(s|s\_neighbors).

MCMC algorithms generally have a burn-in period, during which samples may not be accurate. Therefore, samples are collected after the burn-in period, and used to estimate the posterior using the Monte Carlo method.

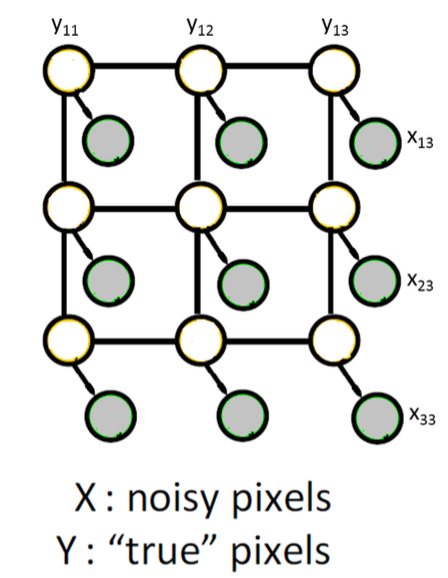
# **Mathematical Deduction;**

For our image denoising problem, we are given a noisy image X and the goal is to restore it to the original image Y, which is unknown.We know a noisy image array X = {xij}, where xij ∈ {−1, +1} represents the pixel at row i and column j. The image is black-and-white, with xij = 1 corresponding to black color and xij = -1 to white.

The original image (without noise) Y has the same size as the noisy image X, with yij ∈ {−1, +1} indicating the value of xij before noise was added. Denoising can be treated as a probabilistic inference, where we perform maximum a posteriori (MAP) estimation by maximizing the a posteriori distribution **p(Y|X)**. By Bayes Theorem we get:*p(Y|X) = p(X|Y)p(Y)/p(X)*. At log-space, this can be rewritten as:

Since X is given, maximizing **log p(Y|X)** is essentially equivalent to minimizing **− log p(X|Y) − log p(Y)**, which is the loss function in this problem.

A classic structure used for image denoising is the pairwise MRF, as shown in the graph below. Each node *yij* is connected with its corresponding output xij and 4 direct neighbors (up, down, left, right). Therefore, given the 5 neighbors of a pixel *yij,*we can determine the probability distribution of *yij*without looking into other pixels. Note that the pixels on the edges have fewer neighbors. For example, the neighbors of y11 are y12, y21 and x11. For convenience, we can pad the edges with 0s when we implement the algorithm so all pixels have 5 neighbors.



Assuming our posterior preference is black, the posterior we want to maximize is **p(Y=1|Y\_neighbors)**, where **Y= {***yij***} for i = 1, …, N and j = 1, …, M**. The joint probability of **Y** and **X** is given as:

Where ita(*η*) and beta(*β*) are our hyperparameters and Z is the normalization constant. **N(ij)** is the corresponding neighbors of *yij* except *xij*.

Using Bayes Rules, we can get the posterior distribution from the joint distribution as:  
3)

(4)

(5)

(6)

Where

We can also get the loss function **− log p(X|Y) − log p(Y),**written as:

(7)

We call this the energy as the loss is equivalent to energy in Boltzmann distribution where states with lower energy will always have a higher probability.

**Energy:**

The distribution over random variables can be expressed in terms of an energy function E. Here we define the energy E for variable yi,j as a sum of loss (L), sparse gradient prior (R) and window-wise products (W)

Eq (8)

The probability distributions of yi,j is defined as

(9)

where Z is the normalization factor. Therefore, higher energy means lower chance of getting sampled, the way we compute the energy depends on MRF.

**Loss:**

Loss is defined as the closeness of yi,j to its neighbors and corresponding xi,j . Therefore, the closer they are, the smaller the loss. We can define loss as the sum of the norm of distances:

(10)

The first term penalizes the difference between yi,j and its neighbors in y, and the second term penalizes the difference between yi,j and the noisy pixel xi,j . I found that λ = 1 is a good setting, and also helps with vectorization. The problem with L-2 norm is that it could be largely effected by outliers, but L-1 could create unnecessary patches. Lorentzian function is robust to outliers, and does not generate as many patches as L-1 norm.

By plugging this into the loss function, we get:

(11)

σ is a hyper parameter here, it controls the bandwidth.

**Sparse gradient prior**

We can also add our assumptions about images as a prior term here. I assumed the image has sparse gradient. Therefore, we also penalize for large gradients. The prior term is:

(11)

Where here we approximated the gradient between yi,j and z as the ratio of their difference to value of z, because this makes vectorization easy. Again, there is a trade-off between L-1 and L-2 norms.

**Window Wise Product :**

The advantage of doing this is it saves time, and could be scene-specific. Although the disadvantage is also obvious: could be biased. If we can find ’patterns’ within a m × n window, then dot product these patterns with the window the Gibbs sampler is currently looking at could yield some useful information. The most straight forward pattern is just the mean: go through the image with a m×n sliding window and then average the sum of pixel values that the window has seen. This turned out to be not helpful, since the mean tend to be plain. The second thing I tried was PCA: taking out the first s principle components and their corresponding explained variance. I denote components as filters f, the variance explained value as α, and the current window as w. Therefore, the third term in our energy function becomes:

(12)

**MRF with different neighbor size**

Expanding the Markov blanket of pixel yi,j such that it is also connected to its diagonal neighbors, i.e., yi−1,j−1, yi−1,j+1, yi+1,j−1 and yi+1,j+1.call this a 3 × 3 window. And test on 5 × 5 window. The finding is that in general, 3 × 3 window is better and more robust than simple pairwise and 5 × 5 window for all types of norms. At very high noise level (e.g., σ = 100 for a 0 to 255 value range), 5 × 5 gives higher PSNR value, but it is not visually good, since it creates too many ’patches’.

**Code:**

import math

import numpy as np

import matplotlib.pyplot as plt

def load\_image(filename):

my\_img = plt.imread(filename)

img\_gray = np.dot(my\_img[..., :3], [0.2989, 0.5870, 0.1140])

img\_gray = np.where(img\_gray > 0.5, 1, -1)

img\_padded = np.zeros([img\_gray.shape[0] + 2, img\_gray.shape[1] + 2])

img\_padded[1:-1, 1:-1] = img\_gray

return img\_padded

def sample\_y(i, j, Y, X):

markov\_blanket = [Y[i - 1, j], Y[i, j - 1], Y[i, j + 1], Y[i + 1, j], X[i, j]]

w = ITA \* markov\_blanket[-1] + BETA \* sum(markov\_blanket[:4])

prob = 1 / (1 + math.exp(-2\*w))

return (np.random.rand() < prob) \* 2 - 1

def get\_posterior(filename, burn\_in\_steps, total\_samples, logfile):

X = load\_image(filename)

posterior = np.zeros(X.shape)

print(X.shape)

Y = np.random.choice([1, -1], size=X.shape)

energy\_list = list()

for step in range(burn\_in\_steps + total\_samples):

for i in range(1, Y.shape[0]-1):

for j in range(1, Y.shape[1]-1):

y = sample\_y(i, j, Y, X)

Y[i, j] = y

if y == 1 and step >= burn\_in\_steps:

posterior[i, j] += 1

energy = -np.sum(np.multiply(Y, X))\*ITA-(np.sum(np.multiply(Y[:-1], Y[1:]))+np.sum(np.multiply(Y[:, :-1], Y[:, 1:])))\*BETA

if step < burn\_in\_steps:

energy\_list.append(str(step) + "\t" + str(energy) + "\tB")

else:

energy\_list.append(str(step) + "\t" + str(energy) + "\tS")

posterior = posterior / total\_samples

file = open(logfile, 'w')

for element in energy\_list:

file.writelines(element)

file.write('\n')

file.close()

return posterior

def denoise\_image(filename, burn\_in\_steps, total\_samples, logfile):

posterior = get\_posterior(filename, burn\_in\_steps, total\_samples, logfile=logfile)

denoised = np.zeros(posterior.shape, dtype=np.float64)

denoised[posterior > 0.5] = 1

return denoised[1:-1, 1:-1]

def plot\_energy(filename):

x = np.genfromtxt(filename, dtype=None, encoding='utf8')

its, energies, phases = zip(\*x)

its = np.asarray(its)

energies = np.asarray(energies)

phases = np.asarray(phases)

burn\_mask = (phases == 'B')

samp\_mask = (phases == 'S')

assert np.sum(burn\_mask) + np.sum(samp\_mask) == len(x), 'Found bad phase'

its\_burn, energies\_burn = its[burn\_mask], energies[burn\_mask]

its\_samp, energies\_samp = its[samp\_mask], energies[samp\_mask]

p1, = plt.plot(its\_burn, energies\_burn, 'r')

p2, = plt.plot(its\_samp, energies\_samp, 'b')

plt.title("energy")

plt.xlabel('iteration number')

plt.ylabel('energy')

plt.legend([p1, p2], ['burn in', 'sampling'])

plt.savefig('%s.png' % filename)

plt.close()

def save\_image(denoised\_image):

plt.imshow(denoised\_image, cmap='gray')

plt.title("denoised image")

plt.savefig('output/denoise\_image.png')

plt.close()

if \_\_name\_\_ == '\_\_main\_\_':

ITA = 1

BETA = 1

total\_samples = 1000

burn\_in\_steps = 100

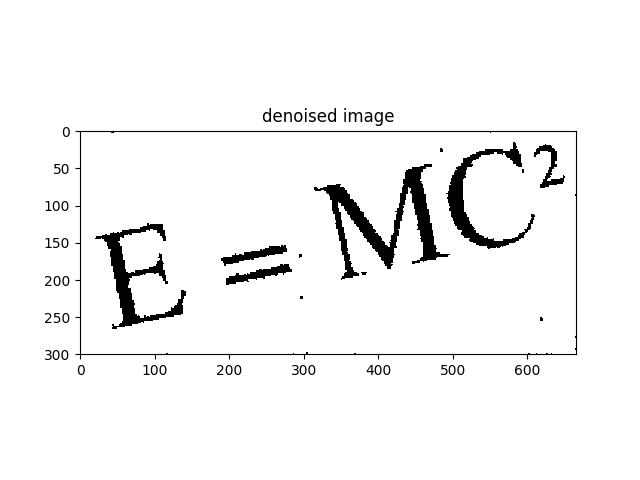
logfile = "output/log\_energy"

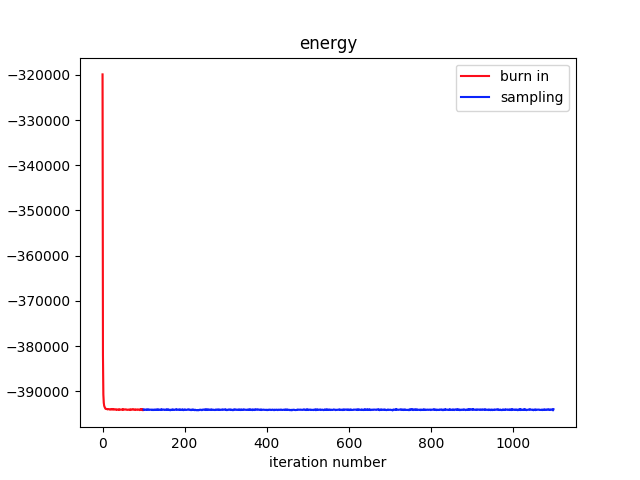
denoised\_img = denoise\_image("data/img\_noisy.png", burn\_in\_steps=burn\_in\_steps,

total\_samples=total\_samples, logfile=logfile)

plot\_energy(logfile)

save\_image(denoised\_img)





**REFERENCE**

[1] [**http://stanford.edu/class/ee367/Winter2018/yue\_ee367\_win18\_report.pdf**](http://stanford.edu/class/ee367/Winter2018/yue_ee367_win18_report.pdf)

[2] [**https://towardsdatascience.com/image-denoising-with-gibbs-sampling-mcmc-concepts-and-code- implementation-11d42a90e153**](https://towardsdatascience.com/image-denoising-with-gibbs-sampling-mcmc-concepts-and-code-%20%20%20implementation-11d42a90e153)

[3] [**https://inf.u-szeged.hu/~ssip/2008/presentations2/Kato\_ssip2008.pdf**](https://inf.u-szeged.hu/~ssip/2008/presentations2/Kato_ssip2008.pdf)

[4] [**https://github.com/AliMorty/Markov-Random-Field-Project**](https://github.com/AliMorty/Markov-Random-Field-Project)

[5] [**https://towardsdatascience.com/markov-random-fields-and-image-processing-20fb4cf7e10d#:~:text=A%20Markov%20Random%20Field%20is,the%20connectivity%20of%20the%20graph**](https://towardsdatascience.com/markov-random-fields-and-image-processing-20fb4cf7e10d#:~:text=A%20Markov%20Random%20Field%20is,the%20connectivity%20of%20the%20graph)**.**

[6] Zoltan Kato and Josiane Zerubia (2012), "Markov Random Fields in Image Segmentation"

[7] **MRF and CRF based Image Denoising and Segmentation,** Wei Zhang, Min Li 2014 International Conference Digital Home