Поля Ґіббса та їх застосування в сегментації зображень (Gibbs fields and their application in image segmentation).

Левченко Іларія Сергіївна

Керівник: Чорней Руслан Констянтинович

Aim of the paper

- develop and programme GRF model, and develop programme for it

Main task

- analyse results and limitations of it for image segmentation using developed model

Gibbs random fields

Gibbs random field a set of random variables $X = \{X_i, i \in S\}$ defined on finite set $= \{1, \dots, N\}$, with respect to neighbourhood $\aleph = \{\aleph(\mathrm{i}), \mathrm{i} \in S\}$, which obeys Gibbs distribution (also term Gibbs measure is is used):

$$P(X = x) = \frac{1}{Z} * \exp\{-\frac{1}{T} E(x)\}$$

where Z is a constant used to normalise distribution: $Z = \sum_x {}_X \exp\{-\frac{1}{T} E(x)\}$. T is temperature constant which is generally assumed to be equal to 1. $E(x)$ is energy function which is equal to the sum of clique potentials.

$$E(x) = \sum_c {}_C \phi_C(x_C)$$

Clique $C \subseteq S$

Markov random fields

All nodes in S are related to each other through the neighbourhood system $\aleph = \{\aleph(\mathrm{i}), \mathrm{i} \in \mathrm{S}\}$, where $\aleph(\mathrm{i})$ is a subset of S, which consists of neighbouring to i, nodes. A node cannot be a neighbour to itself. For a finite set of nodes $S = \{1, \dots, N\}$, MRF is a family of random variables $X_i, i \in S$,which probability function, with relation to the neighbourhood system $\aleph$ satisfies the following conditions [1]:

1) $P[X = x] > 0$ – positivity property, which ensures that all possible values of X have a positive probability of occurring;

2) $P\left[X_i = x_i | X_j = x_i, j \neq i\right] = P[X_i = x_i | X_j = x_j, j \in \aleph(\mathrm{i})]$ – Markov property

The drawback of representing MRFs through local conditional probabilities is that complete system representation is represented through joint probability, and there is no direct method to derive joint probability $P[X_1, \dots, X_N]$ from conditional $P[X_i | X_j, j \in \aleph(\mathrm{i})]$.

Hammersley-Clifford theorem

An MRF is characterized by its Markov property which is a local property, while Gibbs random fields are characterized by the Gibbs distribution, which is a global property.

The Hammersley-Clifford theorem allows us to equate these properties.

Hammersley-Clifford theorem states the equivalence of Markov Random Field and Gibbs Random Field defined on the same graph, with the same set of random variables F set on the set S, with respect to the same neighbourhood system $\aleph$

Redefined Gibbs random fields (1)

Gibbs random fields are models for a set of random variables $X = (x_1, x_2, \dots, x_n)$ on an undirected graph, where these variables are organized in cliques X_c ,where C is a set of cliques; and for each clique the compatibility function $\Psi_C(X_C)$ is defined, such that the probability distribution p over random variables, has the following form:

$$P(X = x) = \frac{1}{Z} \prod_{c\ C} \Psi_c(X_c), \qquad (1)$$

The probability of x is taken with the respect to the joint distribution of the X_c[5]

A clique is a subset of nodes which all are connected with each node in a subset.

$_Z$ is a normalization function to ensure probability distribution, called partition.

Redefined Gibbs random fields (2)

Probability distribution is positive for all domains and can be represented as a Gibbs measure.

$$P(X = x) = \frac{1}{Z} * \exp\{-\frac{1}{T}E(x)\}$$

$$E(x) = \sum_{c\ C} \phi_C(x_C)$$

$$\phi_C(x_C) = -\log(\Psi_C(x_C)) - potential\ on\ clique\ C$$

On such graphs, the Markov property is ensured

Segmentation as pixel labeling task

For each pixel in the picture $s \in S$ the feature vector is defined.

For the picture in general it can be represented as $f = \{\vec{f_s} : s \in S\}$.

The features most commonly include different colour characteristics and brightness.

The set of labels is defined, and for each pixel, a label is assigned $x = \{x_s : s \in S\}$, $x_s \in \Lambda$.

Which results in a $|\Lambda|^{NM}$ variants of labelling, where N *M is the size of the image.
Using GRF for segmentation, means to find optimal labeling from the set of all possible labelings, by maximasing posteriory probability of getting label given feature.

$$x^{MAP} = \arg\max_{x \in \Lambda} P(x|f) = \arg\min_{x \in \Lambda} E(x)$$

Pixel labels

Pixel labels are represented through Gaussian distribution, where mean and variance are estimated through empirical means.

$$P(f_s|x_s) = \frac{1}{\sqrt{2\pi}\sigma_{x_s}} \exp(-\frac{\left(f_s - \mu_{x_s}\right)^2}{2\sigma_{x_s}^2})$$

$$\forall x \in \Lambda:$$

$$\mu_x = \frac{1}{|S_x|} \sum_{s \in S_x} f_s$$

$$\sigma_{x_s}^2 = \frac{1}{|S_x|} \sum_{s \in S_x} (f_s - \mu_x)^2$$

S_x is a set of pixels.

Problem formulation

- Let X be coloured image we observe and which we consider a representation of random field. The problem, is to find true labeling field for this image, where each of the labels correspond to one of the pixels.
- Two layer model was used: one layer of pixels and its values,; and second is label layer.

Energy function

ϕ_C denotes clique potential of clique C with labeling x_C.

We focus only on cliques of two sizes, assigning for all other potential functions value zero, which is considered satisfactory to model spatial dependencies: singleton (c_0 – (node itself, and doubleton $\{c_1, c_2, c_3, c_4$ – {cliques of two nodes (node and it's neighbour).

$$E(x) = \sum_{c\ C} \phi_c(x) = \sum_{iC_1} \phi_{C_1}(x_i) + \sum_{(i,j)C_2} \phi_{C_2}(x_i, x_j) + \cdots$$

Singleton here is proportional to the probability of features given label: log(P(f | x)).

Doubleton prefers smoothness, meaning that neighbouring pixels belong to the same label.

$$\phi_{C_2}(x_i, x_j) = \beta\delta(x_i, x_j) = \begin{cases} -\beta & if \quad x_i = x_j \\ +\beta & if \quad x_i \neq x_j \end{cases}$$

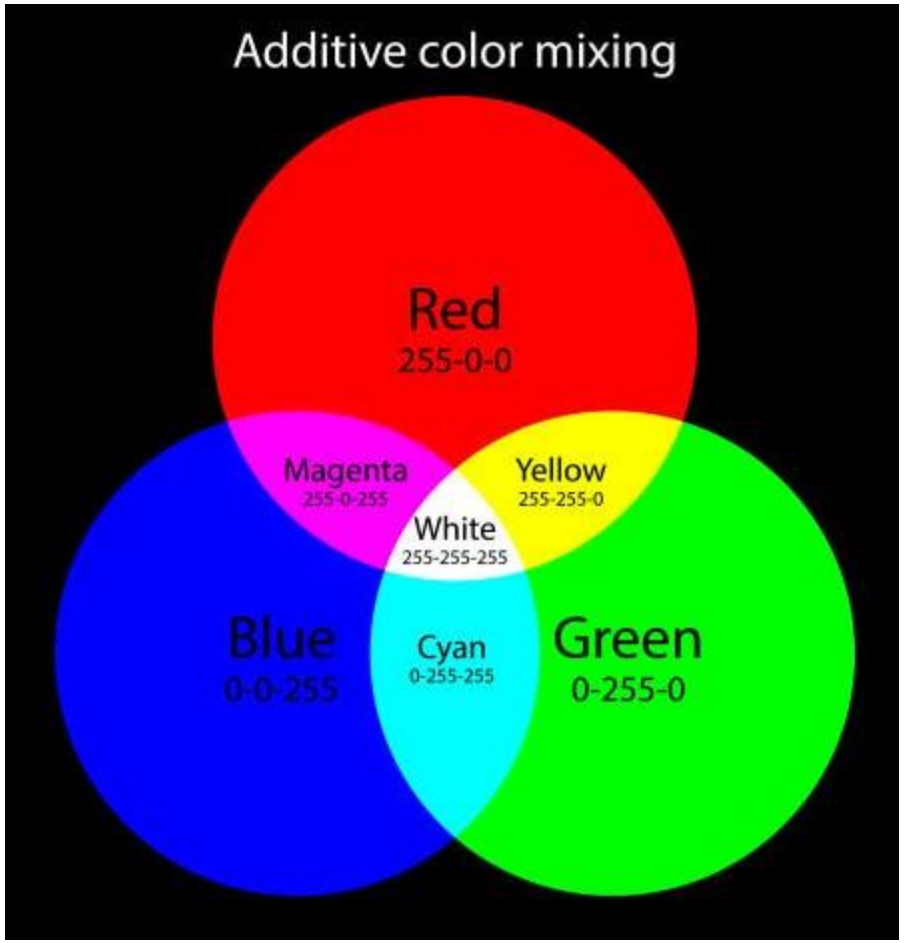
We can now define

$$E(x) = \sum_s \left(\sqrt{2\pi}\sigma_{x_s} + \frac{(f_s - \mu_{x_s})^2}{2\sigma_{x_s}^2} \right) + \sum_{s,w} \beta\delta(x_s, x_w)$$

Colour spaces: RGB

It is coordinate system represented by three primary colours (Red, Green, Blue) and each colour is represented by a vector stating intensity of each of the values (from 0 to 255).
However, while RGB colour space is useful in displaying colour, is not perfect. Disadvantages of RGB colour space include:

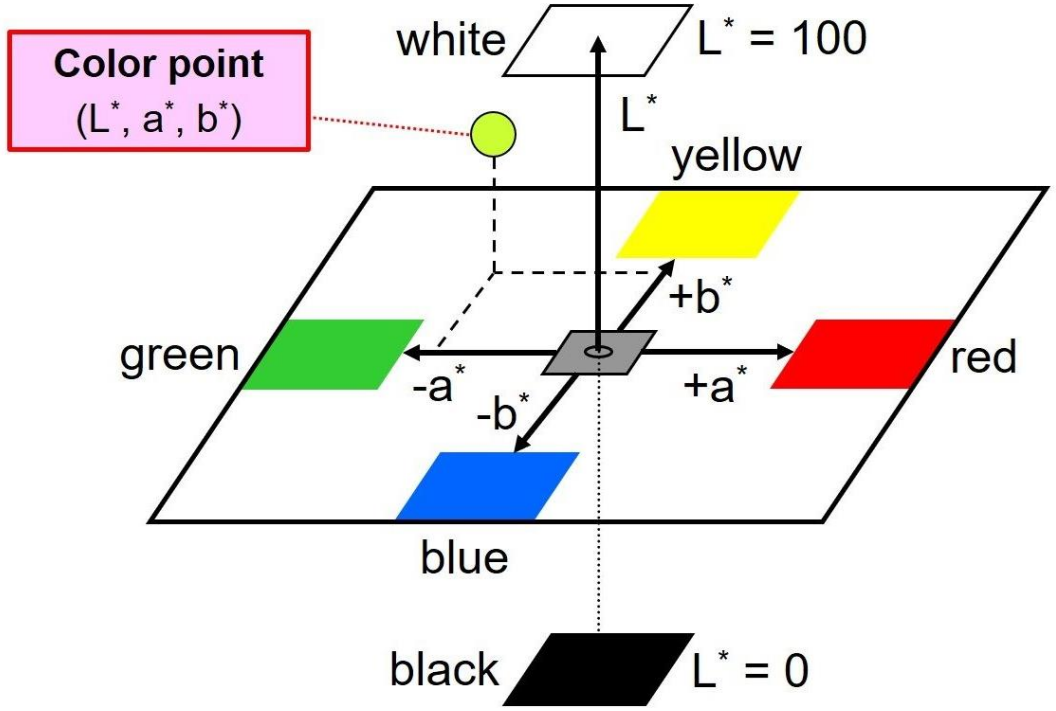
- Differently displayed on different devices,
- not perceptually uniform,
- the difference between colours is not linear.



Colour spaces: CIELAB

CIELAB color space, also referred to as L*a*b*.

This colour space also present colours in vectors of size three: L indicates lightness of the colour, while a and b representing two scales from red to green and from blue to yellow. While it is not perfect it was created to be perceptually uniform and one of the most commonly used.



Colour spaces: Ohta's colour space

Another colourspace used was Ohta's colour space also known as I_1 I_2 I_3, which is less known but was developed for segmentation and is easy to transform to from RGB system

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{2} & 0 & -\frac{1}{2} \\ -\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Algorithm

1. Initialise
2. Optimize
3. Reconstruct and save segmented image

Initialise: initial parameters

BETA = 1

TEMPERATURE = 5.0

COOLRATE = 0.95

NEIGHBORHOOD= [(-1,0) , (1,0) , (0,-1) , (0,1)]

beta_variances, beta_neighbours, coefcol, coefbrightnes, diffort=dt

segmentss=[2,3,4,5,6,10]

iterationss=[100000,1000000,2000000,3000000,5000000]

Initialise: image

```
def input_bw (imagepath):
    original = cv2.imread(imagepath)
    return cv2.cvtColor(original,cv2.COLOR_BGR2GRAY)

def input_RGB (imagepath):
    return cv2.imread(imagepath)

def input_LAB (imagepath):
    original = cv2.imread(imagepath)
    return cv2.cvtColor(original.astype(np.float32)/255, cv2.COLOR_RGB2Lab)

def input_OHTA (imagepath):
    original=cv2.imread(imagepath)
    for i in original:
        for j in i:
            R=j[0]
            G=j[1]
            B=j[2]
            i1=float(R*(1/3)+G*(1/3)+B*(1/3))
            i2=float(R*(1/2)+B*(-1/2))
            i3=float(R*(-1/4)+G*(1/2)+B*(-1/4))
            j=[i1,i2,i3]
    return original
```

Initialise: energy function

```
def initialize_bw(img):
    labels = np.zeros(shape=img.shape ,dtype=np.uint8)
    nos = [0.0]*SEGMENTS
    sums = [0.0]*SEGMENTS
    squares = [0.0]*SEGMENTS
    for i in range(len(img)):
        for j in range(len(img[0])):
            #print(labels)
            l = randint(0,SEGMENTS-1)
            #print(img[i][j])
            sums[l] += img[i][j]
            squares[l] += img[i][j]**2
            nos[l] += 1.0
            labels[i][j] = l
    return (sums,squares,nos,labels)
```

```
def initialize_colour(img):
    labels = np.zeros(shape=(img.shape[0],img.shape[1]) ,dtype=np.uint8)
    nos = [0.0]*SEGMENTS
    sums0 = [0.0]*SEGMENTS
    squares0 = [0.0]*SEGMENTS
    sums1 = [0.0]*SEGMENTS
    squares1 = [0.0]*SEGMENTS
    sums2 = [0.0]*SEGMENTS
    squares2 = [0.0]*SEGMENTS
    for i in range(len(img)):
        for j in range(len(img[0])):
            #print(labels)
            l = randint(0,SEGMENTS-1)
            #print(img[i][j])
            sums0[l] += img[i][j][0]
            sums1[l] += img[i][j][1]
            sums2[l] += img[i][j][2]
            squares0[l] += img[i][j][0]**2
            squares1[l] += img[i][j][1]**2
            squares2[l] += img[i][j][2]**2
            nos[l] += 1.0
            labels[i][j] = l
    return (sums0,squares0,sums1,squares1,sums2,squares2,nos,labels)
```

```
def variance(sums1,squares1,nos1):
    t=(squares1-((sums1/nos1)**2))/nos1
    return t
```

Optimization algorithm: simulated annealing

1. Initiate the Temperature T.
2. Compute the energy of the current state of the system.
3. Introduce slight changes to the current state of system.
4. Compute the new energy of the new state and compare it with energy of previous state.
5. If energy improved (is lower for new state), accept new state of the system. Else, accept new state of the system with some small probability
6. Decrease the temperature.
7. Repeat from step 3, till temperature decreased sufficiently (or the set number of iterations are used).

Energy functions

```
def calculateEnergy_bw(img,variances,labels):
    energy = 0.0
    for i in range(len(img)):
        for j in range(len(img[0])):
            l = labels[i][j]
            energy += math.log(math.sqrt(2*math.pi*variances[l]))+(((img[i][j]-(sums[l]/nos[l]))**2)/(2*variances[l]))
            for (p,q) in NEIGHBORS:
                if isSafe(img.shape[0],img.shape[1],i+p,j+q):
                    energy += (delta(l,labels[i+p][j+q]))
    return energy
```

```
def calculateEnergy_colour(img,variances,labels):
    energy = 0.0
    for i in range(len(img)):
        for j in range(len(img[0])):
            l = labels[i][j]
            energy += math.log(math.sqrt(2*math.pi*variances[l]))
            energy += ((((img[i][j][0]-(sums0[l]/nos[l]))**2)+
                       ((img[i][j][1]-(sums1[l]/nos[l]))**2)+((img[i][j][2]-(sums2[l]/nos[l]))**2))/(2*variances[l]))
            for (p,q) in NEIGHBORS:
                if isSafe(img.shape[0],img.shape[1],i+p,j+q):
                    energy += (delta(l,labels[i+p][j+q])/2.0)
    return energy
```

Energy functions

```
def calculateEnergy_clr(img,variances,labels):
    energy = 0.0
    energy2=0.0
    for i in range(len(img)):
        for j in range(len(img[0])):
            l = labels[i][j]
            energy += math.log(math.sqrt(2*math.pi*variances[l]))
            energy += ((((img[i][j][0]-(sums0[l]/nos[l]))**2)+
                        ((img[i][j][1]-(sums1[l]/nos[l]))**2)+((img[i][j][2]-(sums2[l]/nos[l]))**2))/(2*variances[l]))
            for (p,q) in NEIGHBORS:
                if isSafe(img.shape[0],img.shape[1],i+p,j+q):
                    energy2 += (delta(l,labels[i+p][j+q])/2.0)
    return beta_variances*energy+beta_neighbours*energy2
```

Energy functions

```
def calculateEnergy_clr2(img,variances,labels):
    energy = 0.0
    energy2=0.0
    for i in range(len(img)):
        for j in range(len(img[0])):
            l = labels[i][j]
            energy += math.log(math.sqrt(2*math.pi*variances[l]))
            energy += ((((img[i][j][0]-(sums0[l]/nos[l]))**2)+
                        ((img[i][j][1]-(sums1[l]/nos[l]))**2)+((img[i][j][2]-(sums2[l]/nos[l]))**2))/(2*variances[l]))
            for (p,q) in NEIGHBORS:
                if isSafe(img.shape[0],img.shape[1],i+p,j+q):
                    t=0
                    if img[i][j][0]-img[i+p][j+q][0]<coef and img[i][j][0]-img[i+p][j+q][0]>coef:
                        t=t+1
                    if img[i][j][1]-img[i+p][j+q][1]<coef and img[i][j][1]-img[i+p][j+q][1]>coef:
                        t=t+1
                    if img[i][j][2]-img[i+p][j+q][2]<coefbrightnes and img[i][j][2]-img[i+p][j+q][2]>coefbrightnes:
                        t=t+1
                    delt=delta(l,labels[i+p][j+q])
                    if t>=diffort and l==labels[i+p][j+q]:
                        energy2 += ((-BETA))
                    else:
                        energy2 += ((BETA))


    return beta_variances*energy+beta_neighbours*energy2
```

Energy functions

```
def calculateEnergy_clr22(img,variances,labels):
    energy = 0.0
    energy2=0.0
    for i in range(len(img)):
        for j in range(len(img[0])):
            l = labels[i][j]
            energy += math.log(math.sqrt(2*math.pi*variances[l]))
            energy += ((((img[i][j][0]-(sums0[l]/number[l]))**2)+
                        ((img[i][j][1]-(sums1[l]/number[l]))**2)+((img[i][j][2]-(sums2[l]/number[l]))**2))/(2*variances[l]))
            for (p,q) in NEIGHBORHOOD:
                if isSafe(img.shape[0],img.shape[1],i+p,j+q):
                    t=0
                    tb=0
                    temp=float(img[i][j][0])-float(img[i+p][j+q][0])
                    if (temp<coefcol0 and temp>-coefcol0) :
                        t=t+1
                    if (temp>difbord0 and temp<-difbord0) :
                        tb=tb+1
                    temp=float(img[i][j][1])-float(img[i+p][j+q][1])
                    if (temp<coefcol1 and temp>-coefcol1) :
                        t=t+1
                    if (temp>difbord1 and temp<-difbord1) :
                        tb=tb+1
                    temp=float(img[i][j][2])-float(img[i+p][j+q][2])
                    if (temp<coefcol2 and temp>-coefcol2) :
                        t=t+1
                    if (temp>difbord2 and temp<-difbord2) :
                        tb=tb+1

                    if (t>=diffort and l==labels[i+p][j+q]) or (tb>=diffort and l!=labels[i+p][j+q]):
                        energy2 += ((-BETA))
                    else:
                        energy2 += ((BETA))


    return beta_variances*energy+beta_neighbours*energy2
```

Reconstruct and save

```
def reconstruct(labs):
    labels = labs
    for i in range(len(labels)):
        for j in range(len(labels[0])):
            labels[i][j] = (labels[i][j]*255)/(SEGMENTS-1)
    return labels
```

```
def save (colour, energy_function, neighbours):
    string=""
    #create string for the name with information on the parameters used#
    plt.imshow(reconstruct(labels) , interpolation='nearest',cmap='gray')
    cv2.imwrite(string+'.jpg',labels)
    print(string)
```

Results: colourspace

Colourspace - greyscale

Segments – 2

Energy function – basic (calculateEnergy_bw)

Iterations 1000000

T-4.0

Colourspace – RGB

Segments – 2

Energy function – calculateEnergy_colour

Iterations - 1000000

T-4.0

Colourspace – L*a*b*

Segments – 2

Energy function – calculateEnergy_colour

Iterations - 1000000

T-4.0

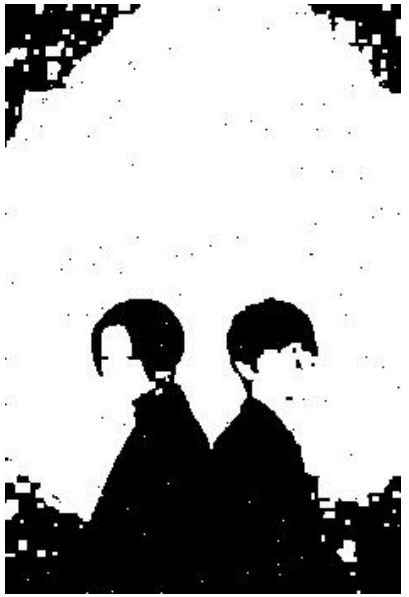
Colourspace – L*a*b*

Segments – 2

Energy function – calculateEnergy_colour without utilizing lightness parameter

Iterations - 1000000

T-4.0



Colourspace – Ohta

Segments – 2

Energy function – calculateEnergy_clr (with coefficients bv-0.25_bn2-0.75)

Iterations - 1000000

Results: number of iterations

Colourspace - greyscale

Segments – 2

Energy function – basic (calculateEnergy_bw)

Iterations - 2000000

T-10.0

Colourspace – L*a*b*

Segments – 2

Energy function – calculateEnergy_clr (with coefficients bv-0.25_bn2-0.75)

Iterations - 2000000

T - 5

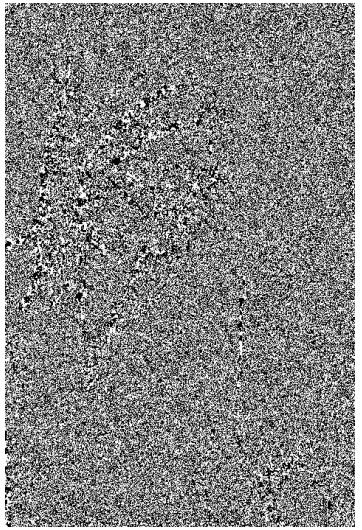
Colourspace – OHTA

Segments – 2

Energy function – calculateEnergy_clr (with coefficients bv-0.25_bn2-0.75)

Iterations - 2000000

T - 5



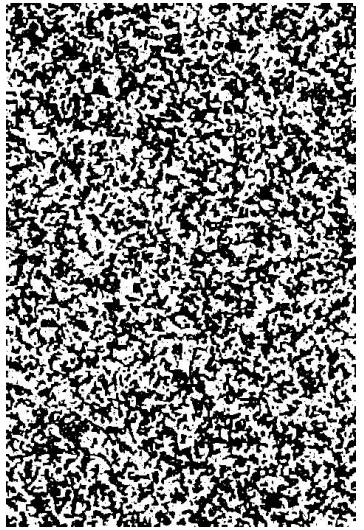
Colourspace – OHTA

Segments – 2

Energy function – calculateEnergy_clr2 (with coefficients coefcol-0.1_coefb-0.1_dif-2_bv-0.25_bn2-0.75)

Iterations - 2000000

T - 5



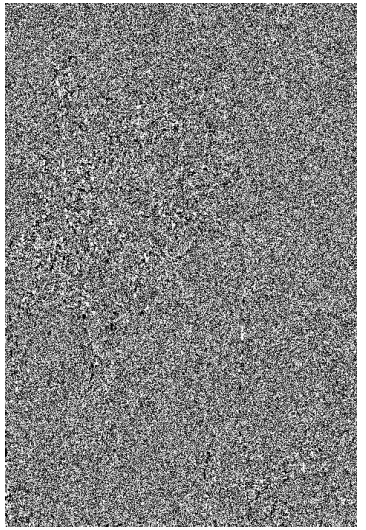
Colourspace – L*a*b*

Segments – 3

Energy function – calculateEnergy_clr22 (with coefficients coefcol-0.01_dif1-1_difborders-0.1-_bv-0.5_bn2-0.5)

Iterations - 2000000

T - 5



Colourspace – L*a*b*

Segments – 3

Energy function – calculateEnergy_clr22 (with coefficients coefcol-0.01dif1-1_difborders-0.1_bv-0.5_bn2-0.5)

Iterations - 2000000

T - 5

Results: weighted energy function

Colourspace – L*a*b*

Segments – 3

Energy function – calculateEnergy_clr22

Coefficient for first sum - bv-0

Coeficient for second sum in energy function - bn2-1

Iterations - 1000000

T - 4

Colourspace – L*a*b*

Segments – 3

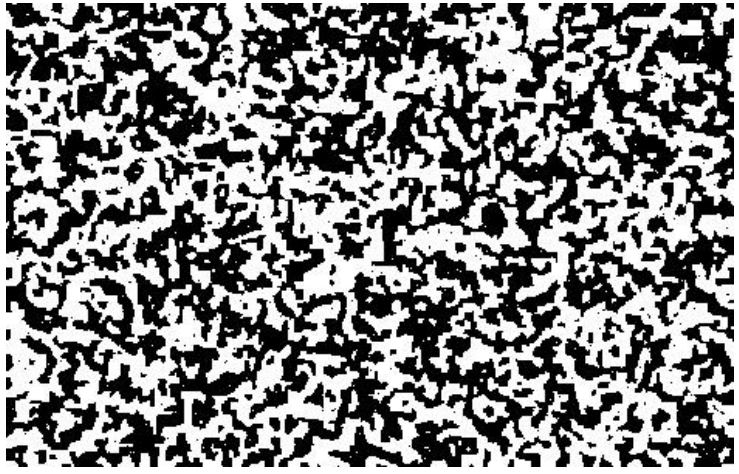
Energy function – calculateEnergy_clr22

Coefficient for first sum - bv-1

Coeficient for second sum in energy function - bn2-0

Iterations - 1000000

T - 4



Colourspace – OHTA

Segments – 2

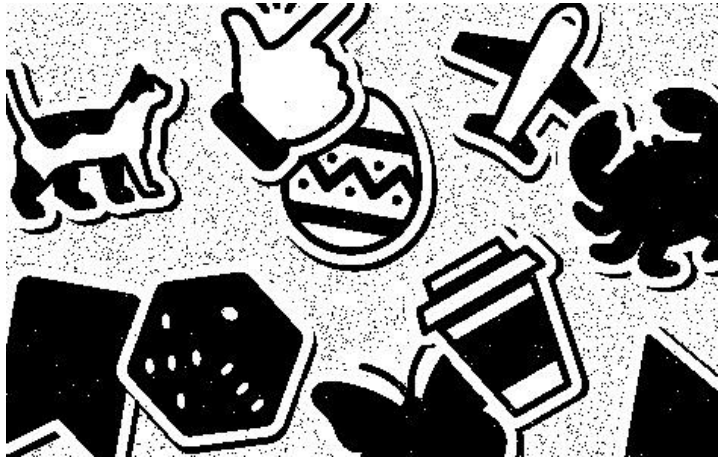
Energy function – calculateEnergy_clr (with coefficients coefcol-10_coefb-20_dif-2_bv-0_bn2-1)

Coefficient for first sum - bv-0

Coeficient for second sum in energy function - bn2-1

Iterations - 2000000

T - 4



Colourspace – OHTA

Segments – 2

Energy function – calculateEnergy_clr

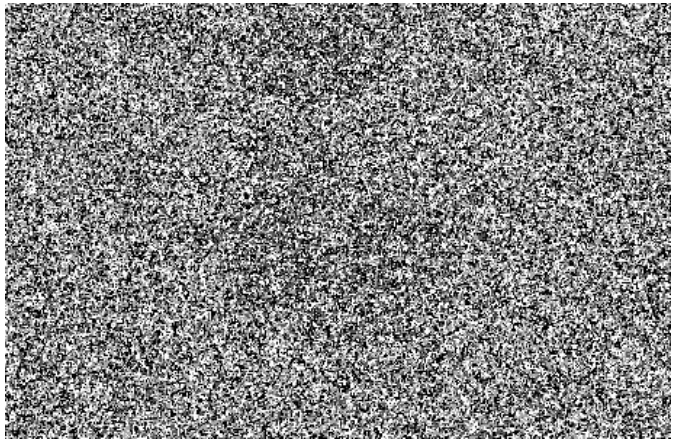
Coefficient for first sum - bv-1

Coeficient for second sum in energy function - bn2-0

Iterations - 2000000

T - 4

Results: iterations



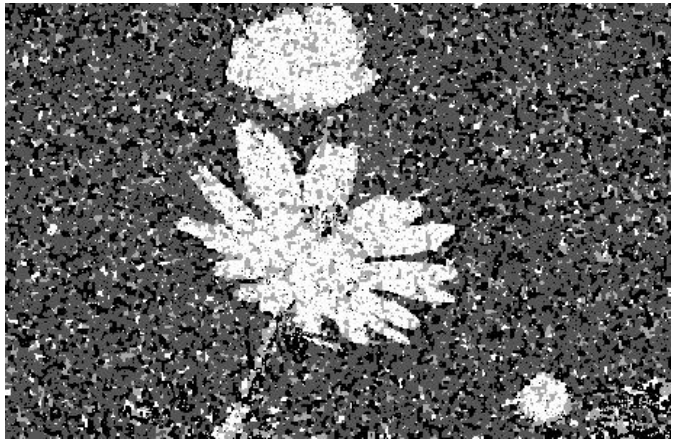
Colourspace – RGB

Segments – 4

Energy function – calculateEnergy_colour

Iterations - 100000

T – 4



Colourspace – RGB

Segments – 4

Energy function – calculateEnergy_colour

Iterations - 1000000

T – 4

Colourspace – L*a*b*

Segments – 2

Energy function – calculateEnergy_colou r

Iterations - 100000

T - 4

Colourspace – L*a*b*

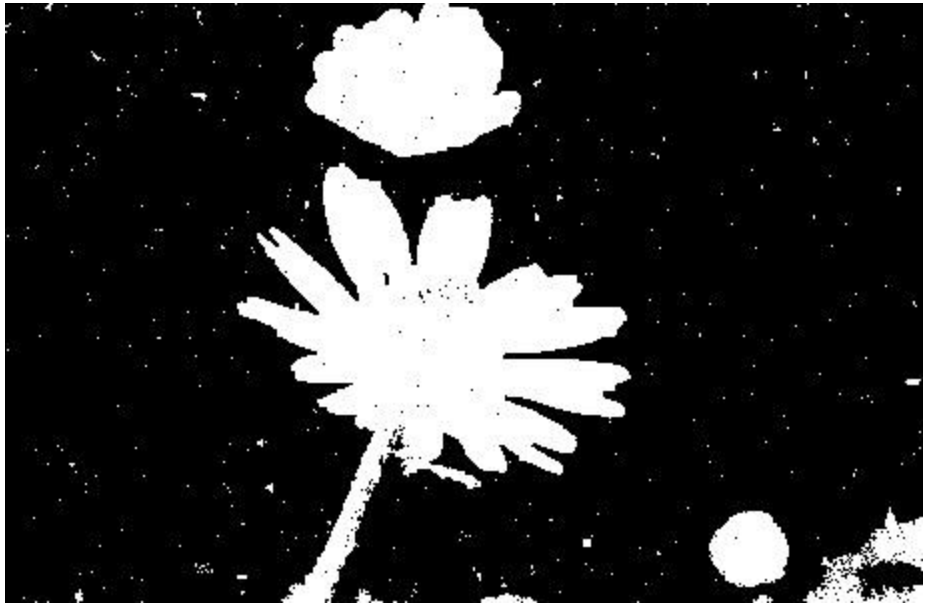
Segments – 2

Energy function – calculateEnergy_colou r

Iterations - 1000000

T - 4

Results: neighborhood



Colourspace - greyscale

Segments – 2

Energy function – basic (calculateEnergy_bw)

Iterations - 1000000

T-4.0

Neighbourhood includes four pixels

Colourspace - greyscale

Segments – 2

Energy function – basic (calculateEnergy_bw)

Iterations - 1000000

T-4.0

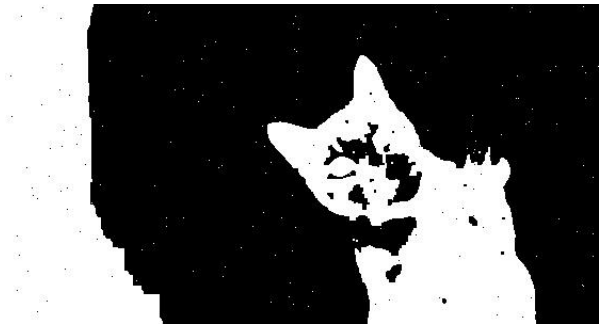
Neighborhood includes eight pixels

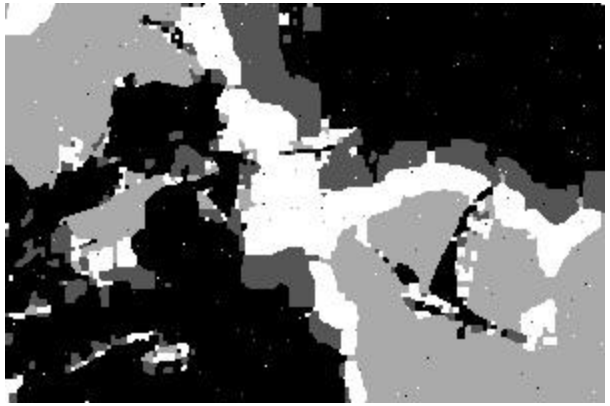
Conclusions: simple images

The best results were achieved on flat images, without gradient or significant shadows or much details. Greyscale is enough for such images and demonstrates great results.

Conclusions: gradient

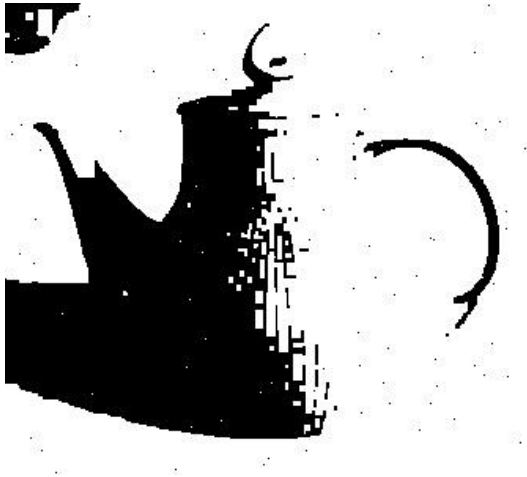
Simple gradient in pictures is not recognized as one segment





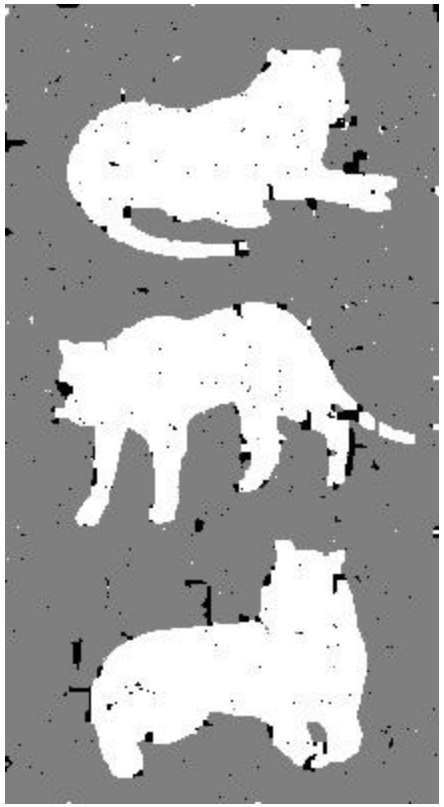
Conclusions: shadows

While people recognize the colour in shadow and in light as one colour, none of the colourspaces reflect that. None of the energy functions or colourspaces used were efficient in combating the problem



Conclusions: number of labels

Another problem, is inability to effectively control number of labels, or more precisely, segmentation of the image on the greater amount of labels.



Colourspace - greyscale

Segments – 3

Energy function – basic (calculateEnergy_bw)

Iterations - 2000000

T-10.0

Colourspace - greyscale

Segments – 4

Energy function – basic (calculateEnergy_bw)

Iterations - 4000000

T-10.0

Colourspace - greyscale

Segments – 4

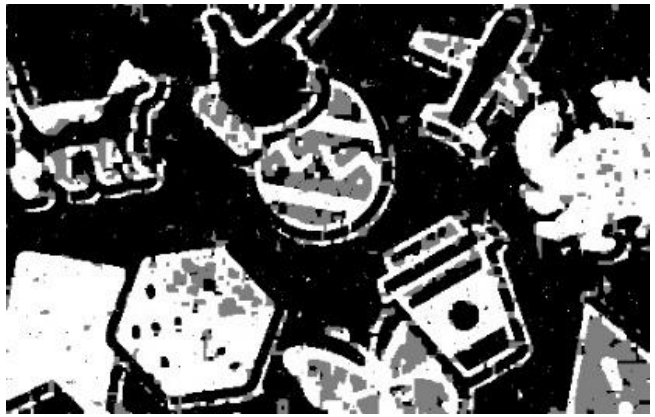
Energy function – basic (calculateEnergy_bw)

Iterations - 100000000

T-4

Conclusions: segments

The last identified problem of universal usage of MRFs for image segmentation, is how image is being segmented itself. As there are no element of control or learning, and energy function are dependent only on colour feature (even if in different formats), segments identified are homogemeus but segmentation is not semantic.



Conclusions

While GRFs introduce benefits in a way of presenting images and have definitely use in image processing and segmentation, they are in no way universal tool and cannot be applied to any image without preprocessing, some form of supervision or learning.

Such model is much more useful for simple images with clear foreground and background, and only two main segments.

Sources

[1] - Oliver C. Ibe, in Markov Processes for Stochastic Modeling (Second Edition), 2013

[2] Hammersley, J. M.; Clifford, P., Markov fields on finite graphs and lattices. 1971

[3] Koralov, L., Sinai, Y.G.. Gibbs Random Fields. In: Theory of Probability and Random Processes. Universitext. Springer, Berlin, Heidelberg. 2012. https://doi.org/10.1007/978-3-540-68829-7_22

[4] Li, S. Z. . Markov Random Field Modeling in Image Analysis. In Computer Science Workbench. Springer Japan. 2001. https://doi.org/10.1007/978-4-431-67044-5

[5] Markov Random Fields W.G.H.S. Weligampola (E/14/379) June 2020

[6] Ian H. Witten, ... Christopher J. Pal, in Data Mining (Fourth Edition), 2017

[7] Shun-Zheng Yu, in Hidden Semi-Markov Models, 2016

[8] - Huawu Deng∗, David A, Unsupervised image segmentation using a simple MRF model with a new implementation scheme. Clausi Department of Systems Design Engineering, University of Waterloo, 200 University Avenue West, Waterloo, Ont., Canada N2L 3G1 Received 15 October 2003; accepted 27 April 2004

[9] - S.Z. Li, Markov Random Field Modeling in Computer Vision, Springer, New York, 2001

[10] G. Aubert, P. Kornprobst, in Encyclopedia of Mathematical Physics, 2006

[11] Mohapatra, Parthajit et al. “Color Image Segmentation Using MRF Model and Simulated Annealing.” 2006.

[12] - Jiebo Luo, Chang Wen Chen, Kevin J. Parker, "Applications of Gibbs random field in image processing: from segmentation to enhancement," Proc. SPIE 2308, Visual Communications and Image Processing '94, 1994; doi: 10.1117/12.185954

[13] Yuri Boykov, Olga Veksler and Ramin Zabih, Fast Approximate energy minimization via Graph Cuts, PAMI 2001

[14] Introduction to Markov Random Fields. In Markov Random Fields for Vision and Image Processing. The MIT Press. 2011https://doi.org/10.7551/mitpress/8579.003.0001

[15] Julien Stoehr, Richard Everitt, Matthew T. Moores. A review on statistical inference methods for discrete Markov random fields. 2017. ffhal-01462078v2f

[16] Kato, Z., & Pong, T.-C. A Markov random field image segmentation model for color textured images. In Image and Vision Computing (Vol. 24, Issue 10, pp. 1103–1114). Elsevier BV. 2006. https://doi.org/10.1016/j.imavis.2006.03.005

Thank you for your attention!