CS5340 Project Report

Team Members:
Zheng Pingxia - A0105662L
Hui Hui - A0105566E

Task 1: Gibbs Sampling

Algorithm

First algorithm used to denoise image is Gibbs Sampling. Local evidence is modelled as normal distribution with standard deviation as 2. That is, $\psi_t(x_t) = N(y_t|x_t, 4)$. The edge strength is set to 1. The edge potential is therefore defined as $\psi(x_s, x_t) = exp(x_s x_t)$. Sequential update strategy is applied. For example, we first update x at coordinate (i, j) then x at coordinate (i, j+1) with the updated value of its left neighbor.

Library

The external packages used are numpy and scipy. Numpy is used for math and matrix operations, and scipy is used for gaussian probability density function.

Output

Output is generated by running gibbs_sampler.py. Left is the original image, right is the denoised image.

Image 1:

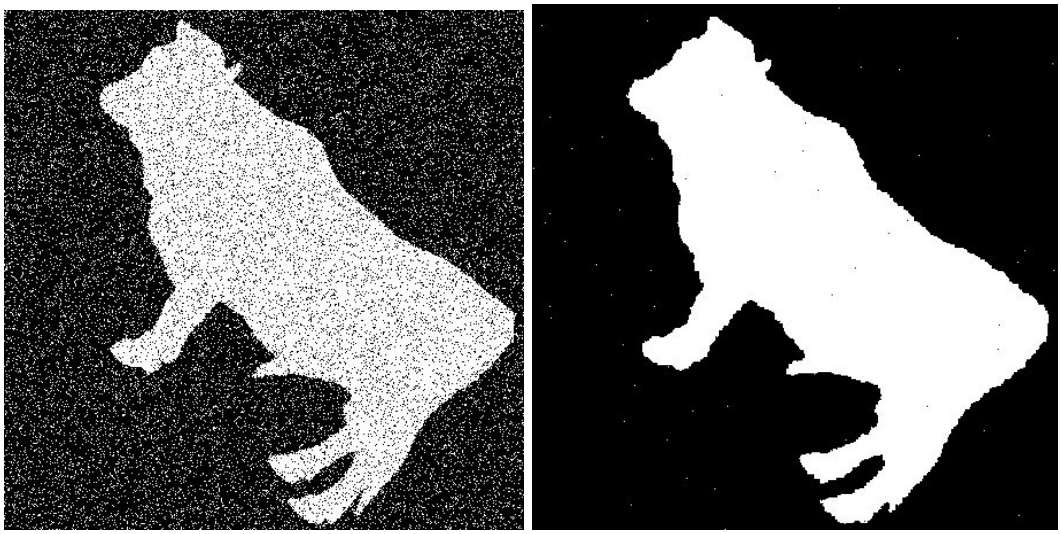


Image 2:

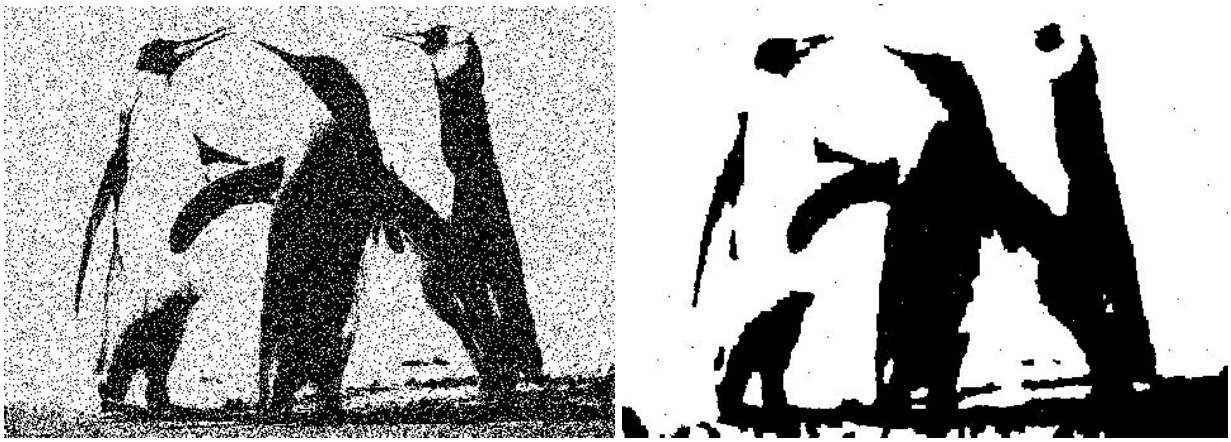


Image 3:

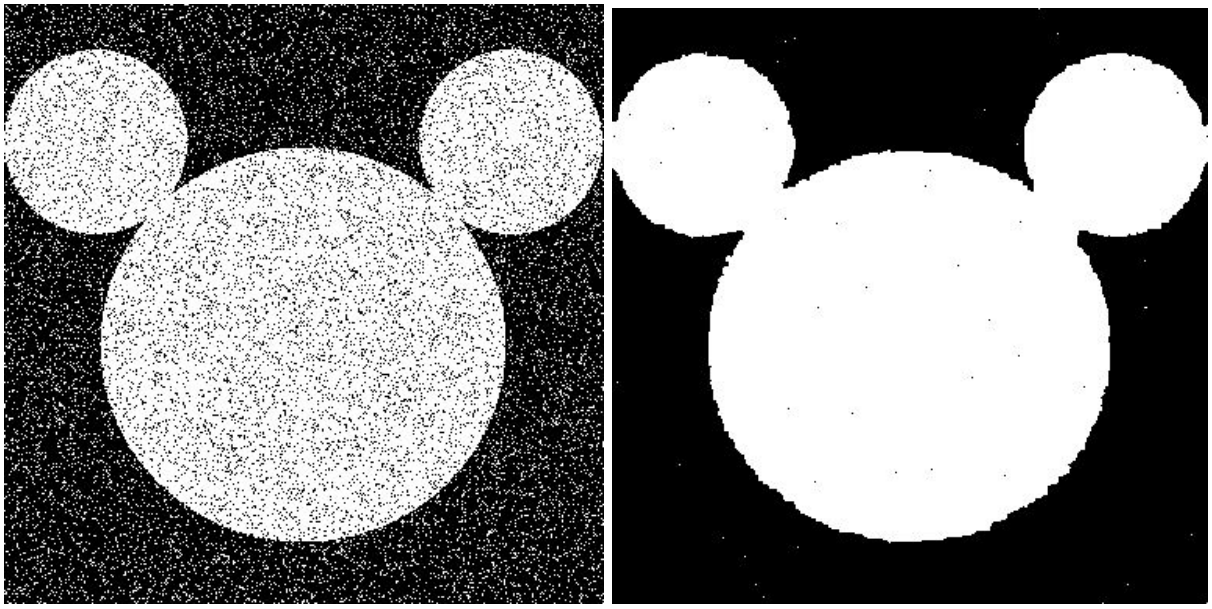
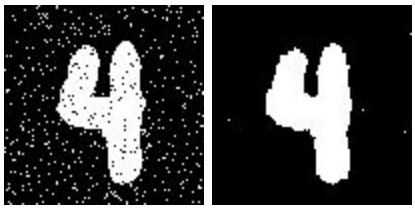


Image 4:



Task 2: Variational Inference

Algorithm

Variational inference is used to denoise the image as well. The update rule is shown below

$$\mu_i^t = \lambda\mu_i^{t-1} + (1-\lambda)tanh(\sum_{j \in neighbor(i)} W_{ij}\mu_j^{t-1} + 0.5(L_i(1) - L_i(-1)))$$

To simplify the notation, the data is assumed to be 1-d array. It can be easily extended to support 2-d image by changing μ_i^t to μ_{im}^t, W_{ij} to W_{imj}, L_i to L_{im} and j represents 2-d coordinator of all neighbors of (i,m).

Damped update form is used and λ is set to 0.5. W is set to 1 for all values. $L_i(x_i) = \log(N(y_i|x_i, 2))$.

Mean value is first initialized to be all zero. At each iteration, we calculate the mean value for a position based on the mean values for its neighbors and itself produced in previous iteration. The algorithm is run 15 iterations to produce the denoised image.

Library

The external packages used are numpy and scipy. Numpy is used for math and matrix operations, and scipy is used for gaussian probability density function.

Output

Output is generated by running variational_sampler.py. Left is the original image, right is the denoised image.
Image 1:

Image 2:

Image 3:

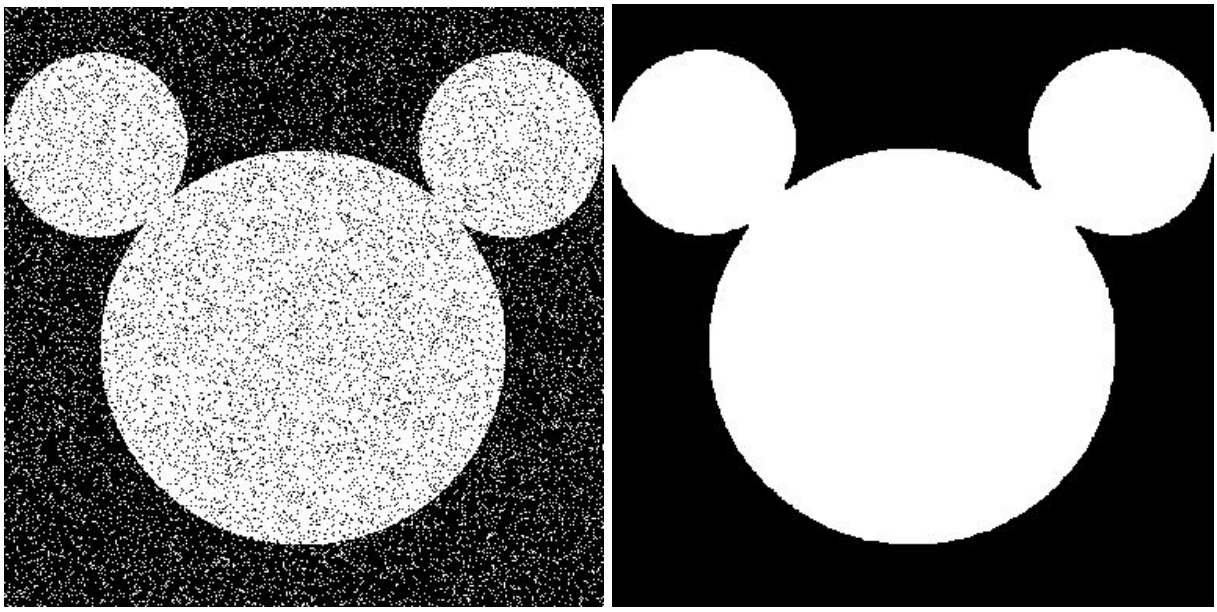
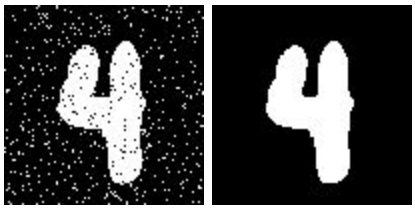


Image 4:



Task 3: EM Segmentation

Algorithm

In this task, an image is provided as input to the algorithm, and the number of segments is 2. Each segment is modeled as a Gaussian distribution with parameter $\theta_k(\mu_k,\Sigma_k)$, and also has weight α_k. The Gaussian mixture model is $p(x|\theta) = \sum_{k=1}^{k} \alpha_k p_k(x|\theta_k)$.

The algorithm starts with random initialization of each θ_k, and then computes the responsibility matrix in E step, followed by re-estimating parameter θ_k based on responsibility matrix in M step. The process is repeated until convergence of p(x|θ) log likelihood is met.

Each pixel is converted into a vector with 3 values, namely the RGB values. The pixel values of the entire image are used as features in EM algorithm.

Random initialization of parameters $(\alpha_k,\mu_k,\Sigma_k)$ is as follows:

$$\alpha_k = \frac{1}{2}$$
$$\mu_k = \textit{the pixel value of a random pixel}$$
$$\Sigma_k = \textit{Identity Matrix of size } (3,3)$$

When convergence is met, for every pixel, the segment label with higher probability in responsibility matrix is assigned.

Library

The external library used is numpy, which is used for matrix and math operation, such as np.log, np.sum, np.multiply etc.

Output

Output is generated by running em.py.

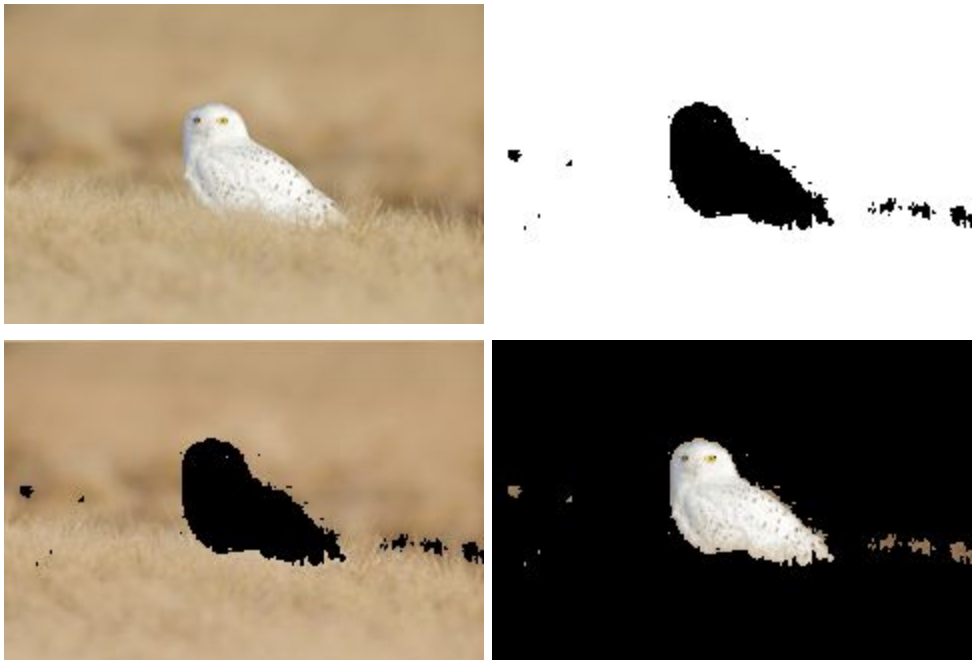
Cow:



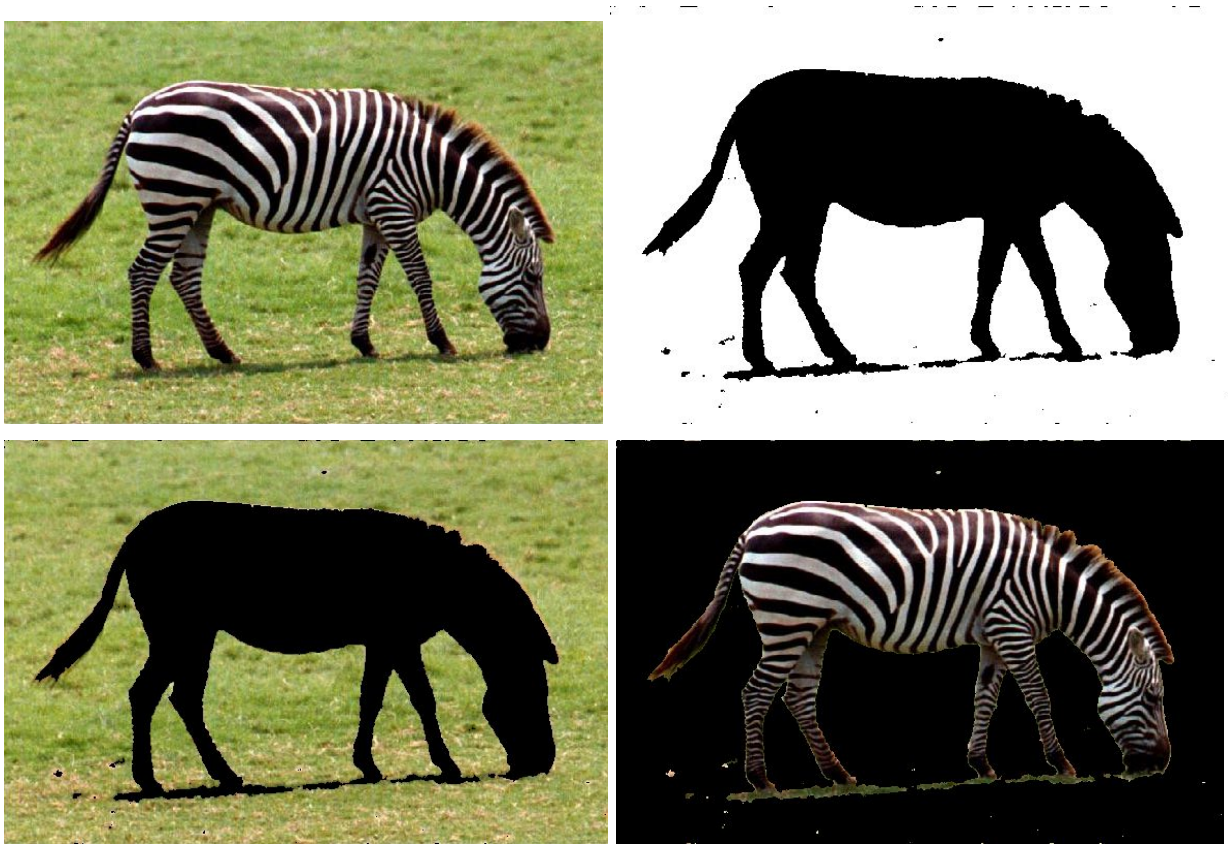

Fox:




Owl:



Zebra:



Submission

The code implementation and outputs are zipped together into submission file A0105662L-A0105566E.zip. Please find the instructions to run the code inside README.md.