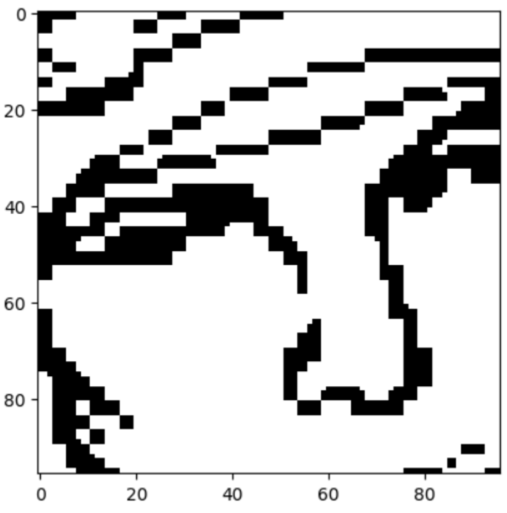
Image Denoising

Akash Santhanakrishnan

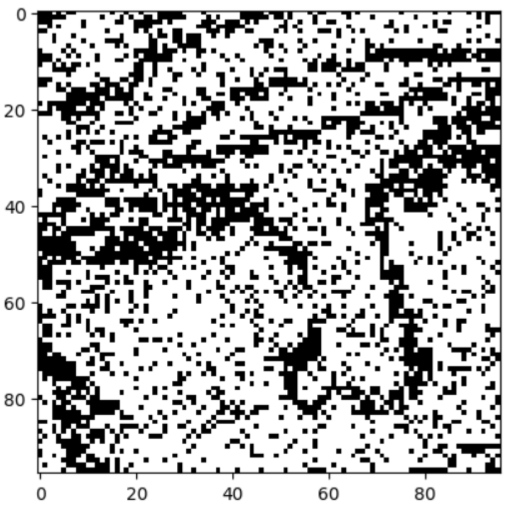
1 Introduction

In this problem, we aim to improve the quality of binary images by removing noise, which can be represented as random fluctuations in pixel values. To do this, we will denoise images of size $\sqrt{n} \times \sqrt{n}$ using the **Loopy belfief propogation method**. Each element of the matrix can be either 1 (White Pixels) or -1 (Black Pixels).

Here is a sample image that is converted into a binary matrix of 1 and -1:



To introduce noise into the image, for each pixel, we swap its value between 1 and -1 by some rate 0.2



1.1 The Loopy BP Algorithm

The Loopy-BP algorithm iterative updates the messages of each node through a sum-product operation. In **sum-product** belief propagation, joint inbound messages are calculated by multiplying relevant factors together, followed by the marginalization of these factors through summation. This approach differs from max-product belief propagation, where the focus is on determining the maximum a-posteriori value for each variable by selecting the maximum value over variable

Initialization:

For discrete node x_j with 2 possible states, $m_{i\to j}$ can be written as a 2 dimensional real vector $\mathrm{m}_{i,j}$ with $m_{i\to j}(x_j) = \mathrm{m}_{i,j}[index(x_j)]$. We initialize them uniformly to $m_{i\to j}(x_j) = 1/2$. (Aside: for continuous cases, $m_{i\to j}(x_j)$ is a real valued function of x_j. We only need to deal with the discrete case here.)

For a number of iterations:

For node x_j in $\{x_s\}_{s=1}^n$:

1. Compute the product of inbound messages from neighbours of x_j:

$$\prod_{k\in N(j)\neq i} m_{k\to j}(x_j)$$

2. Compute potentials $\psi_j(x_j) = \exp(\beta x_j y_j)$ and $\psi_{ij}(x_i, x_j) = \exp(J x_i x_j)$. This expression specifically holds when $x \in \{-1, +1\}$.

3. Marginalize over $x_j = \{-1, +1\}$ to get $m_{j\to i}(x_i)$:

$$m_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)\neq i} m_{k\to j}(x_j)$$

4. Normalize messages for stability $m_{j\to i}(x_i) = m_{j\to i}(x_i)/\sum_{x_i} m_{j\to i}(x_i)$.

Compute beliefs after message passing is done.

$$b(x_i) \propto \psi_i(x_i) \prod_{j\in N(i)} m_{j\to i}(x_i).$$

1.2 Initialization

Initialize the message between neighbor pixels uniformly as $m_{j\text{ß}i}(x_i) = 1/k$. Since each pixel can only be 1 or -1, message has two values $m_{j\text{ß}i}(1)$ and $m_{j\text{ß}i}(-1)$. We also initialize hyperparameters J and β. Find the neighboring pixels around a given pixel, which will be used for BP updates.

1.3 Message Passing in BP

Implement the function ‘get_message()‘ that computes the message passed from node j to node i:

$$m_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)\neq i} m_{k\to j}(x_j)$$

get_message() will be used by step_bp() to perform one iteration of loopy-BP: it first normalizes the returned message from get_message(), and then updates the message with momentum ‘1.0 - step‘.

1.4 Computing belief from messages

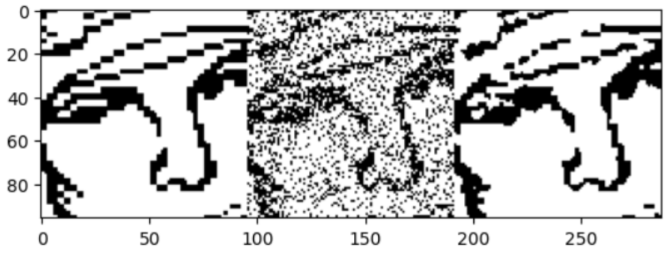
Now, calculate the unnormalized belief for each pixel

$$\tilde{b}(x_i) = \psi_i(x_i)\Pi_{j\in N(i)} m_{j\to i}(x_i),$$

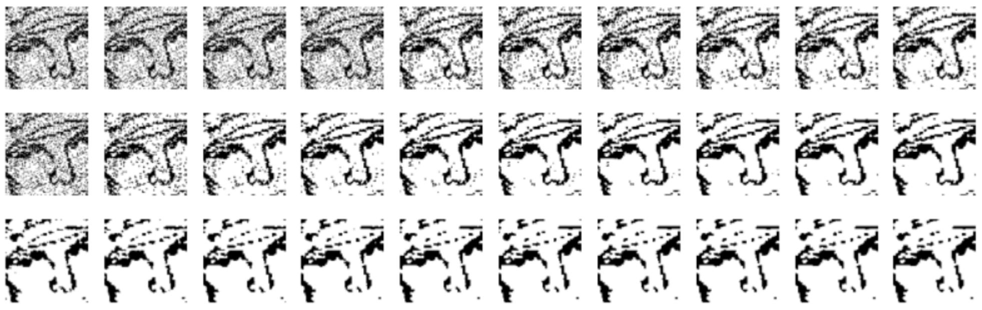
and normalize the belief across all pixels

$$b(x_i) = \frac{\tilde{b}(x_i)}{\sum_{x_j} \tilde{b}(x_j)}.$$

Finally, to get the denoised image, we use 0.5 as the threshold and consider pixel with belief less than threshold as black while others as white.



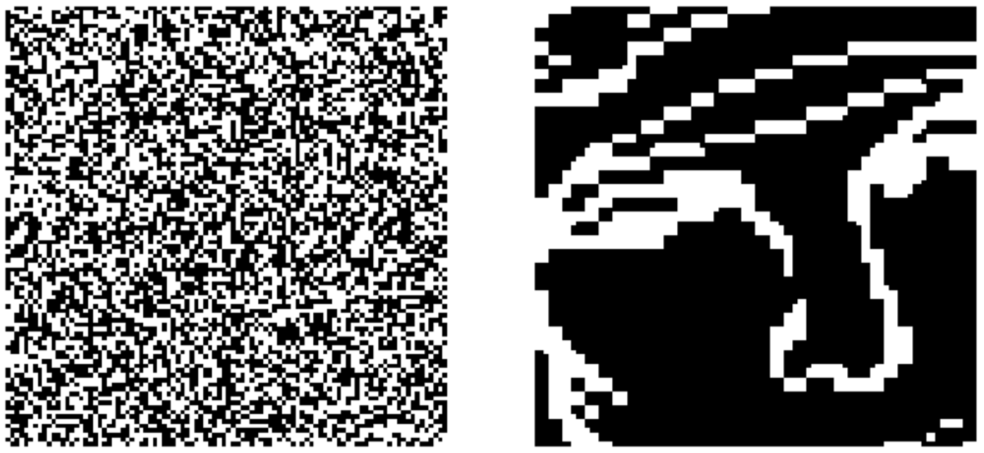
1.5 Momentum in belief propagation



Fine details in the image such as right eye are removed due to smoothing effect of message passing

1.6 Noise Level

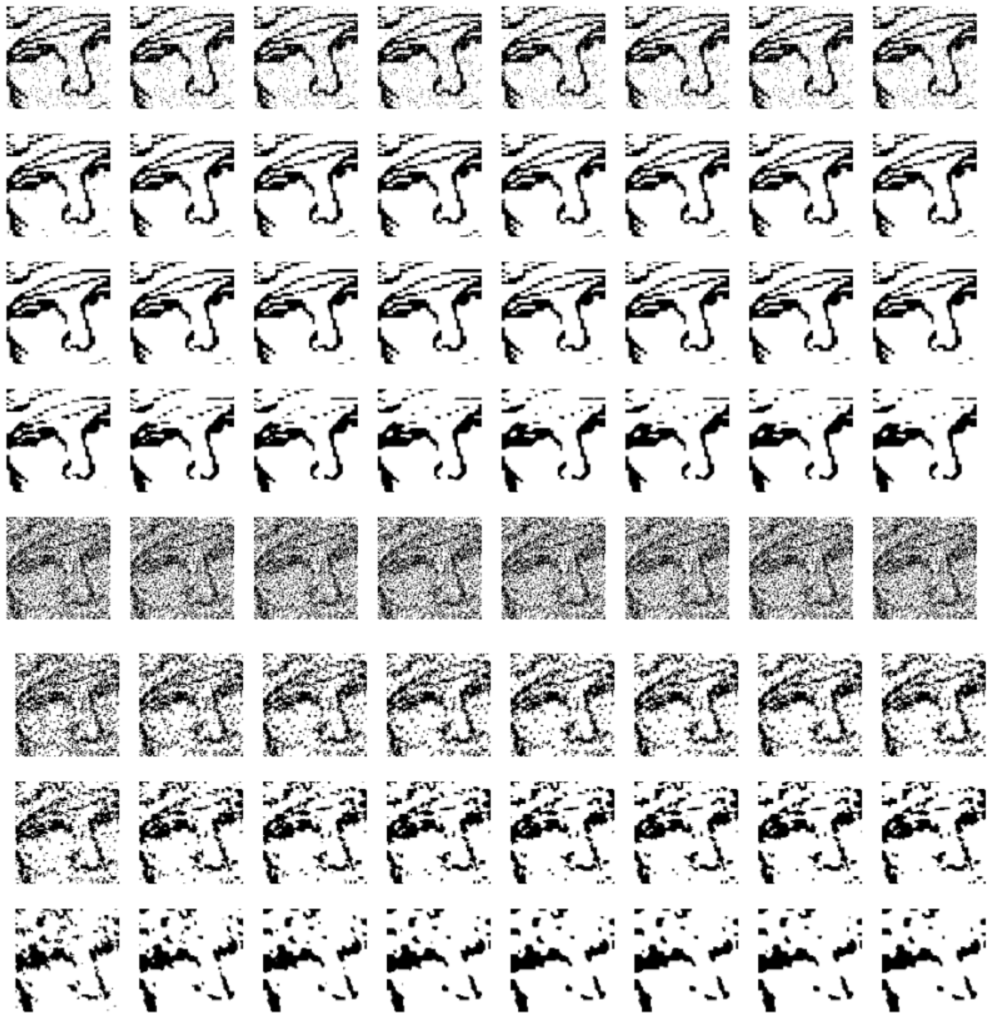
Now let's look at how the level of noise in the image influennces our choice in the hyperparamater J



With noise of 0.5, we should flip the pixels with a rate of 0.5. This would mean half the pixels would be flipped.

With noise of 1, we would flip all pixels. This would give us an inverted image, where black pixels are changed to white and white pixels are changed to black

Now let's observere this on a larger scale with increasing hyperparamters



The model was able to perform a lot better on the first 4 series compared to the following 4 series. This makes sense since the noise levels for the first 4 series was a lot lower, thus the chance of making mistakes was also lower.

As J is increased, we see the model gets better till J = 1.0. When J = 5.0, the model performs worse. This may be because we have over fitted the model to the noisy image, and more specifically. Since J is large, the potential between nodes is increased. The model also stabilizes very quickly. J