4 Part C: Graphical Models

C.1 Design and implementation of image denoising

The **noisy image** stored in an array with binary pixels as their data y_i with $y_i \in \{-1, 1\}$. Variable i is the index value of the state of the pixels i = [1, 2, 3 .. N], with N the number of pixels in the array. If the noisy image is to be consider corrupt then an assumption that there exist an **original** (uncorrupted) image denoted by x_i with $x_i \in \{-1, 1\}$.

If the number of pixels within both the **corrupted** and **uncorrupted** images only differed by 10 % then it can be assumed that there is a high correlation between the two sets of pixels x_i and y_i [1]. The **Markov random field** (MRF) has been used to illustrate undirected graphical models of the pixels in a 3D space. The correlation between the two set of pixels $\{x_i, y_i\}$ is calculated by using its **associated energy** [1]. Denoising is achieved by **minimizing** the energy function by comparing states of the pixels found in x_i and y_i. Each pixel is indexed by its axis variables i and j. These are used to identify neighbouring pixels. The energy function

$$E(\mathbf{x}, \mathbf{y}) = h \sum_{i=1}^{N} x_i - \beta \sum_{i \sim j} x_i x_j - \eta \sum N_{i=1} x_i y_i, \tag{2}$$

is accomplished by a simple method of subtracting marginal values according to the value obtained from their respective associated energies. It is shown in equation 2 by the three model parameters h, β and η. The negative values are used to give a lower energy to pixels which have the same sign [1]. To increase performance an additional model parameter **h** is used to bias the energy function towards pixels which hold the same state as those from the noise free image x_i. The joint distribution probability,

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \exp(-E(\mathbf{x}, \mathbf{y})) \tag{3}$$

is a conditional distribution $p(x|y)$. By substituting equation 2 into equation 3, the join probability is expanded to,

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \exp(-h \sum_{i=1}^{N} x_i - \beta \sum_{i \sim j} x_i x_j - \eta \sum N_{i=1} x_i y_i). \tag{4}$$

To find an image that has the highest probability of having the same pixels as the original image, the algorithm creates multiple different images derived from the initial noisy image. This method is iterated conditional model (ICM) which is coordinate-wise gradient ascent [1].

Algorithm 1 Data pre-processing and initialization

```
Require: Import noisy image "Y"
Require: Import uncorrupted image
1: Tone image for simple manipulation (Grey scale)
2: height, length = Y.shape                              ▷ Determine length and height
3: pixel_count = height × length                         ▷ Number of pixels
4: alpha = 0.05                                          ▷ h
5: beta = 0.5                                            ▷ $\beta$
6: eta = 0.1                                             ▷ $\eta$
7: Iterations = 9                                        ▷ Convergence stop criteria
8: algorithm = Processing(Y, alpha, beta, eta)           ▷ Begin algorithm
```

The raw data must be pre-processed to match the inputs of the ICM model which is shown in Algorithm 1. The length and height are extracted and used to determine the number of pixels. Initial values for the three hyperparameters and number of iterations are predetermined and initialized. The algorithm is then called passing the values from the noisy image and the three hyperparameters. The ICM model will run until convergence or some desired stopping criteria is met. The argument that gives the maximum value will be used for the final decision,

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} p(\mathbf{x}, \mathbf{y}) \tag{5}$$

The augmented maximum is used to determine x^* (predicted value of a pixel). The program will show the current state of the restored image after each and every iteration (all pixels). The stopping criteria will run the program until at least once to run through all the pixels and if there are not noticed changes then the algorithm has converged. This does not imply that the final outcome is the optimum solution but potentially a local maximum. In order to make an accurate assumption the predicted value needs to be rounded up or down to the nearest integer value before determining the performance of the model.

The error loss $L(x_{GT}, y)$ will be used to determine the overall performance of the algorithm. The aim is to make the loss function as small as possible by choosing the best parameters and lowest energy functions. The loss function used in this implementation is a fraction of incorrectly recovered pixels,

$$L(\mathbf{u}, \mathbf{v}) = \frac{1}{N} \sum_{i=1}^{N} [u_i - v_i \neq 0] \, . \tag{6}$$

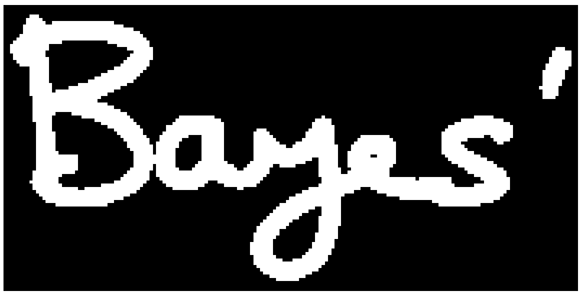
Equation 6 shows the loss function which is calculated by comparing the pixels from the original image to the denoised image for each and every corresponding index. An average will be taken across all iterations. P will be a Boolean data type which is indicated by a 1 if true and 0 otherwise.

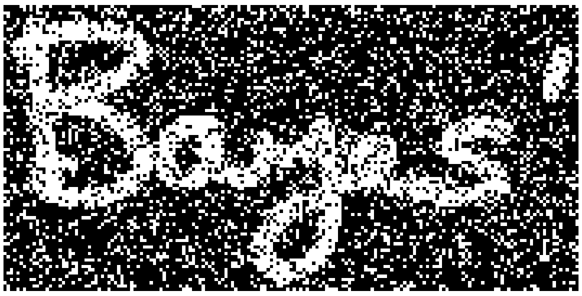
The ICM model was initialized and called in Algorithm 1. The pseudo code shown by Algorithm 2 is the approach used to correct the corrupted pixels in the noisy image and compute the performance metrics along with a denoised image.

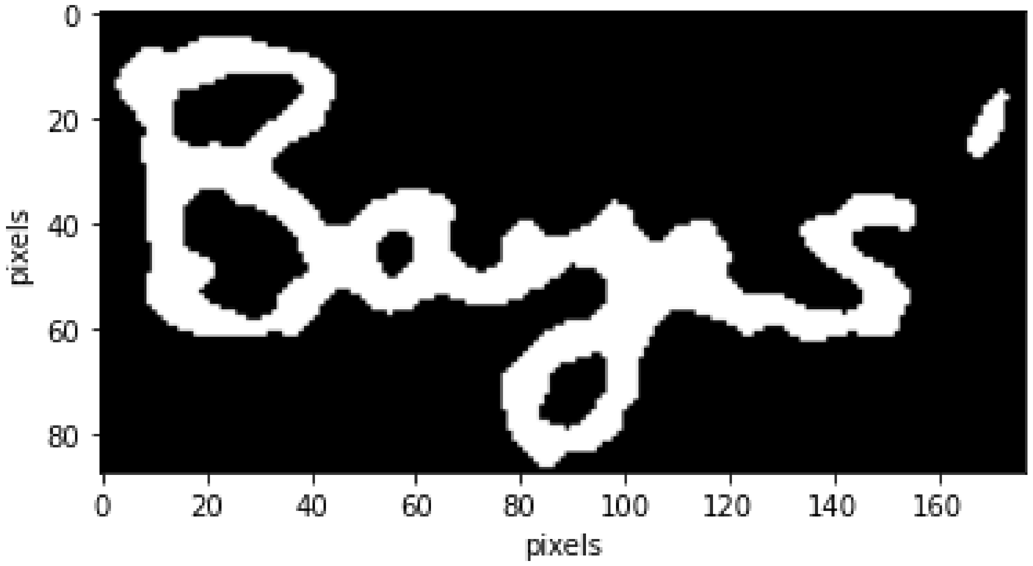
Algorithm 2 Algorithm implementation

```
1: while Stop Criteria not met do                                    ▷ has not yet converged
2:     for i in range(0, height) do                                  ▷ j
3:         for i in range(0, length) do                              ▷ i
4:             algorithm.selection([i,j])                            ▷ Iterate through each pixel
5:             prediction += algorithm.X
6: def selection(self, in_var):
7:     index_x, index_y = in_var
8:     prob_1, prob_2 = self.probability(index_x, index_y)
9:     self.X[i, j]= 0 if 0.5 $\leq$ prob_1 else 1

10: def probability(self, index_x, index_y):                         ▷ Conditional probability
11:     alpha = self.alpha
12:     beta = self.beta $\times$ np.sum( self.X[x_val,y_val] for (x_val,y_val) in neighbours)
13:     eta = self.eta $\times$ self.X[index_x,index_y]
14:     neighbours = self.neighbours(index_x,index_y)
15:     energy = alpha -beta -eta                                    ▷ Energy function
16:     ComponentA = float(np.exp(-energy))/(np.exp(-energy)+np.exp(energy))
17:     return [1.0 - ComponentA, ComponentA]

18: prediction = np.argmax(prediction)                               ▷ Bayes estimator
19: prediction = prediction.astype(np.int)          ▷ Rounding up or down to nearest integer
20: loss_function = 1/len(X)*np.sum(im2 - prediction)/pixel_count*Size
```

The first 5 lines of code are used to iterate through the x and y co-ordinates of the noisy image and passing their values into the selection function where the conditional probability will be calculated based on the energy function. The pixel's are evaluated against their neighbours (pixel for pixel) where the energy function will be the lowest over the two outcomes on the basis of $x_j = 1$ and $x_j = -1$ and by making x_j equal to the state that minimizes the energy function. This is seen in the algorithm shown above from lines 10 to 17. The idea is to increase the probability by lowering the energy function to its minimum. No changes will occur if there is no sign change for neighbouring pixels. Line 18 is the argument that gives the maximum value that will be used for the final decision which is then converted to an integer value shown in line 19. The final step is to calculate the loss function and determine the performance of the ICM model and selected hyperparameters.


(a) Original uncorrupted image


(b) Observed noisy image



(c) Denoised image

Figure 8. Predicted result compared to original and noisy image

To make a direct comparison between the original, noisy and denoised images a sub figure with all three images is shown above to allow for easy visual inspection. Figure 8a is the uncorrupted image which we wish to retrieve as closely as possible. This was done by optimizing the 3 hyperparameters, h, β and η to ideally minimize the loss function seen in equation 6 which is the fraction of incorrectly recovered pixels. Figure 8b is the observed noisy image $\mathbf{y}$. Through inspection, it can be seen that many of the pixels were flipped, black pixels became white and white pixels became black. The outcome of using **Iterated Conditional Modes** until **convergence** is shown in Figure 8c which was the best recovered image.

The recover image is highly similar to that of the original uncorrupted image with a **loss function = 0.01144461759350049**. It is visible that all pixels which were originally black have been restored except the few bordering the original white pixels. The algorithm struggled to restore the gap in the alphabetical letter 'e' which is due to the space separating the black and white pixels being tiny (up to 4 pixels in wide). The energy function is directly derived from the three hyperparameters which were set to h = 0.05, β = 0.5 and η = 0.1. The absolute value of the energy function is, **energy = 75.549999999999997**.

The goal in mind was always to lower the loss function which is the average of the difference between the original uncorrupted pixels and denoised pixels. This was tuned by adjusting the values of all three hyperparameters with some interesting observations noted. There were many variations of these parameters which worked reasonably well. Setting η much less than β ($\geq \times 10^2$) would results in a distorted image. Additionally if the algorithm was allowed to continue past convergence, the results wold have been over corrected and result in a loss of uncorrupted pixels.

By increasing the number of neighbour pixels, the algorithm would detect rounded protruding lines more accurately similar to the one found in the letter 'B' in Figure 8. This is due to the algorithm having a wider search range and thus increasing its understanding of what is to be expected from neighbouring pixels. By maximizing the hyperparameters, h = 0.05, $\beta = 10000$ and $\eta = 2000$. The absolute value of the energy function would be approximately, **energy = 72K** with a **loss function = 0.011582504552458333**. This shows the wide range of possible values for the hyperparameters although the former set of results were better.

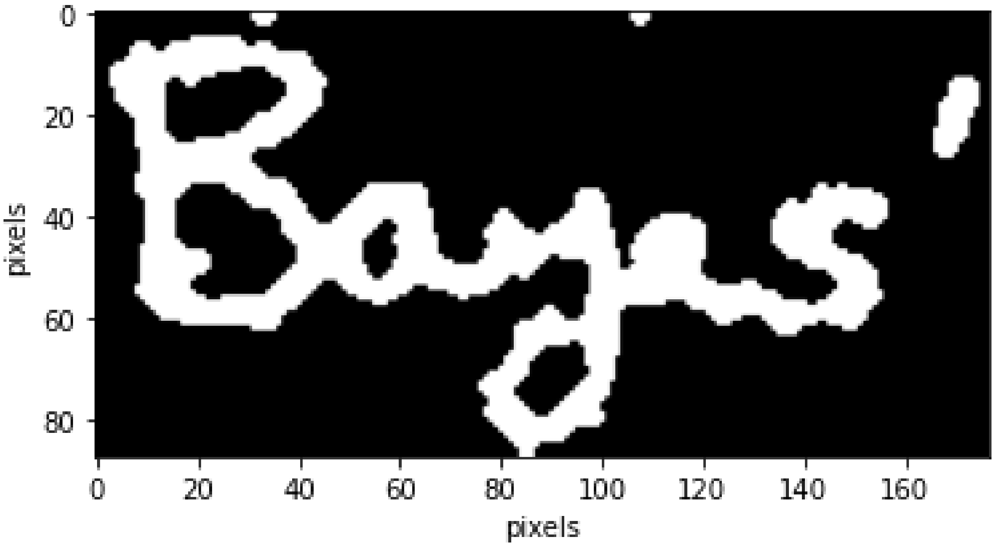
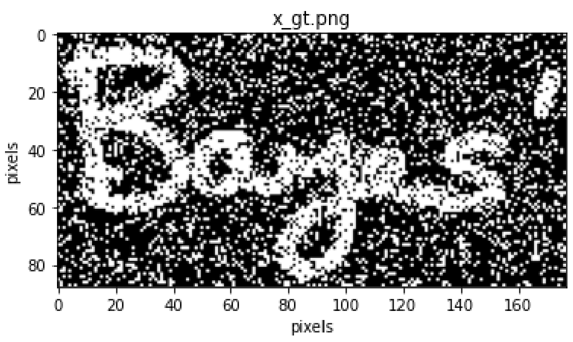
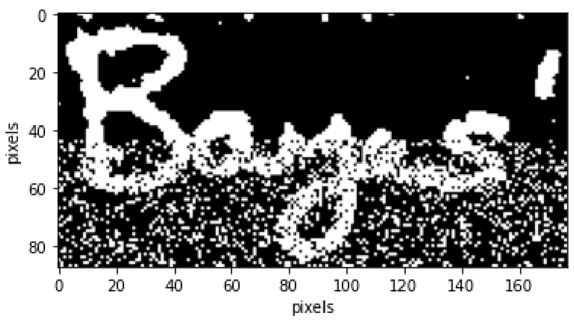


Figure 9. Denoised image using alternative parameters

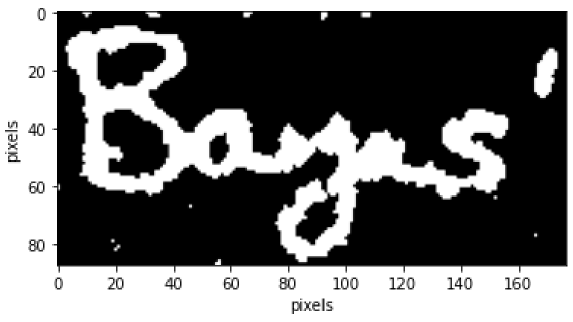
The denoised image for the alternative set of parameters is shown in Figure 9 where the pixels around the body of the text seems to be more jaggered. Although the results are similar to that of Figure 8c. The current state of the ICM after every iteration is shown in Figure 10. With the first Figure 10a iteration zero (noisy image) and the last Figure 10j the denoised image. Through each iteration, the image is cleared of corrupted pixels with a finer correction made after the third iteration Figure 10d.



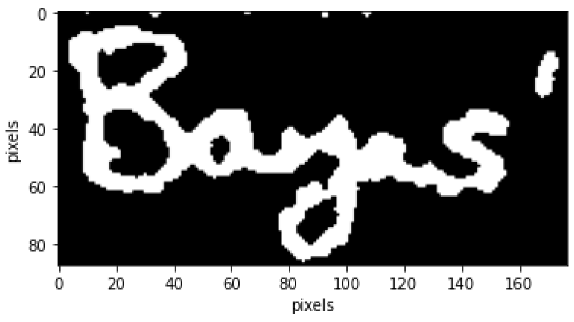
(a) Initial noisy image



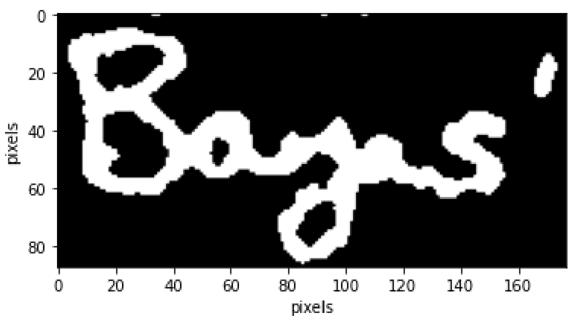
(b) First iteration



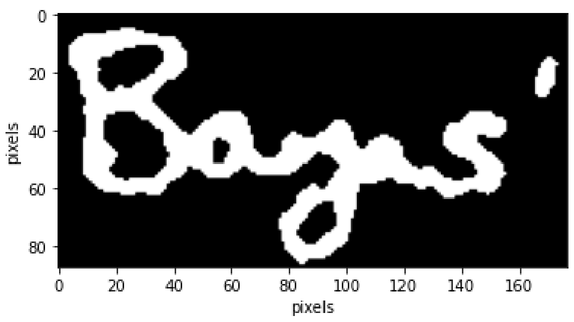
(c) Second iteration



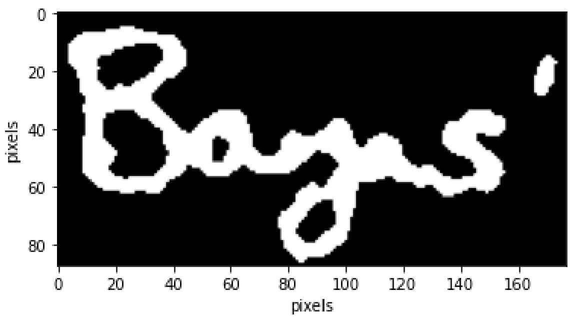
(d) Third iteration



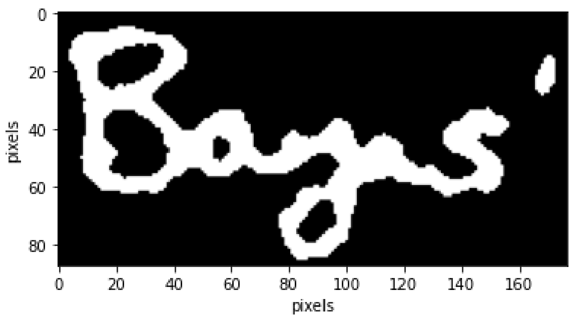
(e) Fourth iteration



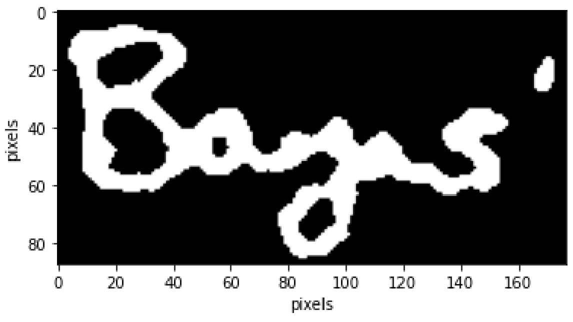
(f) Fifth iteration



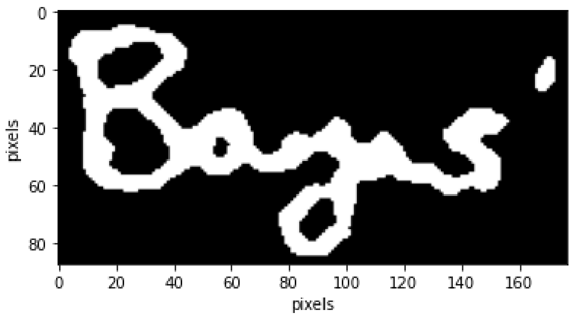
(g) Sixth iteration



(h) Seventh iteration



(i) Eighth iteration



(j) Denoised image

Figure 10. current state of the restored image after ICM iteration

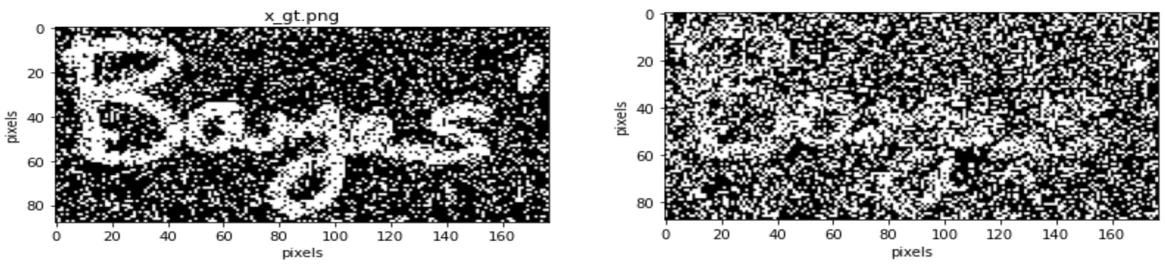
C2 Graphical models

C.2.a Explain the role of the three factor types. For each of parameters, explain what happens when they are set to 0.

The three factor types, $hx_i, -\beta x_i x_j$ and $-\eta x_i y_i$ which influence the performance of the algorithm are variable based and can be adjusted to increase the probability of correcting a corrupted pixel. An additional term is added for the noise free image $\mathbf{h}x_i$ which is the summation of all pixel values index by i multiplied by x_i. This term is used to create a bias between the noise free image and noisy image. The bias has a particular preference of sign taken from the noise free image. By making $\mathbf{h = 0}$, the prior probability of the two possible states will be equally likely [1].

η derived from $\eta x_i y_i$ is a positive constant, which is subtracted in the energy function to ideally lower the energy which increases the probability of correcting a corrupted pixel. By lowering the energy when the states of x_i and y_i have the same sign (lower energy is better) and increase the energy when their signs are opposite. These help express the correlation between the associated energy as it is rewarding to have the same sign.

Similar to that of η, β is also a **positive constant** which is **subtracted** in the energy function. We know that the **energy level is small** between **neighbouring pixels** x_i and x_j. This allowed us to assume that neighbouring pixels do have a **strong correlation**. $\beta x_i x_j$ acts as the primary influencer for the correlation between neighbouring pixels in the energy function. By setting $\beta = \mathbf{0}$, we are severing the connection between neighbouring pixels. The disconnection between neighbouring pixels results in a global optimal solution of $y_i = x_i$ for all values index by i found in the noisy image [1]. Through experimentation is was realized by setting β to zero the denoised image worsened (no adjustments were made to either h or η) with the result shown in Figure 11.



(a) Noisy image **(b)** Setting β to zero

Figure 11. Effect of setting a hyperparameter to zero