SIMC$^{2.0}$ Report 2024

Dunman High School
Chee Justin Suwattana, Ethan Wang Jun Qi, Ryan Joo Rui An

Task 1

(a)

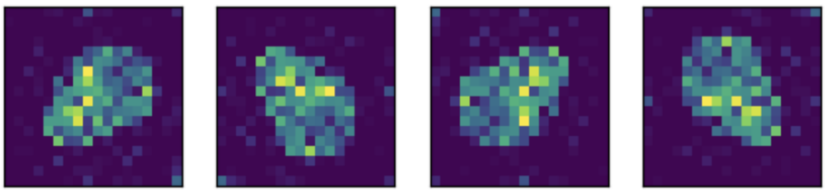

Figure 1: Reference pattern in its four clockwise rotations: 0°, 90°, 180° and 270°.

(b) There are 6, 10, 6 and 3 patterns that are rotated 0°, 90°, 180° and 270° with respect to the reference pattern (first pattern) respectively.

(c) **Part (a):**

We wrote a function `rotate()`, which takes a 2D NumPy array – in this case, the reference pattern (first pattern) – as input and returns a NumPy array containing all four clockwise rotations of the input pattern. The function works by iterating over the range `(0,4)` to rotate the input array by four angles, namely 0, 90, 180, and 270 degrees. The rotated arrays are then appended to the `rotations` list. After all rotations are completed, the `rotations` list is converted to a NumPy array and returned.

We used this function to find all four rotations of the reference pattern. The `plt.subplots()` function was used to create a figure and a set of subplots with a grid of 1×4, and the `imshow()` function was used to display each rotation in the corresponding subplot.

Part (b):

We first initialised a list `count` with 4 zeros to store the count of patterns in each orientation. Then the `rotate()` function was called to generate the rotated versions of the reference pattern. We iterated over the first 25 patterns in `task1`, checking if the current pattern is equal to any of the rotated versions of the reference pattern using the `np.array_equal()` function. If a match was found, the count of patterns in the corresponding orientation was incremented by one. Finally, we printed the count of patterns in each orientation.

To verify our answer, we displayed all 25 patterns of `task1` using the `imshow()` function, and subsequently counted the number of patterns for each orientation by visual inspection.

Task 2

(a)

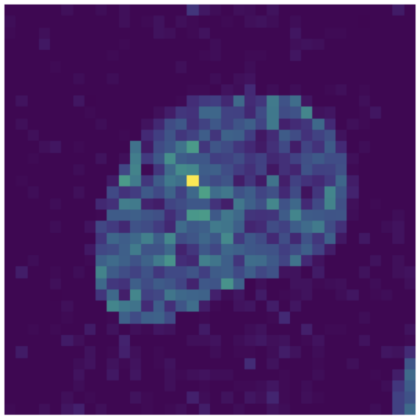

Figure 2: 2D master image for `task2`.

(b) There are 22, 32, 21 and 25 patterns that are rotated $0°$, $90°$, $180°$ and $270°$ with respect to the reference pattern respectively.

(c) **Part (a):**

We reshaped the flattened NumPy array to a 36×36 array, then used the `imshow()` function to display the master image as shown in Figure 2.

Part (b):

We first initialised a `count` list with 4 zeros to store the count of patterns in each orientation. The `rotate()` function was called to generate the four rotated versions of the master image.

We then iterated through all the patterns and reshaped them to 36×36 arrays. Following this, the `np.array_equal()` function was used to check if the current pattern is equal to any of the rotated versions of the master image. If a match is found, the count of patterns in the corresponding orientation is incremented by one.

Task 3

(a)

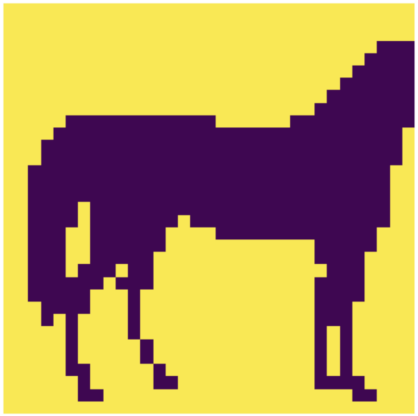

Figure 3: 2D master image for `task3`.

(b) There are 254, 236, 250 and 260 patterns that are rotated 0°, 90°, 180° and 270° with respect to the reference pattern respectively.

(c)

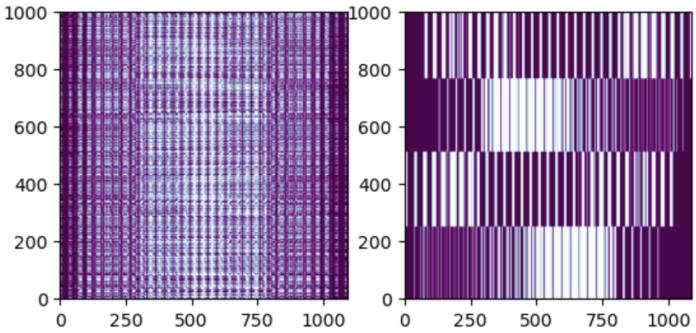

Figure 4: **LEFT**: Unsorted version of the design matrix for `task3`. **RIGHT**: Sorted version of the design matrix.

(d) **Part (a):**

Assuming the image follows previous patterns of being a square image, the length and width of the image can be calculated by $\sqrt{1089} = 33$. Hence the 2D master image is a 33×33 pixel image.

We thus reshaped the NumPy array to a 33×33 array to produce the master image shown in Figure 3. By visual inspection, the mystery animal is a horse.

Part (b):

We used `rotate()` to find all four rotations of the master image and iterated through all patterns, while reshaping them to 33×33 arrays, in order to count the number of patterns in each orientation.

Part (c):

The `plt.subplots()` function was used to create a figure and a set of subplots with a grid of 1×2. The original unsorted version of the design matrix was displayed in the first subplot using the `imshow()` function.

The `np.lexsort()` function was used to sort the design matrix based on the columns in reverse order. The sorted version of the design matrix was displayed using the `imshow()` function.

Both the unsorted and sorted design matrices are displayed in Figure 4. The sorted design matrix is clearly segmented into four distinct groups, thus showing that the patterns are correctly classified.

Task 4

(a) The average 2D pattern's total pixel value is $\boxed{1251.0}$.

(b)

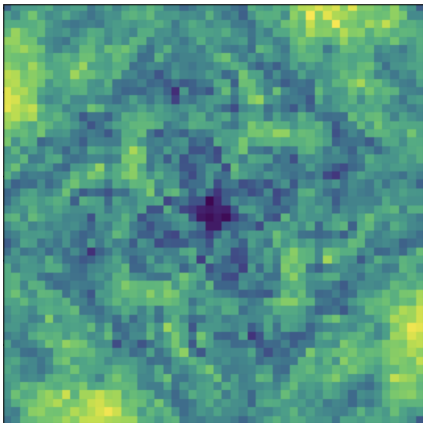

Figure 5: Average 2D pattern for `task4`, assuming they were all the same orientation as the first pattern.

(c) By visual inspection, the master image displays a cat.

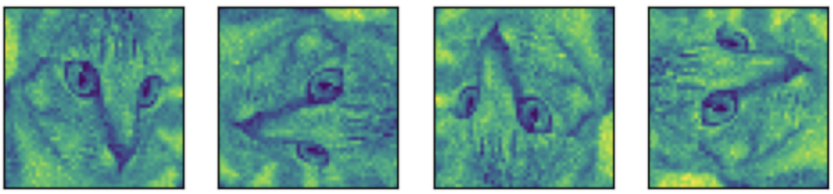

Figure 6: Final 2D orientation class average patterns for each of the four orientation classes.

(d) **Part (a):**

To calculate the average 2D pattern's total pixel value, we summed up the pixel values of all patterns and divided it by the number of patterns, i.e. 1000.

Part (b):

At each pixel, we took the average of pixel values across all 1000 patterns to produce the average 2D pattern, as shown in Figure 5.

Part (c):

We performed a KMeans clustering algorithm to group the datasets into four orientation classes. The algorithm provided us with the labels for each pattern and we grouped all the patterns of the same label together.

Then, we took the average of all the patterns in each group to produce an image for every orientation class, as shown in Figure 6.

In our initial attempt, our results were rather inaccurate as the images produced were blurry. We increased the parameter `n_init` from 10 to 100 to increase the number of iterations the KMeans algorithm ran, which increased the accuracy of our results as seen in Figure 6.

Task 5

(a) The likelihood is given by

$$\begin{aligned}\mathcal{L}(r = 90^\circ \mid K_{(4)}, \mu_{(4)}) &= \Pr\big(k_1 = 1 \mid \beta\lambda\big)\Pr\big(k_2 = 0 \mid \lambda\big)\Pr\big(k_3 = 0 \mid \beta\lambda\big)\Pr\big(k_4 = 0 \mid \beta\lambda\big)\\ &= \beta\lambda(1-\lambda)(1-\beta\lambda)(1-\beta\lambda)\\ &= \boxed{\beta\lambda(1-\lambda)(1-\beta\lambda)^2}.\end{aligned}$$

(b) The ratio is given by $\dfrac{\mathcal{L}(r = 0^\circ \mid K_{(4)}, \mu_{(4)})}{\mathcal{L}(r = 90^\circ \mid K_{(4)}, \mu_{(4)})} = \dfrac{\lambda(1-\beta\lambda)^3}{\beta\lambda(1-\lambda)(1-\beta\lambda)^2} = \boxed{\dfrac{1-\beta\lambda}{\beta-\beta\lambda}}$.

(c) Equating the expression in Part (b) to 1 gives $\dfrac{1-\beta\lambda}{\beta-\beta\lambda} = 1$, on solving which yields $\boxed{\beta = 1}$.

In the given context, this means that as the average values of non-outlier pixels become more similar to that of outlier pixels, outlier pixels become less identifiable, thus the aligned orientation becomes increasingly similar to misaligned orientations, thereby lowering the likelihood of determining the orientation of a pattern.

(d)

$$\begin{aligned}\mathcal{L}(\text{aligned}) &= \mathcal{L}(r = 0^\circ \mid K_{(16)}, \mu_{(16)})\\ &= \Pr\big(k_1 = 1 \mid \lambda\big)\Pr\big(k_2 = 0 \mid \beta\lambda\big)\cdots\Pr\big(k_{16} = 0 \mid \beta\lambda\big)\\ &= \lambda(1-\beta\lambda)^{15}\end{aligned}$$

and

$$\begin{aligned}\mathcal{L}(\text{misaligned}) &= \mathcal{L}(r = 90^\circ \mid K_{(16)}, \mu_{(16)})\\ &= \Pr\big(k_1 = 1 \mid \beta\lambda\big)\Pr\big(k_2 = 0 \mid \beta\lambda\big)\Pr\big(k_3 = 0 \mid \beta\lambda\big)\Pr\big(k_4 = 0 \mid \lambda\big)\cdots\Pr\big(k_{16} = 0 \mid \beta\lambda\big)\\ &= \beta\lambda(1-\lambda)(1-\beta\lambda)^{14}.\end{aligned}$$

Therefore $\dfrac{\mathcal{L}(\text{aligned})}{\mathcal{L}(\text{misaligned})} = \boxed{\dfrac{1-\beta\lambda}{\beta-\beta\lambda}}$.

(e) Since it is given that λ_i can only take on two values, namely λ and $\beta\lambda$, we are essentially dealing with the following four cases:

- Case A: $k_i = 1$ given $\mu_i = \lambda$,
- Case B: $k_i = 0$ given $\mu_i = \lambda$,
- Case C: $k_i = 1$ given $\mu_i = \beta\lambda$,
- Case D: $k_i = 0$ given $\mu_i = \beta\lambda$.

Let $J_n \coloneqq \{i \in \mathbb{Z}^+ : i \le n\}$. Let

$$\begin{aligned}A &\coloneqq \{i \in J_n : k_i = 1 \mid \lambda\},\\ B &\coloneqq \{i \in J_n : k_i = 0 \mid \lambda\},\\ C &\coloneqq \{i \in J_n : k_i = 1 \mid \beta\lambda\},\\ D &\coloneqq \{i \in J_n : k_i = 0 \mid \beta\lambda\}.\end{aligned}$$

In the case where $r = 0°$, $A = \{1, 3, 13\}, B = \varnothing, C = \varnothing, D = (A \cup B \cup C)^c = A^c$, where for a set X, X^c denotes the complement of X in J_n. Thus $|A| = 3$, $|B| = 0$, $|C| = 0$, $|D| = 13$, where for a set X, $|X|$ denotes the cardinality of X. Hence

$$\begin{aligned}\mathcal{L}(\text{aligned}) &= \mathcal{L}(r = 0° \mid K_{(16)}, \mu_{(16)}) \\ &= \prod_{s\in A} \Pr\big(k_s = 1 \mid \lambda\big) \times \prod_{s\in D} \Pr\big(k_s = 0 \mid \beta\lambda\big) \\ &= \Big[\Pr\big(k_s = 1 \mid \lambda\big)\Big]^{|A|} \Big[\Pr\big(k_s = 0 \mid \beta\lambda\big)\Big]^{|D|} \\ &= \lambda^3(1 - \beta\lambda)^{13}.\end{aligned}$$

In the case where $r = 90°$, $A = \varnothing, B = \{4, 6, 14\}, C = \{1, 3, 13\}, D = (A \cup B \cup C)^c$, thus $|A| = 0$, $|B| = 3$, $|C| = 3$, $|D| = 10$. Hence

$$\begin{aligned}\mathcal{L}(\text{misaligned}) &= \mathcal{L}(r = 90° \mid K_{(16)}, \mu_{(16)}) \\ &= \prod_{s\in B} \Pr\big(k_s = 0 \mid \lambda\big) \times \prod_{s\in C} \Pr\big(k_s = 1 \mid \beta\lambda\big) \times \prod_{s\in D} \Pr\big(k_s = 0 \mid \beta\lambda\big) \\ &= \Big[\Pr\big(k_s = 0 \mid \lambda\big)\Big]^{|B|} \Big[\Pr\big(k_s = 1 \mid \beta\lambda\big)\Big]^{|C|} \Big[\Pr\big(k_s = 0 \mid \beta\lambda\big)\Big]^{|D|} \\ &= (1 - \lambda)^3(\beta\lambda)^3(1 - \beta\lambda)^{10}.\end{aligned}$$

Therefore $\dfrac{\mathcal{L}(\text{aligned})}{\mathcal{L}(\text{misaligned})} = \boxed{\left(\dfrac{1 - \beta\lambda}{\beta - \beta\lambda}\right)^3}$.

(f) Using the notation above, M pixels belong to Case A, and $N - M$ pixels belong to Case D.

Hence the log-likelihood is given by

$$\begin{aligned}\ln\Big(\mathcal{L}(\text{aligned} \mid \vec{\mu}, \vec{k})\Big) &= \sum_{i=1}^{N} \big[k_i \ln(\mu_i) + (1 - k_i)\ln(1 - \mu_i)\big] \\ &= \underbrace{\sum_{i=1}^{M} \ln\lambda}_{M \text{ pixels belonging to Case A}} + \underbrace{\sum_{i=M+1}^{N} \ln(1 - \beta\lambda)}_{N-M \text{ pixels belonging to Case D}} \\ &= M\ln\lambda + (N - M)\ln(1 - \beta\lambda).\end{aligned}$$

Therefore the average log-likelihood is given by $\boxed{\dfrac{M\ln\lambda + (N - M)\ln(1 - \beta\lambda)}{N}}$.

Task 6

(a) The average pattern's total pixel value is $\boxed{23.745}$.

(b)

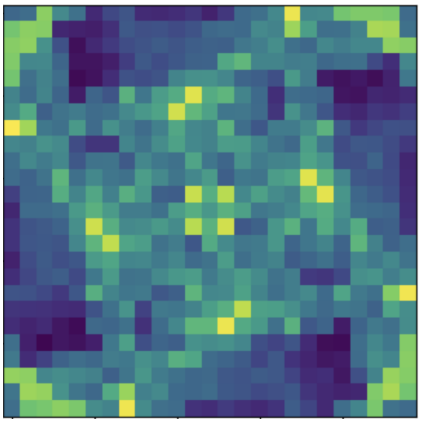

Figure 7: Average 2D pattern for `task6`, assuming they were all the same orientation as the first pattern.

(c) By visual inspection of Figure 8, the master image is a person's face.

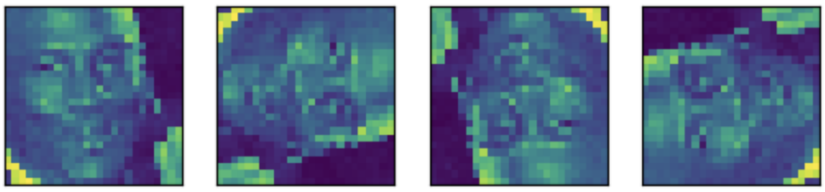

Figure 8: 2D orientation class average patterns for each of the four orientation classes.

(d) **Part (a):**

To find the pattern's average total pixel value, we divided the length of vector `a` (which indicates the total number of 1s in all the patterns) by the number of patterns, i.e. 65535.

Part (b):

We first observed that vector `a` stored the pattern index, while vector `b` stored the index of the pixel in the pattern. Using this, we mapped the values in vectors `a` and `b` to their respective patterns and positions to produce a 65535×625 design matrix, which contained all the patterns.

Then we took the average of every pixel to produce the average pattern as shown in Figure 7.

Part (c):

We used a KMeans clustering algorithm to classify the patterns based on their orientation class.

We then sorted the patterns into their respective orientation classes and took the average of pixels of each pattern in the orientiation class, producing an image for each group, as shown in Figure 8. To ensure a high accuracy, we set `n_init` as 100.

Task 7

(a)

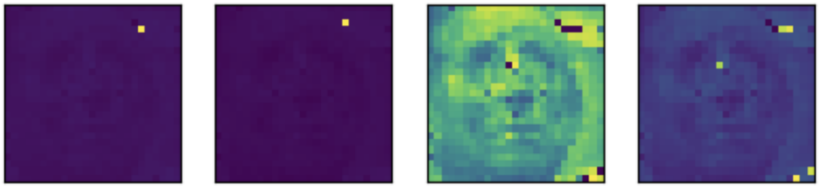

Figure 9: Reconstruction of the 2D master image for `task7`. The third image from the left shows the master image most clearly.

(b) Similar to Task 6, we first used the vectors provided to create the design matrix of 100000 flattened 25×25 images. Then we used a KMeans algorithm to cluster the datasets into 4 groups. However, this approach did not yield any substantial results as there was too much noise in the images and the clustering algorithm was unable to accurately cluster the images by their orientation class. The results from this can be seen in Figure 10.

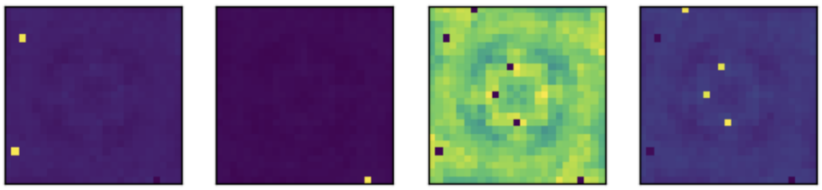

Figure 10: First attempt of reconstructing the 2D master image.

As such, we decided to make our dataset 4 times as large by adding all 4 rotations of each image. This meant that for each orientation there were 4 times as many datasets which we believed would be useful in reducing the noise if we could average it out across more datasets. Upon experimentation, we discovered that when setting `n_init` as 100 as before, the results were not much different than before as there was always one cluster which seemed to be all the images combined while the rest were just noise that the KMeans algorithm filtered out. However, upon setting `n_init` as 10, the algorithm managed to clearly split the data into two groups, both appearing to be rotations of the master image, as shown in Figure 11.

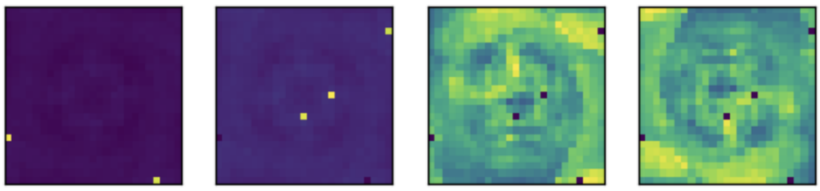

Figure 11: Second attempt of reconstructing the 2D master image.

As such, we rotated all the images in one of the groups and concatenated them to the other group to create one group of images that are mostly in the same orientation, making it easier to denoise. Then we performed another round of KMeans clustering on this dataset, yielding the image shown in Figure 9 where the master image is seen in the third image from the left, while the rest of the images seem to be noise filtered out from the dataset. From visual inspection, the master image resembles the face of a person whose identity cannot be immediately determined.

Remark. When running our source code, the results might differ due to the probabilistic aspect of the KMeans algorithm.