SIMC$^{2.0}$ 2024: On Classification of Noiseless and Noisy Pattern Orientations

Philippine Science High School – Main Campus (**SIMC Endeavor Team 04**)
Hans Gabriel D. De Vera, Benjamin L. Jacob, Davis Nicholo A. Magpantay

This paper explores a multi-part lower-dimensional toy model inspired by the computational methods behind single-particle imaging (SPI). This is a simple case in 2 dimensions, where only rotations with integer multiples of 90˚ are considered. There are two main sections to this paper: on noiseless patterns and on noisy patterns, which utilize different methods for orientation classification. We also summarize possible improvements in this work. All relevant processes and further proofs of implementation are presented in-depth in the attached code.

Section 1: On Classifying Noiseless Patterns

1.1 Task 1: Classifying a Few Noiseless Patterns

This first section of the paper revolves around the classification of noiseless patterns. The magnitude of the number of operations increases throughout the three tasks in this section, which merits different levels of implementation with faster time complexities. We show that a simple, deterministic algorithm involving hashing the patterns suffices in this idealistic case (with order of magnitudes 10^3, 10^4, 10^6). Throughout this paper, except for Task 5, we denote the number of patterns as A and we denote the number of pixels in the image as B. Task 1 deals with 25 patterns composed of 25×25-pixel images.

1.1.1 Task 1(a)

Throughout this paper, we define all rotations clockwise from a fixed axis (e.g., the $+y$-axis), and with only four possible rotations of the reference image (0˚, 90˚, 180˚, 270˚). The rotations of the reference pattern are shown in Figure 1. We note that `numpy` rotates matrices anti-clockwise by default.

1.1.2 Task 1(b)

Using the labelling of the orientations in Figure 1, there are **(a)** $\boxed{6 \text{ patterns}}$ of rotation 0˚, **(b)** $\boxed{10 \text{ patterns}}$ of rotation 90˚, **(c)** $\boxed{6 \text{ patterns}}$ of rotation 180˚, and finally, there are **(d)** $\boxed{3 \text{ patterns}}$ of rotation 270˚. The sum of all of these counts is 25 images, which is the number of elements in the dataset. The complete classification of the images can be found in the source code.

1.1.3 Task 1(c)

Since our dataset is small ($A = 25$ patterns and $B = 625$ pixels) and idealized (i.e., noiseless), we decided to implement a simple algorithm that hashes each master image rotation using a hash function with $p = 10^9 + 7$ possible hashes, at a time complexity of $\mathcal{O}(B)$. This produces a "reference hash" via Eq. 1.1, which serves as a unique "identification number" for a certain orientation.

$$H = \sum_{i=1}^{B} a_i \cdot 2^i \pmod{p} \tag{Eq. 1.1}$$

Because most master images are assumed to not possess rotational symmetry, the hash of each rotated version is unique[1]. In our implementation, each of the $A = 25$ sample patterns was hashed via the

[1] As long as less than $p = 10^9 + 7$ images are produced, the hashes are unique since there are p residues (modulo p). There are $1000 \ll p$ images, so the probability of a hash collision is extremely low.

same method and then classified to the matching reference hash. This was then used to count the number of images that fell under each rotation. Therefore, this implementation has a time complexity of $\mathcal{O}(AB)$. The magnitude of the number of operations is $10 \cdot 10^2 \sim 10^3$. While a brute force approach that performs pairwise comparisons between each pattern to each of the master image's rotations in Figure 1 would work, such a method would be more computationally inefficient. In fact, our hashing method proves fast enough for future tasks.

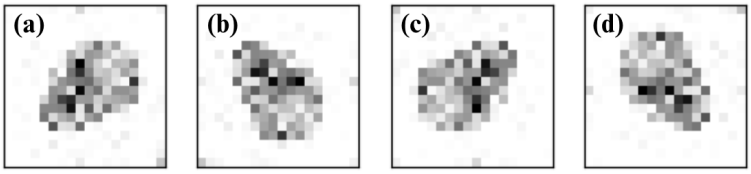

Figure 1: Rotations at **(a)** 0˚, **(b)** 90˚, **(c)** 180˚, **(d)** 270˚ of the Reference Image for Task 1(a) (§1.1)

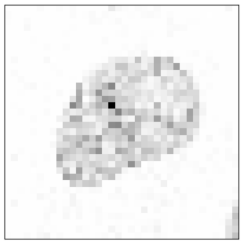

Figure 2: Reconstruction of Image for Task 2(a) (§1.2)

1.2 Task 2: Flattening 2D Patterns Into 1D Representations

The `task2` dataset contains $A = 100$ images with dimensions $B = 32 \times 32$. However, unlike the `task1` dataset, all images were flattened to one-dimensional vectors.

1.2.1 Task 2(a)

We reconstructed the 2D master image from its flattened version by reshaping the flattened version of dimensions 1024×1 into a square array of dimensions 32×32 using the `numpy` python library, since the number of elements of each vector is the same. The result of this transformation is shown in Figure 2.

1.2.2 Task 2(b)

From the master image in Figure 2 to be the reference image, marked as rotation 0˚. Our algorithm then showed that there are $\boxed{22 \text{ patterns}}$ of rotation 0˚, $\boxed{32 \text{ patterns}}$ of rotation 90˚, $\boxed{21 \text{ patterns}}$ of rotation 180˚, and $\boxed{25 \text{ patterns}}$ of rotation 270˚. This adds up to a total of 100 patterns, as expected from the given dataset.

1.2.3 Task 2(c)

This task involves a larger dataset and introduces the concept of transforming 2D image data into 1D design arrays via "flattening," as described in the earlier subtasks. However, the size of the dataset is now in the order of magnitude $10^2 \cdot 10^2 \sim 10^4$. Despite this, we used the same prime hashing algorithm described in Task 1c (§1.1.3). The time complexity of our algorithm remains resilient to this larger dataset.

1.3 Task 3: Scaling Thousands of Patterns

The `task3` dataset is similar to the `task2` dataset. It contains $A = 1000$ flattened images with $B = 1089$ pixels. However, the dimensions of the images were not given in advance.

1.3.1 Task 3(a)

Inspecting the Task 3 dataset revealed that the flattened image is of length $B = 1089 = 33^2$. To reconstruct the master image, we had to guess the number of rows r, which can only be among [1, 3, 9, 11, 33, 99, 121, 363, 1089] as r should divide B. By iteratively reshaping the flattened array,

we found the best value to be $r = 33$ (as this also produces a square image), which as seen in Figure 3, clearly shows the silhouette of a horse!

1.3.2 Task 3(b)

Using the same algorithm from Task 1(b) (§1.1) and Task 2(b) (§1.2), with master image Figure 3 defined as the 0˚ rotation image, we found that there are 254 patterns of rotation 0˚, 236 patterns of rotation 90˚, 250 patterns of rotation 180˚, and 260 patterns of rotation 270˚. This adds up to a total of 1000 patterns, as expected from the given dataset. If this were to be implemented using k-means clustering, the clusters are approximately equal, which is highly optimal.

1.3.3 Task 3(c)

Figure 4 shows that our algorithm has indeed classified the original design matrix into four clusters, representing each of the four orientations. We demonstrate this by plotting the sorted and unsorted clusters with the `BuPu` colormap, also shown in Figure 4.

1.3.4 Task 3(d)

We use the same hashing-based and comparison algorithm we have previously described in §1.1 and §1.2 has been shown to be resilient to the thousand-magnitude of pattern data, as the size of the dataset is now in the order of magnitude $10^3 \cdot 10^3 \sim 10^6$. This is also because the data remains noiseless even at this stage of the implementation.

Figure 3: Rendered 2D Master Image of a Horse for Task 3(a) (§1.3)

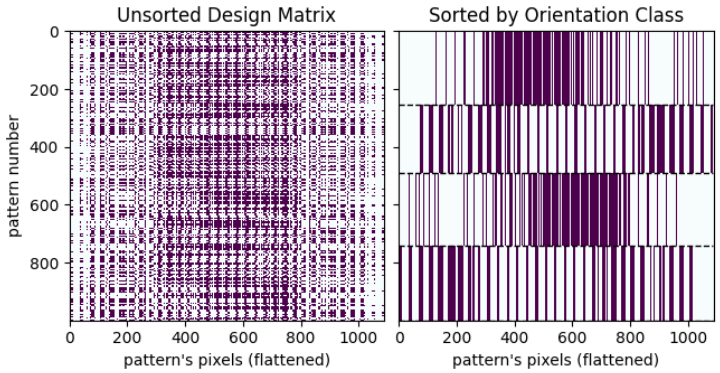

Figure 4: Unsorted and Sorted Design Matrices for Task 3(c) (§1.3)

However, if each pattern produced in the dataset were to have some noise, comparing hashes would no longer the most efficient or accurate solution since the introduction of noise results in extremely different values from the hash function. Therefore, this is where we introduce the use of clustering methods in order to guess which groups are "closest" to each other, one of which is the k-means clustering algorithm which will be used in the upcoming tasks.

Section 2: On Classifying Noisy Patterns

2.1 Task 4: Noisy Patterns

The `task4` dataset is similar in shape to the `task3` dataset. It contains $A = 1000$ images with dimensions $B = 50 \times 50$ represented as flattened pixel vectors. However, the images in the `task4` dataset contain noise that affects the values of the hash function, so the hash-based algorithm we used in Tasks 1 to 3 will not work for Task 4.

2.1.1 Task 4(a)

We simply calculate the sum of all of the values in the 50×50 pattern and take its average over 1000 such patterns using native `numpy` implementations. From this, we calculate that the average value of the sums of each image is $\boxed{1251.047}$.

2.1.2 Task 4(b)

We render the average value of all 1000 patterns throughout the implementation, and what we get is Figure 5 as a result. Upon visual inspection, there is an approximate four-fold rotational symmetry in the resulting figure. This could be attributed to the resulting average that was received upon rotating the master image. This also hints at the possibility of using $k = 4$ clusters in our implementation of the k-nearest means.

2.1.3 Task 4(c)

Due to the noise that is inherent in the dataset for this section, we implement the k-means clustering algorithm with $k = 4$ on the flattened images, which correspond to the rotations of the master image. Figure 6 shows the orientation class means from the k-means algorithm, which is clearly an image of a cat.

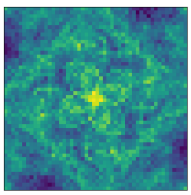

Figure 5: Average of Images without Clustering in Task 4(b) (§2.1)

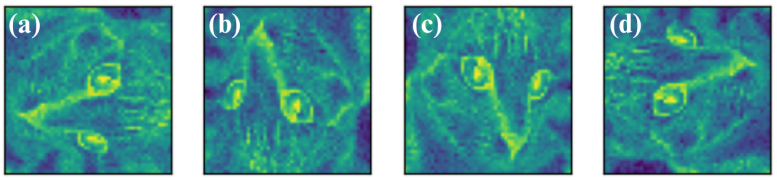

Figure 6: Average of Images with k-Means Clustering in Task 4(c) (§2.1). (*Note: All four orientations of the master image are shown.*)

2.1.4 Task 4(d)

We used the `KMeans` model from the `scikit-learn` library to identify clusters within the data. The k-means clustering algorithm is an unsupervised clustering algorithm that alternates between computing cluster means and reassigning points to the nearest mean until convergence. We ran this k-means clustering algorithm on the `task4` dataset with $k = 4$ clusters and $n\_init = 10$ initializations. We used the `k-means++` initialization method for all 10 initializations to address possible issues of clusters being close to each other.

2.2 Task 5: Likelihood to Succeed with Noisy Patterns

2.2.1 Task 5(a)

Throughout this task, we denote the probability of an event E occurring as $\mathbf{Pr}(E)$, and the expected value of a random variable X as $\mathbf{E}(X)$. We number the pixels from left to right, then top to bottom. The probability can simply be expressed by means of aligning the master image oriented at 90° to the measured image (Figure 7, Problem Statement):

$$\begin{aligned}\mathscr{L}(r = 90^\circ | K_{(4)}, \mu_{(4)}) &= \mathbf{Pr}(k_1 = 1|\beta\lambda) \cdot \mathbf{Pr}(k_2 = 0|\lambda) \cdot \mathbf{Pr}(k_3 = 0|\beta\lambda) \cdot \mathbf{Pr}(k_4 = 0|\beta\lambda) \\ &= \boxed{\beta\lambda(1-\lambda)(1-\beta\lambda)^2} \qquad \text{(Eq. 2.1)}\end{aligned}$$

2.2.2 Task 5(b)

Using (Eq. 2.1) and (Eq. 3, Problem Statement), the likelihood ratio from aligned cases (with the case 0°) to misaligned cases (where the case 90° is sufficient representation) can be expressed as:

$$\frac{\mathscr{L}(r=0^\circ|K_{(4)},\mu_{(4)})}{\mathscr{L}(r=90^\circ|K_{(4)},\mu_{(4)})}=\frac{\lambda(1-\beta\lambda)^3}{\beta\lambda(1-\lambda)(1-\beta\lambda)^2}=\boxed{\frac{1-\beta\lambda}{\beta(1-\lambda)}} \tag{Eq. 2.2}$$

2.2.3 Task 5(c)

We use our expression in (Eq. 2.2). If $\frac{1-\beta\lambda}{\beta(1-\lambda)}=1$, rearranging, $1-\beta\lambda=\beta(1-\lambda)\Rightarrow\boxed{\beta=1}$. Here, all pixels have the same value, so obviously we cannot differentiate any groups from each other.

2.2.4 Task 5(d)

We refer to Figure 8 (Problem Statement). In the aligned case, there is 1 pixel that corresponds to a value of λ with an activating photon ($k=1$), and the remaining 15 pixels that corresponds to a value of $\beta\lambda$ with no activating photon ($k=0$). In the misaligned case, there is 1 pixel that corresponds to a value of $\beta\lambda$ with an activating photon ($k=1$), 1 pixel that has a value of λ with no activating photon ($k=0$), and the remaining pixels having a value of $\beta\lambda$ and no activating photon ($k=0$). Therefore, the likelihood ratio is:

$$\begin{aligned}\frac{\mathscr{L}(r=0^\circ|K_{(16)},\mu_{(16)})}{\mathscr{L}(r=90^\circ|K_{(16)},\mu_{(16)})}&=\frac{\mathbf{Pr}(k=1|\lambda)\cdot\mathbf{Pr}(k=0|\beta\lambda)^{15}}{\mathbf{Pr}(k=1|\beta\lambda)\cdot\mathbf{Pr}(k=0|\lambda)\cdot\mathbf{Pr}(k=0|\beta\lambda)^{14}}\\&=\frac{\lambda(1-\beta\lambda)^{15}}{\beta\lambda(1-\lambda)(1-\beta\lambda)^{14}}=\boxed{\frac{1-\beta\lambda}{\beta(1-\lambda)}}\end{aligned} \tag{Eq. 2.3}$$

which is the same as the likelihood ratio in §2.2.2.

2.2.5 Task 5(e)

We list down the alignment/misalignment matrix in Table 1, which lists the measured image and the values of each of the coordinates. Red flattened values in the spiral sense are assigned a value of λ and white flattened values are assigned a value of $\beta\lambda$ (see Figure 9, Problem Statement).

master	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
0° (aligned)	λ	$\beta\lambda$	λ	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	λ	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$
90° (misaligned)	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	λ	$\beta\lambda$	λ	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	$\beta\lambda$	λ	$\beta\lambda$	$\beta\lambda$

Table 1: Alignment Matrix for Task 5(e) (§2.2.5)

Therefore, the likelihood ratio is:

$$\begin{aligned}\frac{\mathscr{L}(\text{aligned})}{\mathscr{L}(\text{misaligned})}&=\frac{\mathbf{Pr}(k=1|\lambda)^3\cdot\mathbf{Pr}(k=0|\beta\lambda)^{13}}{\mathbf{Pr}(k=0|\lambda)^3\cdot\mathbf{Pr}(k=1|\beta\lambda)^3\cdot\mathbf{Pr}(k=0|\beta\lambda)^{10}}\\&=\frac{\lambda^3(1-\beta\lambda)^{13}}{(1-\lambda)^3(\beta\lambda)^3(1-\beta\lambda)^{10}}=\boxed{\frac{(1-\beta\lambda)^3}{\beta^3(1-\lambda)^3}}\end{aligned} \tag{Eq. 2.4}$$

2.2.6 Task 5(f)

We define M, N as in the Problem Statement. The probability of a pixel having a value of λ is $p_\lambda = M/N$, while the probability of a pixel having a value of $\beta\lambda$ is $p_{\beta\lambda} = 1 - M/N$. Suppose $f(\mu_i, k_i) = k_i \ln\mu_i + (1-k_1)\ln(1-\mu_1)$. We note that there are four values possible: $f(\lambda,1) = \ln\lambda$, $f(\lambda,0) = \ln(1-\lambda)$, $f(\beta\lambda,1) = \ln(\beta\lambda)$, $f(\beta\lambda,0) = \ln(1-\beta\lambda)$. Therefore, by the linearity of

expectation theorem,

$$\begin{aligned}\mathbf{E}(\text{likelihood}) &= \mathbf{E}\left(\sum f(\mu_i, k_i)\right) = N \cdot \mathbf{E}(f(\mu_i, k_i)) \\ &= N\left[\frac{M}{N}\lambda \cdot \ln\lambda + \frac{M}{N}(1-\lambda)\cdot\ln(1-\lambda) + \left(1-\frac{M}{N}\right)\beta\lambda\cdot\ln(\beta\lambda) + \left(1-\frac{M}{N}\right)(1-\beta\lambda)\cdot\ln(1-\beta\lambda)\right] \\ &= \boxed{M(\lambda\ln\lambda + (1-\lambda)\ln(1-\lambda)) + (N-M)(\beta\lambda\ln(\beta\lambda) + (1-\beta\lambda)\ln(1-\beta\lambda)} \qquad \text{(Eq. 2.5)}\end{aligned}$$

When graphed in `matplotlib`, we see a graph similar to that of the problem statement.

2.2.7 Remarks

We discuss briefly how this would be useful in the clash of the previous question: on whether the images differ because of noise or orientation. In Task 6 (§2.3) and Task 7 (§2.4), we notice that all of the images have binary signals (either values of 1 or 0), which correspond to our pattern image $\vec{k}$ in this question. Even though the signals generated by the master image $\vec{\mu}$ are non-binary (they have multiple values possible), we can then determine the log-likelihood that each pattern image $\vec{k}$ matches an orientation of the master $\vec{\mu}$. This could be a distance metric[2] for clustering algorithms but is not used in this report, the reason for which is explained in our Remarks (§3).

2.3 Task 6: Scaling Up with Sparse Data Formats

The Task 6 dataset is divided into two one-dimensional numpy arrays `task6a` and `task6b`, both of shape (1556153,). There are $A = 65535$ images in the dataset, each with dimensions $B = 25 \times 25$. For this section, we denote the ith entry of the `task6a` and `task6b` datasets as x_i and y_i, respectively. x_i represents the pattern number, while y_i specifies the index of the pixel in the flattened pixel array. All values in these datasets satisfy $0 \le x_i \le 65534$ and $0 \le y_i \le 624$.

Our implementation of k-means clustering thus far has been using a dense design matrix representation. Unfortunately, this has its limitations in storing larger pattern sizes (number of pixels) or millions of measured patterns. Overcoming this is necessary to handle larger datasets, such as the Task 6 dataset. In this task, we improve upon our previous implementation by using a sparse data representation, which leverages the tendency of the measured pattern to exhibit very few signals individually.

2.3.1 Task 6(a)

We calculated the average pixel value (i.e., average row-sum of the sparse data) to be $\boxed{23.745}$. This was calculated by applying mean sum on the sparse array representation of the `task6a` dataset.

2.3.2 Task 6(b)

We can generate an image similar to the design matrix mean image in Figure 5, but instead via sparse data format. The final result for Task 6, without regards to the correct orientation of the data, is shown in Figure 7.

2.3.3 Task 6(c)

Similar to Task 2.1, we also used k-means clustering (with $k = 4$) on the sparse data to get the mean images of each orientation class. This final results for Task 6, with regards to the correct orientation of the data, are shown in Figure 8. Our results resemble a person.

2.3.4 Task 6(d)

In this section, we were able to overcome the storage limitations of standard k-means by using the sparse representation of its data. To do this, we turned the provided datasets into coordinate sparse

[2]This expected value is widely known as the **cross-entropy** of the dataset.

arrays (`coo_array`) using the `scipy.sparse` library. The input to the `coo_array` constructor was a list of coordinates $S = \{(x_0, y_0), (x_1, y_1), ..., (x_{n-1}, y_{n-1})\}$ ($0 \leq x_i \leq 65534$ and $0 \leq y_i \leq 624$) from the `task6a` and `task6b` datasets. The value of the sparse matrix at each point $P \in S$ was set to 1. The increase in the dataset size allows our algorithm to accommodate more measurements, which translates to improved signal-to-noise ratio in real-world applications.

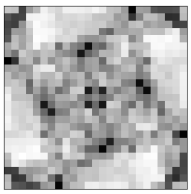

Figure 7: Average Image from Sparse Data in §2.3 without Clustering

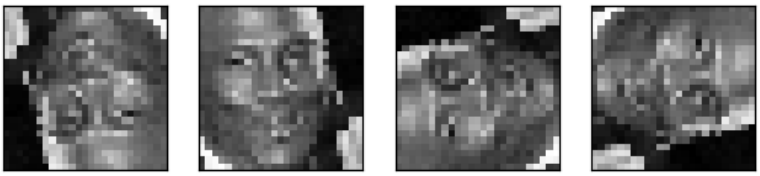

Figure 8: Average Image from Sparse Data in §2.3 with Clustering. (*Note: All four orientations of the master image are shown.*)

2.4 Task 7: Sharing Data Among Orientations

The Task 7 dataset is subdivided into two smaller datasets `task7a` and `task7b`; however, the dataset now contains $A = 100000$ images, with the same dimensions $B = 25 \times 25$ pixels.

2.4.1 Task 7(a)

The images that result in the implementation of the modified k-means algorithm in Algorithm 1 can be seen in Figure 9 below.

While there are four produced orientations, we generate the most likely master image from these four orientations. Note that this master image is orientation-agnostic, so we guess that the "master image" is $M_{T(1)}$. The complete methodology is described in Task 7(b). But, in summary, for each image, we find the orientation with the largest cosine similarity to $M_{T(1)}$. The cosine similarity $\cos\theta$ is defined as:

$$\cos\theta = \frac{\vec{\mu}_1 \cdot \vec{\mu}_2}{\|\vec{\mu}_1\|\|\vec{\mu}_2\|} \qquad \text{(Eq. 2.6)}$$

where a higher value of $\cos\theta \to 1$ indicates the two vectors are mostly aligned in vector space. The orientation with the highest $\cos\theta$ is said to match the orientation $M_{T(1)}$. Taking the average of each of the correct orientations with respect to $M_{T(1)}$ produces the master image as shown in Figure 10.

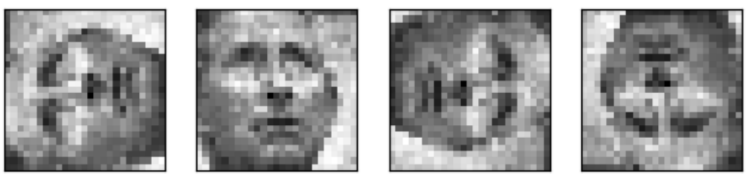

Figure 9: Orientations of the Image Resulting from Modified k-Means Algorithm in Task 7(a) (§2.4). (*Note: All four orientations of the master image are shown.*)

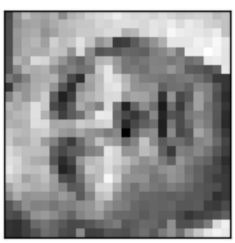

Figure 10: Master Image Resulting from Modified k-Means Algorithm in Task 7(a) (§2.4)

2.4.2 Task 7(b)

In the first draft of our k-means algorithm, we note two observations with failed attempts: (1) some cluster means had sparse signals (e.g., white signals like in Panel 1A, Figure 11), and (2) some clusters had overlapping rotations (e.g., Panel 1B, Figure 11), which are both low entropy solutions. This is a problem with the initialization of means. Therefore, we discuss two modifications to the k-means algorithm (see Algorithm 1).

Algorithm 1 Modified K-means Algorithm for Task 7b. The number of iterations to run the algorithm for, T, is a hyperparameter.

procedure ModifiedKMeans
  Initialize the master image μ_0' with each entry uniformly sampled between 0 and 1.
  Predict the means of the cluster to be the four rotations $\mu_{0(0^\circ)}, ..., \mu_{0(270^\circ)}$ of μ_0'.
  for iterations $i = 1, 2, 3, ..., T$, **do**
    Implement the k-means algorithm using $\mu_{(i-1)(0^\circ)}, ..., \mu_{(i-1)(270^\circ)}$ as the initial means.
    Store the four master images $\vec{M}_{i(1)}, ..., \vec{M}_{i(4)}$ from the k-means algorithm.
    Select the cluster μ_i' with the highest entropy $S(\vec{\mu}')$ (Eq. 2.7) as the new master image.
    Set the new means μ_i to be the 90° rotations of μ_i'.
  end for
  for images $j = 1, 2, 3, 4$ **do**
    Define $\vec{A}_j$ to be a rotation of the vector $\vec{M}_{T(j)}$ that maximizes the cosine similarity with $\vec{M}_{T(1)}$.
  end for
  Take the predicted master image to be $\vec{M} = \frac{\vec{A}_1+\vec{A}_2+\vec{A}_3+\vec{A}_4}{4}$

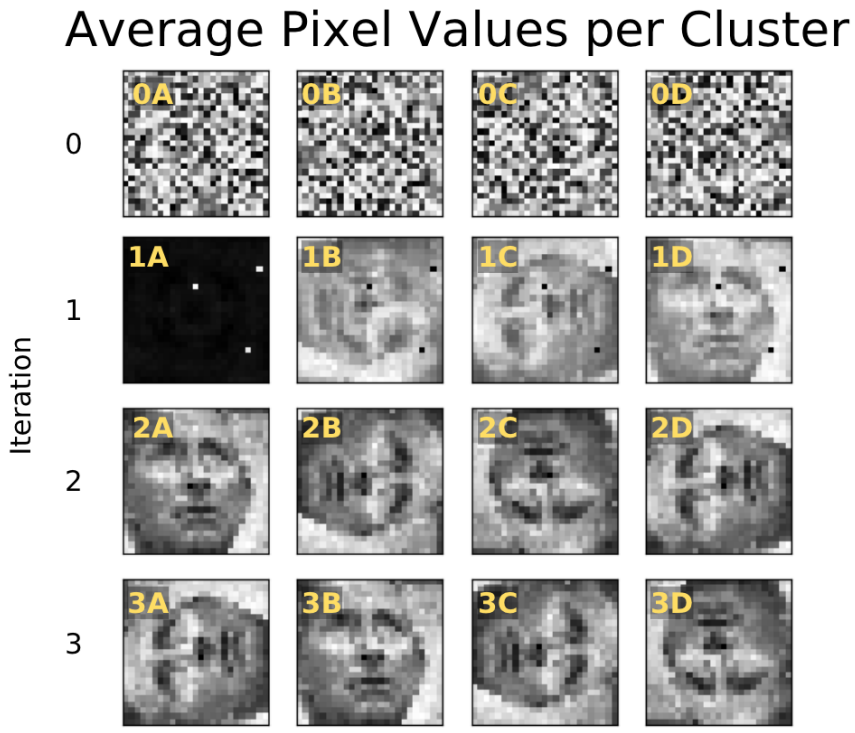

Figure 11: Iterations of the Modified k-Means Algorithm for §2.4. (Labels Added with Canva)

First, to accommodate a larger dataset, we used a dictionary of keys sparse matrix (`dok_matrix`) from `scipy.sparse` since the `scikit-learn` implementation of k-means does not support coordinate sparse matrices with large indices.

Second, we used the cluster whose mean had the highest modified Shannon entropy (Eq. 2.7) as the seed for the next iteration of k-means. This step aims to penalize means that have low entropy. For a proposed master image $\vec{\mu}$ with individual elements μ_i, we defined the entropy as:

$$S(\vec{\mu}) = -\sum \left[\mu_i \log_2(\mu_i) + (1-\mu_i)\log_2(1-\mu_i)\right] \qquad \text{(Eq. 2.7)}$$

After multiple iterations of the modified algorithm, we see the results of each iteration in Figure 11. All four cluster means produce a discernible image. Afterwards, we shared the information from each of the orientations to produce a final master image (Figure 10) as described in Algorithm 1 and Task 7(a)

2.4.3 Remarks

Since the k-means model successfully produces a discernible image for all clusters, we conclude that the model meets the goal for this question, which was to produce an image of a person (although we are not too sure who this is). With modifications to this model, we believe that this model could find useful applications in imaging, such as in Single Particle Imaging.

Section 3: Recommendations and Remarks

There are multiple recommendations that the authors would like to make in order to improve the use of k-clustering algorithms. First, the cross-entropy of the dataset (discussed in §2.2) is a better metric to determine the difference in entropy between two arrays instead of analyzing the entropy of a certain state of the master image. However, the default implementation of `scikit-learn` does not allow for the application of a custom distance metric. Second, the authors were unable to find the implementation of a cross-entropy system that satisfied all properties of a distance metric $d(\vec{x}, \vec{y})$ between vectors $\vec{x}$, $\vec{y}$, and $\vec{z}$:

- Non-negativity: $d(\vec{x}, \vec{y}) \geq 0$
- Symmetry: $d(\vec{x}, \vec{y}) = 0$ if and only if $\vec{x} = \vec{y}$
- Commutativity: $d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x})$
- Triangle Inequality: $d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z}) \geq d(\vec{x}, \vec{z})$

Combining these two recommendations, we can implement the k-means algorithm using a custom distance metric without the use of libraries. Yet, as the k-means algorithm is a heuristic algorithm, any method of attaining an answer makes the result successful enough. Third, a measure of convergence for the iterations in Task 7(b) could be implemented to determine convergence. Finally, it is worth noting that novel methods should be observed to implement single-particle imaging (SPI), as non-integer rotations could be accounted for in the motion of objects such as biomolecules.