Computer Vision
Task 4
Team 06

Team Members:

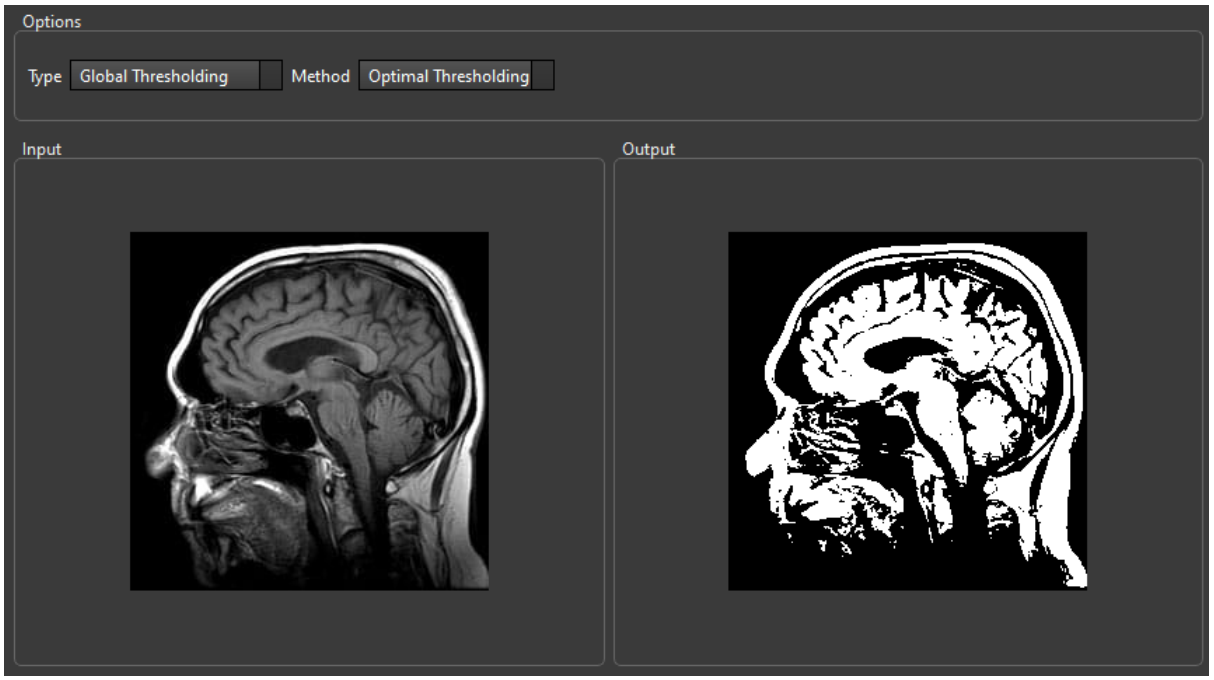
Name	Section	BN
Romaisaa Shrief	1	36
Kamel Mohamed	2	11
Youssef Shaban Mohamed	2	56

Optimal Thresholding

Algorithm:

The main goal of this algorithm is to find the best threshold value, so first we assume any random value to be the threshold, then we calculate the mean of each class by dividing *f(i, j)* by total number of pixels in each class, then calculate the new value of threshold, and repeat till the new value is equal the old value, this algorithm is applied on the whole image.

Result Sample:



Otsu Thresholding

Algorithm:

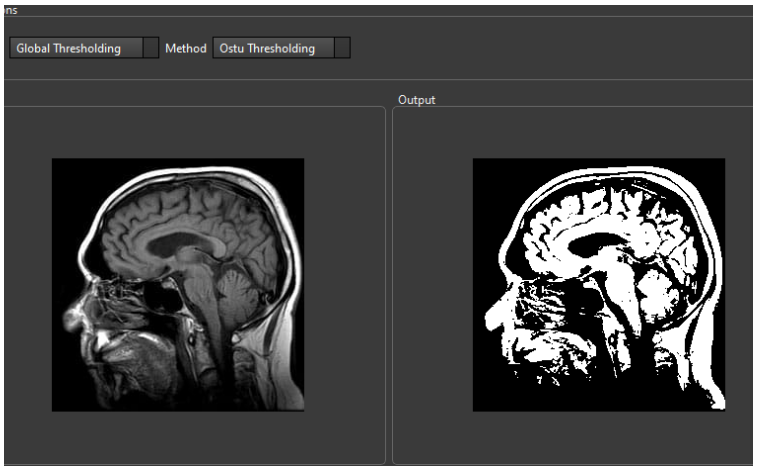
In Otsu, we use the histogram of image, to calculate the best variance, and then get best T, we follow these equations to calcuate variance:

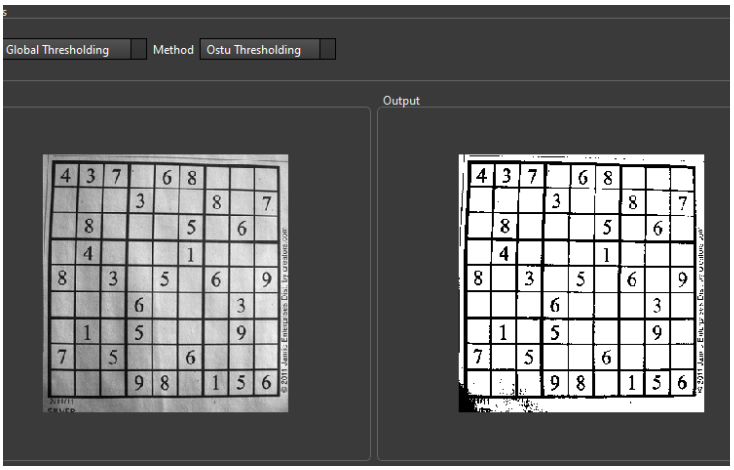
$$\omega_0(t) = \sum_{i=0}^{t-1} p(i) \qquad \mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i) \qquad \mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = \omega_0(t)(\mu_0 - \mu_T)^2 + \omega_1(t)(\mu_1 - \mu_T)^2$$
$$= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

Result Sample:

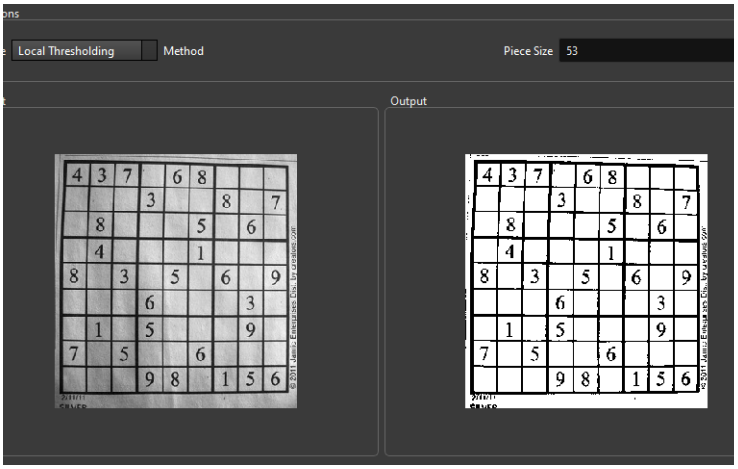




Local Thresholding

Algorithm:

In this algorithm we apply any binary thresholding algorithm in small pieces of image, not the whole image like global thresholding, we first get the piece, apply our function and get T, apply binarization on this piece only and then put it in its place.



Spectral Thresholding

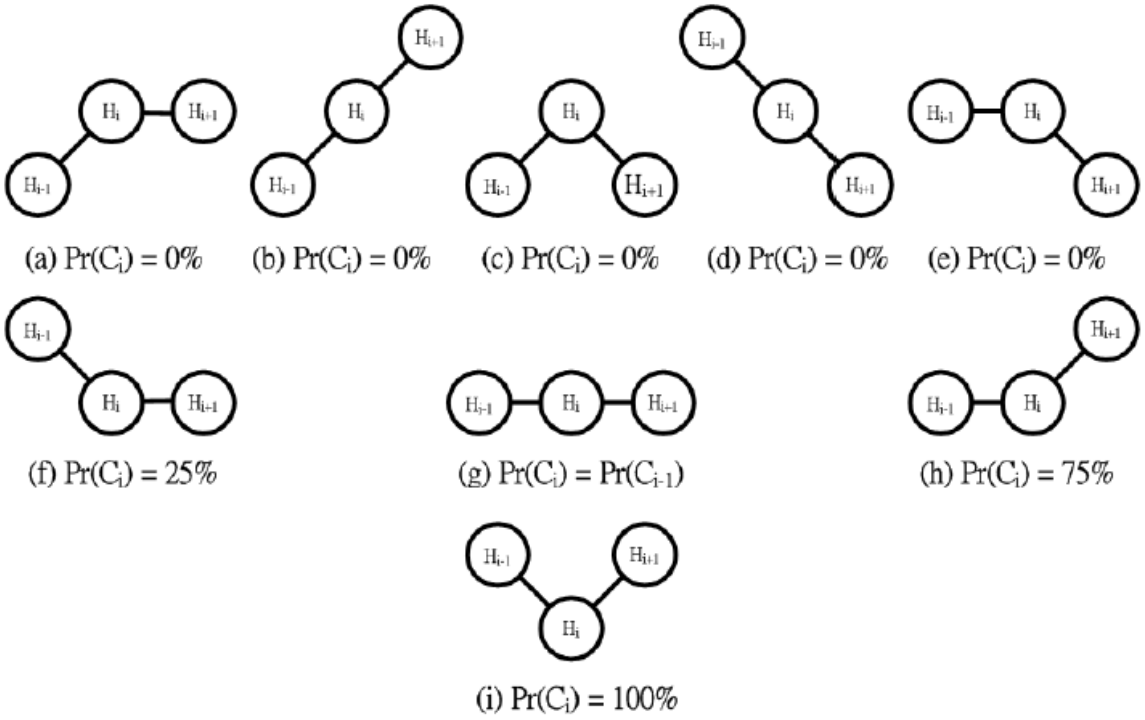
Algorithm:

After searching we find that nothing called spectral thresholding, but after asking, we find that this algorithm called Multilevel Thresholding, which means having multiple thresholds not just one, in this part instead of implementing conventional way that's need so many iterations to get the write values,

1- we first start with histogram of the image, calculate normalized version of it .
2- split histogram into 32 piece and iterate over and calculate H for each piece as H can be determined by:

$$H_i = \frac{h_i}{\max(h_i)} \times 100, \quad i \in \{0, 1, \cdots, 31\}, \quad \text{where} \quad h_i = \sum_{i \in C_i} f_i$$

And to decide if there is a peak in this piece or it will be a shape of these 9 combinations.



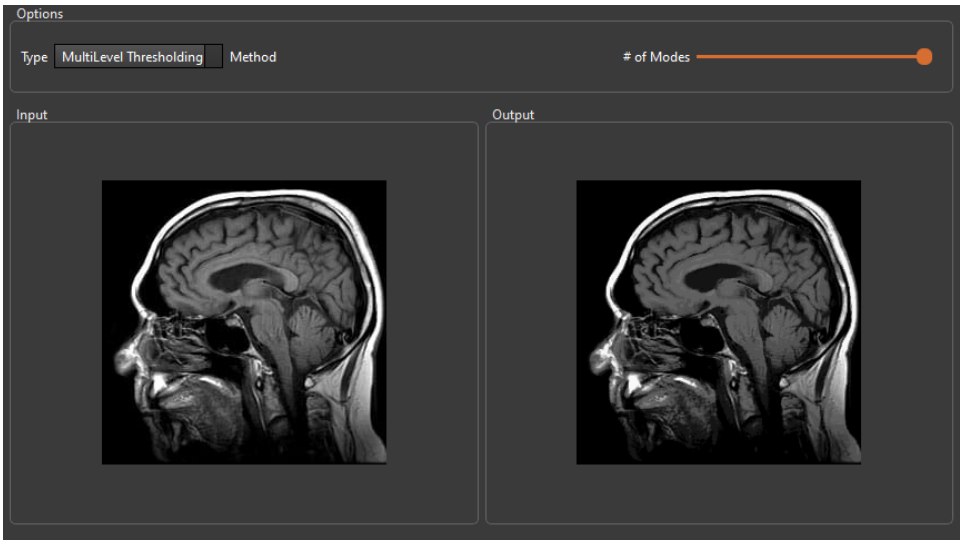
3- after this iteration, we make another one but in reverse order, from C31 to C0, and add the result also to the list of peaks, this peaks called valleys and this method called valleys estimation.
4- we then calculate TSMO a modified version of Otsu, that bins histogram to less bins than 256, and calculate threshold of the image, this method is faster than original otsu.
5- we now have valleys so we split histogram to parts and calculate TSMO for each part, and get the threshold value.
6- now we have thresholds, and can apply our thresholding method with these values.
What make's this method better:

1- less computations.

2- automatic determination of number of classes, but can be modified by changing number of bins in histogram.

This is an implementation of This Paper.

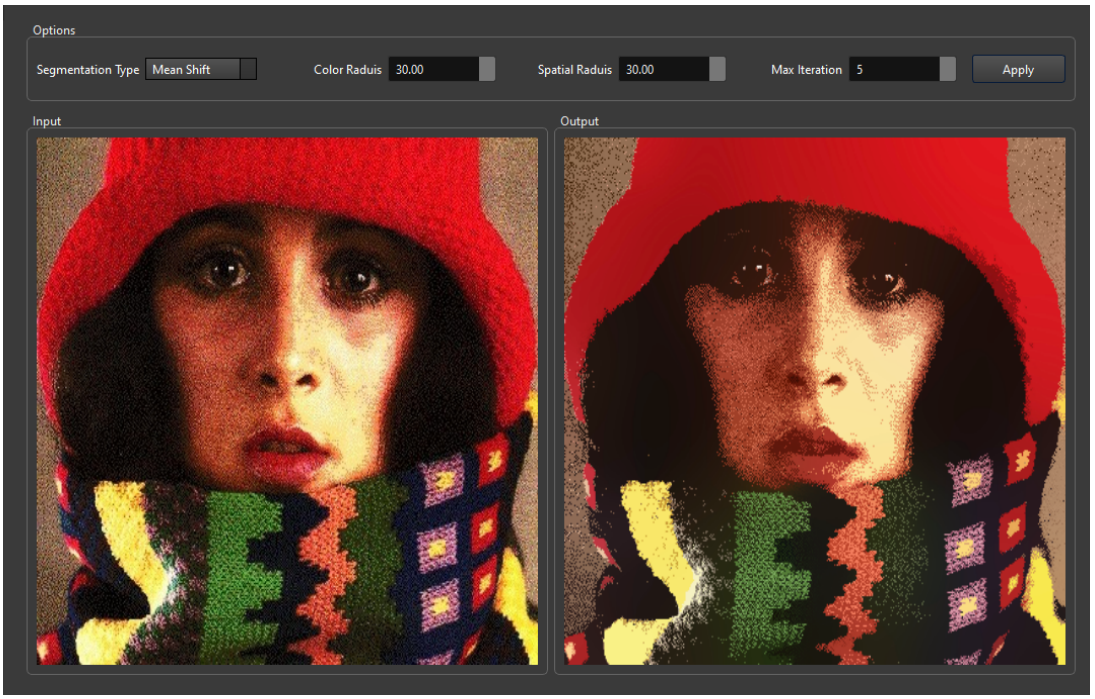
Result Sample:

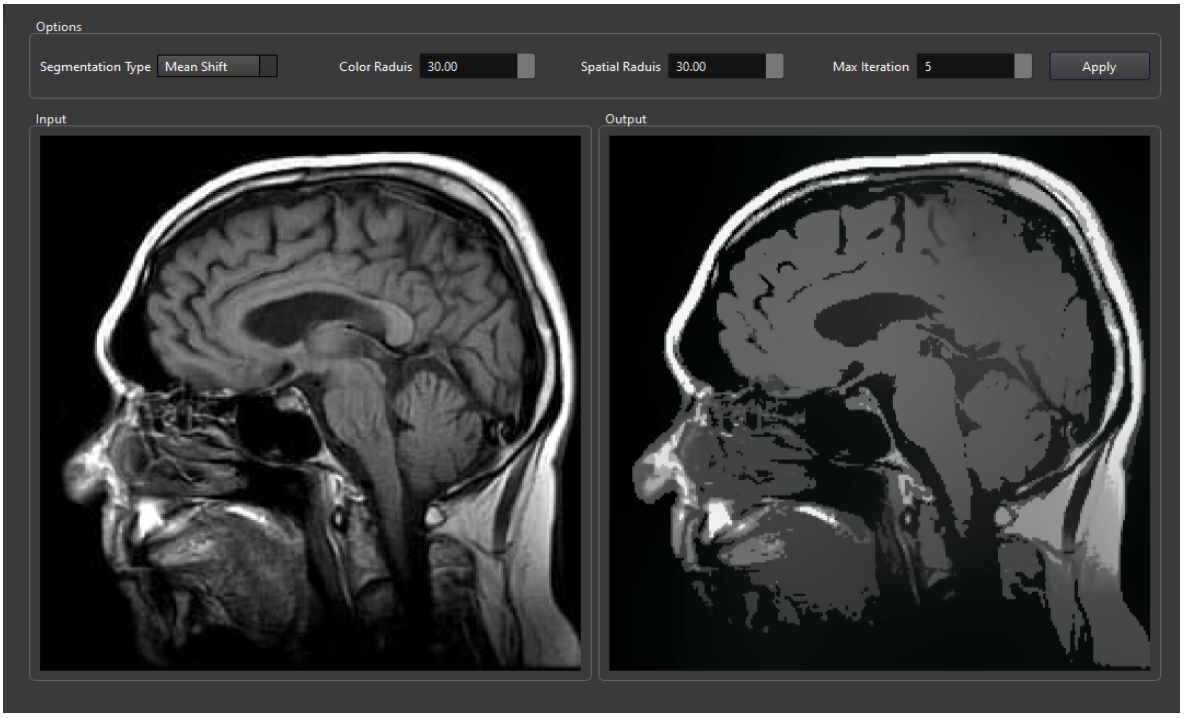


Mean Shift Segmentation:

Algorithm and parameters:

Iteratively shift each pixel towards the mean of its neighbouring pixels having similar colour values to the current pixel(based on colour radius), until convergence is reached(Max Itr in our algorithm).
The colorspace to work was LUV colour space.
For mean calculations, we use a kernel density estimator to estimate the density of the data points in the feature space to assign weights to neighbouring data points based on their distance from the current data point(spatial and colour distance). We used a Gaussian function that assigns higher weights to neighbouring data points that are closer to the current data point in the feature space.
Weight of each neighbour is calculated by:
weight = exp(-spatialDistance^2 / spatialRadius^2 -colorDistance^ 2 / colorRadius^2)
As spatialRadius, colorRadius given by user.




Options
Segmentation Type
Mean Shift
Color Raduis
30.00
Spatial Raduis
30.00
Max Iteration
5
Apply
Input
Output

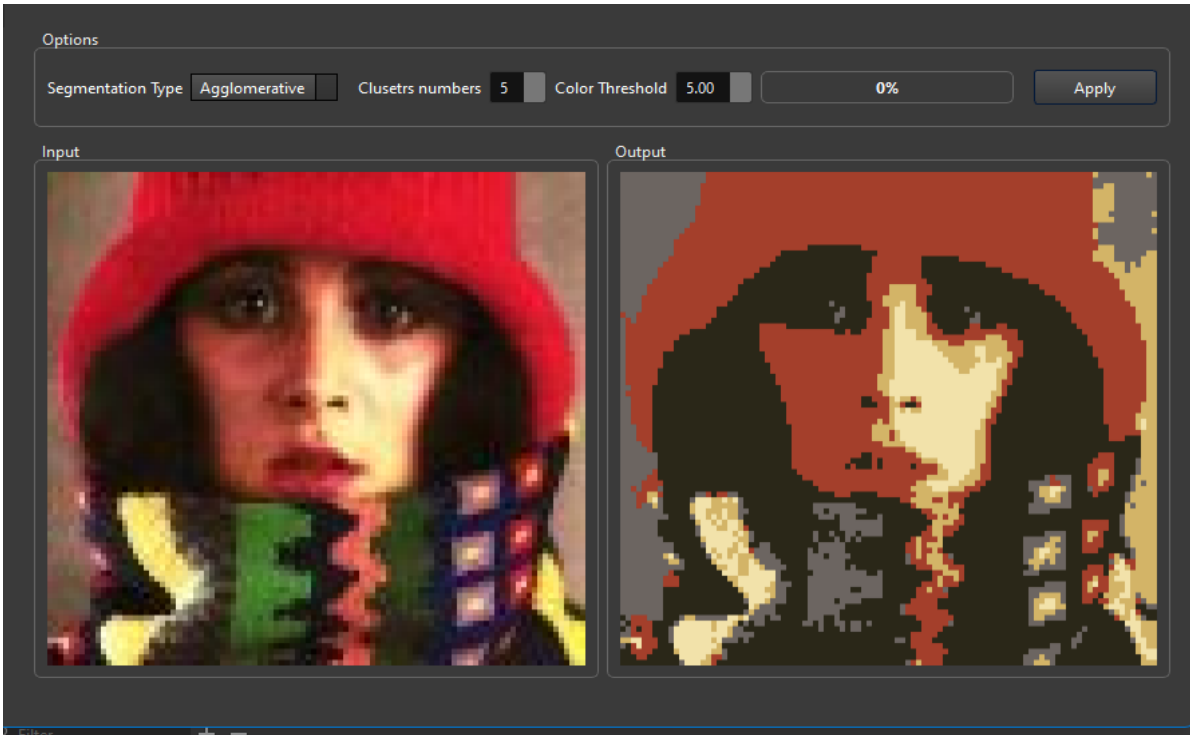

Options
Segmentation Type
Mean Shift
Color Raduis
30.00
Spatial Raduis
30.00
Max Iteration
5
Apply
Input
Output

Agglomerative Segmentation:

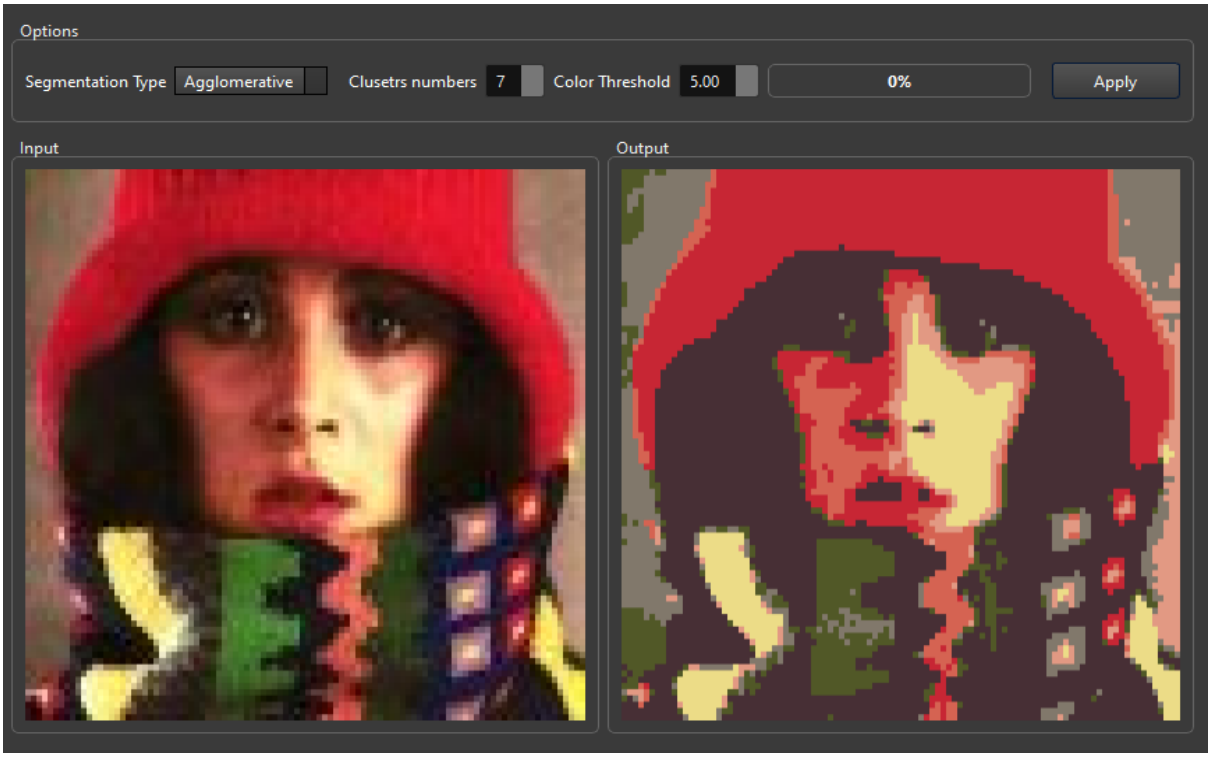
Algorithm and parameters:

The idea behind the agglomerative algorithm is to deal with each pixel as a single cluster then start to merge each 2 clusters together based on their similarity (euclidean distance between them in LUV colour space).
This algorithm is known to be slow, so we used multithreading to speed up the process. Instead of finding global min distance in the iteration (most 2 similar clusters). We create a vector of local min distances(calculated in parallel). Then loop over this vector to get the global min distance of the pixels and merge them.

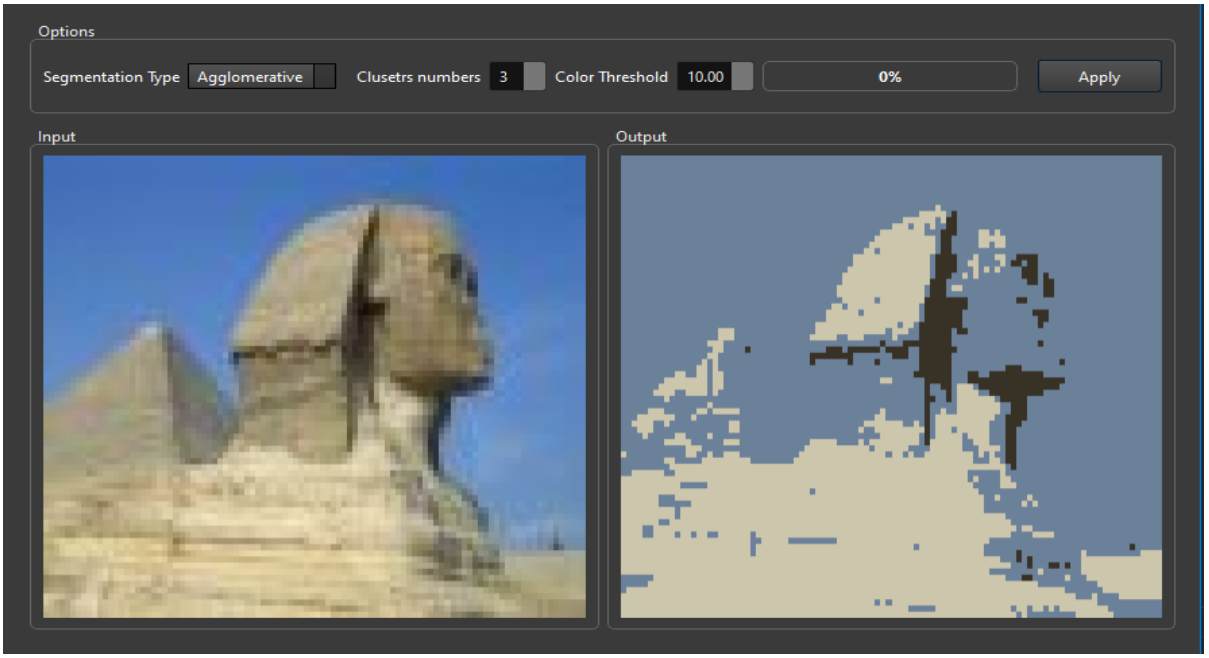
Results samples:



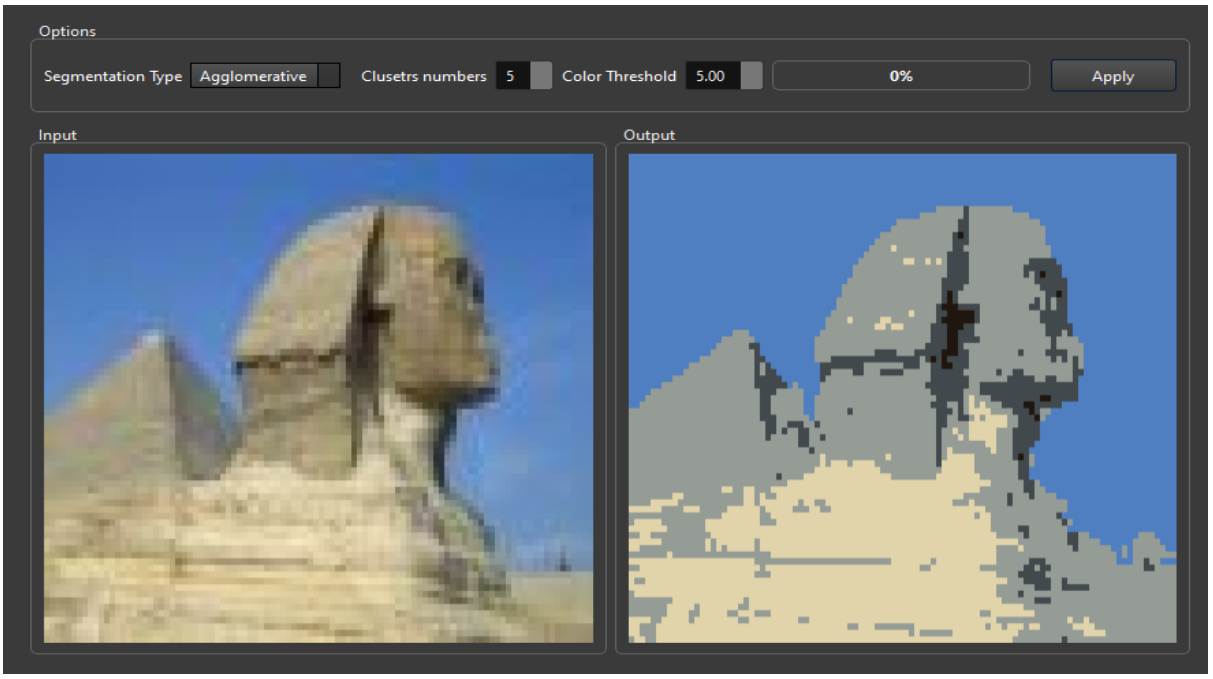
In this sample, number of clusters=5, we can observe that most similar colours (green, purple) became one cluster with another colour (average of them).



Increasing number of clusters shows (differentiate) colours accurately



In this sample, the number of clusters=3, widen the colour threshold for similarity. We observe that a higher threshold can merge two unsimilar colours together as one cluster given that the number of clusters are not enough.



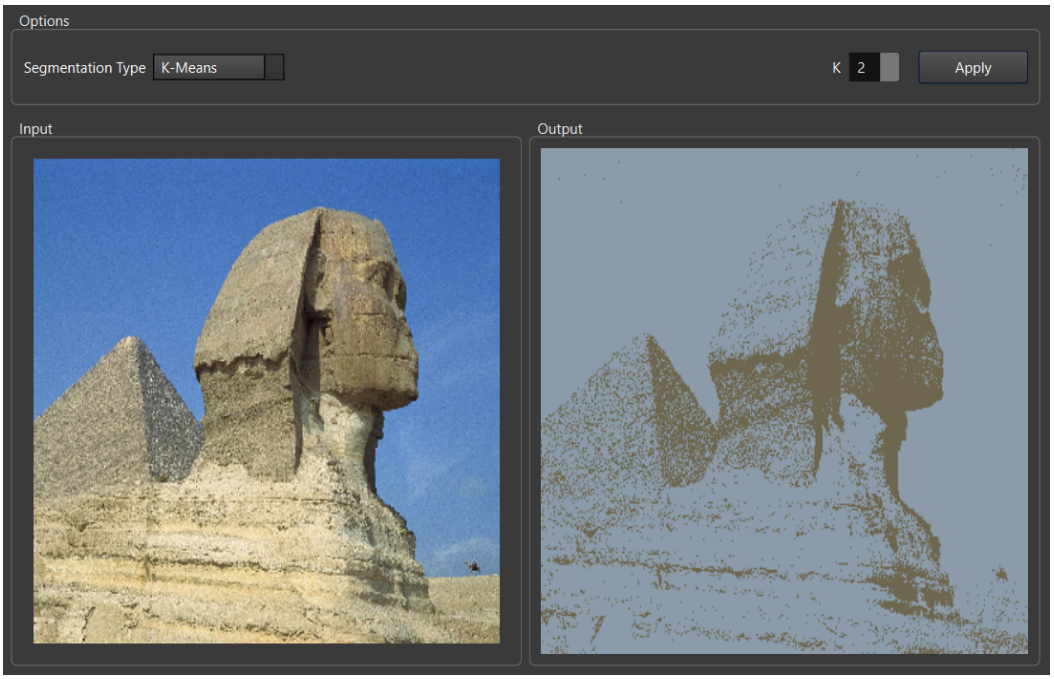
Decreasing colour threshold and increasing (over-clustering) clusters numbers leads to more obvious changes for colours, and more accurate colouring (almost same as real one) as in sky blue.

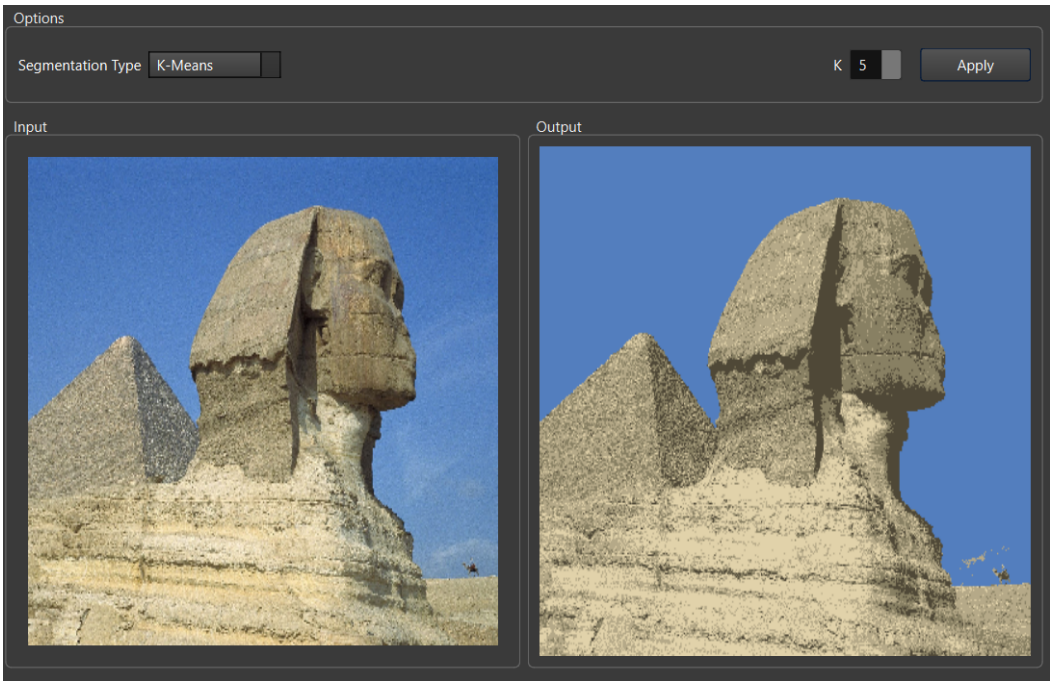
K-means Clustering

KMeans segmentation is a clustering algorithm used to group similar pixels in an image based on their feature similarity. The algorithm takes the number of clusters desired as input and assigns each pixel to its nearest centroid. The centroids are then updated based on the mean value of the pixels assigned to them, and the assignment process is repeated until convergence.
The quality of the segmentation depends heavily on the initialization of the centroids, which can be random or chosen using other methods such as k-means++.

Results:

This is the effect of the output when we increase the k number.





Region Growing

Region growing segmentation is a pixel-based segmentation algorithm that groups neighboring pixels together into regions based on some similarity criterion. The algorithm starts by selecting a seed pixel and then grows the region by adding neighboring pixels that satisfy a similarity condition. This process continues until no more pixels can be added.

The similarity condition can be defined based on various features such as color, texture, or intensity. The parameters for the algorithm include the seed pixel selection criteria, the similarity criterion, and the stopping criterion. The stopping criterion is usually based on some threshold value that determines the maximum dissimilarity allowed between neighboring pixels in the region.

Results: