ISyE 6740 – Summer 2020
Final Report

Team Member Names: Jessica Staley (903551584), Henry Staley (902959442)

Project Title: Puzzle Masters (Team #46)

Problem Statement

At the time of this report's creation, there has been a global pandemic raging for a few months, requiring many to stay inside their homes for extended periods of time. Given this increased time at home, many people have turned to a pastime that dates back to the 18th century: jigsaw puzzles. Originally made of wood, jigsaw puzzles have gone up and down in popularity over the years, seeing another notable surge during the Great Depression. Jigsaw puzzles, however, are not for the impatient, as they can require very tedious work, particularly in regards to the task around which our analysis will be centered: sorting the pieces.

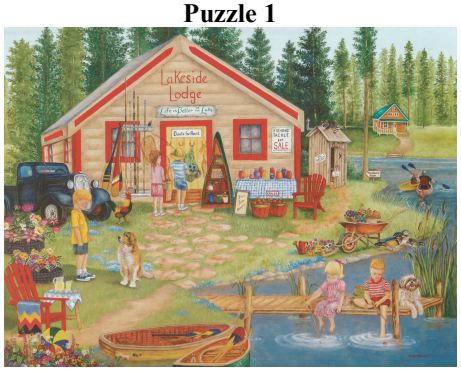
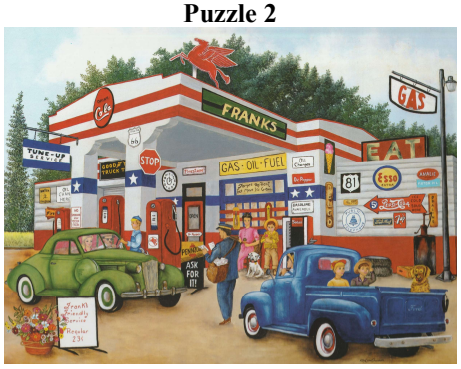
For any puzzle larger than a few hundred pieces, it becomes too difficult to parse through all the pieces when searching for a specific one. To address this, people typically sort the puzzle pieces into different piles based on appearance. That way, when searching for a particular piece, someone would only need to search the pile in which the piece should be sorted based on its appearance. This task of sorting however, can be very tedious and it may not be initially clear how the pieces should be divided. Our goal is to develop an algorithmic approach that emulates the process of manually sorting puzzle pieces by appearance before the puzzle is assembled. An ideal algorithm for this purpose would, without any human intervention, sort the puzzle pieces into the optimal number of groups based on the same characteristics that a person would focus on when manually sorting puzzle pieces.

Data Source

Our data source consisted of three self-selected, 300 piece puzzles with irregularly shaped pieces (as opposed to forming neat rows and columns). The puzzle pieces were laid, in batches of 20, within a carefully measured grid and scanned; these batch scans were then separated into individual pieces and used as the input images to our algorithm. Naturally, as the number of pieces in a puzzle increases, the individual pieces become more visually homogeneous and likely to only portray a single color or pattern. Given this, it can be assumed that a sorting algorithm will perform better as the size of the puzzle increases. We acknowledge that the problem statement is most relevant to larger puzzles (e.g, 2000+ pieces), however, due to the manual effort and time required to generate the input datasets, as well as the desire to use multiple puzzles to ensure generalization, we determined 300 piece puzzles were sufficient to develop a prototype.

Once the desired puzzle size had been determined, the specific puzzles were chosen by prioritizing images with a mix of relatively homogenous sections, such as the sky or a uniformly colored building, in addition to sections containing more variation in terms of color and pattern. This was a very subjective selection process that was somewhat restricted by the availability of 300 piece puzzles, which fall between the two primary categories of puzzles: childrens' puzzles (100-200 pieces) and adult puzzles (500+ pieces).

An image of each of the three completed puzzles is shown below:

Puzzle 1	Puzzle 2	Puzzle 3
		

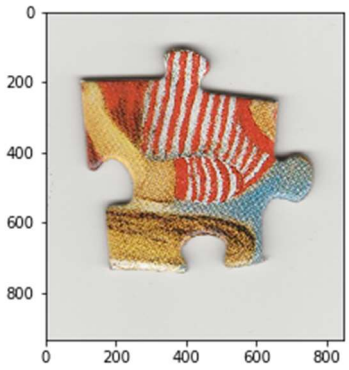
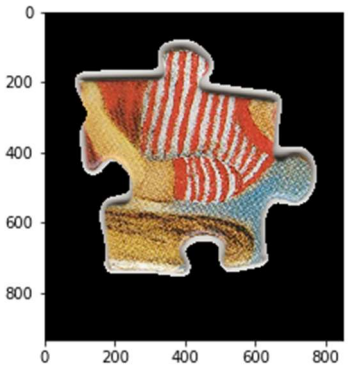
Clustering Methodology

There is no objectively perfect grouping of puzzle pieces for any given puzzle, thus, we are faced with an unsupervised learning problem for which a clustering algorithm would appear to be the obvious solution. We considered several different clustering models, and the process used for evaluating and comparing them will be discussed in a later section.

Featurization

In order to determine which features to use in the clustering models, we considered the aspects that would be most important to someone manually sorting the puzzle pieces. These aspects broadly boil down to color and pattern/texture. It should be acknowledged that in practice, people usually separate the "edge" pieces and construct the border of the puzzle before sorting the rest of the pieces. While this could be emulated using a separate algorithm that identifies the "edge" pieces before feeding the remaining pieces into the sorting algorithm, our models simply ignore this and sort the "edge" pieces along with the others based on their color and texture features.

In order to calculate the features related to color and texture, we had to first remove the background from each image so that we only consider the pixels that represent the puzzle piece. This is achieved through the use of a Sobel filter, with all of the pixels determined to be background being replaced with black. This allows us to isolate the puzzle piece for the calculation of the features, as explained below. It is worth noting that this methodology may cause issues if used on a puzzle that contains a significant amount of true black (rgb=[0,0,0]) pixels, such as in the portrayal of a night sky. An image of a puzzle piece before and after the Sobel filter was applied is included below.

Before	After
	

Color Features

Because each puzzle piece image had the same number of pixels and each pixel is represented as a 1x3 array of RGB values, our initial thought was to treat each pixel as a feature and cluster the pieces on all 1000+ features. The problem with this is that the pieces can have drastically different sizes and shapes, and there is no way to determine if the puzzle pieces are facing the "same" direction, therefore, the pixel to pixel comparison would not be valid. We circumvented this issue by engineering 12 features that capture the overall color of the piece as well as the most prominent clusters of colors within each piece *after* the true black pixels identified by the Sobel filter were removed.

The first 3 of the 12 color features capture the average RGB value for the entire piece. These 3 features are most useful and telling when the entirety of the piece is generally the same color (e.g. a piece of the sky that is all light blue). When the puzzle piece has a large variation in colors and patterns, like the piece shown above, the value driven by these 3 features is lesser.

To account for the different colors present in most puzzle pieces, we performed KMeans clustering on the pixels within each piece. The KMeans model identified 3 clusters of pixels within each piece; our thought was that these 3 clusters would provide insight into the most common colors present in each puzzle piece. The 3 clusters within each piece were then sorted in decreasing order by cluster size and the RGB values associated with each clusters' centroids were used as features (making a total of 9 additional features per puzzle piece).

Texture Features

While color is likely the most influential factor when sorting puzzle pieces, it does have its limitations, particularly when using computed color features instead of visual inspection. For pieces that display an irregular pattern or texture, such as foliage or clouds, the average RGB values can be unstable and potentially quite different between pieces that are immediately adjacent to each other and displaying the same pattern or texture. To accommodate for this, we introduced a number of "texture" features.

The texture features used in our clustering models are calculated using a statistical method known as the Gray-Level Co-Occurrence Matrix (GLCM). This produces a matrix representation of patterns displayed in an image. To illustrate the concept, please consider a simple 16 pixel image containing three colors (red, blue, green) as displayed in Figure 1. The resulting GLCM will have one row and one column for each color present in the image, as seen in Figure 2. Given a displacement vector, which in this demonstration will be **[1,-1]** , the numbers within each cell of the matrix represent counts for pairs of pixels that are 1 down and 1 to the right of each other and whose colors are represented by the row and column.

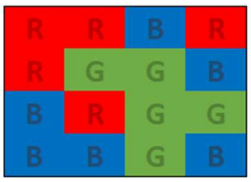


Figure 1

	R	B	G
R	1	0	3
B	0	2	0
G	0	1	2

Figure 2

For example, observe the number 3, highlighted in Figure 3, that sits at the intersection of the red row and the green column. This number represents the number of instances in the image where a green pixel was 1 down and 1 to the right of a red pixel, portrayed in Figure 4.

	R	B	G
R	1	0	3
B	0	2	0
G	0	1	2

Figure 3

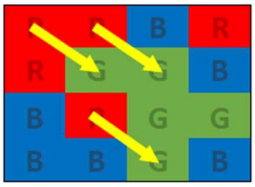


Figure 4

Although this concept was easiest to demonstrate using color pixels, as the name suggests, the Gray-Level Co-Occurrence Matrix is actually calculated on gray-scale images, where the columns and rows of the matrix represent the different shades of gray present in an image. For our purposes, this was best accomplished by formatting the pixels in our gray-scale images as integers and having one row and column for each integer value between 0 (black) and 255 (white), as displayed in Figure 5. Remember, though, that our images have had the backgrounds removed and replaced with black, which, in a gray-scale image, will be represented by the integer value 0. In order to remove the background from the calculation of our texture features, we simply remove the first row and column from our GLCM, with the remaining matrix (**P**), represented by the highlighted section in Figure 6, being used in the calculation of the features as detailed below.

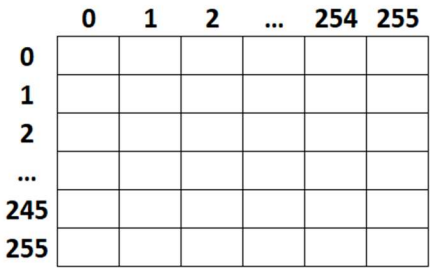


Figure 5

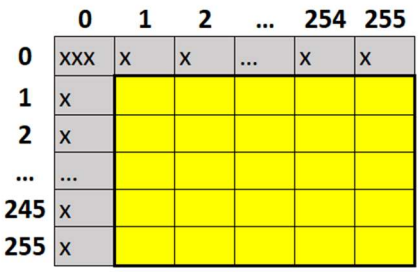


Figure 6

Once this matrix (**P**) has been constructed, there are numerous texture features that can be extracted. However, for our clustering models, we only include the 6 metrics provided through the scikit-image python package. These metrics are detailed below.

Metric	Equation	Description
Contrast	$\sum_{i,j=0}^{levels-1} P_{i,j}(i-j)^2$	Measures the difference in color between adjacent pixels, scores higher for images where adjacent pixels are very different from each other
Dissimilarity	$\sum_{i,j=0}^{levels-1} P_{i,j}\lvert i-j\rvert$	Similar to Contrast, but the difference in color between adjacent pixels is weighted linearly instead of exponentially
Homogeneity	$\sum_{i,j=0}^{levels-1} \frac{P_{i,j}}{1+(i-j)^2}$	Measures how similar the colors of adjacent pixels are, scores higher for images where adjacent pixels are close in terms of color
Angular Second Moment (ASM)	$\sum_{i,j=0}^{levels-1} P_{i,j}^2$	Measures the "orderliness" of an image, scores higher when colors or neatly grouped/arranged within an image as opposed to being scattered randomly
Energy	$\sqrt{ASM}$	Similar to ASM, but weighted linearly instead of exponentially.
Correlation	$\sum_{i,j=0}^{levels-1} P_{i,j}\left[\frac{(i-\mu_i)\,(j-\mu_j)}{\sqrt{(\sigma_i^2)(\sigma_j^2)}}\right]$	Measures how well a pixel's color can be linearly predicted by its adjacent pixel, scores higher for images consisting of a small number of numerously repeated patterns

Recall that the matrix ($\mathbf{P}$) was calculated using a specific displacement vector, and choosing a different displacement vector will result in a different GLCM. In the scikit-image package, the direction of this vector is expressed in radians, and for our clustering models, we decided to calculate the 6 texture features described above at four different angles (corresponding to 0°, 45°, 90°, 135°), resulting in 24 total texture features for each puzzle piece. Furthermore, by allowing the GLCM to be symmetrical (ignoring which pixel is on which end of the displacement vector), the resulting texture features are much more stable. Regardless, some of the metrics will error out for certain puzzle pieces given certain displacement vectors, in which case they are filled with 0.

Dimensionality Reduction

Combining the color and texture features brought the total number of features to 36. However, due to varying importance and potential relationships between our features, we decided to reduce the feature space that would be used in our clustering models. We used both Isomap and Principal Component Analysis (PCA) for this task, creating several models for each method, with varying numbers of reduced features.

Clustering Models

In order to build each clustering model, we needed to first determine how many clusters to find. In order to estimate the optimal number of clusters, we used the Gap-Statistic. The Gap-Statistic is an objective way to quantify what the "elbow" method subjectively determines. The "elbow" method plots the percentage of variance explained by the clusters W_k against the number of clusters k. The point in the graph at which the increase in percent of variance explained starts to taper off is the "elbow" of the curve and the number of clusters associated with this point is the optimal number of clusters. The visual inspection required in this determination, however, does not lend itself well to dynamic decision making. The Gap-Statistic serves as a metric that our algorithm can use to independently determine the optimal number of clusters without this subjective human intervention.

The Gap-Statistic compares the percent of variance explained by the clusters to the percent of variance explained by a distribution with no obvious clustering. The estimate for the optimal number of clusters is the value of k for which W_k falls the furthest below this reference curve. The formula to calculate the Gap-Statistic is shown below:

$$Gap_n(k) = E_n^*\{logW_k\} - logW_k$$

It should be noted that we did put a cap on the maximum number of clusters (k = 10) to test. We felt that this cap was necessary to emulate a manual clustering of the pieces, due to the fact that most people don't have room on their table to sort the puzzle into more than 10 groups.

We could then build our models, through KMeans and Spectral Clustering, given the optimal number of clusters k determined by the Gap-Statistic. We only implemented the dimensionality reduction, using Isomap or PCA, for the KMeans models since the Spectral Clustering algorithm effectively performs its own dimensionality reduction prior to clustering, in a way quite similar to Isomap.

Model Evaluation/Comparison

The most appropriate way to evaluate the performance of our clustering models, in regards to the problem statement, would be to visually inspect the resulting clusters and determine how close they are to what we would produce if sorting manually. However, due to the high level of subjectivity involved in this method, we desired some objectively calculated metric(s) that could be used in combination with visual inspection to evaluate clustering results. Although a perfect performance metric would be impossible to define for this problem, we developed a couple metrics that correspond to how closely the pieces within a given cluster are grouped together in the finished puzzle. Again, this does not exactly line up with how a person would manually sort puzzle pieces (given the presence of only 4 red pieces in an entire puzzle, a reasonable person could sort those pieces together even if they will not be adjacent or even close to each other in the finished puzzle), but it does give some insight into how the clustering models are performing.

These metrics are created by representing the finished puzzle as a graph, defined by an adjacency matrix, where each piece is a node and edges are between pieces that are touching each other on the finished puzzle, as illustrated in Figure 7.

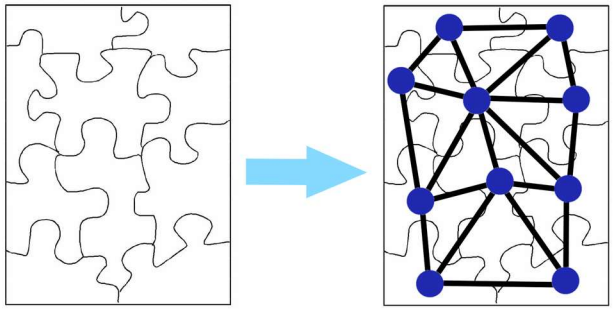


Figure 7

It should be noted that, because our puzzle pieces are irregularly shaped, many of the corners do not line up cleanly. For the purposes of constructing the adjacency matrix of the finished puzzle, two pieces are considered “adjacent” if they are touching each other, even if only at a single point with their corners. However, if those corners are separated by even the smallest space, they are not considered adjacent.

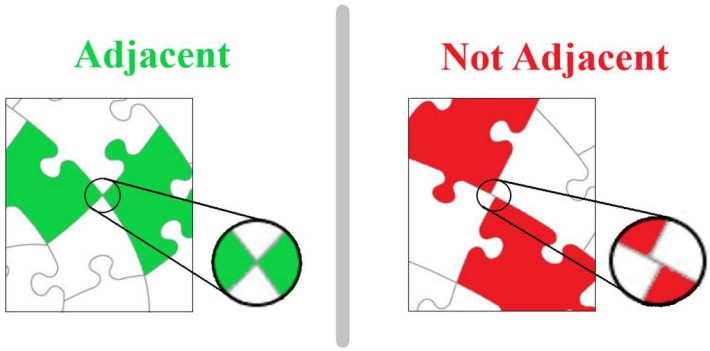


Figure 8

Once we have our cluster assignments and our finished puzzle represented as an adjacency matrix, we can then utilize metrics that are typically used for evaluating the results of clustering algorithms performed on graphs, such as Spectral Clustering. Intuitively, a graph representation of a jigsaw puzzle will be relatively evenly distributed, and would not lend itself well to being clustered. However, given the cluster assignments determined by our color and texture features, we can still use these graph-clustering evaluation metrics to determine how closely grouped our clusters are in the finished puzzle.

There are numerous metrics available to evaluate graph-clustering models, but we decided to utilize two of the most common: conductance and modularity (specifically Newman-Girvan Modularity). These scores are calculated slightly differently, with the conductance being calculated for each cluster individually and then having the minimum taken, while modularity is calculated over all clusters at the same time. However, they both roughly measure the relative number of internal edges within clusters and external edges between clusters, with higher values being preferred for modularity and lower values preferred for conductance. These two metrics, when used in combination with visual inspection of the clustering results, allow us to evaluate and compare our different clustering models with some added degree of objectivity.

Final Results

We ultimately ended up with a collection of clustering models that differed from each other in terms of the clustering algorithm, the method of dimensionality reduction, and the number of reduced features. After running all of these models on each of the three puzzles, it was immediately clear, based on both visual inspection and our evaluation metrics, that Spectral Clustering performed significantly worse than KMeans. A detailed breakdown of how the models performed in terms of conductance and modularity can be found in Table 1 of the Appendix.

Amongst the KMeans models, it was also apparent, particularly in regards to the evaluation metrics, that the models using Isomap for dimensionality reduction performed better than those using PCA, though this difference is not as pronounced as the difference in performance between KMeans and Spectral Clustering. The difference does, however, indicate that there are likely some non-linear relationships between our features. Figure 9, below, plots the pieces of one puzzle in a feature space reduced down to two dimensions through Isomap. This representation gives an idea of how clusters may be formed in the reduced space.

Figure 9

Within the KMeans models using Isomap for dimensionality reduction, there was relatively little variation and it was difficult to determine through visual inspection whether any model performed better than others. However, if we are to judge by the evaluation metrics, then the model that implements 4 reduced dimensions performed the best. Some of the cluster assignments produced by this model can be seen in Table 2 of the Appendix. Realistically, it can be assumed that all models using KMeans, Isomap, and around 4-8 reduced features will perform roughly the same.

From a visual perspective, even the best model performed suboptimally, with most clusters containing at least one piece (often a few) that likely would have been placed elsewhere if the sorting had been done manually. However, given the previously discussed constraints of using relatively small puzzles, this was the anticipated outcome. With our 300 piece puzzles, a single piece may contain several different colors, patterns, objects, or even text. This amount of potential variation within a single piece makes clustering more difficult. However, given the results of our clustering

models on the 300 piece puzzles, it does appear that the same methodology could be quite successful when applied to larger puzzles with less variation in each piece.

Future Studies

There are several improvements and extensions that could be explored within this methodology. The most useful improvement would almost definitely be the development of a more efficient process for scanning the puzzle pieces as well as constructing the adjacency matrix from the finished puzzle. The manual effort required by these tasks under our current methodology is what prevents the use of the larger puzzles on which clustering algorithms would perform best.

In regards to processing the puzzle piece images, a better method of removing the background could likely be implemented. The current method still leaves some background pixels around the edges of the puzzle piece and, as mentioned earlier, will likely cause issues with any puzzle displaying significant amounts of black.

The featurization can also be improved upon to some degree. One idea would be to use a windowing function on each piece to determine its dominant color or texture. This would reduce the effect of small variations in each piece. While the RGB clustering performed on each puzzle piece in our methodology attempts to do something similar, it is not ideal.

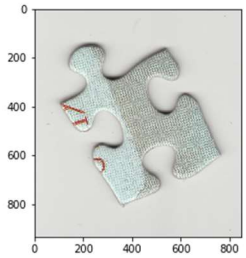
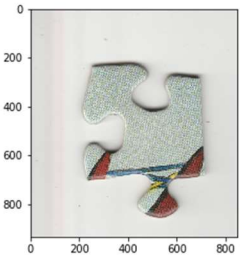
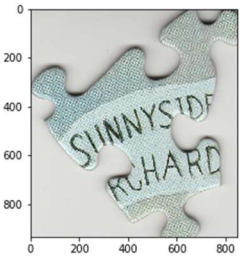
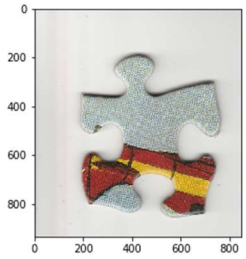
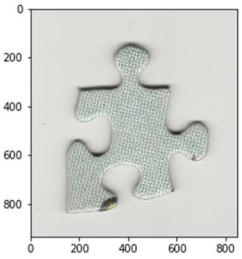
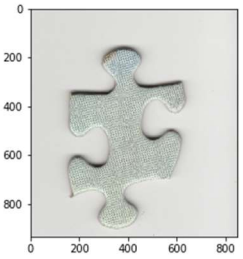
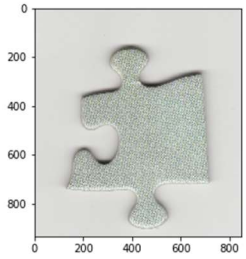
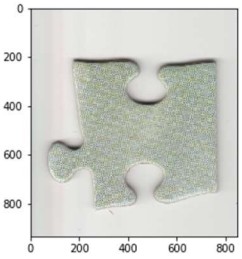
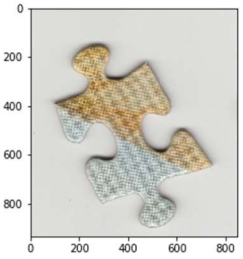
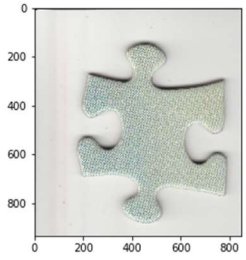
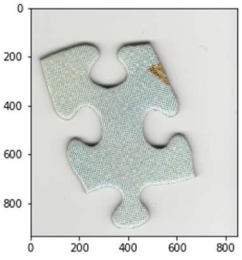
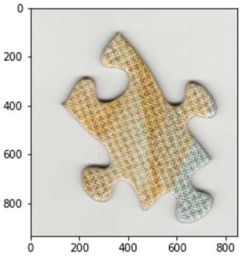
One final improvement, which could fundamentally change the nature of the problem, would be to develop some metric that better encapsulates how well the model is performing compared to manual sorting. The graph-clustering evaluation metrics we used in our methodology are a good starting point, but fail to fully represent what someone is striving for when sorting puzzle pieces. If this ideal performance metric could be developed, however, the problem could then turn into a supervised (or at least semi-supervised) learning problem.

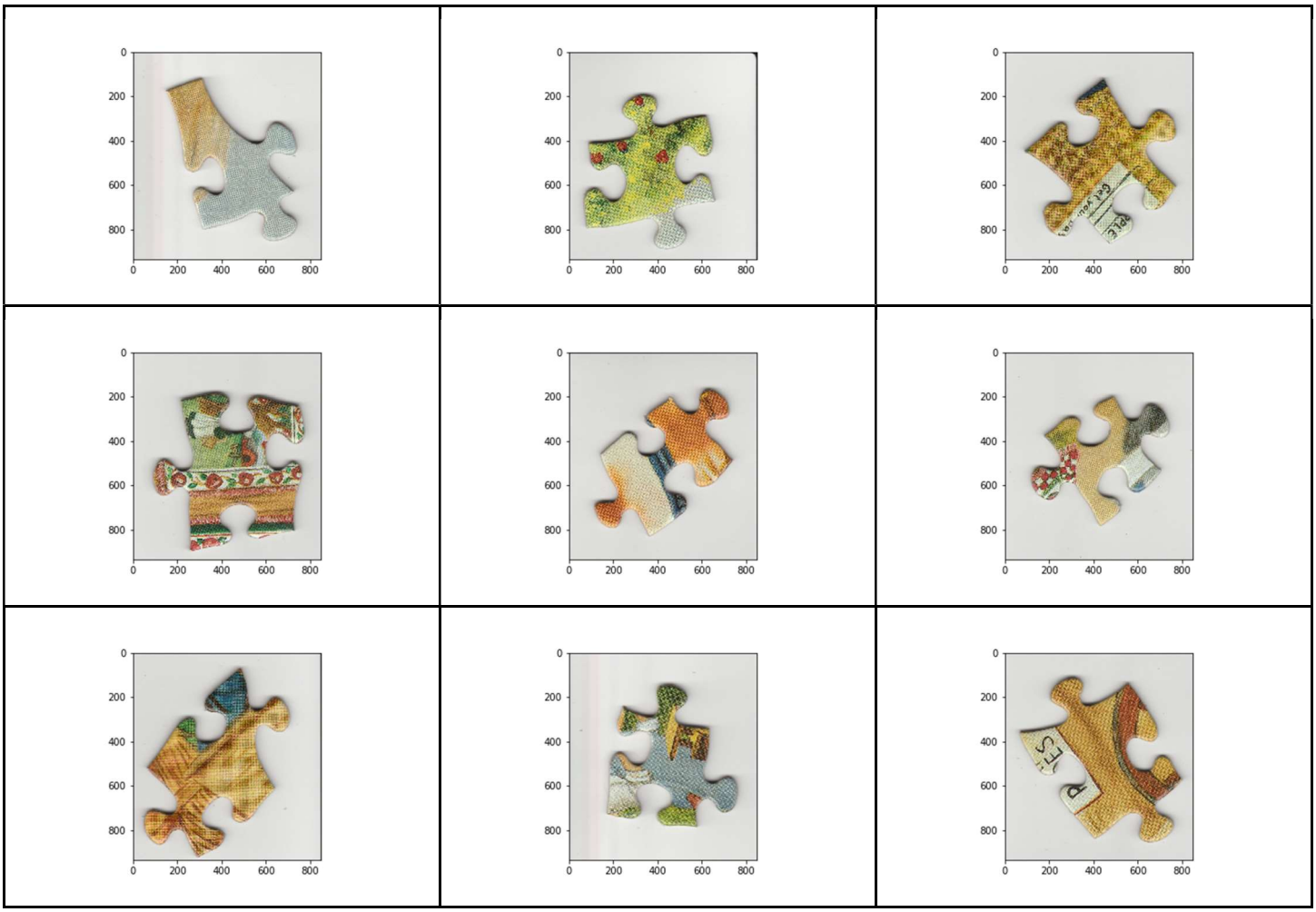
Appendix

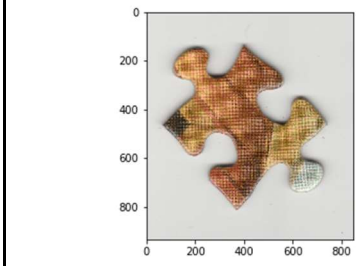
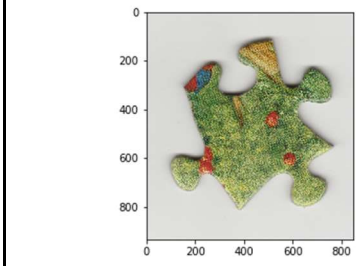
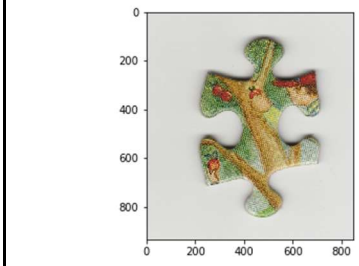
Table 1

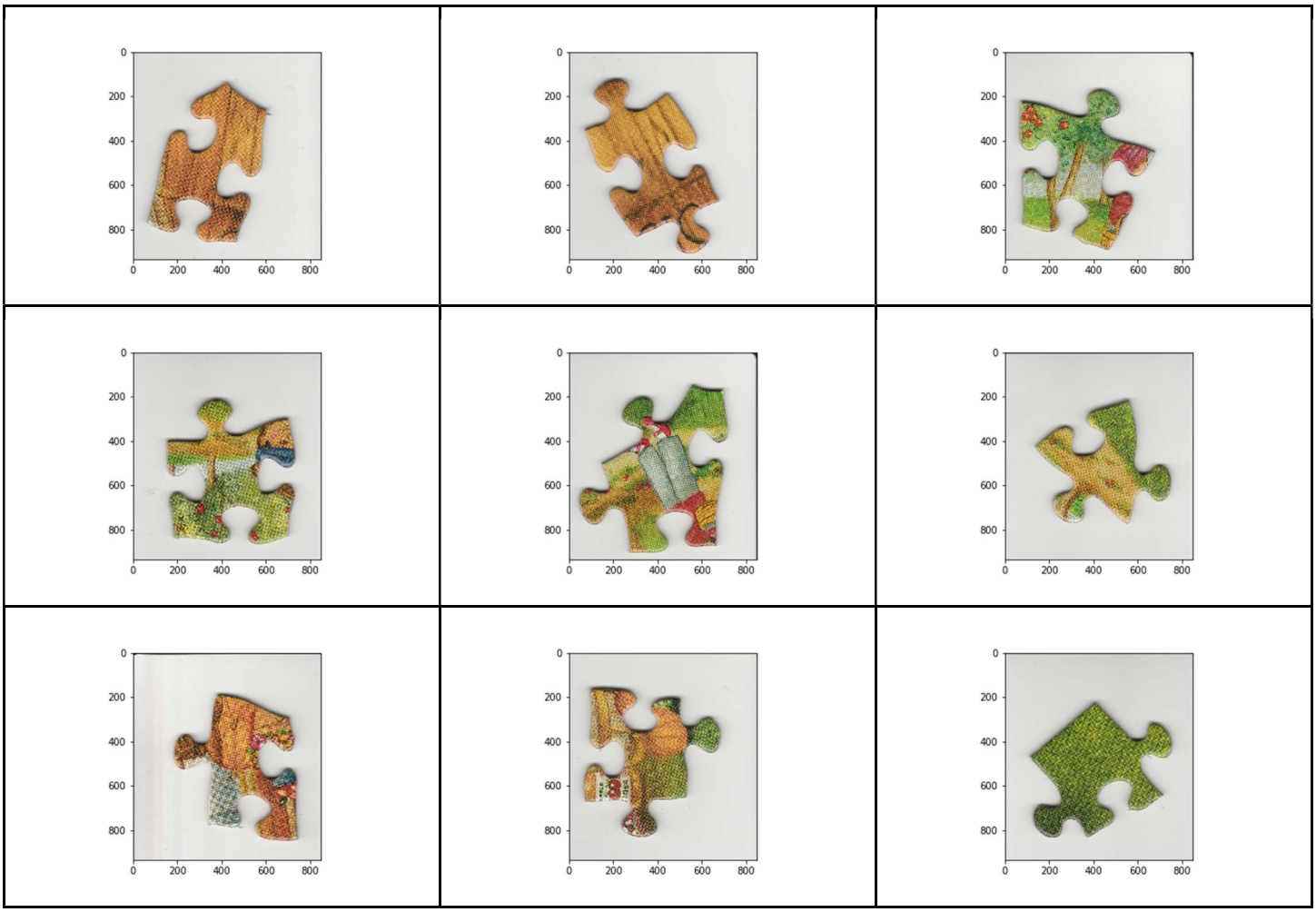
model_name	modularity_score	conductance_score
isoKM_4	0.1532	0.7264
isoKM_13	0.1512	0.7253
isoKM_12	0.1463	0.7315
isoKM_8	0.1435	0.7352
isoKM_5	0.1422	0.7362
isoKM_9	0.1416	0.7362
isoKM_6	0.1410	0.7396
isoKM_14	0.1406	0.7344
isoKM_11	0.1401	0.7357
isoKM_10	0.1355	0.7395
isoKM_3	0.1316	0.7508
isoKM_7	0.1283	0.7508
KMeans	0.1270	0.7671
isoKM_2	0.1267	0.7597
pcaKM_9	0.1198	0.7555
pcaKM_10	0.1196	0.7779
pcaKM_13	0.1120	0.7599
pcaKM_5	0.1103	0.7809
pcaKM_12	0.1036	0.7725
pcaKM_14	0.1028	0.7859
pcaKM_4	0.1017	0.7747
pcaKM_6	0.1009	0.7827
pcaKM_7	0.0994	0.7838
pcaKM_8	0.0911	0.7823
pcaKM_11	0.0758	0.7894
pcaKM_2	0.0737	0.7933
pcaKM_3	0.0648	0.8035
SpectralClustering	-0.0052	0.5578

Table 2

Images from Puzzle 3, Cluster 7		
		
		
		
		



Images from Puzzle 3, Cluster 8		
		


0 200 400 600 800