Lab 5: Texture Classification

Daniel Sherman (0954083), *Student, University of Guelph*

***Abstract*—Manual feature extraction of 1024 blocks of 16x16 pixels was performed for 10 features. Supervised classification was successfully implemented for 16 different classes. A custom routine to implement k-Means clustering was written, and had little success, as the choice of initial seed was random and did not utilize *kmeans++ Algorithm* that MATLAB utilizes. Using a Euclidean classifier had proven more successful than a Mahalanobis classifier for k-Means clustering as the covariance matrices that were calculated from MATLAB were near singular.**

***Index Terms*—Texture Classification, Feature Extraction, Supervised Classification, Unsupervised Clustering, k-Means, Machine Learning, Mahalanobis Classifier, Euclidean Classifier**

I. Introduction

TEXTURE is a key characteristic in virtually all surfaces, and discerning the differences between two textures can allow a system to distinguish between two objects. Texture classification plays an important role in computer vision applications (1). In the area of Medical Image Processing, differentiating textures in an image can allow computers to determine the boundary between adjacent organs, or be able to detect if a tumor is present in a scan that a radiologist may not be able to detect visually, and can be done relatively efficiently with machine learning.

Within texture classification, two very important, but distinct problems exist: extracting features and classifying them. Extracting proper features can make the difference between a powerful classifier and a weak classifier (1). In a way that is analogous to a carpenter being only as good as their tools, a classification algorithm can only be effective with effective features. For many machine learning algorithms, the choice of optimization criterion can heavily affect the efficacy of the system. Bayesian optimization has proven an incredibly effective optimization method (2). The most common machine learning algorithms are Supervised and Unsupervised learning (3). Supervised learning consists of mapping data to known labels while minimizing error (3). Unsupervised learning algorithms attempts to classify data together without the aid of labels based on similarity (3). Common ways to implement Supervised and Unsupervised learning include k-Nearest Neighbour, Naive Bayes, Decision Trees, Linear Regression, Support Vector, Machines, Neural Networks, and k-Means (4).

Manual feature extraction was performed on a test image in order to implement Supervised and Unsupervised Classification on areas in a test image with a Bayesian classifier.

II. Methods

A test image was obtained and loaded into MATLAB for processing, seen in Figure 1.

Lab Report received April 12, 2020

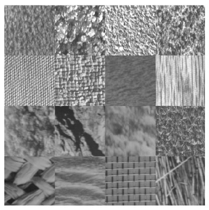

Fig. 1: Image 'brodatz.tif' (512x512 pixels) that was used as a test image

The classes will be referred to by taking the textures to be a 4x4 grid of textures (on Figure 1). The top left texture will be designated as Class 1,1, the top right texture as Class 1,4, the bottom right as Class 4,4, and so on.

A. Feature Extraction

The entire image was subdivided into 16 classes, visually seen in Figure 1 (each 128x128 pixels). 3 classes were selected for further investigation, seen in Figure 2, Figure 3, and Figure 4. They were selected as they visually look distinct from one another.

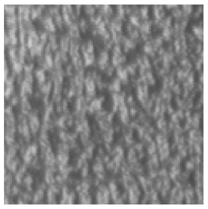

Fig. 2: Class 1,1 (128x128 pixels)

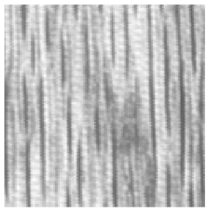

Fig. 3: Class 2,4 (128x128 pixels)

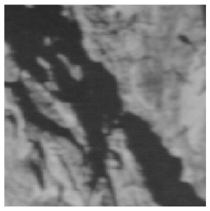

Fig. 4: Class 3,2 (128x128 pixels)

Each of the 16 classes in the total image were further subdivided into 16 blocks (each 16x16 pixels, totaling 64 blocks per class, and 1024 blocks in the entire image).

In each block, 10 features were extracted: mean grey level (μ), standard deviation of the grey level (σ), the mean magnitude of high frequency components at varying angles ($\bar{\Omega}_{high,\theta}$, for $\theta = 0°$, $45°$, $90°$, and $135°$), and the square root of the Auto-correlation function for $\tau = 1, 4$ for both the horizontal and vertical directions ($\sqrt{R_{xx,h}(\tau)}$, $\sqrt{R_{xx,v}(\tau)}$, respectively).

For each class (m, n), the Covariance Matrix ($\Sigma_{m,n}$), its Eigenvalues ($\lambda_{m,n}$), and the Mean Vector of each class's features ($\mu_{m,n}$) were calculated.

B. Supervised Classification

The Bayes classifier was utilized to implement a supervised classification algorithm for 2 cases: with all 16 classes, and with the 3 classes for the features outlined in subsection II-A.

To do so, the Mahalanobis distance was calculated for each block in 10-Dimensional space (for the 10 features). For each block, the Mahalanobis distance from the block's features to each class's average was calculated, for 16 distances per block in the case of all the classes, and 3 distances per block in the case where only 3 classes were investigated.

For each block, the minimum Mahalanobis distance was determined and the class average that produced the minimum distance was taken to be the classified class of said block.

Results from supervised classification were presented in confusion matrices.

C. Unsupervised Clustering

A k-Means algorithm was applied to assign all of the 1024 blocks to $k = 3, 8, 16$ classes by randomly seeding the initial class guesses. Euclidean and Mahalanobis distance classifiers were utilized.

Results were presented in silhouettes to compare MATLAB's `kmeans()` function to the custom one that was written, the effect of changing the number of classes, and to compare the Euclidean and Mahalanobis classifiers. Code used in this lab can be seen in section D.

III. Results and Discussion

A. Feature Extraction

The elements in each feature vector can be seen in Equation 1.

$$\begin{bmatrix} \mu, \sigma, \bar{\Omega}_{high,0°}, \bar{\Omega}_{high,45°}, \bar{\Omega}_{high,90°}, \bar{\Omega}_{high,135°}, \cdots \\ \sqrt{R_{xx,h}(1)}, \sqrt{R_{xx,v}(4)}, \sqrt{R_{xx,h}(1)}, \sqrt{R_{xx,v}(4)} \end{bmatrix}^T \quad (1)$$

The covariance matrix for the features in Class 1,1 can be seen in Equation 2. All the elements are positive, which indicates that all the features in Class 1,1 are positively proportional to each other. It should be noted that the auto-correlations are all heavily related to each other, as the elements that relate the horizontal and vertical auto-correlations for 1 and 4 all have order of magnitude 5 (Equation 2). This is to be expected, because they are all calculated the same way, and should change relatively similar to each other. Another interesting to note is the relatively weak correlation between the mean and the standard deviation of the grey level (Equation 2). This is interesting to note because the standard deviation uses the mean to calculate that value.

The square root of the Eigenvalues of Equation 2 are seen in Equation 3. As MATLAB's `eig()` function returns them in ascending order, nothing can be stated on what Eigenvalue correlates to what feature (5). However, it should be noted that the spread of the Eigenvalues is 5 orders of magnitude, from 10^{-2} to 10^3.

The average feature vector for Class 1,1 is seen in Equation 4.

The covariance matrix for the feature in Class 2,4 can be seen in Equation 5. Not all elements are positively correlated with each other. μ is positively correlated with $\bar{\Omega}_{high,90°}$, and strongly (on the order of 10^3) with all the auto-correlations. Its negative correlations are relatively weak (on the order of 1 to 10^2). σ is weakly positively correlated with $\bar{\Omega}_{high,45°}$, $\bar{\Omega}_{high,90°}$, and $\bar{\Omega}_{high,135°}$, and weakly negatively correlated with the rest of the features. $\bar{\Omega}_{high,0°}$ is weakly positively correlated with the rest of the mean high frequencies at varying angles (on the order of 1 to 10^2), weakly negatively correlated with μ and σ (on the order of 1 to 10^1), and strongly negatively correlated with the auto-correlations (on the order of 10^3). $\bar{\Omega}_{high,45°}$ is weakly positively correlated with the rest of the mean high frequencies at varying angles and σ (on the order of 1 to 10^2), and weakly negatively correlated with the other features (on the order of 1 to 10^2). $\bar{\Omega}_{high,90°}$ is correlated weakly positively with all other features (on the order of 1 to 10^2). $\bar{\Omega}_{high,135°}$ is weakly positively correlated with the other mean high frequency components at varying angles and σ (on the order of 1 to 10^2), and weakly negatively correlated with the rest of the features (on the order of 1 to 10^2). The auto-correlations share the same correlations. They are all strongly positively correlated with each other (on the order of 10^5), weakly positively correlated with $\bar{\Omega}_{high,90°}$ (on the order of 10^2), moderately negatively correlated with the rest of the mean high frequencies at varying angles (on the order of 10^2 to 10^3), strongly positively correlated with μ (10^3), and weakly negatively correlated with σ (on the order of 10^1 to 10^2).

The square root of the Eigenvalues of Equation 5 are seen in Equation 6, in ascending order (5). No conclusions can be made about specific features due to `eig()`'s function to reorder the Eigenvalues. They spread 5 orders of magnitude, the same ones as Class 1,1, 10^{-2} to 10^3.

The average feature vector for Class 2,4 is seen in Equation 7.

The covariance matrix for Class 3,2 can be seen in Figure 4. Certain elements were positively correlated, but not all. μ is strongly positively correlated with the auto-correlations (on the order of 10^5), but weakly negatively correlated with all other features (on the order of 10^1 to 10^2). σ is weakly positively correlated with the auto-correlations (on the order of 10^1 to 10^2). $\bar{\Omega}_{high,\theta}$ for $0°$, $45°$, $90°$, and $135°$ are weakly positively correlated with the other angular components (on the order of 10^2 to 10^3) and with σ (10^1 to 10^2). They are strongly negatively correlated with the auto-correlations (on the order of 10^3), and weakly negatively with μ (on the order of 10^2). The auto-correlations are all strongly positively correlated with themselves and μ (on the order of 10^5) and moderately negatively correlated with σ and the mean high angular frequencies (on the order of 10^2 to 10^3).

The square root of the Eigenvalues of Equation 8 are seen in Equation 9. They are in ascending order, so no comments can be made about the specific Eigenvalue of a specific feature (5). The Eigenvalues span 4 orders of magnitude, from 10^{-1} to 10^3.

The average feature vector for Class 3,2 is seen in Equation 10.

B. Supervised Classification

The confusion matrices for the classification of 3 classes and 16 classes are seen in Figure 5 and Figure 6. The number of correctly sorted blocks into classes was 91.67%, three times better than the standard random 33.33%. This is due to hand picking classes that were as visually diverse as possible. Just looking at the grey levels, Figure 3 looks to be the brightest, Figure 2 looks to be darker than that, and Figure 4 (especially the line through the diagonal) appears to be the darkest. Furthermore, the textures of the three classes appear to go in different directions, where Figure 3 has textures aligned vertically, Figure 4 has texture diagonally, and Figure 2 has texture almost stippled. All these visual features, and more help distinguish each class.

When classifying all 16 classes, the number of correctly sorted blocks was 56.84%, slightly less than an order of magnitude better than the 6.25% sorting them randomly. That being said, some classes were sorted much better than others. Looking at the columns of Figure 6, the algorithm had a tough time differentiating Class 1,2 (2 on Figure 6), and for Class 3,2 (10 on Figure 6).

Specifically, the algorithm confused Class 1,2 (2 on Figure 6) with Classes 2,2 and 3,4 (4 and 12 on the Figure 6, respectively). This could be due to the fact that all 3 of the aforementioned classes look relatively stippled in their own right. The three classes look more similar to each other than any of the other classes, visually.

Furthermore, the algorithm confused Class 3,2 (10 on Figure 6) with Class 3,2 (7 on Figure 6), Class 3,1 (9 on Figure 6), Class 3,3 (11 on Figure 6), Class 4,1 (13 on Figure 6), and Class 4,2 (14 on Figure 6). This could be because in the true class, Class 3,2 has a few textures in the subdivision itself. There are dark spots and lighter spots, and that streak going through the diagonal. All the classes that the algorithm confused with Class 3,2 have streaks of dark grey levels in them, which could have caused the sorting confusion.

The Mahalanobis distance classifier is useful because it accounts for the correlation between features, but using the covariance matrix can cause computational challenges for many features and large data sets (6). For this investigation, 10 features and 1024 data points is not enough to make the additional computation time a limitation in using the Mahalanobis distance.

C. Unsupervised Clustering

Depending on the method, the Unsupervised and Supervised results differ. Segregating the data into 16 classes, yields 16 clusters of approximately the same size when using MATLAB's `kmeans()` (Figure 9). This particular case of Unsupervised Clustering utilized a Euclidean classifier rather than the Mahalanobis one used for Supervised Classification. The other two cases using MATLAB's `kmeans()`, 8 clusters and 3 clusters, yield similar results, an appropriate number of classes, of approximately the same size (Figure 8, Figure 7). In all cases, the silhouette numbers were fairly high, indicating that the algorithm sorted most of the blocs properly.

The results when using a custom routine are not as good. For 3 clusters and a Euclidean classifier, most of the blocks appear to be sorted correctly in classes 1 and 3, but class 2 appears to be sorted incorrectly (Figure 10). That silhouette appears to have classes of approximately the same size. The positives mentioned above cannot be said when sorting into 8 and 16 classes with a Euclidean classifier however. It appears that many of the blocks were sorted incorrectly, and the classes are not approximately the same size. Furthermore, some classes appear to be so small, they do not appear on the y-axis (Figure 11, Figure 12).

The large difference in the results from MATLAB versus the custom routine appears to be the choice of an initial seed. `kmeans()` uses the *kmeans++ Algorithm*, which determines the best centroids by maximizing the minimum distance between clusters (7). This algorithm is much more robust and effective than randomly seeding initial clusters, as was done in the code from section D.

Utilizing a Mahalanobis distance classifier, the results are not good at all. Nearly all of the blocks have been classified incorrectly. This can be seen on the silhouettes (Figure 13, Figure 14, Figure 15). This is due to the added complexity of utilizing the covariance matrix when calculating the Mahalanobis distance classifier. The covariance matrix cannot always be obtained accurately, reducing the efficacy of using the Mahalanobis distance as an optimization criterion (6). When executing the code to run the program, the covariance matrices that MATLAB calculates are near singular, which drastically reduces the accuracy of the results. Furthermore, the number of features that were selected was small, and incorporating more features, or automatically extracting many could have improved the results of Unsupervised Clustering in general.

IV. Conclusion

For a test image, manual feature extraction was performed for 1024 blocks of 16x16 pixels, for 10 different features. The mean grey level, grey level standard deviation, mean magnitude of high frequency components at varying angles, and horizontal and vertical auto-correlations for two different τ values were taken as features. Supervised classification based on the minimum Mahalanobis distance was performed for 3 known classes and for 16 known classes. The success rate was double the performance of randomly assigning classes for 3 classes, and an order of magnitude better for assigning all 16.

Unsupervised clustering did not have much success due to poorly choosing an initial seed. Utilizing a Mahalanobis classifier instead of a Euclidean classifier did not help, as the covariance matrices that were calculated were near singular, and the number of features selected was small.

APPENDIX A
SUPERVISED CLASSIFICATION RESULTS VISUALIZED

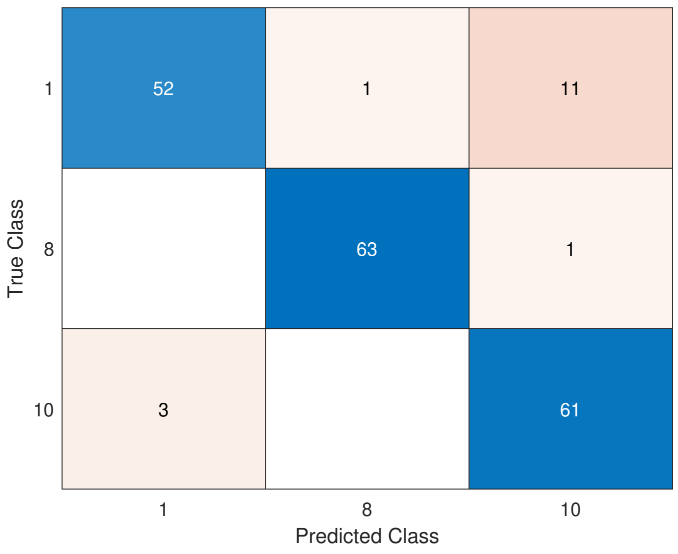

Fig. 5: Confusion Matrix for Supervised Classification of 3 classes. It should be noted that Class 1 on the figure is the same class as Class 1,1, Class 8 on the Figure is the same as Class 2,4, and Class 10 on the figure is the same as Class 3,2

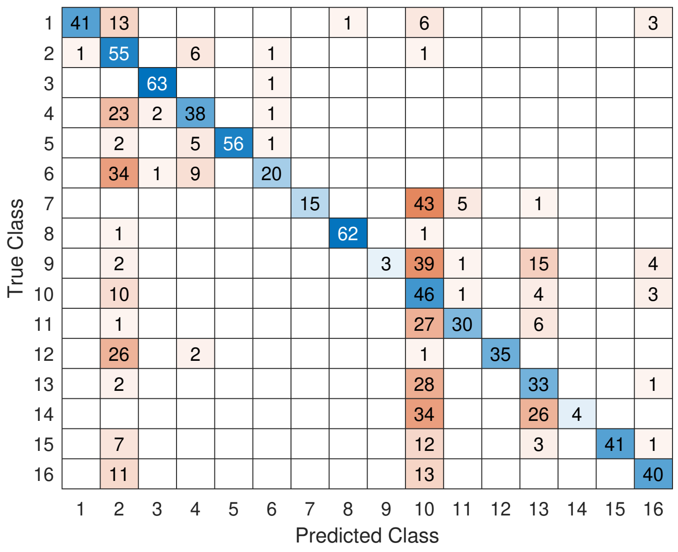

Fig. 6: Confusion Matrix for Supervised Classification of all 16 classes. It should be noted that the class labels on the figure correspond with the nomenclature used throughout the report. Class 1 on the figure is Class 1,1, Class 4 is Class 1,4, Class 5 is Class 2,1, and Class 16 is Class 4,4. The rest of the classes follow the same naming conventions.

APPENDIX B
UNSUPERVISED CLUSTERING RESULTS VISUALIZED

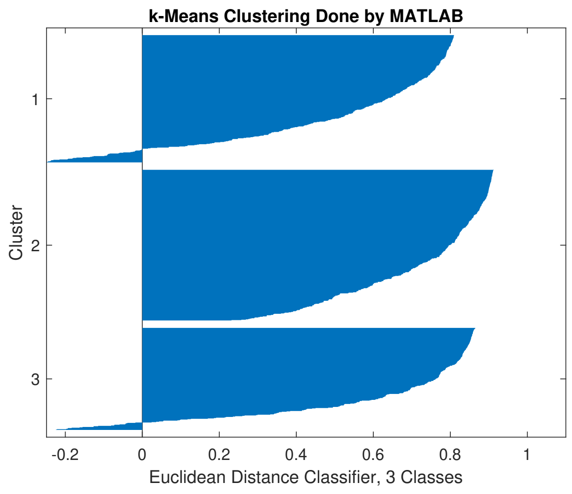

Fig. 7: k-Means Clustering Results using MATLAB's `kmeans()` function with a Euclidean distance classifier, sorting data into 3 clusters

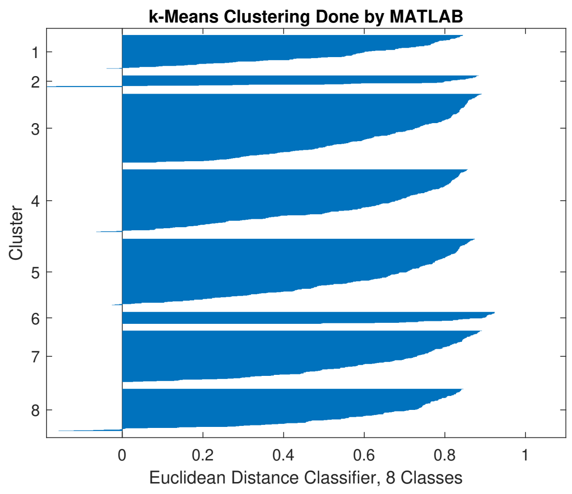

Fig. 8: k-Means Clustering Results using MATLAB's `kmeans()` function with a Euclidean distance classifier, sorting data into 8 clusters

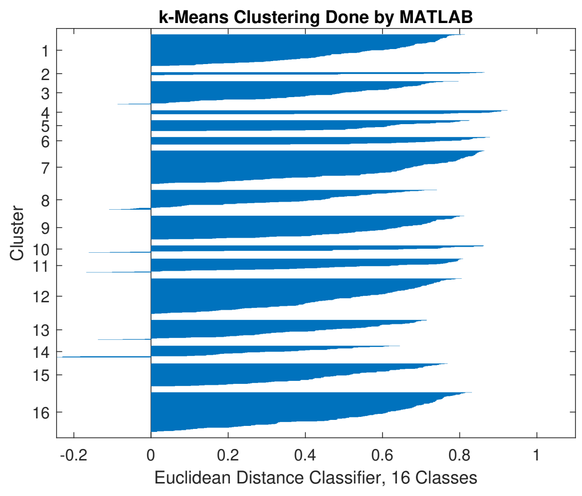

Fig. 9: k-Means Clustering Results using MATLAB's `kmeans()` function with a Euclidean distance classifier, sorting data into 16 clusters

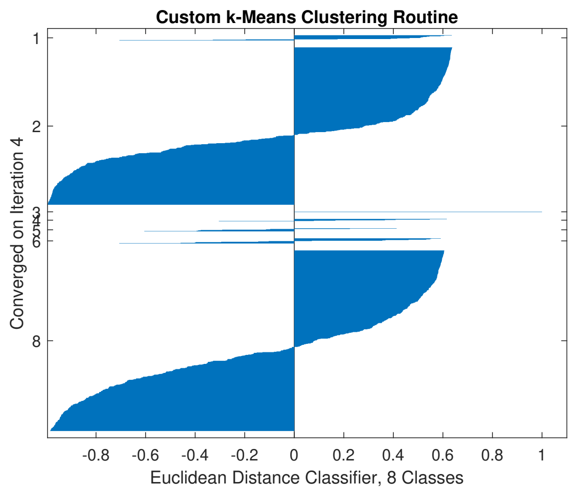

Fig. 11: k-Means Clustering Results using a custom clustering routine with a Euclidean distance classifier, sorting data into 8 clusters

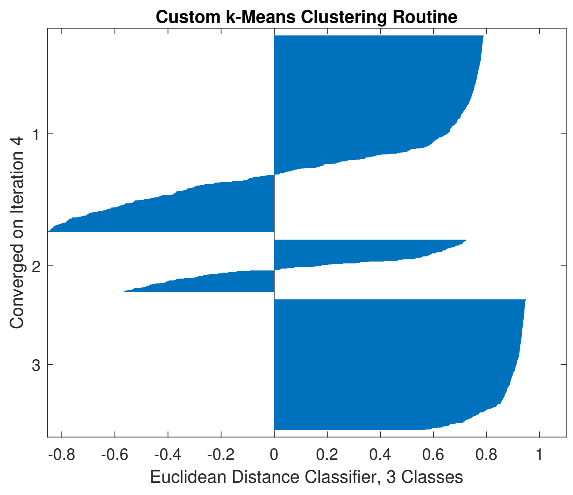

Fig. 10: k-Means Clustering Results using a custom clustering routine with a Euclidean distance classifier, sorting data into 3 clusters

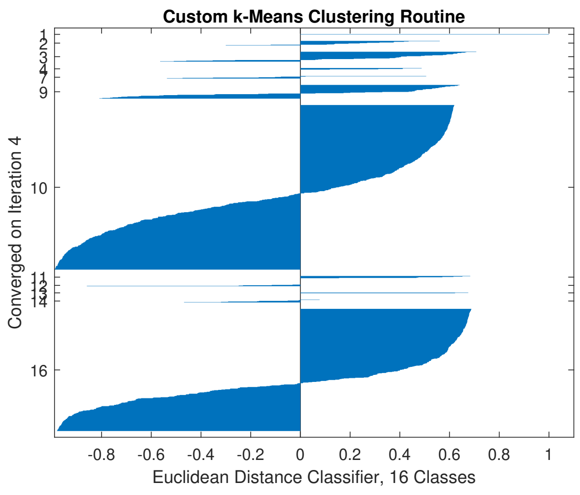

Fig. 12: k-Means Clustering Results using a custom clustering routine with a Euclidean distance classifier, sorting data into 16 clusters

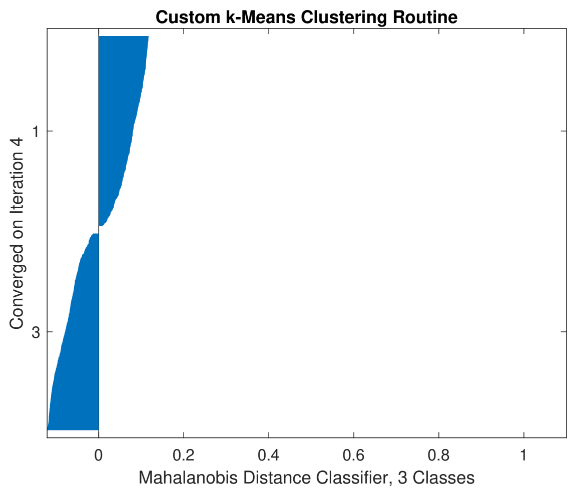

Fig. 13: k-Means Clustering Results using a custom clustering routine with a Mahalanobis distance classifier, sorting data into 3 clusters

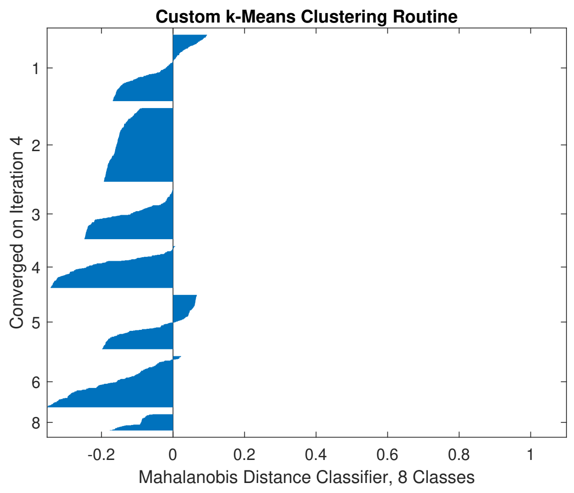

Fig. 14: k-Means Clustering Results using a custom clustering routine with a Mahalanobis distance classifier, sorting data into 8 clusters

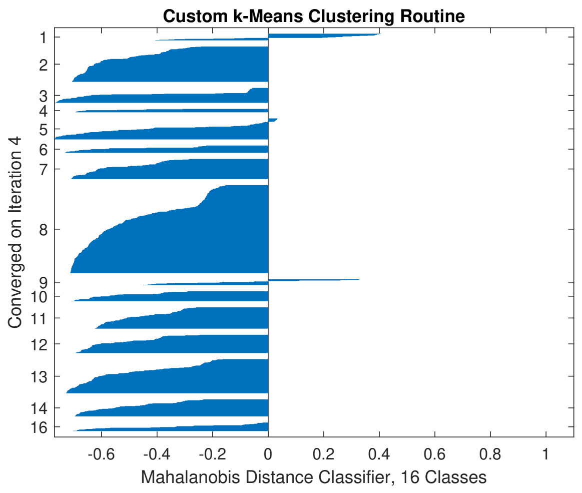

Fig. 15: k-Means Clustering Results using a custom clustering routine with a Mahalanobis distance classifier, sorting data into 16 clusters

APPENDIX C
COVARIANCE MATRICES, MEAN FEATURES, AND EIGENVALUES FOR SELECTED CLASSES

A. *Class 1,1*

$$\Sigma_{1,1} = \begin{bmatrix} 69.13311 & 8.384621 & 49.11402 & 23.75174 & 37.15791 & 12.59118 & 1112.881 & 1106.502 & 1103.32 & 1093.649 \\ 8.384621 & 6.766309 & 9.481823 & 1.129277 & 18.12431 & 1.72962 & 149.7756 & 148.2828 & 142.6018 & 132.7364 \\ 49.11402 & 9.481823 & 562.3233 & 73.91853 & 118.7475 & 43.9807 & 791.9942 & 777.9501 & 781.6874 & 779.6531 \\ 23.75174 & 1.129277 & 73.91853 & 104.2051 & 129.6154 & 42.9862 & 372.7999 & 373.7986 & 367.1869 & 378.1211 \\ 37.15791 & 18.12431 & 118.7475 & 129.6154 & 723.0363 & 54.55073 & 598.8605 & 615.0803 & 545.9493 & 575.2993 \\ 12.59118 & 1.72962 & 43.9807 & 42.9862 & 54.55073 & 41.15879 & 199.4305 & 198.9153 & 191.8816 & 198.5499 \\ 1112.881 & 149.7756 & 791.9942 & 372.7999 & 598.8605 & 199.4305 & 17956.31 & 17850.51 & 17792.41 & 17607.88 \\ 1106.502 & 148.2828 & 777.9501 & 373.7986 & 615.0803 & 198.9153 & 17850.51 & 17748.08 & 17686.29 & 17507.72 \\ 1103.32 & 142.6018 & 781.6874 & 367.1869 & 545.9493 & 191.8816 & 17792.41 & 17686.29 & 17657.21 & 17460.6 \\ 1093.649 & 132.7364 & 779.6531 & 378.1211 & 575.2993 & 198.5499 & 17607.88 & 17507.72 & 17460.6 & 17323.86 \end{bmatrix} \tag{2}$$

$$\sqrt{\lambda_{1,1}} = \left[0.0563, 0.8790, 1.2974, 4.6359, 6.1846, 9.3693, 22.1205, 27.8721, 266.0116\right]^T \tag{3}$$

$$\mu_{1,1} = \left[125.7123, 21.2417, 125.7232, 48.5885, 104.3650, 34.9172, 2027.6566, 2002.6416, 1919.4390\right] \tag{4}$$

B. *Class 2,4*

$$\Sigma_{2,4} = \begin{bmatrix} 238.1881 & -7.30474 & -75.1223 & -12.6541 & 49.37159 & -12.5305 & 3741.441 & 3766.92 & 3716.899 & 3757.691 \\ -7.30474 & 22.09662 & -9.76512 & 3.010355 & 19.52645 & 2.004953 & -63.6654 & -82.3631 & -60.8388 & -130.685 \\ -75.1223 & -9.76512 & 873.4686 & 179.0799 & 3.469443 & 153.1442 & -1221.66 & -1197.7 & -1229.94 & -1183.01 \\ -12.6541 & 3.010355 & 179.0799 & 163.7394 & 21.74406 & 97.57479 & -191.685 & -190.919 & -189.719 & -191.827 \\ 49.37159 & 19.52645 & 3.469443 & 21.74406 & 523.7944 & 17.25086 & 818.2491 & 821.7315 & 799.6917 & 767.1951 \\ -12.5305 & 2.004953 & 153.1442 & 97.57479 & 17.25086 & 109.3965 & -190.962 & -187.41 & -181.592 & -192.833 \\ 3741.441 & -63.6654 & -1221.66 & -191.685 & 818.2491 & -190.962 & 58894.88 & 59250.85 & 58521.16 & 58994.21 \\ 3766.92 & -82.3631 & -1197.7 & -190.919 & 821.7315 & -187.41 & 59250.85 & 59636.76 & 58871.22 & 59409.87 \\ 3716.899 & -60.8388 & -1229.94 & -189.719 & 799.6917 & -181.592 & 58521.16 & 58871.22 & 58165.22 & 58610.9 \\ 3757.691 & -130.685 & -1183.01 & -191.827 & 767.1951 & -192.833 & 58994.21 & 59409.87 & 58610.9 & 59385.42 \end{bmatrix} \tag{5}$$

$$\sqrt{\lambda_{2,4}} = \left[0.0839, 1.6751, 2.5823, 4.0161, 5.9665, 12.4967, 14.6735, 22.8134, 30.4939, 485.9320\right]^T \tag{6}$$

$$\mu_{2,4} = \begin{bmatrix} 185.3395 & 27.72377 & 104.7308 & 55.75869 & 126.9712 & 42.18558 & 2990.656 & 2980.839 & 2968.507 & 2935.71 \end{bmatrix} \tag{7}$$

C. *Class 3,2*

$$\Sigma_{3,2} = \begin{bmatrix} 1226.233 & -21.3567 & -313.118 & -147.013 & -486.246 & -114.578 & 19215.35 & 19222.07 & 19375.78 & 19401.2 \\ -21.3567 & 101.5167 & 221.4926 & 51.60247 & 427.3835 & 45.22927 & -21.4588 & -39.4 & -206.225 & -293.026 \\ -313.118 & 221.4926 & 1767.755 & 418.9627 & 1314.956 & 347.9326 & -4319.37 & -4292.44 & -4955.09 & -4803.39 \\ -147.013 & 51.60247 & 418.9627 & 184.373 & 484.4861 & 150.9582 & -2202.55 & -2183.66 & -2364.89 & -2304.84 \\ -486.246 & 427.3835 & 1314.956 & 484.4861 & 3865.491 & 442.8221 & -6512.58 & -6430.46 & -7522.86 & -7314.66 \\ -114.578 & 45.22927 & 347.9326 & 150.9582 & 442.8221 & 131.6604 & -1706.77 & -1689.46 & -1851.57 & -1790.73 \\ 19215.35 & -21.4588 & -4319.37 & -2202.55 & -6512.58 & -1706.77 & 302178.2 & 302197.8 & 304136.9 & 304169.1 \\ 19222.07 & -39.4 & -4292.44 & -2183.66 & -6430.46 & -1689.46 & 302197.8 & 302254.7 & 304134.5 & 304304.6 \\ 19375.78 & -206.225 & -4955.09 & -2364.89 & -7522.86 & -1851.57 & 304136.9 & 304134.5 & 306616.3 & 306530.8 \\ 19401.2 & -293.026 & -4803.39 & -2304.84 & -7314.66 & -1790.73 & 304169.1 & 304304.6 & 306530.8 & 307120.5 \end{bmatrix} \tag{8}$$

$$\sqrt{\lambda_{3,2}} = \left[0.3525, 1.6453, 2.1586, 2.4897, 9.7983, 11.9872, 22.2826, 34.3193, 68.8667, 1.1040 \times 10^3\right]^T \tag{9}$$

$$\mu_{3,2} = \begin{bmatrix}109.5576 & 22.27273 & 80.60046 & 27.64012 & 126.1657 & 24.03664 & 1788.943 & 1787.506 & 1753.117 & 1744.855\end{bmatrix} \tag{10}$$

APPENDIX D
MATLAB CODE

Listing 1: Main code used in the lab

```
%% ENGG 4660: MEDICAL IMAGE PROCESSING
% LAB 5: TEXTURE CLASSIFICATION
% DANIEL SHERMAN
% 0954083
% MARCH 23, 2020

%% START OF CODE

close all
clear all
clc

%% LOAD IN FILES

textures = imread('brodatz.tif');
[row, col] = size(textures);

figure()
imshow(textures)

%% SUBDIVIDE INTO CLASSES

for m = 0:3
    for n = 0:3
        eval(strcat(['class', num2str(m + 1), num2str(n + 1), ...
            ' = textures(1 + m*128: 128*(m + 1), 1 + n*128: 128*(n + 1));']));
    end
end

%% SUBDIVIDE A CLASS INTO 16x16 BLOCKS (64 TOTAL PER CLASS)

for m = 0:3
    for n = 0:3
        eval(strcat(['bloc', num2str(1 + m), num2str(1 + n), ...
            ' = double(subdivide_block(class', num2str(1 + m), num2str(1 + n), '));']));
    end
end

%% EXTRACT FEATURE VECTOR FOR EACH BLOCK, AVERAGE FEATURE FOR EACH CLASS,
% COVARIANCE MATRIX FOR EACH CLASS, AND EIGENVALUES FOR EACH CLASS

for m = 0:3
    for n = 0:3
        %find features for each block (each block's features are a row)
        eval(strcat(['bloc_features', num2str(m + 1), num2str(n + 1), ...
            ' = feature_extraction(bloc', num2str(m + 1), num2str(n + 1), ');']));
        %calculate the average value for each feature in a class
        eval(strcat(['average_feature', num2str(m + 1), num2str(n + 1), ...
            ' = mean(bloc_features', num2str(m + 1), num2str(n + 1), ');']));
        %calculate covariance matrix for each class
        eval(strcat(['cov_mat', num2str(m + 1), num2str(n + 1), ...
            ' = cov(bloc_features', num2str(m + 1), num2str(n + 1), ');']));
        %calculate the sqrare root of eignevalues for each covariance
        %matrix, denoting spread
        eval(strcat(['eigen_cov', num2str(m + 1), num2str(n + 1), ...
            ' = sqrt(eig(cov_mat', num2str(m + 1), num2str(n + 1), '));']));
    end
end


%organize average feature value, covariance matricies, and eigenvalues to
%store the data all together

subdiv = 1;

for m = 0:3
    for n = 0:3
        eval(strcat(['all_average(', num2str(subdiv), ...
            ',:) = transpose(average_feature', num2str(m + 1), num2str(n + 1), ');']));
        eval(strcat(['all_cov(:,:,', num2str(subdiv), ...
            ') = cov_mat', num2str(m + 1), num2str(n + 1), ';']));
        eval(strcat(['all_eig(', num2str(subdiv), ...
            ',:) = transpose(eigen_cov', num2str(m + 1), num2str(n + 1), ');']));
        subdiv = subdiv + 1;
    end
end

% 'all_average' is a 16x10 vectors containing the average value of each feature on the column,
% and on the row is each class (row 1 is top left,
% row 4 is top right, ... row 16 is bottom right class)

% 'all_cov' is the covariance matrix for each class, 10x10 for
% each feature, and 16 matricies for each class
% (matrix 1 is top left, matrix 4 is top right ... matrix 16 is bottom right class)

% 'all_eig' is a 16x10 vectors containing the eigenvalues
% for each of the covariance matrix for each class
% row 1 correlates with covariance matrix 1 (top left class),
% row 4 correlates with covariance matrix 4 (top right matrix),
% row 16 correlates with covariance matrix 16 (bottom right class)

%% CALCULATE THE MAHALANOBIS DISTANCE FOR EVERY BLOCK, FOR EVERY AVERAGE FEATURE VECTOR

mahal_distances = [];

for m = 0:3
    for n = 0:3
        eval(strcat(['mahal_holder = find_mahal_dist(bloc_features', ...
            num2str(m + 1), num2str(n + 1), ', all_average, all_cov);']));
        mahal_distances = [mahal_distances ; mahal_holder];
    end
end

[conf_mat, corr_percent] = check_min_mahal_dist(mahal_distances);
[conf_mat3, corr_percent3] = check_min_mahal_dist_3_classes(mahal_distances, 1, 10, 8);

%% MAKE LARGE MATRIX OF ALL BLOCK FEATURES

all_features = [];

for m = 0:3
    for n = 0:3
        eval(strcat(['all_features = [all_features ; bloc_features', ...
            num2str(m + 1), num2str(n + 1), '];']));
    end
end

%% USING MATLAB FUNCTIONS

id3 = kmeans(all_features, 3);
figure()
silhouette(all_features, id3);
```

```
title('k-Means Clustering Done by MATLAB')
xlabel(strcat(['Euclidean Distance Classifier, 3 Classes']))


id8 = kmeans(all_features, 8);
figure()
silhouette(all_features, id8);
title('k-Means Clustering Done by MATLAB')
xlabel(strcat(['Euclidean Distance Classifier, 8 Classes']))

id16 = kmeans(all_features, 16);
figure()
silhouette(all_features, id16);
title('k-Means Clustering Done by MATLAB')
xlabel(strcat(['Euclidean Distance Classifier, 16 Classes']))

%% USING MY_KMEANS

[id_my3] = my_kmeans(all_features, 3, 'Euclidean');
[id_my8] = my_kmeans(all_features, 8, 'Euclidean');
[id_my16] = my_kmeans(all_features, 16, 'Euclidean');

[id_my3ma] = my_kmeans(all_features, 3, 'Mahalanobis');
[id_my8ma] = my_kmeans(all_features, 8, 'Mahalanobis');
[id_my16ma] = my_kmeans(all_features, 16, 'Mahalanobis');
```

Listing 2: Code used to divide a class into 64 blocks of size 16x16 pixels

```
function bloc = subdivide_block(class)
%% DOCUMENTATION

% FUNCTION FOR IMPLEMENTATION FOR LAB 5
% FUNCTION ACCEPTS A CLASS (SIZE 128x128) AND SUBDIVIDES IT INTO BLOCKS (SIZE 16x16)
% FUNCTION OUTPUTS A 16x16x64 ARRAY OR MATRICIES CONTAINING THE PIXEL
% VALUES FOR EACH BLOCK (BLOCK 1 BEING THE TOP LEFT AND 64 BEING BOTTOM RIGHT)

% MADE BY: DANIEL SHERMAN
% MARCH 23, 2020

%% START OF CODE
iter = 1;

for m = 0:7
    for n = 0:7
        eval(strcat(['bloc(:,:,', num2str(iter), ...
            ') = double(class(1 + m*16: 16*(m + 1), 1 + n*16: 16*(n + 1)));']));
        iter = iter + 1;
    end
end
```

Listing 3: Code used to extract 10 features from a block

```
function bloc_features = feature_extraction(bloc)
%% DOCUMENTATION

% FUNCTION ACCEPTS AN ARRAY OF BLOCKS (SIZE 16x16)
% FOR EACH BLOCK, FUNCTION CALCULATES THE FOLLOWING:
% MEAN, STANDARD DEVIATION,
% MEAN MAGNITUDE OF HIGH FREQUENCY REPRESENTATIONS (THETA =
% 0, 45, 90, 135 BOTH HORIZONTALLY AND VERTICALLY),
% AND SQUARE ROOT OF THE AUTOCORRELATION FOR TAU = 1,4, HORIZONTALLY AND
% VERTICALLY

%% START OF CODE

[row, col, mat] = size(bloc); %size of block
Rxx_h = zeros(mat,4); %initialize horizontal autocorrelation
Rxx_v = zeros(mat,4); %initialize vertical autocorrelation
```

```
for i = 1:mat
    bloc_transpose(:,:,i) = bloc(:,:,i)'; %find transpose of
    %every single matrix in the bloc array
end

bloc_reshape_H = reshape(bloc, 1, row*col, mat);
bloc_reshape_V = reshape(bloc_transpose, 1, row*col, mat);

for k = 1:mat

    %% AVERAGE AND STANDARD DEVIATION

    avg_bloc(k) = mean(bloc(:,:,k), 'all'); %mean grey level

    std_bloc(k) = std(bloc(:,:,k), 1, 'all'); %standard deviation of grey level

    %% MEAN FREQUENCY AT VARYING ANGLES

    mag_bloc(:,:,k) = abs(fftshift(fft2(bloc(:,:,k)))); %frequency domain transform
    mag_0(k) = mean(mag_bloc(5:8, 13:16, k), 'all'); %0 degree average
    mag_45(k) = mean(mag_bloc(1:4, 13:16, k), 'all'); %45 degree average
    mag_90(k) = mean(mag_bloc(1:4, 9:12, k), 'all'); %90 degree average
    mag_135(k) = mean(mag_bloc(1:4, 1:4, k), 'all'); %135 degree average

    %% AUTOCORRELATION

    for tau = 1:4 %iterate for tau vales 1:4
     for m = 1:row*col - tau
          Rxx_h(k, tau) = Rxx_h(k, tau) + bloc_reshape_H(1, m, k)...
              .*bloc_reshape_H(1, m + tau, k); %horizontal autocorrelation
          Rxx_v(k, tau) = Rxx_v(k, tau) + bloc_reshape_V(1, m, k)...
              .*bloc_reshape_V(1, m + tau, k); %vertical autocorrelation
     end
    end

    bloc_features(:,k) = [avg_bloc(k); ... %average pixel value in bloc
     std_bloc(k);... %standard deviation of pixel value in bloc
     mag_0(k);... %mean frequency at 0 degrees
     mag_45(k);... %mean frequency at 45 degrees
     mag_90(k);... %mean frequency at 90 degrees
     mag_135(k);... %mean frequency at 135 degrees
     sqrt(Rxx_h(k, 1));... %horizontal autocorrelation, tau = 1
     sqrt(Rxx_v(k, 1));... %vertical autocorrelation, tau = 1
     sqrt(Rxx_h(k, 4));... %horizontal autocorrelation, tau = 4
     sqrt(Rxx_v(k, 4));].'; %vertical autocorrelation, tau = 4
end

bloc_features = bloc_features.';
```

Listing 4: Code used to find the Mahalanobis distance from a block to the average value of all classes

```
function mahal_dist = find_mahal_dist(bloc_features, mu_vects, cov_mats)
%% DOCUMENTATION

% FUNCTION CALCULATES THE MAHALANOBIS DISTANCE BETWEEN 1 BLOCK'S FEATURE VECTOR AND THE
% AVERAGE FEATURE VECTOR FOR EACH OF 16 CLASSES
% FUNCTION OUTPUTS THE MAHALANOBIS DISTANCE MATRIX OF 16 DISTANCES FOR 64 CLASSES

% MADE BY: DANIEL SHERMAN
% MARCH 27, 2020

%% START OF CODE

[block_num, ~] = size(bloc_features); %get number of blocks
[~, ~, class_num] = size(cov_mats); %get number of classes

bloc_features = bloc_features.'; %block features down the column
mu_vects = mu_vects.';
```

```
for i = 1:block_num
    for j = 1:class_num
        %matrix of squared Mahalanobis distances for each block from each average
        %of each class
        mahal_dist(i, j) = sqrt(transpose((bloc_features(:,i) - ...
            mu_vects(:,j)))*inv(cov_mats(:,:,j))*(bloc_features(:,i) - mu_vects(:,j)));
    end
end
```

Listing 5: Code used to find the minimum Mahalanobis distance for all 16 classes

```
function [conf_mat, corr_percent] = check_min_mahal_dist(mahal_dist)
%% DOCUMENTATION

% FUNCTION ACCPETS A MATRIX OF MAHALANOBIS DISTANCES FOR EVERY BLOCK IN THE IMAGE (1024 BLOCKS)
% TO EVERY CLASS (16 CLASSES)
% FUNCTION CHECKS THE MINIMUM MAHALANOBIS DISTANCE AND COUNTS HOW MANY
% WERE CORRECTLY SORTED IN EACH CLASS
% FUNCTION RETURNS A CONFUSION MATRIX DISPLAYING THE TOTAL NUMBER THAT WERE SORTED CORRECTLY

% MADE BY: DANIEL SHERMAN
% MARCH 27, 2020

%% START OF CODE

%% FIND THE MINIMUM MAHALANOBIS DISTANCE FROM A BLOC TO AN AVERAGE FEATURE IN A CLASS

[blocs, class] = size(mahal_dist);

for i = 1:blocs
    [~, min_index(i)] = min(mahal_dist(i,:));
end

%% BUILD THE CONFUSION MATRIX

conf_mat = zeros(class,class);

for m = 1:class %iterate through the classes
    for n = 1 + (m - 1)*64 : 64 + (m - 1)*64 %iterate in blocks of 64
        switch min_index(n)
            case 1
                conf_mat(m,1) = conf_mat(m,1) + 1;
            case 2
                conf_mat(m,2) = conf_mat(m,2) + 1;
            case 3
                conf_mat(m,3) = conf_mat(m,3) + 1;
            case 4
                conf_mat(m,4) = conf_mat(m,4) + 1;
            case 5
                conf_mat(m,5) = conf_mat(m,5) + 1;
            case 6
                conf_mat(m,6) = conf_mat(m,6) + 1;
            case 7
                conf_mat(m,7) = conf_mat(m,7) + 1;
            case 8
                conf_mat(m,8) = conf_mat(m,8) + 1;
            case 9
                conf_mat(m,9) = conf_mat(m,9) + 1;
            case 10
                conf_mat(m,10) = conf_mat(m,10) + 1;
            case 11
                conf_mat(m,11) = conf_mat(m,11) + 1;
            case 12
                conf_mat(m,12) = conf_mat(m,12) + 1;
            case 13
                conf_mat(m,13) = conf_mat(m,13) + 1;
            case 14
                conf_mat(m,14) = conf_mat(m,14) + 1;
            case 15
```

```
                conf_mat(m,15) = conf_mat(m,15) + 1;
            case 16
                conf_mat(m,16) = conf_mat(m,16) + 1;
            otherwise
                error('NUMBER IN MINIMUM MAHALANOBIS DISTANCES THAT SHOULD NOT BE THERE')
        end
    end
end

%% PLOT CONFUSION CHART AND ASSESS CORRECTNESS OF ASSIGNMENTS

figure()
confusionchart(conf_mat)

correct = 0;

for i = 1:class
    correct = correct + conf_mat(i,i);
end

corr_percent = correct/sum(conf_mat, 'all')
```

Listing 6: Code used to find the minimum Mahalanobis distance for 3 selected classes from a block to the average value for 3 classes

```
function [conf_mat3, corr_percent] = check_min_mahal_dist_3_classes(mahal_dist, c1, c2, c3)
%% DOCUMENTATION

% FUNCTION ACCPETS A MATRIX OF MAHALANOBIS DISTANCES FOR EVERY BLOCK IN THE IMAGE (1024 BLOCKS)
% TO EVERY CLASS (16 CLASSES)
% FUNCTION ALSO ACCEPTS THE CHOICE OF 3 CLASSES
% FUNCTION CHECKS THE MINIMUM MAHALANOBIS DISTANCE AND COUNTS
% HOW MANY WERE CORRECTLY SORTED IN EACH CLASS
% FUNCTION RETURNS A CONFUSION MATRIX DISPLAYING THE TOTAL NUMBER THAT WERE
% SORTED CORRECTLY FOR 3 CLASSES

% MADE BY: DANIEL SHERMAN
% MARCH 30, 2020

%% START OF CODE

%% CHOOSE OUT CLASSES

parse_mahal = [mahal_dist(1:64, c1), mahal_dist(1:64, c2), mahal_dist(1:64, c3); ...
    mahal_dist(65:128, c1), mahal_dist(65:128, c2), mahal_dist(65:128, c3); ...
    mahal_dist(449:512, c1), mahal_dist(449:512, c2), mahal_dist(449:512,c3)];

%% FIND THE MINIMUM MAHALANOBIS DISTANCE FROM A BLOC TO AN AVERAGE FEATURE IN A CLASS

[blocs, class] = size(parse_mahal);

for i = 1:blocs
    [~, min_index(i)] = min(parse_mahal(i,:));
end

%% BUILD THE CONFUSION MATRIX

conf_mat3 = zeros(class, class);

for m = 1:class %iterate through the classes
    for n = 1 + (m - 1)*64 : 64 + (m - 1)*64 %iterate in blocks of 64
        switch min_index(n)
            case 1
                conf_mat3(m,1) = conf_mat3(m,1) + 1;
            case 2
                conf_mat3(m,2) = conf_mat3(m,2) + 1;
            case 3
                conf_mat3(m,3) = conf_mat3(m,3) + 1;
            otherwise
```

```
                error('NUMBER IN MINIMUM MAHALANOBIS DISTANCES THAT SHOULD NOT BE THERE')
            end
        end
    end

%% PLOT CONFUSION CHART AND ASSESS CORRECTNESS OF ASSIGNMENTS

figure()
confusionchart(conf_mat3, [c1, c2, c3])

correct = 0;

for i = 1:class
    correct = correct + conf_mat3(i,i);
end

corr_percent = correct/sum(conf_mat3, 'all')
```

Listing 7: Code used to find the Euclidean distance from a block to the average value of all classes

```
function euclid_dist = find_euclid_dist(all_features, all_average, k)
%% DOCUMENTATION

% FUNCTION CALCULATES THE EUCLIDEAN DISTANCE BETWEEN A
% GIVEN SET OF FEATURE VECTORS AND THE AVERAGE CLASS VECTORS
% FOR USE WITH KMEANS FUNCTIONS

% MADE BY: DANIEL SHERMAN
% MARCH 30, 2020

%% START OF CODE

[block_num, ~] = size(all_features); %get number of blocks
class_num = k; %number of classes

for i = 1:block_num
    for j = 1:class_num
        V = all_features(i,:) - all_average(j,:);
        euclid_dist(i,j) = norm(V*V');
    end
end
```

Listing 8: Code used to implement k-Means clustering

```
function [idx] = my_kmeans(all_features,k, distance)
%% DOCUMENTATION

% TRYING TO EMULATE MATLAB'S KMEANS() FUNCTION
% FUNCTION ACCEPTS A FEATURE MATRIX, NUMBER OF CLASSES K, AND A STRING
% DECLARING WHAT TYPE OF DISTANCE CLASSIFIER IS DESIRED
%(ONLY EUCLIDEAN AND MAHALANOBIS ARE SUPPORTED)

% MADE BY: DANIEL SHERMAN
% MARCH 30,2020

%% START OF CODE

[num_bloc, num_feature] = size(all_features);
%getting size, n is number of blocks, p is number of features

idx_rand = randi(k, num_bloc, 1); %assign initial blocks to random classes

%% SPLIT INTO k CLASSES

feat_aug_class(:,:,1) = [all_features idx_rand];
%augment the feature matrix with the randomly assigned classes

sort_aug_class(:,:,1) = sortrows(feat_aug_class, [11 1:10]);
%sort rows by ascending class column (group classes together)
```

```
%initialize variables to get information on length of classes
class_index = zeros(k + 1, 1);
class_index(1) = 1;
class_index(k + 1) = num_bloc;
iter = 1;

%get class-changing indexes in the feature matrix
for i = 2:num_bloc
    if sort_aug_class(i,11,1) ~= sort_aug_class(i - 1,11,1);
        class_index(iter + 1) = i;
        iter = iter + 1;
    end
end

%% FIND INITIAL AVERAGES

for i = 1:k
    eval(strcat(['mu(', num2str(i), ...
        ', 1:num_feature, 1) = mean(all_features(class_index(', ...
        num2str(i), '):class_index(', num2str(i + 1), ') - 1, 1:num_feature));']));
end

%% CALCULATE EUCLIDEAN DISTANCE FROM BLOCK TO EACH AVERAGE, AND REASSIGN CLASSES

count = 2;

dummy_check = 0;

tic
while dummy_check == 0

    %calculates euclidean distance from each block's feature vector to each
    %average

    switch distance
        case 'Euclidean'
            euclid_dist = find_euclid_dist(all_features(:,1:num_feature), mu(:,:,count - 1), k);
        case 'Mahalanobis'
            for i = 1:k
                cov_mat(:,:,i) = cov(sort_aug_class(class_index(i):class_index(i+1), ...
                    1:num_feature));
            end
            euclid_dist = find_mahal_dist(sort_aug_class(:,1:num_feature), ...
                mu(:,:, count - 1), cov_mat);
        otherwise
            error('That distance type is not supported')
    end



    %determine the minimum distance from a bloc to the class average
    for i = 1:num_bloc
        [~, min_index(i)] = min(euclid_dist(i,:));
    end

    feat_aug_class(:,11, count) = min_index(:).';
    %augment the organized features with the newly calculated classes
    feat_aug_class(:, 1:10, count) = feat_aug_class(:, 1:10, count - 1);
    %carry over features from previous iterations
    sort_aug_class(:,:, count) = sortrows(feat_aug_class(:,:, count), [11 1:10]);
    %sort by new classification

    %get class-changing indexes in the feature matrix
    for i = 2:num_bloc
        if sort_aug_class(i,11, count) ~= sort_aug_class(i - 1,11, count);
            class_index(iter) = i;
            iter = iter + 1;
        end
    end

    for i = 1:k
        eval(strcat(['mu(', num2str(i), ', 1:num_feature,' , ...
            num2str(count), ') = mean(sort_aug_class(class_index(', ...
            num2str(i), '):class_index(', num2str(i + 1), ') - 1, 1:num_feature));']));
    end

    dummy_check = isempty(find(sort_aug_class(:,:,count) - sort_aug_class(:,:, count - 1)));
    count = count + 1

end %while loop
toc

figure()
silhouette(sort_aug_class(:,1:10,count - 1), sort_aug_class(:,11,count - 1))
title('Custom k-Means Clustering Routine')
ylabel(strcat(['Converged on Iteration ', num2str(count - 1)]))
xlabel(strcat([distance, ' Distance Classifier, ', num2str(k), ' Classes']))

idx = sort_aug_class(:,11, count - 1);
```

References

[1] L. Liu, *BRINT: Binary Rotation Invariant and Noise Tolerant Texture Classification*, IEEE Transactions on Image Processing, 2014.

[2] J. Snoek, H. Larochelle, and R.P. Adams, *Practical Bayesian Optimization of Machine Learning Algorithms*, Neural Information Processing Systems Conference, 2012.

[3] F. Chollet, *Deep Learning with Python*, Manning Publications Co., 2018

[4] D. Fumo, *Types of Machine Learning Algorithms You Should Know*, Towards Data Science. [Online]. Available: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

[5] MATLAB, r2019b, *eig*, MATLAB MathWorks, 2020.

[6] F. Cincotti et. al, *Classification of EEG Mental Patterns by Using Two Scalp Electrodes and Mahalanobis Distance-Based Classifiers*, Methods of Information in Medicine, 2002.

[7] MATLAB, r2019b, *kmeans*, MATLAB MathWorks, 2020.