Machine Learning in Image Segmentation of Scanning Electron Microscopy Images of Nuclear Material

MARY E. NWOSU*, DoD Center of Excellence: Howard University, USA

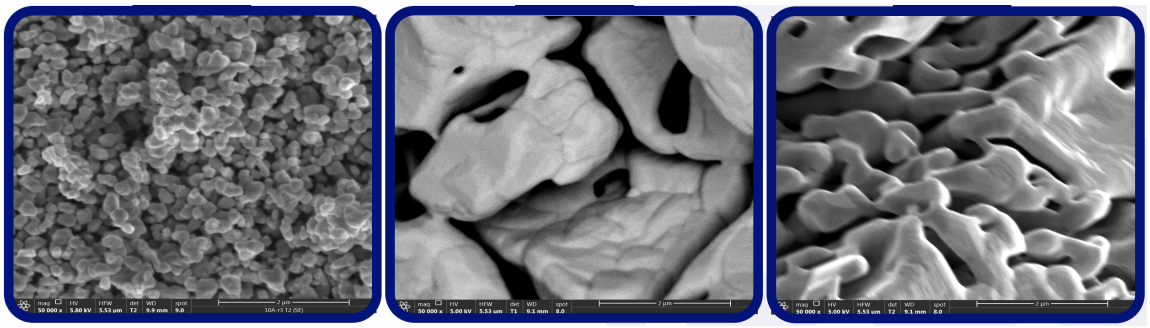

Fig. 1. Three Scanning Electron Microscopy (SEM) Images from University of Utah (Luther McDonald)

The Morphological Signatures Project at the Los Alamos National Laboratory (LANL) conducts quantitative morphological analyses of microstructural features visible in scanning electron microscopy (SEM) images of nuclear material. Particle morphology analysis has been developed and applied with image analysis software to examine - specifically, to segment and quantify - fully visible nuclear particles in SEM images. Segmentation (the process of partitioning or delineating a digital image into multiple segments) and quantification (the process of measuring attributes of an image and mapping them into members of some set of numbers) of nuclear particles are performed with a deployed LANL software, Morphological Analysis of Materials (MAMA). Pre-clustering of SEM images into appropriate groups associated with how MAMA performs particle morphology could facilitate and even improve the segmentation and quantification workflow. This report describes the implementation of several such clusterings of 13 SEM images provided by Luther McDonald at the University of Utah. In particular, this report examines the clustering of images by the amount of background and/or percentage of pixel color and documents (with figures) the working code (in Python) that clusters and sorts by background according to best practices.

CCS Concepts: • **Artificial Intelligence** → **Machine Learning**; • **Digital Image Processing** → *Clustering Images*; • **Artificial Intelligence/Machine Learning** → Neural Networks; • **Principal Component Analysis** → K-means Clustering.

Additional Key Words and Phrases: nuclear material forensics, machine learning, clustering images, image segmentation, feature extraction, K-means, principal component analysis, convolutional neural networks, visual similarity, image quantification, particle morphology

Author's address: Mary E. Nwosu, mary.nwosu@bison.howard.edu, DoD Center of Excellence: Howard University, 2400 6th Street., Washington, DC, USA, 20059.

Manuscript submitted to ACM

ACM Reference Format:
Mary E. Nwosu. 2022. Machine Learning in Image Segmentation of Scanning Electron Microscopy Images of Nuclear Material. 1, 1 (December 2022), 20 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

An important objective in pre-detonation nuclear forensics (where scientific techniques are used in connection with the detection of a crime) is to determine critical information as to the origin and age of certain radiological material. Particle morphological (size, shape, and texture) signatures of nuclear material can be used to infer the processing history of interdicted materials. Traditional signatures in certain materials, such as morphological changes based on calcination temperature, can lead to the identification of source and route of nuclear material and devices before they are used. Therefore, understanding and prediction of these radiological processes remains a crucial interest of national and international security.

Towards this objective, the Morphological Signatures Project at the Los Alamos National Laboratory (LANL) conducts quantitative morphological analyses of microstructural features visible in scanning electron microscopy (SEM) images of nuclear material. Figure 1 highlights some example SEM images of nuclear material acquired by Luther McDonald, a member of the Morphological Signatures Project and who is based at the University of Utah. Particle morphological analysis has been developed and applied with image analysis software to examine - specifically, to segment and quantify - fully visible nuclear particles in SEM images.

In digital image processing, image segmentation is the process of partitioning or delineating a digital image into multiple segments, aka image regions or image objects or sets of pixels; where the goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze [16]. Segmentation and quantification (the process of characterizing and measuring attributes of an image, by human sense or machine observations, and mapping them into members of some set of numbers or coded concepts [4]) of nuclear particles are performed with a deployed LANL software, Morphological Analysis of Materials (MAMA). With MAMA, SEM images of nuclear particles are manually segmented (located, recognized, and assigned boundaries) and then their attributes are quantified - attributes such as circularity, area, perimeter, and ellipse aspect ratio.

Workflow data to MAMA captures both algorithmic and/or human input (from a subject matter expert) pertaining to the type of segmentation required for a particular morphological analysis. The difficulty of manual quantification would be significantly eased if the SEM images were somehow "pre-clustered" into groups associated with how MAMA performs particle segmentation. Clustering of SEM images into appropriate groups could facilitate and even improve the segmentation and quantification workflow. This report describes the implementation of several such clusterings of 13 SEM images provided by Luther McDonald at the University of Utah. In particular, this report examines the "clustering of images by the amount of background and/or percentage of pixel color" and documents (with figures) the "working code that clusters and sorts by background (Luther's Data) according to best practices".

1.1 The Objective

The following report compares a hand-sorted, manually-designed feature clustering of 13 SEM images transformed with an image thresholding function [13] (meant to simulate the input of a subject matter expert, providing the "ground truth" of the desired categorizations) to an automated clustering process using a pre-trained neural network, a principal

component analysis, and a K-means clustering algorithm [3]. Both clustering processes are meant to serve as the "pre-clustering" of SEM images of nuclear material into groups associated with how MAMA performs particle segmentation and quantification.

1.2 The Configuration

The configuration of these implementations includes two binary image thresholding functions of "cv.threshold", from the OpenCV open-source library [7]. The pre-trained model used here is the VGG16 convolutional neural network (CNN) with a principal component analysis (PCA) and a "KMeans" clustering algorithm.

1.3 The Dataset

The dataset used in these implementations is 13 Scanning Electron Microscopy (SEM) images of nuclear material - .tif files at 50,000x magnification - of Luther McDonald's original set of 35 images from the University of Utah [5]. Figure 2 displays Luther McDonald's 13 SEM images at 50kx magnification.

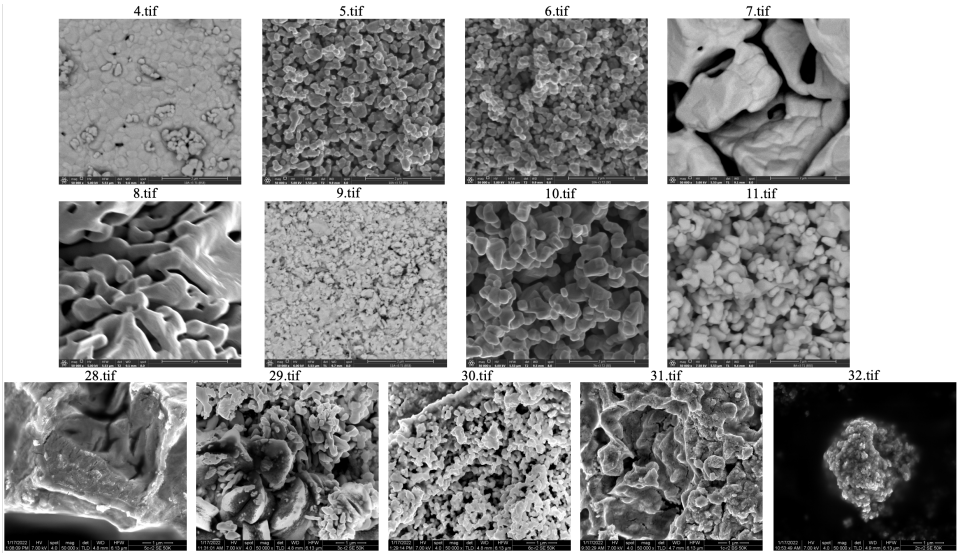

Fig. 2. Luther McDonald's 13 SEM Images of Nuclear Material at 50kx Magnification

2 CLUSTERINGS BY A SUBJECT MATTER EXPERT (SME)

Quantitative morphological analyses as applied to nuclear forensics currently requires subject matter experts (SMEs) in both microscopy and nuclear production processes. Their analyses are time-consuming and the partially manual tasks often involve multiple software applications. One of the goals of the Morphological Signatures Project at LANL is to facilitate the methods of extracting and encoding domain knowledge associated with SEM imagery of nuclear material so that parts of these analyses can be automated. If information derived from human input can be cumulatively learned by algorithms over a period a time, the need for time-consuming and tailored analyses by an image-by-image basis may be reduced.

For example, multiple object types and features of a single SEM image may indicate the kind of segmentation the image requires. Usually, an SME, like a chemist or a materials microscopist, would provide input as to how SEM images of nuclear particles should be clustered prior to segmentation - where each cluster preliminarily represents a type of segmentation to be performed. Using SME knowledge to determine the desired clustering of SEM images is a significant driver for advanced segmentation tool development with the Morphological Analysis of Materials (MAMA) software application; and if automated, this process could potentially consume less time and less human capital.

In the Morphological Signatures Project, Ian Schwerdt is one of the SMEs that conducts quantitative morphological analyses on SEM images of nuclear material. He is a ... Using his domain expert knowledge, this report will preliminarily group the 13 Original SEM images into small clusters; clusters, which will serve, not only as indicators of the type of segmentation that will occur using MAMA, but also as indicators of the type of processing history of the particular cluster.

2.1 Thresholding

The Morphological Signatures Project often relies on subject matter expert (SME) clustering before using MAMA and these clusterings often use thresholding, the most basic type of image segmentation. Thresholding methods, for example, can convert an SEM image from color or grayscale into a binary image or to one that is simply black and white - and therefore, possibly, more easily analyzed. For this report, the OpenCV function - **cv.threshold** - is used to apply two types of thresholding: simple thresholding and adaptive thresholding.

2.1.1 Simple Thresholding. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold (127, in this implementation), it is set to 0 (black), otherwise it is set to a maximum value of 255 (white). The simple thresholding types used here are **cv.THRESH-BINARY** (binary thresholding) and **cv.THRESH-BINARY-INV** (inverse-binary thresholding). The 13 original SEM images and their corresponding thresholding images are displayed in Figure 3.

2.1.2 Adaptive thresholding. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold (127, in this implementation), it is set to 0 (black), otherwise it is set to a maximum value of 255 (white). The simple thresholding types used here are **cv.THRESH-BINARY** (binary thresholding) and **cv.THRESH-BINARY-INV** (inverse-binary thresholding). The 13 original SEM images and their corresponding thresholding images are displayed in Figure 4.

2.2 Subject Matter Expert (SME) Clusterings

The SME has the option of using thresholding, imbued with his or her domain expert knowledge, in order to create the pre-clusterings of SEM images. In this implementation, however, the SME groups the 13 original SEM images (without thresholding) into clusters which preliminarily represent the type of segmentation to then be performed with the MAMA software application.

2.2.1 The SME Clusters of the Original 13 SEM Images. For this report, the SME categorizes the 13 original SEM images into the 8 clusters **[SME, G0, k=8]** displayed in Figure 5.

2.2.2 SME Thresholding Clusters as "Ground Truth". For this report, the above configuration of the 13 Original SEM images of nuclear material into 8 clusters **[SME, G0, k=8]** will serve as the chosen formation of "ground truth". In this

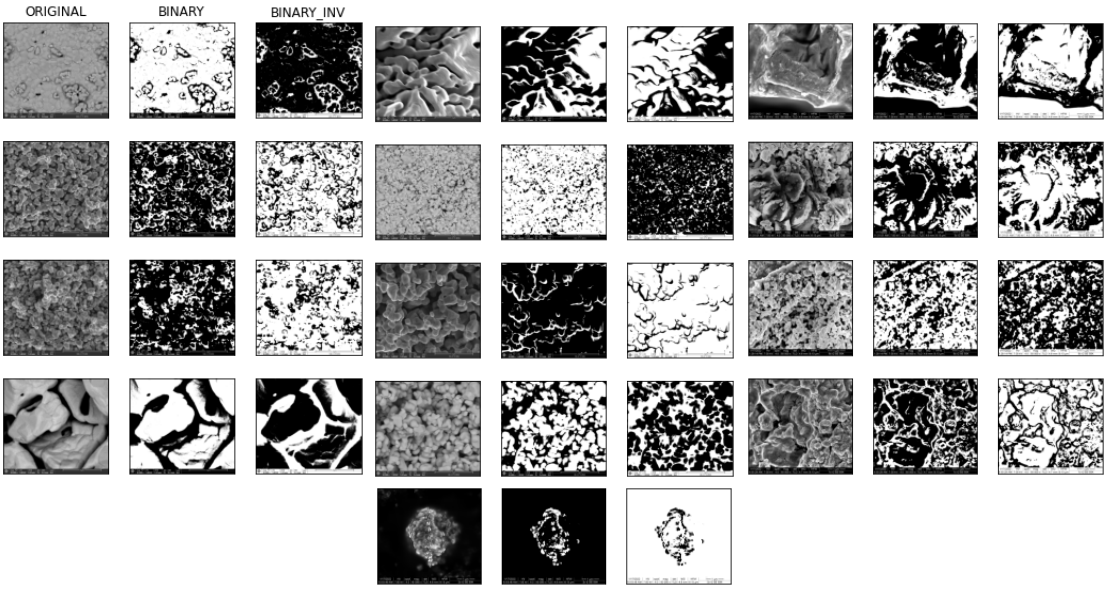

Fig. 3. Binary and Inverse-Binary Thresholdings of the 13 Original SEM Images

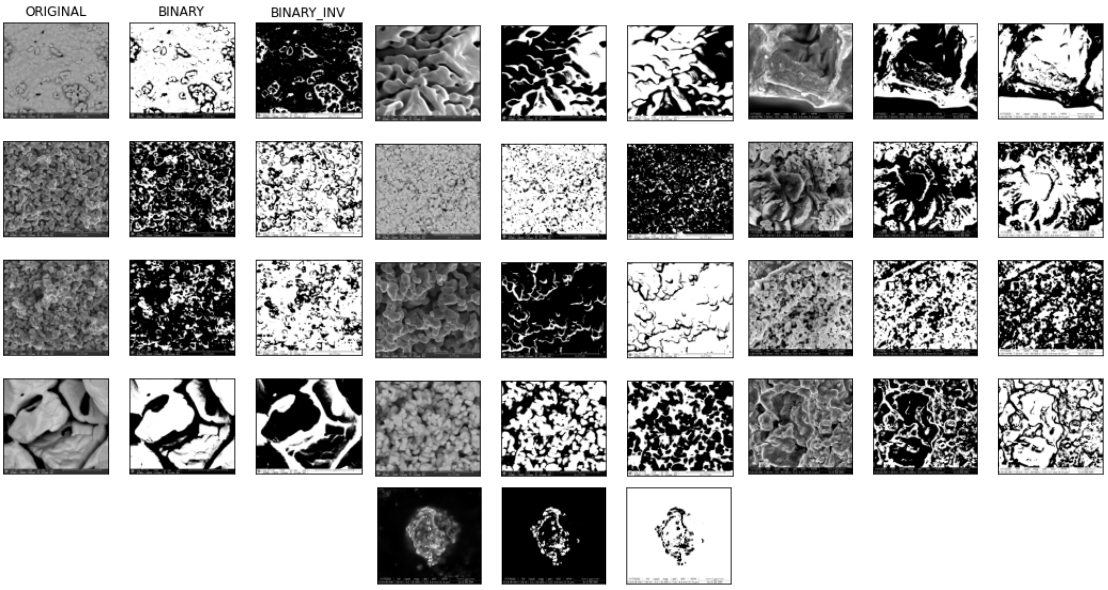

Fig. 4. Binary and Inverse-Binary Thresholdings of the 13 Original SEM Images

way, the SME-selected clustering of the 13 Original SEM images, **[SME, G0, k=8]**, represents the desired clustering by which the results of all other clustering processes are referenced.

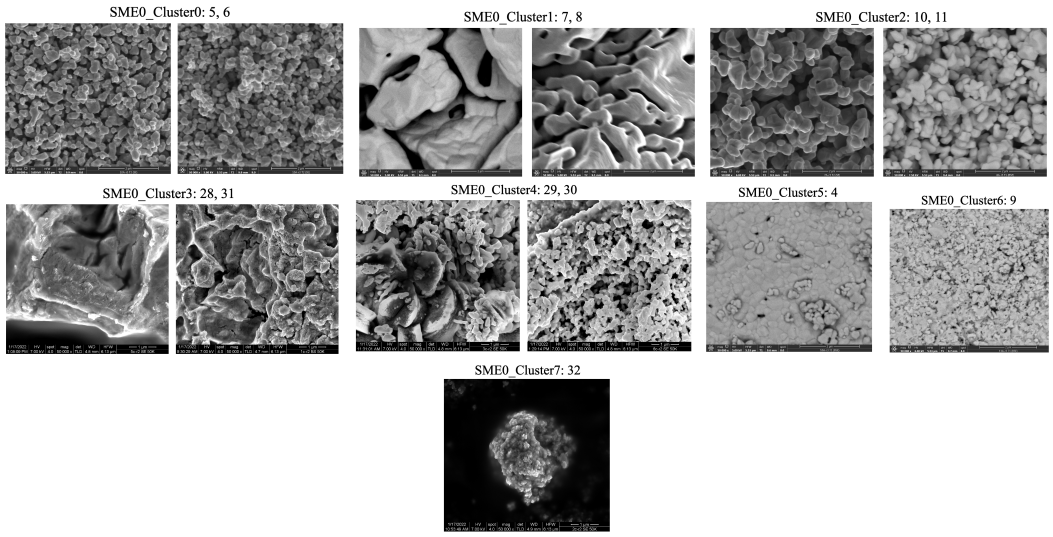

Fig. 5. SME-Selected Clustering of the Original 13 SEM Images **[SME, G0, k=8]**

3 THE AUTOMATED CLUSTERING PROCESS

The following section describes an automated clustering process involving a pre-trained neural network for feature extraction, a dimensionality reduction step using principal component analysis, and a K-means clustering algorithm. The results of this automated clustering process are meant to be compared to the preceeding binary thresholding clustering, **[SME, G1, k=6]**, established by the simulated subject matter expert (SME).

3.1 Feature Extraction

The VGG16 model (architecture displayed in Figure 6) is a simple, widely-used, and pre-trained convolutional neural network (CNN) that was originally developed in 2014 for ImageNet, a large visual database project used in visual object recognition software research [8]. Considered to be state of the art for image recognition tasks, the VGG16 model is used in this implementation as a feature extractor. Undisturbed, the VGG16 is a 16-layer architecture (Figure 7); there are three fully-connected layers that follow a stack of convolutional ones and the final dense layer is used for classification. In this implementation, the final dense output layer is manually removed (see Appendix C.1, Figure 19 for the Python code) from the VGG16 model in order to obtain a 4,096-dimensioned feature vector. This means that the new final layer is a fully-connected layer with 4,096 output nodes and that output vector will be used in clustering the 13 SEM images.

3.2 Dimensionality Reduction: Principal Component Analysis (PCA)

For high-dimensional datasets (i.e., 4,096 components), dimensionality reduction is usually performed in order to transform the data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data [15]. This is typically done to avoid the effects of the "curse of dimensionality" - an unfortunate expression referring to the various unfortunate phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings [14].

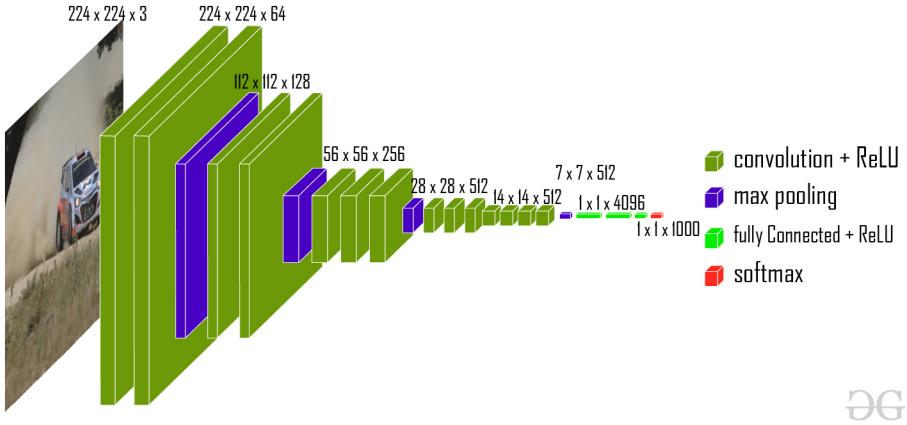

Fig. 6. VGG16 Architecture. Image via Medium.com. (https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615)

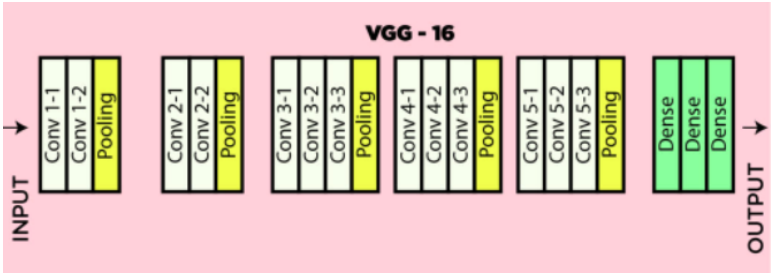

Fig. 7. VGG16 Architecture Map. Image via GeeksforGeeks.org. (https://www.geeksforgeeks.org/vgg-16-cnn-model/)

The initial feature vector of this implementation has 4,096 dimensions, but the feature space cannot simply *shorten* by slicing it or using some subset of it because information will be lost. The way to reduce the dimensionality while keeping as much information as possible can be done with principle component analysis (PCA). The main linear technique for dimensionality reduction, PCA performs a linear mapping of the data to a lower-dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized [1]. In other words, PCA allows the reduction of the number of dimensions (or components) while preserving as much information from the original vector as possible.

In order to examine the impact of PCA on clustering, this report displays the results of a parameter study conducted where the number of dimensions (or components) of the feature vector were manipulated and the number of clusters were held constant. The number of clusters (k=6) were chosen to match the configuration of the preceding SME-selected binary thresholding clusterings, **[SME, G1, k=6]** - shown here again in Figure 8. This SME-selected clustering configuration represents the "ground-truth" reference to which the following results will be compared.

3.2.1 No PCA, k=6. The first parameter manipulated in the study was simply to not use PCA at all. In this part of the study, the initial feature vector of 4,096 dimensions was input directly into the k-means algorithm (described in the next

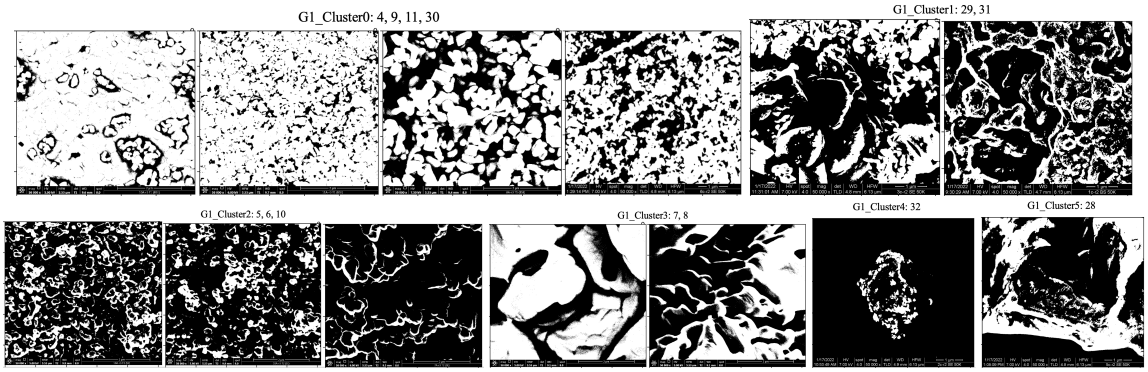

Fig. 8. SME-Selected Clustering of Binary Thresholding SEM Images, **[SME, G1, k=6]**

section) in order to cluster the original group of 13 SEM images - without any dimensionality reduction. The analysis produced the following configuration of 6 clusters for the original group, G0, **[No-PCA, G0, k=6]** - shown in Figure 9.

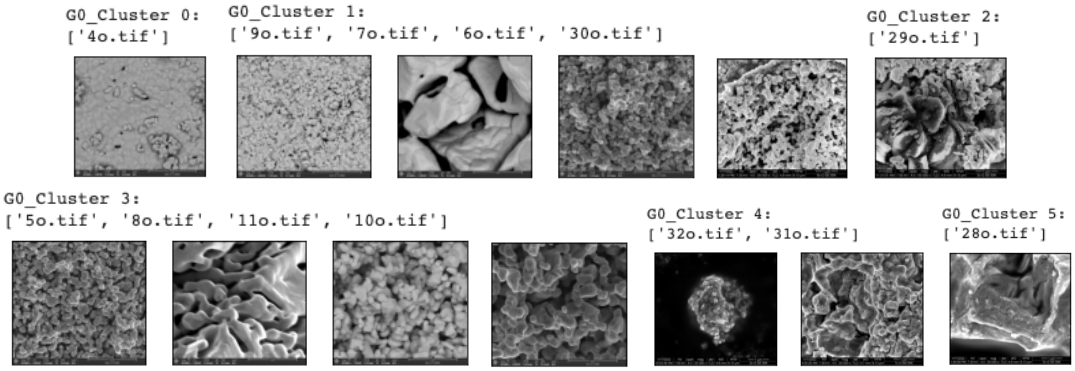

Fig. 9. "No-PCA" Clusters of G0, the Original Group of 13 SEM Images, **[No-PCA, G0, k=6]** – Not Similar to SME Configuration

This "No-PCA" **[No-PCA, G0, k=6]** analysis may well *have* experienced the "curse of dimensionality" as a result of it's 4,096 dimensional space because the configuration of clusters is not similar at all to the above binary thresholding clustering configured by the simulated SME **[SME, G1, k=6]**. Although in both configurations, Image 28 is in its own separate cluster, the resemblance ends there. For example, in the SME configuration **[SME, G1, k=6]**, Images 5 and 6 are in one cluster - as their particle morphologies are ostensibly identical; yet, not so, in the No-PCA configuration **[No-PCA, G0, k=6]**. The comparisons of the two configurations are replete with differences, and have very little similarities.

Although the No-PCA, binary thresholding clustering configuration **[No-PCA, G1, k=6]** *also* showed little resemblance to the SME configuration **[SME, G1, k=6]**; the No-PCA, inverse-binary thresholding clustering configuration **[No-PCA, G2, k=6]** *did* show much similarity - as displayed in Figure 10. In both the **[No-PCA, G2, k=6]** and the **[SME, G1, k=6]** configurations, Images 4, 9, 11, 30 are in one cluster. In both configurations, Images 32 28 form their own individual clusters. Similarly, Images 5 6 are in the same cluster for both configurations.

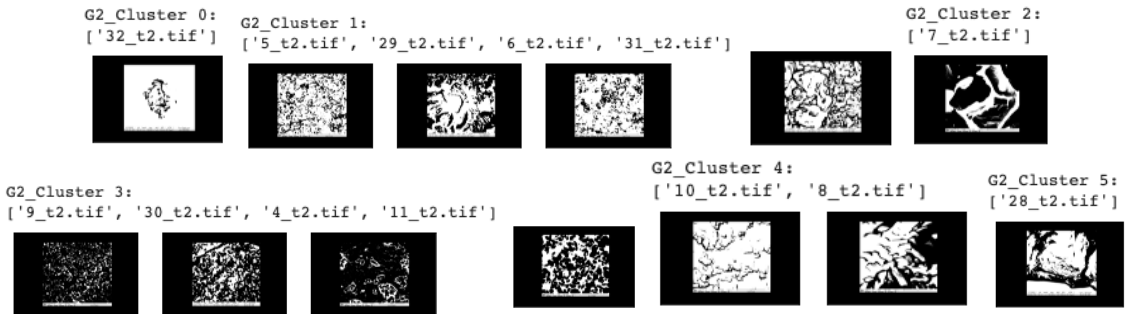

Fig. 10. "No-PCA" Clusters of G2, the Inverse-Binary Thresholding Group of 13 SEM Images, **[No-PCA, G2, k=6]** – Similar to SME Configuration

In short, the SME clustering configuration, **[SME, G1, k=6]**, is similar - though not identical - to the **[No-PCA, G2, k=6]** clustering configuration. The **[SME, G1, k=6]** configuration is NOT similar to **[No-PCA, G0, k=6]** configuration NOR is it similar to the **[No-PCA, G1, k=6]** configuration.

3.2.2 n=4096, k=6. The next parameter manipulated in the study was the maximum number of components (n) that could theoretically be used in the PCA, n=4096. However, when the PCA was set to n=4096 (see Figure 11), the algorithm produced an error (Figure 12).

```
# reduce dimensions of each feature vector
pca = PCA(n_components=4096, random_state=22)
```

Fig. 11. Python Code Used to Set PCA at n=4096

```
File /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/sklearn/decomposition/_pca.py:475,
in PCA._fit_full(self, X, n_components)
    471         raise ValueError(
    472             "n_components='mle' is only supported if n_samples >= n_features"
    473         )
    474 elif not 0 <= n_components <= min(n_samples, n_features):
--> 475     raise ValueError(
    476         "n_components=%r must be between 0 and "
    477         "min(n_samples, n_features)=%r with "
    478         "svd_solver='full'" % (n_components, min(n_samples, n_features))
    479     )
    480 elif n_components >= 1:
    481     if not isinstance(n_components, numbers.Integral):

ValueError: n_components=4096 must be between 0 and min(n_samples, n_features)=13 with svd_solver='full'
```

Fig. 12. Error Message Produced when n=4096

As stated in the error message, the number of n components, in the PCA, may not exceed the number of samples of the problem; therefore, in this implementation, the number of n components may not exceed the number of SEM image samples, 13.

3.2.3 n=13, k=6. The next parameter chosen for the study was n=13. The number of vector components, n, chosen for the PCA model could only go as high as the number of images samples. Subsequently, the initial feature vector of 4,096 dimensions was input into the PCA model at n=13 and that output of the PCA was then input into the k-means algorithm (described in the next section).

The analysis produced a configuration of 6 clusters for the original group, G0 - **[n=13, G0, k=6]**; a configuration of 6 clusters for the binary thresholding group, G1 - **[n=13, G1, k=6]**; and a configuration of 6 clusters for the inverse-binary thresholding group, G2 - **[n=13, G2, k=6]**.

All three of these clustering configurations exactly matched their "No-PCA" counterparts; meaning,

[n=13, G0, k=6] = **[No-PCA, G0, k=6]** and

[n=13, G1, k=6] = **[No-PCA, G1, k=6]** and

[n=13, G2, k=6] = **[No-PCA, G2, k=6]**.

As a result, these clustering configurations have the same relationship to the SME clustering configuration, **[SME, G1, k=6]**, as did the "No-PCA" configurations. As such, **[SME, G1, k=6]** is similar - though not identical - to the **[n=13, G2, k=6]** clustering configuration. The **[SME, G1, k=6]** configuration is NOT similar to **[n=13, G0, k=6]** configuration NOR is it similar to the **[n=13, G1, k=6]** configuration.

3.2.4 n=8, k=6. All three of these clustering configurations exactly matched their "No-PCA" counterparts, as well.

This means that -

[n=13, G0, k=6] = **[No-PCA, G0, k=6]** and

[n=13, G1, k=6] = **[No-PCA, G1, k=6]** and

[n=13, G2, k=6] = **[No-PCA, G2, k=6]**.

This also means that these clustering configurations have the same relationship to the SME clustering configuration, **[SME, G1, k=6]**, as did the "No-PCA" configurations.

3.2.5 n=4, k=6. All three of these clustering configurations exactly matched their "No-PCA" counterparts, as well.

This means that -

[n=4, G0, k=6] = **[No-PCA, G0, k=6]** and

[n=4, G1, k=6] = **[No-PCA, G1, k=6]** and

[n=4, G2, k=6] = **[No-PCA, G2, k=6]**.

This also means that these clustering configurations have the same relationship to the SME clustering configuration, **[SME, G1, k=6]**, as did the "No-PCA" configurations.

3.2.6 PCA Impact on Clustering. From this simple parameter study, it seems that the PCA used in this implementation is an extremely robust algorithm. Regardless of the depth of the dimensionality reduction (n=13 or 8 or 4) and holding the number of clusters constant (k=6); the clustering configurations produced, remained the same across all n's. And one of the three configurations produced for each n, was at least similar to the SME-selected, "ground-truth" clustering configuration.

3.3 K-means Clustering

The K-means clustering is one of the simplest and most-popular unsupervised machine learning algorithms and it allows for grouping feature vectors into k clusters. Typically, unsupervised algorithms make inferences from datasets

using only input vectors without referring to known, or labelled, outcomes. In this implementation, each cluster should contain images that are visually similar.

In order to examine the impact of the K-means algorithm on clustering, this report displays the results of another parameter study conducted where the number of clusters were adjusted. The number of components set in the PCA analysis preceding the clustering were also taken into account. Finally, the clustering configurations produced in this analysis are compared to the SME-selected clustering configuration, **[SME, G1, k=6]** - which represents "ground-truth".

3.3.1 No-PCA, k=5. The analysis that was run with "No-PCA" on the original group of 13 SEM images, G0, produced the following 5-clustered configuration, **[No-PCA, G0, k=5]**.

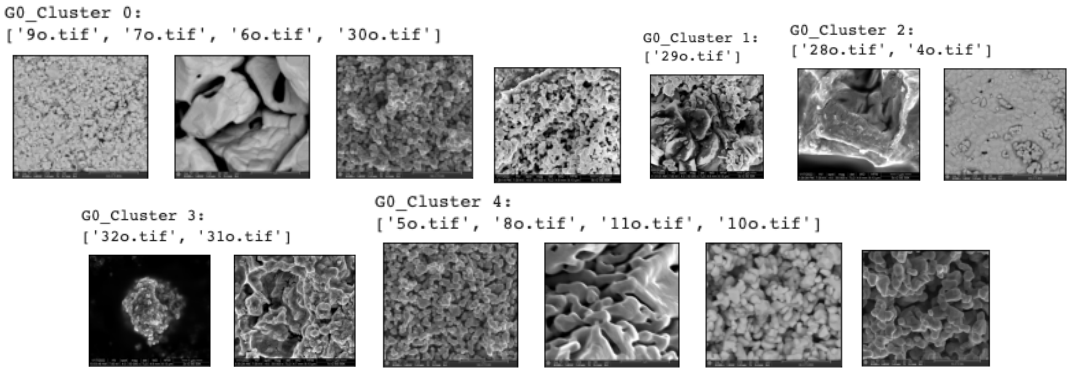

Fig. 13. The 5-Clustered Configuration of G0, the Original Group of 13 SEM Images, **[No-PCA, G0, k=5]**

The "No-PCA" analysis on the group of 13 images transformed by the binary thresholding function, G1, produced the following 5-clustered configuration - **[No-PCA, G1, k=5]** (Figure 14).

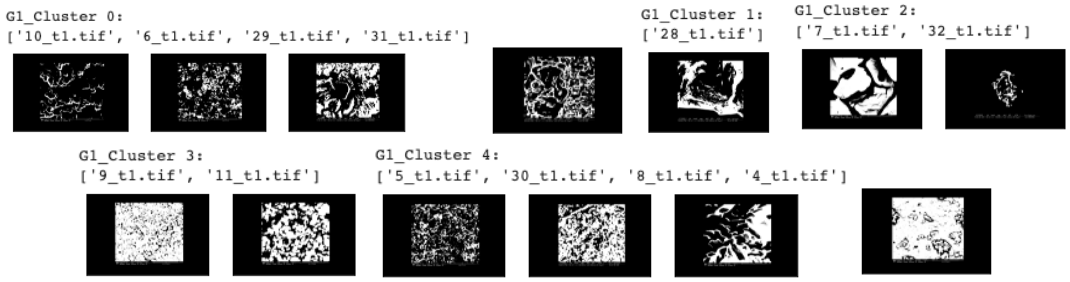

Fig. 14. The 5-Clustered Configuration of G1, the Binary Thresholding Group of 13 SEM Images - **[No-PCA, G1, k=5]**

And the "No-PCA" analysis on the final group of 13 images transformed by the inverse-binary thresholding function, G2, produced the following 5-clustered configuration - **[No-PCA, G2, k=5]** (Figure 15).

3.3.2 n=13, k=5. The next parameter chosen for the study was n=13 at k=5. The number of vector components, n, chosen for the PCA model could only go as high as the number of images samples. And as in the previous section, the initial feature vector of 4,096 dimensions was input into the PCA model at n=13 and that output of the PCA was then input into the k-means algorithm.

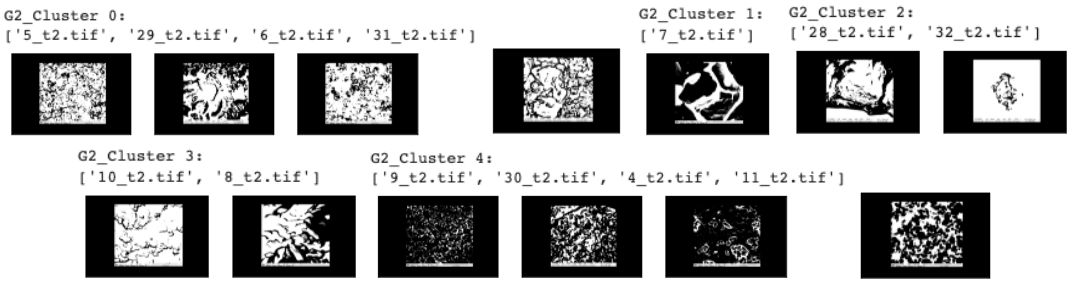

Fig. 15. The 5 Clusters of G2, the Inverse-Binary Thresholding Group of 13 SEM Images

The analysis produced a configuration of 5 clusters for the original group, G0 - **[n=13, G0, k=5]**; a configuration of 5 clusters for the binary thresholding group, G1 - **[n=13, G1, k=5]**; and a configuration of 5 clusters for the inverse-binary thresholding group, G2 - **[n=13, G2, k=5]**.

All three of these clustering configurations exactly matched their "No-PCA" counterparts; meaning,

[n=13, G0, k=5] = **[No-PCA, G0, k=5]** and

[n=13, G1, k=5] = **[No-PCA, G1, k=5]** and

[n=13, G2, k=5] = **[No-PCA, G2, k=5]**.

3.3.3 n=8 4, k=5. Although the n=8 configurations are exactly the same as the "No-PCA" configurations, the n=4 configurations did vary slightly.

4 CONCLUSION

The Morphology Signatures Project at the Los Alamos National Laboratory (LANL) conducts quantitative morphological analyses of microstructural features visible in scanning electron microscopy (SEM) images of nuclear material. Particle morphology analysis has been developed and applied with image analysis software to examine - specifically, to segment and quantify - fully visible nuclear particles in SEM images. Segmentation (the process of partitioning or delineating a digital image into multiple segments) and quantification (the process of measuring attributes of an image and mapping them into members of some set of numbers) of nuclear particles are performed with a deployed LANL software, Morphological Analysis of Materials (MAMA). Pre-clustering of SEM images into appropriate groups associated with how MAMA performs particle morphology could facilitate and even improve the segmentation and quantification workflow. This report describes the implementation of several such clusterings of 13 SEM images provided by Luther McDonald at the University of Utah. In particular, this report examines the "clustering of images by the amount of background and/or percentage of pixel color" and documents (with figures) the "working code that clusters and sorts by background (Luther's Data) according to best practices".

[2, 6, 9–12]

REFERENCES

[1] Jason Brownlee. 20120. *Introduction to Dimensionality Reduction for Machine Learning.* Retrieved March 27, 2022 from https://machinelearningmastery.com/dimensionality-reduction-for-machine-learning/

[2] Tom Burr, Ian Schwerdt, Kari Sentz, Luther MacDonald, and Marianne Wilkerson. 2021. Overview of Algorithms for Using Particle Morphology in Pre-Detonation Nuclear Forensics. *Algorithms* 14 (August 2021), 21 pages. https://doi.org/10.3390/a14120340

[3] Gabe Flomo. 2020. *How to Cluster Images Based on Visual Similarity.* Retrieved March 20, 2022 from https://towardsdatascience.com/how-to-cluster-images-based-on-visual-similarity-cd6e7209fe34

[4] Google. 2022. *Quantification in Image Processing.* Retrieved March 27, 2022 from https://www.google.com/search?q=quantification+in+image+processing&rlz=1C5CHFA_enUS914US914&ei=kAJAYu_LMdCZptQP9seS4AE&oq=quantification+in+image&gs_lcp=Cgdnd3Mtd2l6EAEYADIFCAAQgAQyCAgAEBYQChAeMgYIABAWEB4yBggAEBYQHjIGCAAQFhAeMgYIABAWEB4yBggAEBYQHjIGCAAQFhAeMgYIABAWEB4yBQgAEIYDOgUIABCR sclient=gws-wiz

[5] Luther McDonald. 2022. MODE Workshop Unknown Images 2022. SEM Images. Retrieved March 8, 2022 from https://app.box.com/folder/154806766301?s=99uzkqrgct1bpt3zrlhyhwm4bc8ppi91

[6] NVIDIA. 2021. *Deep Learning Institute: Fundamentals of Deep Learning Course.* Retrieved March 10, 2022 from https://courses.nvidia.com/courses/course-v1:DLI+C-FX-01+V3/courseware/

[7] OpenCV. 2021. *OpenCV - Python Tutorials.* Retrieved February 28, 2022 from https://docs.opencv.org/4.5.5/d6/d00/tutorial_py_root.html

[8] Tinsy John Perumanoor. 2021. *What is VGG16?—Introduction to VGG16.* Retrieved March 20, 2022 from https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615

[9] Reid Porter, Christy Ruggiero, Rhys Leahy, Don Hush, Neal Harvey, Patrick Kelly, Wayne Scoggins, and Lav Tandon. 2011. Interactive Image Quantification Tools in Nuclear Material Forensics. In *Proceedings of SPIE (SPIE '11)*. SPIE, Bellingham, Washington, USA, 9 pages.

[10] Christy Ruggiero, Amy Ross, and Reid Porter. 2016. *Segmentation and Learning in the Quantitative Analysis of Microscopy Images.* Los Alamos National Laboratory Technical Report. Los Alamos National Laboratory, Los Alamos, NM.

[11] Aishwarya Singh. 2019. *3 Beginner-Friendly Techniques to Extract Features from Image Data using Python.* Retrieved March 20, 2022 from https://www.analyticsvidhya.com/blog/2019/08/3-techniques-extract-features-from-image-data-machine-learning-python/

[12] Abdulhamit Subasi. 2020. Chapter 5: Other classification examples. In *Practical Machine Learning for Data Analysis Using Python*. Academic Press, Cambridge, Massachusetts, 323–390. https://doi.org/10.1016/B978-0-12-821379-7.00005-9

[13] OpenCV Python Tutorials. 2021. *Image Processing in OpenCV: Image Thresholding.* Retrieved March 10, 2022 from https://docs.opencv.org/4.5.5/d7/d4d/tutorial_py_thresholding.html

[14] Wikipedia. 2022. *Curse of Dimensionality.* Retrieved March 27, 2022 from https://en.wikipedia.org/wiki/Curse_of_dimensionality

[15] Wikipedia. 2022. *Dimensionality Reduction.* Retrieved March 27, 2022 from https://en.wikipedia.org/wiki/Dimensionality_reduction

[16] Wikipedia. 2022. *Image Segmentation.* Retrieved March 26, 2022 from https://en.wikipedia.org/wiki/Image_segmentation

A PYTHON CODE FOR IMPORTING AND PREPROCESSING DATA

A.1 Imports

The first step is to import all of the modules and models needed to load and process the 13 images as well to extract and cluster the feature vectors.

where -

- **load-img** loads an image from a file as a PIL object;
- **img-to-array** converts the PIL object into a NumPy array;
- **preproccess-input** prepares an image into the format the model requires;
- **VGG16** is the pre-trained model;
- **KMeans** is the clustering algorithm; and
- **PCA** reduces the dimensions of the feature vector.

A.2 Preprocessing Data

Python points to the directory where the images are located. In the following code, the original and thresholding images are saved and the name of the file is used, instead of loading a whole file path.

And then lists are created to hold the filenames of all of the images -

B PYTHON CODE FOR SME CLUSTERING

(see Figure 18)

```python
# import the modules needed to load and process the images
import cv2 as cv
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import os

# for loading/processing the images
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input

# models
from keras.applications.vgg16 import VGG16
from keras.models import Model

# for clustering and dimension reduction
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# for everything else
from random import randint
import pickle
%matplotlib inline
```

Fig. 16. Python Code for Importing Modules and Models Needed for Clustering

```python
# save original and transformed images
for i in range(4,12):
    flow = cv.imread(str(i)+'.tif',0)
    ret,thresh1 = cv.threshold(flow,127,255,cv.THRESH_BINARY)
    ret,thresh2 = cv.threshold(flow,127,255,cv.THRESH_BINARY_INV)
    plt.imshow(flow,'gray'), plt.savefig(str(i)+'_t0.tif', dpi=300)
    plt.imshow(thresh1,'gray'), plt.savefig(str(i)+'_t1.tif', dpi=300)
    plt.imshow(thresh2,'gray'), plt.savefig(str(i)+'_t2.tif', dpi=300)

for i in range(28,33):
    flow = cv.imread(str(i)+'.tif',0)
    ret,thresh1 = cv.threshold(flow,127,255,cv.THRESH_BINARY)
    ret,thresh2 = cv.threshold(flow,127,255,cv.THRESH_BINARY_INV)
    plt.imshow(flow,'gray'), plt.savefig(str(i)+'_t0.tif', dpi=300)
    plt.imshow(thresh1,'gray'), plt.savefig(str(i)+'_t1.tif', dpi=300)
    plt.imshow(thresh2,'gray'), plt.savefig(str(i)+'_t2.tif', dpi=300)
```

Fig. 17. Python Code for Saving Original, Binary, and Inverse-Binary Thresholding Images

C PYTHON CODE FOR THE AUTOMATED CLUSTERING PROCESS

C.1 The CNN Model for Feature Extraction

The following code uses the feature-extraction function in order to extract the features from all of the images and store the features in a dictionary with the filenames as the keys.

When loading the images, the target size is set to (224, 224) because the VGG model expects the images it receives to be 224x224 NumPy arrays. Currently, the arrays have only 3 dimensions (rows, columns, channels) and the model operates in batches of samples. So, in the following code, the arrays are expanded to add the dimension that will let the model know how many images will be inputted (num-of-samples, rows, columns, channels). The last step is to pass the reshaped array to the preprocess-input method and the image is then ready to be loaded into the model.

```
# create lists to hold all the image filenames
flowers = []
candy = []
bbq = []

# create a ScandirIterator aliased as files
with os.scandir(path) as files:
  # loops through each file in the directory
    for file in files:
        if file.name.endswith('_t0.tif'):
          # adds only the image files to the flowers list
            flowers.append(file.name)
        elif file.name.endswith('_t1.tif'):
          # adds only the image files to the flowers list
            candy.append(file.name)
        elif file.name.endswith('_t2.tif'):
          # adds only the image files to the flowers list
            bbq.append(file.name)

# view the entries of each list
print('flowers:\n', flowers), print('candy:\n', candy), print('bbq:\n', bbq)
```

Fig. 18. Python Code for Creating Lists to Hold the Image Filenames

C.2 Python Code for Dimensionality Reduction (PCA)

C.3 Python Code for K-means Clustering

The additional code creates dictionaries that hold the clusters of images.

D PYTHON CODE FOR DISPLAYING THE CLUSTERS

The following code displays the 5 clusters for each group of images: the original group of 13 images, the group of 13 images transformed by the binary thresholding function, and the final group of 13 images transformed by the inverse-binary thresholding function.

```
# change the working directory to the path where the images are located
path = r"/Users/marynwosu/mag_50k"
os.chdir(path)

# plot the original and transformed images
img = cv.imread('4.tif',0)
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)

titles = ['ORIGINAL','BINARY','BINARY_INV']
images = [img, thresh1, thresh2]

for i in range(3):
    plt.subplot(1,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()

for j in range(5,12):
    img = cv.imread(str(j) + '.tif',0)
    ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
    ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)

    images = [img, thresh1, thresh2]
    for k in range(3):
        plt.subplot(1,3,k+1),plt.imshow(images[k],'gray',vmin=0,vmax=255)
        plt.xticks([]),plt.yticks([])
    plt.show()

for j in range(28,33):
    img = cv.imread(str(j) + '.tif',0)
    ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
    ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)

    images = [img, thresh1, thresh2]
    for k in range(3):
        plt.subplot(1,3,k+1),plt.imshow(images[k],'gray',vmin=0,vmax=255)
        plt.xticks([]),plt.yticks([])
    plt.show()
```

Fig. 19. Python Code for Binary and Inverse-Binary Thresholding

```
# load the model first and pass as an argument
model = VGG16()

# remove the output layer
model = Model(inputs = model.inputs, outputs = model.layers[-2].output)
```

Fig. 20. Python Code Used to Remove the Output Layer of the VGG16

```
# for holding the data id and the images
data0 = {}
data1 = {}
data2 = {}
p = r"/Users/marynwosu/mag_50k"

# loop through each image in the dataset of original images (G0)
for f in flowers:
    # try to extract the features and update the dictionary
    try:
        feat0 = extract_features(f,model)
        data0[f] = feat0
    # if something fails, save the extracted features as a pickle file (optional)
    except:
        with open(p,'wb') as file:
            pickle.dump(data0,file)
```

Fig. 21. Python Code for Feature Extraction

```
# loop through each image in the dataset of images transformed by binary threasholding (G1)
for c in candy:
    # try to extract the features and update the dictionary
    try:
        feat1 = extract_features(c,model)
        data1[c] = feat1
    # if something fails, save the extracted features as a pickle file (optional)
    except:
        with open(p,'wb') as file:
            pickle.dump(data1,file)
```

```
# loop through each image in the dataset of images transformed by inverse-binary threasholding (G3)
for b in bbq:
    # try to extract the features and update the dictionary
    try:
        feat2 = extract_features(b,model)
        data2[b] = feat2
    # if something fails, save the extracted features as a pickle file (optional)
    except:
        with open(p,'wb') as file:
            pickle.dump(data2,file)
```

Fig. 22. Continuation of Python Code for Feature Extraction

```
# get lists of the filenames
filenames0 = np.array(list(data0.keys()))
filenames1 = np.array(list(data1.keys()))
filenames2 = np.array(list(data2.keys()))

# get lists of just the features
feat0 = np.array(list(data0.values()))
feat1 = np.array(list(data1.values()))
feat2 = np.array(list(data2.values()))
feat0.shape, feat1.shape, feat2.shape
```

Fig. 23. Python Code for Creating Lists of Features

```
# reshape each list so that there are 13 samples of 4096 vectors
feat0 = feat0.reshape(-1,4096)
feat1 = feat1.reshape(-1,4096)
feat2 = feat2.reshape(-1,4096)
feat0.shape, feat1.shape, feat2.shape
```

Fig. 24. Python Code for Reshaping Feature Vectors

```
# load the model first and pass as an argument
model = VGG16()
model = Model(inputs = model.inputs, outputs = model.layers[-2].output)

def extract_features(feature, model):
    # load the image as a 224x224 array
    img = load_img(feature, target_size=(224,224))
    # convert from 'PIL.Image.Image' to numpy array
    img = np.array(img)
    # reshape the data for the model reshape(num_of_samples, dim 1, dim 2, channels)
    reshaped_img = img.reshape(1,224,224,3)
    # prepare image for model
    imgx = preprocess_input(reshaped_img)
    # get the feature vector
    features = model.predict(imgx, use_multiprocessing=True)
    return features
```

Fig. 25. Python Code for Pre-Processing Images to Load the Model

```
# reduce dimensions of each feature vector
pca = PCA(n_components=13, random_state=22)

pca.fit(feat0)
x0 = pca.transform(feat0)

pca.fit(feat1)
x1 = pca.transform(feat1)

pca.fit(feat2)
x2 = pca.transform(feat2)
```

Fig. 26. Python Code for Dimensionality Reduction (PCA)

```
# apply the KMeans clustering algorithm to each dataset group
kmeans = KMeans(n_clusters=5, random_state=22)

kmeans.fit(x0)
kmeans.fit(x1)
kmeans.fit(x2)
```

Fig. 27. Python Code for K-means Clustering (k=5)

```
# for holding the cluster id and the images { id: [images] }
groups0 = {}
groups1 = {}
groups2 = {}

for file, cluster in zip(filenames0,kmeans.labels_):
    if cluster not in groups0.keys():
        groups0[cluster] = []
        groups0[cluster].append(file)
    else:
        groups0[cluster].append(file)
print(str(len(groups0)) + ' clusters in the original group of images, G0')

for file, cluster in zip(filenames1,kmeans.labels_):
    if cluster not in groups1.keys():
        groups1[cluster] = []
        groups1[cluster].append(file)
    else:
        groups1[cluster].append(file)
print(str(len(groups1)) + ' clusters in the binary thresholding group of images, G1')

for file, cluster in zip(filenames2,kmeans.labels_):
    if cluster not in groups2.keys():
        groups2[cluster] = []
        groups2[cluster].append(file)
    else:
        groups2[cluster].append(file)
print(str(len(groups2)) + ' clusters in the inverse-binary thresholding group of images, G2')
```

Fig. 28. Python Code for Clustering Dictionaries

```
# viewing clusters (based on identifier) of each group (G0, G1, and G2) of images
for i in range(1,6):
    cluster0 = list(groups0[i-1])
    print('G0_Cluster ' + str(i-1) + ': ')
    print(cluster0)
    for l in range(len(groups0[i-1])):
        m = len(groups0[i-1])
        if ((m/3)>=0) & ((m%3)!=0):
            n=int(m/3) + 1
        elif ((m/3)>=0) & ((m%3)==0):
            n=int(m/3)
        plt.subplot(n, 3, l+1)
        plt.imshow(cv.imread(cluster0[l],0), 'gray', vmin=0, vmax=255)
        plt.xticks([]),plt.yticks([])
    plt.show()

for j in range(1,6):
    cluster1 = list(groups1[j-1])
    print('G1_Cluster ' + str(j-1) + ': ')
    print(cluster1)
    for l in range(len(groups1[j-1])):
        m = len(groups1[j-1])
        if ((m/3)>=0) & ((m%3)!=0):
            n=int(m/3) + 1
        elif ((m/3)>=0) & ((m%3)==0):
            n=int(m/3)
        plt.subplot(n, 3, l+1)
        plt.imshow(cv.imread(cluster1[l],0), 'gray',vmin=0,vmax=255)
        plt.xticks([]),plt.yticks([])
    plt.show()

for k in range(1,6):
    cluster2 = list(groups2[k-1])
    print('G2_Cluster ' + str(k-1) + ': ')
    print(cluster2)
    for l in range(len(groups2[k-1])):
        m = len(groups2[k-1])
        if ((m/3)>=0) & ((m%3)!=0):
            n=int(m/3) + 1
        elif ((m/3)>=0) & ((m%3)==0):
            n=int(m/3)
        plt.subplot(n, 3, l+1)
        plt.imshow(cv.imread(cluster2[l],0), 'gray',vmin=0,vmax=255)
        plt.xticks([]),plt.yticks([])
    plt.show()
```

Fig. 29. Python Code for Displaying the 5 Clusters for Each Group of Images