Classifying Wild Edible Flowers by Color Segmentation and Histogram of Oriented Gradients

Le, Cody
DePaul University

CSC 481
Fall 2021
November 18, 2021

Abstract

This analysis explores the extraction of texture features using histogram of gradients (HOG) for the classification of wild edible flowers. Plants such as herbs and edible flowers have many benefits for humans such as health and medical benefits as well as benefits in research, agriculture, and human consumption. The main framework for plant recognition includes pre-processing, feature extraction, and classification. HOG feature extraction has been determined as the best method for texture features in classifying herb leaves. Support vector machines (SVM) has been determined as the best method for multi-class classification in plant leaves. This analysis will classify flowers using support vector machines and compares different HOG parameter sizes as well as different flower class sizes to evaluate the SVM model performance. The preprocessing involves color segmentation in which the images are segmented based on a red or blue color channel and converted to binary to remove background noise. The image is than masked and segmented for feature extraction. 5-class, 7-class, and 9-class flower datasets were created with each dataset containing more similar flowers with the 7-class dataset containing similar color and the 9-class dataset containing similar color and shape. Results showed that SVM can be used with HOG feature extraction to successfully classify flowers. The model with the highest accuracy was the 5-class model. With increase in flower classes, the model accuracy decreased, and the estimated error increased. The best HOG feature size was 64 by 64 and determined as the size that captured the most spatial information in the lowest dimensionality. Although the SVM model was able to classify flowers of similar color, the model was less successful when the flowers are both of similar color and of similar shape and texture.

Introduction

In the last half decade, various research and studies have been conducted on plant recognition systems using image processing, pattern recognition, and machine algorithms for classification. Majority of studies have focused on classifying plant leaves due the large database of leaves in botanical references and the availability of leaves year-round. Besides plant leaves, flowers have also been a topic for classification and are considered a challenge to classify since flowers have a large intra-class variation and small inter-class variation present among different classes (Shaparia et al. 2017). In recent years, the need to develop an automated system for plant recognition has been more prominent due to the scarcity of professional taxonomist and botanist and the desire for everyday individuals to easily identify plants as non-professionals (Kho et. al 2017). Plants have many benefits for human such as health and medical benefits that warrant the need to classify them to prevent extinction or diseases (Haque et al. 2019). Plants such as herbs grow in the wild and the ability to detect their species and properties would benefit research in agriculture, herbarium, and provide understanding of their benefits for human consumption (Ibrahim et al. 2018). In addition, being able to identify species of flowers and recognize its type has various benefits in fields such as farming, floriculture, botany, and ayurvadic treatment. The consensus between current research shows that the main framework for developing a plant recognition system involves three steps: image pre-processing, feature extraction, and classification. Plant recognition systems use various features for classification including shape, morphological features, texture features, color features, or a combination. In this analysis, a classifier model using support vector machines and texture features extracted using histogram of oriented gradients are explored with different feature parameters and different class sizes to classify wild edible flowers. Support vector machines and histogram of oriented gradients have previously been shown to accurately recognize plant leaves, this analysis will determine if the methodology also performs well in recognizing wild edible flowers.

Literature Review / Related Works

The development of a plant leaf recognition system with the ability to classify large number of leaves with a large variety of species is first demonstrated by Trishen Munisami et. Al in their study which examines leaf shape features and color histogram with K-Nearest Neighbor (KNN) classification (2015). Their study is significant as they created their own image database called Folio, with images taken of leaves in broad daylight using a smartphone from farms and local precincts around the University of Mauritius. They captured images for 32 different species with twenty different images per species. They used this database to test and evaluate the effectiveness of an already existing leaf recognition algorithm using KNN. Unlike previous studies, their study produced a client and server application where the client allows a user to upload an image of a leaf to a server, and the server uses the shape descriptor and color information to compare the extracted features to the database and provide the user with a potential match. The methodology for the recognition system required first pre-processing of the image including rotation of the image to proper orientation, conversion to grayscale, thresholding to convert to binary, removal of unwanted noise, inverse thresholding to invert binary background to black, edge detection and filtering to isolate the leaf contours. Second, feature extraction was performed to obtain the convex hull information using boundary points of the leaf, morphological information by calculating the minimum and maximum axes coordinates, distance maps were drawn in horizontal and vertical axes, and centroid radial maps were found by intersecting the diagonal axes through the bounding box around the leaf. All feature information extracted were then converted into ratios for the classifier: aspect ratio, white area ratio, perimeter to area, perimeter to hull, hull area ratio, distance map to x coordinate, distance map to y coordinate, and centroid radial distance. Color histogram is computed for a cropped part of the image which is only used in the second stage of the classifier. Lasty, the KNN algorithm classifies in two stages: the first stage compares the leaf image to the Folio database by returning the three closest matches based on feature ratios and the second stage is for instances where the first stage matches contain different species, the color histogram is then compared to generate a result through correlation coefficient. The results showed an accuracy of 83.5% in the first stage and 87.2% in the second stage. Percent accuracy was obtained by seven types of species and only one species had an accuracy of less than 75%. The study shows that increasing the number of leaves per species up to 15 leaves per species, increased the accuracy. Their work set the bench for the number of images per each flower selected in this analysis. When determining the number of images for each flower, 25 was selected since we know that increasing the number of images may increase accuracy.

The research by Trishen Munisami et. Al is limited due to the images being captured in daylight and without challenging backgrounds. Sonali Agarwal et. Al offers a solution to this by proposing a framework for plant identification specifically for leaf images with complex backgrounds using morphological and texture features (2018). In their proposal, they demonstrate that using multiclass classifier with Support Vector Machines (SMV) and a Structured Matrix Decomposition (SMD) algorithm to identify the region of interest results in an efficient and accurate leaf recognition system. Their proposal validates the three-step needed for leaf recognition which is used in all other studies: image pre-processing, feature extraction, and classification. Their methodology begins with image pre-processing in which the leaf object is isolated using SMD and thresholding is performed for binarization. Next, feature extraction is performed to obtain morphological features including area of leaf, perimeter of leaf, convex hull of leaf, area of convex hull, perimeter of convex hull, rectangularity, and circularity. Texture features are extracted as a two-dimensional surface to obtain the spatial structure (pattern) and the contrast (amount of texture). Lastly, classification is performed using a multilevel approach which presents details of classes at each level. The method was performed on the Genus Acer dataset using images from 13 species containing 25 images with challenging issues such as shadow, rotation, and complex natural backgrounds. The results were excellent at 95% accuracy showing that the combination of morphological and texture feature extraction with SMV provides efficient and good results as a leaf recognition technique especially with non-white background images of leaves. This study shows that SMV is most effective for classifying leaf images with non-white and complex backgrounds. Their approach to creating classifier models will be applied in this analysis which also seeks to classify images with complex backgrounds.

Sonali Agarwal et. Al contends that a multiclass SMV provides the most accurate and efficient algorithm for classification of leaves, but a study by Kho et. Al for the automation of plant identification compared SMV to Artificial Neutral Networks (ANN) and shows that ANN has a slight edge in performance (2017). Their study is significant because they performed their classification on leaves of the same species with very similar shape features to compare and see if SMV or ANN would be better recognition models. Their goal was to develop a baseline automated system to identify three species of the Ficus plant which have similar leaf morphology. For their study, 60 herbarium leaf images were used from the Ficus plant from three of their species with 20 images selected for each: F.benjamina, F.pellucidopunctata, and F.sumatrana. Images selected only obtained one leaf object, with leaves perfectly intact, and free from any damages. First, they start with image pre-processing by removing any shadows in the image, then convert the image to grayscale and used histogram to uniform the distribution, and then convert to binary. Next, a median filter was used to reduce image noise, segmentation was performed to isolate the leaf object, and a Canny edge detector is used to detect the edges of the leaf. Elements inside the leaf were removed by applying a binary gradient mask and a horizontal structuring element to till in the interior gaps. Second, feature extraction was performed, and four different types of features were extracted: morphological features, Hu moment invariants features, texture features, and histogram of oriented gradients (HOG). Morphological features include area, perimeter, eccentricity, minor, and major axes. Hu moment invariants include seven moment features that describe rotation, translation, and scaling. Texture is extracted by gray level co-occurrence matrix analysis and HOG is computed as a descriptor to complete all numeric feature extractions. Lastly, ANN models were trained using a 3-layer feed-forward network with only one hidden layer to avoid overfitting. Images were divided into three blocks: 70% training, 10% validation, and 20% testing. SVM models were trained using a radial basis function kernel and images were divided into two blocks: 80% training and 20% testing. The models were compared to each other using three criteria: accuracy, receiver operating characteristics (ROC), and area under the curve (AUC). The results showed that the ANN and SVM models performed equally well and accurately identified 10 out of 12 test samples. The accuracy of both models was 83.3% with ANN having a higher value in identifying F.benjamina and F.pellucidopunctata while SMV had a higher value of identifying F.sumatrana. Although the accuracies are the same, when evaluating the ROC curve and the AUC curve for both models, the results showed that ANN slightly performed better than SVM. This study used the most types of features for feature extraction than any other study and had a more complex classifier with ANN, which results in a more complex model. In model selection, the goal is to select the simplest model that is efficient and yields the highest accuracy. With their ANN model having a similar accuracy result to their SMV model, they do not successful show or explain why an ANN model is overall better or should be used over other models. In this view, the model presented in Sonali Agarwal et. Al's study is considered the more efficient model. Their model would just need to be tested using images of similar leaf shape and plant genus to validate their model.

Neutral network classifiers have also proven to be effective for flower classification specifically with color and texture features as explored by Shapaira et. Al (2017). Their study looks at classifying five classes of flowers (daffodil, snowdrop, lily valley, crocus, and tiger lily) with 40 images per flower from the oxford flower dataset. Their process starts with segmentation by threshold which they apply Otsu's thresholding method which divides the image into foreground and background for segmentation. Noise is eliminated from the resulting binary image by creating a bounding box where all pixels outside of the boundaries are filtered to black. After segmentation, feature extraction is applied using Gray Level Co-occurrence Matrix (GLCM) method which calculates variation in intensity at interested pixels of a grayscale image. Once the GLCM is calculated, properties associated with the GLCM such as correlation, contrast, energy, and local homogeneity can be extracted as features. Color feature is extracted using color moments which are rotation and scaling invariant and provide information about shape and color. Color moment is computed based on RGB, HSV, or CMYK space with each color model having three color moments, resulting in 9 moments in RGB space, in this case, 3 different means and 3 different standard deviations as features. Moreover, eight GLCM values and six color moment are inputted as vectors and given to the neural network for classification. The results show a 95% accuracy when both GLCM and color moment are combined as features. When only GLCM is used, the accuracy is low at 40% and when only color moment is used, the accuracy is 65%. Moreover, the method for flower classification is effective but only when both texture and color features are used in the classifier. The segmentation method used in this study will be implemented in part in this analysis since Otsu's thresholding method is an effective method for image binarization.

Plant recognition models do not need to be complex and instead simpler models have been demonstrated using only one type of feature extraction with high accuracy results. Zaidah Ibrahim et. Al presents such a model using only texture feature and evaluate the three most popular texture feature methods using an SMV classifier in their study (2018). Their study looks at identifying herbal plants using texture only since herbal plants may appear with similar or different colors depending on where it was grown. Their study is unique as previously there has not been studies comparing different texture feature methods to each other for leaf recognition. The texture features evaluated are histogram-oriented gradients (HOG), local binary pattern (LBP), and speeded-up robust features (SURF). Texture features are identified by modeling texture as a two-dimensional gray-level variation within a segmented region. Their methodology begins first with resizing the image and converting to grayscale. Then, performing HOG which counts the occurrence of the gradient orientation in a. localized region, LBP which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number, or SURF which is achieved by relying on integral images and Hessian matrix-based measures.

Lastly, a one-against-one approach for multiclass SVM was used as the classifier which consists of multiple binary, linear SMV learners. Their method was performed on the Flavia dataset which consists of 40 images for training and 10 images for testing from 33 different plant species and performed on their own dataset created using 10 different herb plants with 20 images for each species for training and 5 images for testing. The classification results for HOG and LBP were similar at 99% for the herb plant dataset and 97% for the Flavia dataset. SURF did not perform as well with an accuracy of 75% on the herb plant dataset and 63% on the Flavia dataset. One explanation they provided for the difference in accuracy is that HOG and LBP operate on localized cells which uphold invariance to geometric and photometric transformations and less affected by illumination, whereas SURF is more affected by illumination variations. This study is interesting because the methodology is very simple and yielded very accurate results especially with HOG and LBP. This study is of particular interest since it examines herbs which are edible plants. This study will be used a bench for this analysis and this analysis will explore HOG feature extraction in detail also with a one-against-one approach for multiclass SVM model. This analysis will include flower images with complex backgrounds and will evaluate different SVM models based on varying number of classes.

The review of recent studies on plant recognition systems reveals that complex systems for a more automated approach use ANN or SVM algorithms with at least two feature types and KNN algorithms have also been used successfully to classify large datasets. Simple models using one feature type have also proved to be successful at leaf recognition but may not be useful for automation or for large datasets. The main issues today are leaf and flower mages with lighting variations, images with complex backgrounds, and wide variety of species of plants that make for a robust training and testing for a recognition system more complex and not as easily feasible. This analysis will focus on flower images with complex backgrounds and will apply one feature extraction using HOG with a multi-class SVM classifier model. The models will be trained and tested using a 5-class, 7-class, and 9-class dataset with images of flowers with similar color and shape to determine if the models are robust using the texture feature to accurately predict the flower classes.

Data and Images

The images are taken from the Wild Edible Plants Dataset from Kaggle. The dataset contains 35 classes of wild edible plants with various number of images per class. Images in the dataset were obtained through Flickr and free to use by all members of the public. For this analysis, only wild edible plant flowers will be used for classification. Flower images were chosen for classification since previous works have focus on plant leaves with positive outcomes, but few studies have been conducted on flowers specifically using histogram of gradients (HOG) for feature extraction and support vector machines (SMV) for multi-class classification. Few studies have focuses on more than 5 flower classes or flower classes with similar color, shape, or texture. In this analysis, flower images selected will have similar color and shape and SVM models will be implemented on a 5-class, 7-class, and 9-class dataset. Nine types of wild edible flowers were selected for the analysis which can be separated into three color groups visually:

Yellow Flowers:	Dandelion, Calendula, and Coltsfoot
White Flowers:	Daisy and Gardenia
Violet/Pink Flowers:	Geranium, Knapweed, Chicory, and Common Mallow

25 images were selected for each flower type. Only images with full flower head and flower petals were selected. Flowers were oriented in various positions in the images. Images contained various backgrounds including dark backgrounds, blurred backgrounds, backgrounds with greenery, and natural backgrounds with flowers in their natural habitat. The flowers were separated into three datasets: 5-class, 7-class, and 9-class. The 5-class dataset contains a total of 125 flower images with 5 classes. The 7-class dataset contains a total of 175 flower images with 7 classes. The 9-class dataset contains a total of 225 flower images with 9 classes.

5-Class Dataset:	Daisy, Dandelion, Gardenia, Geranium, Knapweed
7-Class Dataset:	Calendula, Chicory, Daisy, Dandelion, Gardenia, Geranium, Knapweed
9-Class Dataset:	Calendula, Chicory, Coltsfoot, Common Mallow, Daisy, Dandelion, Gardenia, Geranium, Knapweed

Methodology / Overview of Classification System

The methodology used in this analysis follows the main framework for developing a plant recognition system and involves three steps: image pre-processing, feature extraction, and classification. The images are pre-processed and cropped to be equal length and width. Then, the images are segmented to remove noise, eliminate the background, and resize the image to 400 by 400 pixels. Texture feature is extracted using HOG feature parameters. Extracted features at different feature parameter sizes are then used in a one-to-one multi-class SVM model. Models are than compared and evaluated for accuracy across the different class sizes and datasets.

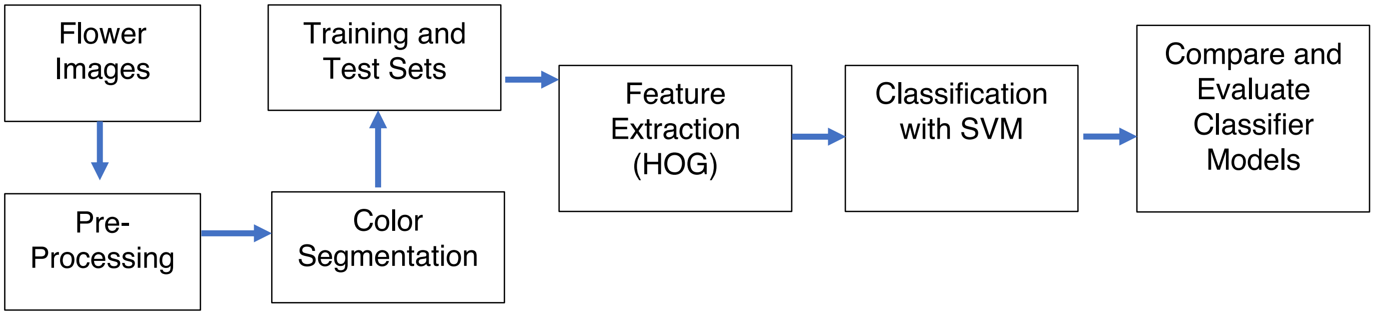


Figure 1.1 Diagram of Methodology Overview

Pre-Processing and Color Segmentation

All images are pre-processed to standardize the size and shape of all images and to segment the flowers for better feature extraction. First, all images are cropped to have equal length and width. Next, the segmentation process begins with one flower class at a time. Flowers are segmented using a red color or blue color channel. Selected flowers visually display a color that respond well to one of the two color channels which is used to separate the flower from the image background. Flowers segmented using the red color channel include Daisy, Dandelion, Calendula, and Coltsfoot. Flowers segmented with the blue color channel include Gardenia, Geranium, Chicory, Knapweed, and Common Mallow. The red or blue color channel is used since all the flower images have mostly green or dark green backgrounds, which when converted into a red or blue color channel, the greens will read as background and the flower will read as foreground. The segmentation process starts with converting the images to grayscale using the red color or blue color channel. Next, grayscale images are converted to binary images using Otsu's method. Otsu's method is an automatic image thresholding method that returns a single intensity threshold separating two class pixels to foreground and background. Next, the holes in the resulting binary image are filled in using a region-fill method that targets holes in the object. Region-filling is a method to fill a bounded region of an image to match the pixel of that region with the surrounding pixels. Then, area opening is performed on the binary image to remove any noise or small objects in the background. Area opening is an opening method that removes all connected objects that have fewer than a set number of pixels from the image. In this case, the set number of pixels chosen is 400. Next, the original image is masked onto the cleaned binary image using an element-wise binary operation function. Lastly, the image is resized to a 400 by 400-pixel size using nearest-neighbor interpolation. Nearest neighbor approach interpolates the image by determinizing the nearest neighboring pixel and assumes the intensity value of that pixel. This approach is the simplest and was chosen for processing efficiency.

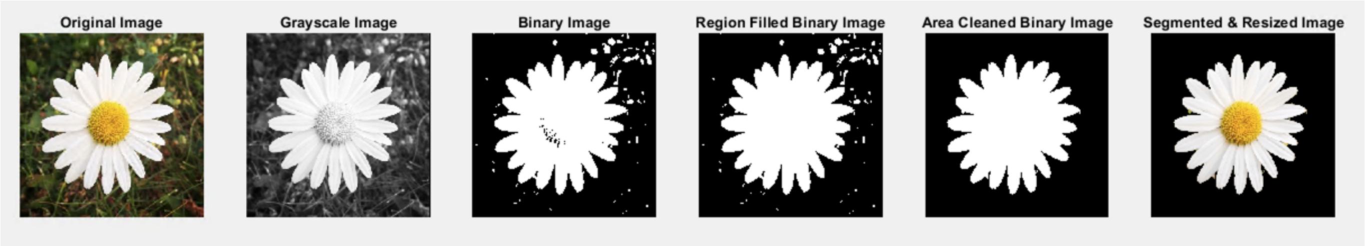

Original Image
Grayscale Image
Binary Image
Region Filled Binary Image
Area Cleaned Binary Image
Segmented & Resized Image

Figure 2.1 Segmentation Process for Daisy using Red Color Channel

Figure 2.1 shows the pre-processing and segmentation process for a sample daisy flower image. First the original image is converted into grayscale. Next, the binary image shows holes in the center of the object and noise in the background. Region-fill method in the next image fills in the holes in the object. Then, area opening cleans the noise in the background in the next image. The resulting segmented image displays the daisy in color with the background removed and cleaned. Below Figure 2.2 shows the same pre-processing and segmentation process for a sample chicory flower image. This image had leas noise in the background and foreground and in the third image the binarized image already has noise removed. Since the same segmentation process is applied to all images, it is acceptable to perform the region-fill and the area open as it does not affect the image negatively and the result of the segmented image is still achieved.

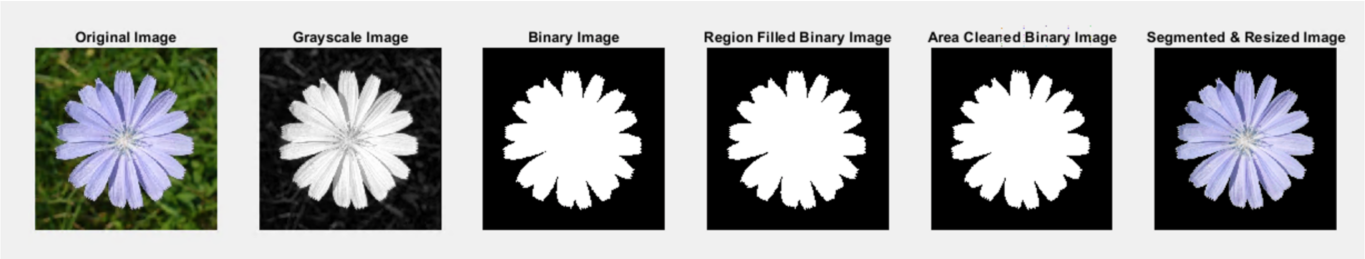

Original Image
Grayscale Image
Binary Image
Region Filled Binary Image
Area Cleaned Binary Image
Segmented & Resized Image

Figure 2.2 Segmentation Process for Chicory using Blue Color Channel

Training and Testing Data Sets

To build the SVM classifier model, the model must be built on training features and then the model is used to predict test features. The images will be split into training images and testing images. The images are split randomly with 80% for training and 20% for testing. The 80/20 training and testing split is selected to mirror the method for splitting data described in Sonali Agrawal et. Al's study. Using the 80/20 split, 20 flower images are selected for training and 5 flower images are selected for testing. The 5-class dataset contains five flowers: daisy, dandelion, gardenia, geranium, and knapweed. This dataset contains flowers with different colors, different shapes, and different textures. 20 of the images are selected as training images for a total of 100 training images and 5 images are selected as testing images for a total of 25 testing images.

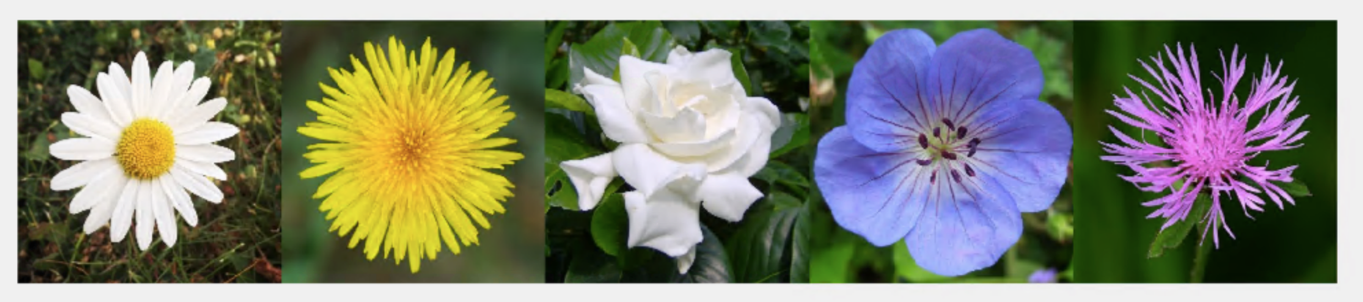


Figure 3.1 Five-Class Dataset Sample Original Images.
From Left-to-Right: Daisy, Dandelion, Gardenia, Geranium, and Knapweed.

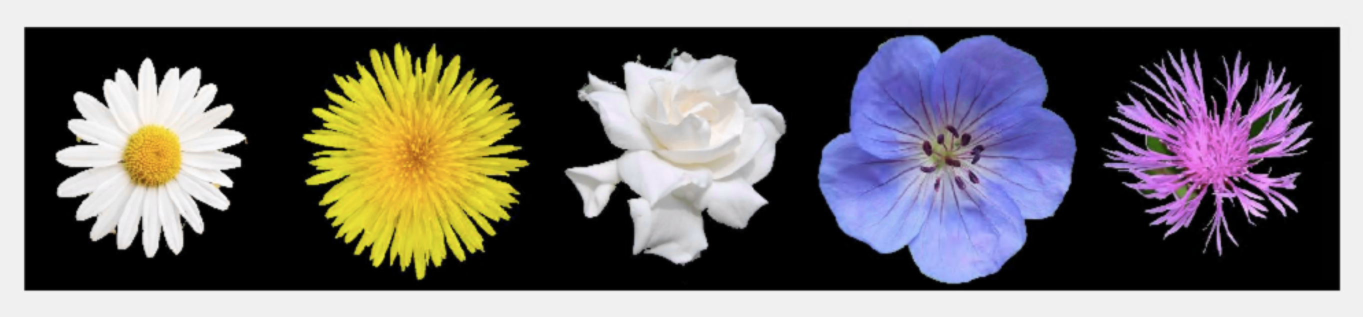


Figure 3.2 Five-Class Dataset Sample Segmented Images.
From Left-to-Right: Daisy, Dandelion, Gardenia, Geranium, and Knapweed.

The 7-class dataset contains seven flowers: calendula, chicory, daisy, dandelion, gardenia, geranium, and knapweed. This dataset introduces two flowers that are similar in color, calendula and dandelion are both yellow flowers and chicory and geranium are both violet flowers. The purpose of this dataset is to evaluate if the SVM classifier model can still classify accurately with the additional of flowers with similar color. 20 of the images are selected as training images for a total of 140 training images and 5 images are selected as testing images for a total of 35 testing images.

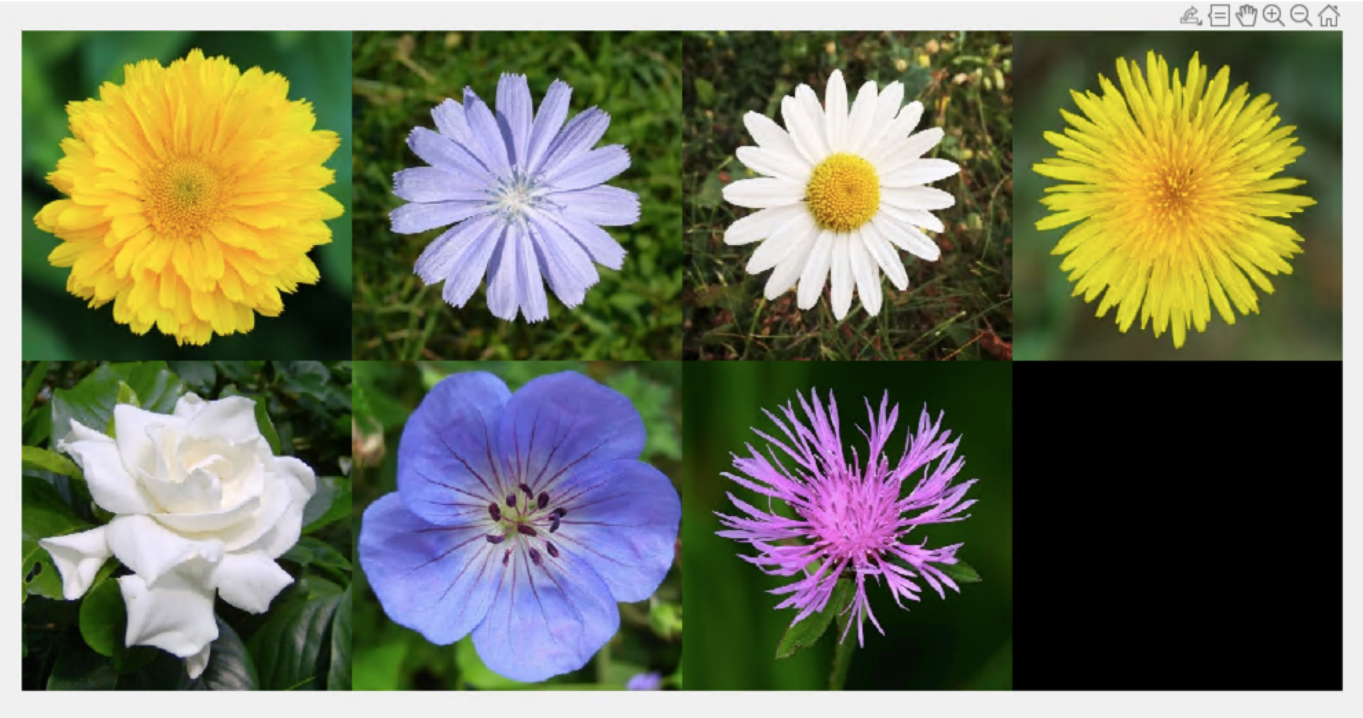


Figure 3.3 Seven-Class Dataset Sample Original Images.
From Left-to-Right 1st Row: Calendula, Chicory, Daisy, Dandelion
From Left-to-Right 2nd Row: Gardenia, Geranium, and Knapweed

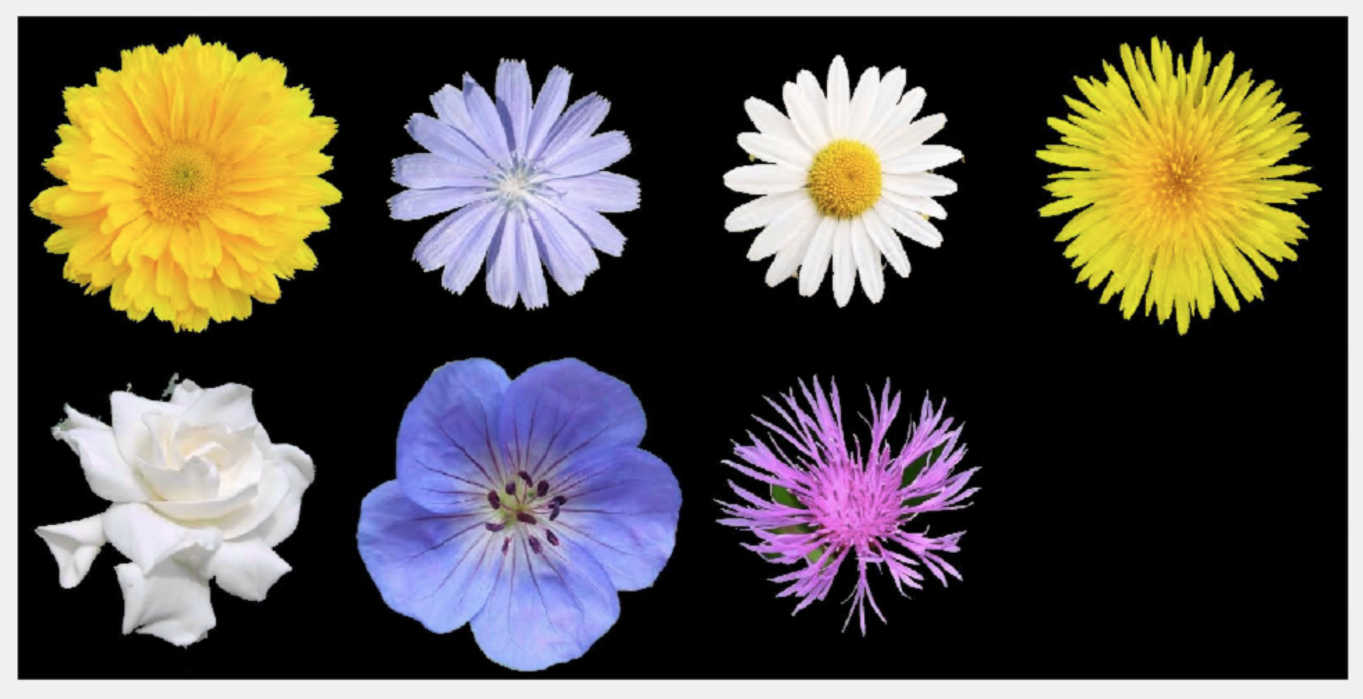


Figure 3.4 Seven-Class Dataset Sample Segmented Images.
From Left-to-Right 1st Row: Calendula, Chicory, Daisy, Dandelion
From Left-to-Right 2nd Row: Gardenia, Geranium, and Knapweed

The 9-class dataset contains nine flowers: calendula, chicory, coltsfoot, common mallow, daisy, dandelion, gardenia, geranium, and knapweed. This dataset introduces two additional flowers that are both similar in color and texture from existing flowers. Coltsfoot is also a yellow flower with similar shape to dandelion. Common mallow is a pink flower that has a similar color to knapweed and a similar shape to geranium. This dataset has three yellow flowers, two white flowers, two violet flowers, and two pink flowers. The purpose of this dataset is to evaluate if the SVM classifier model can still classify accurately with the additional of flowers with similar color and shape. 20 of the images are selected as training images for a total of 180 training images and 5 images are selected as testing images for a total of 45 testing images.

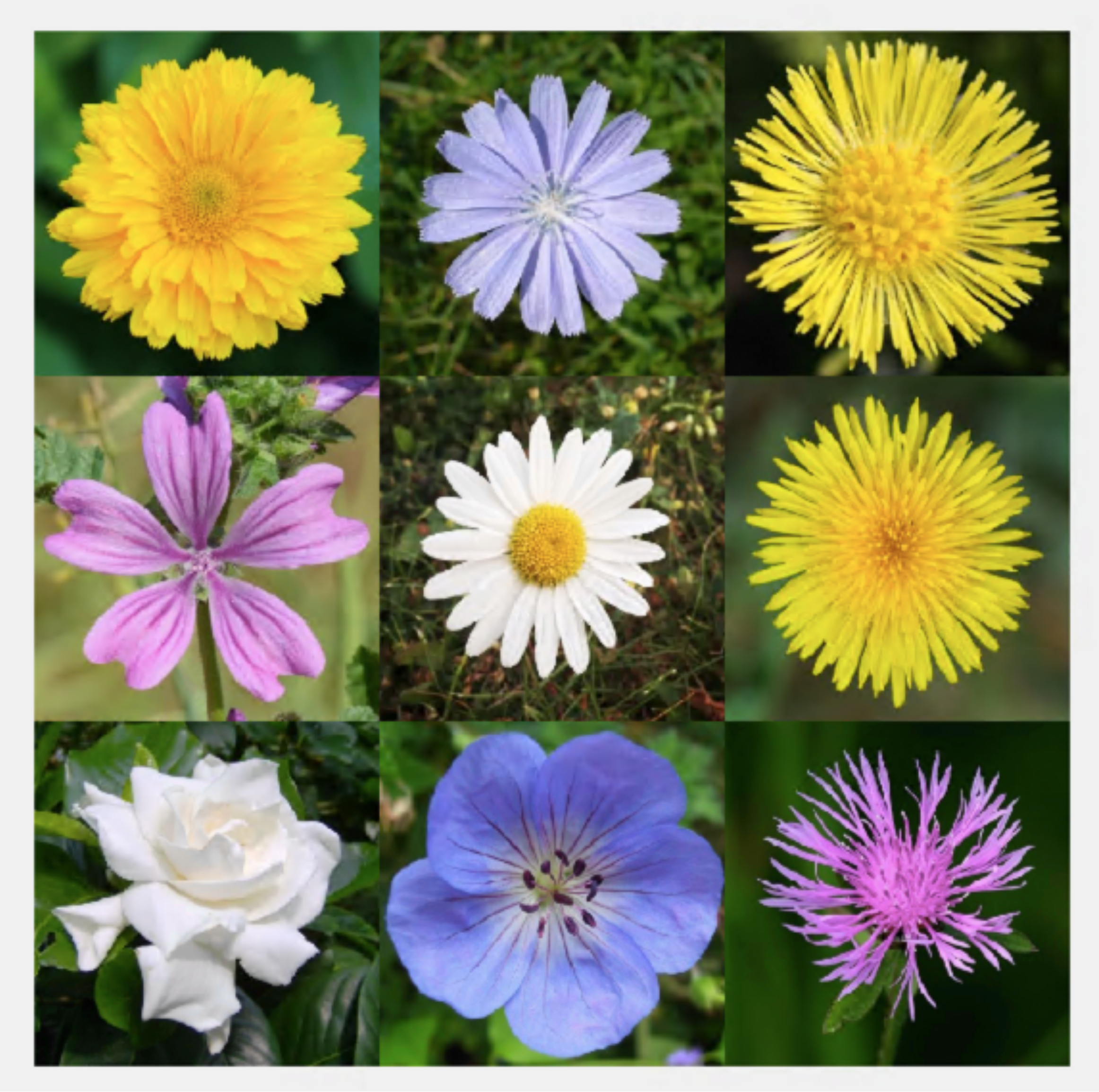


Figure 3.5 Nine-Class Dataset Sample Original Images.
From Left-to-Right 1st Row: Calendula, Chicory, Coltsfoot
From Left-to-Right 2nd Row: Common Mallow, Daisy, Dandelion
From Left-to-Right 3rd Row: Gardenia, Geranium, and Knapweed

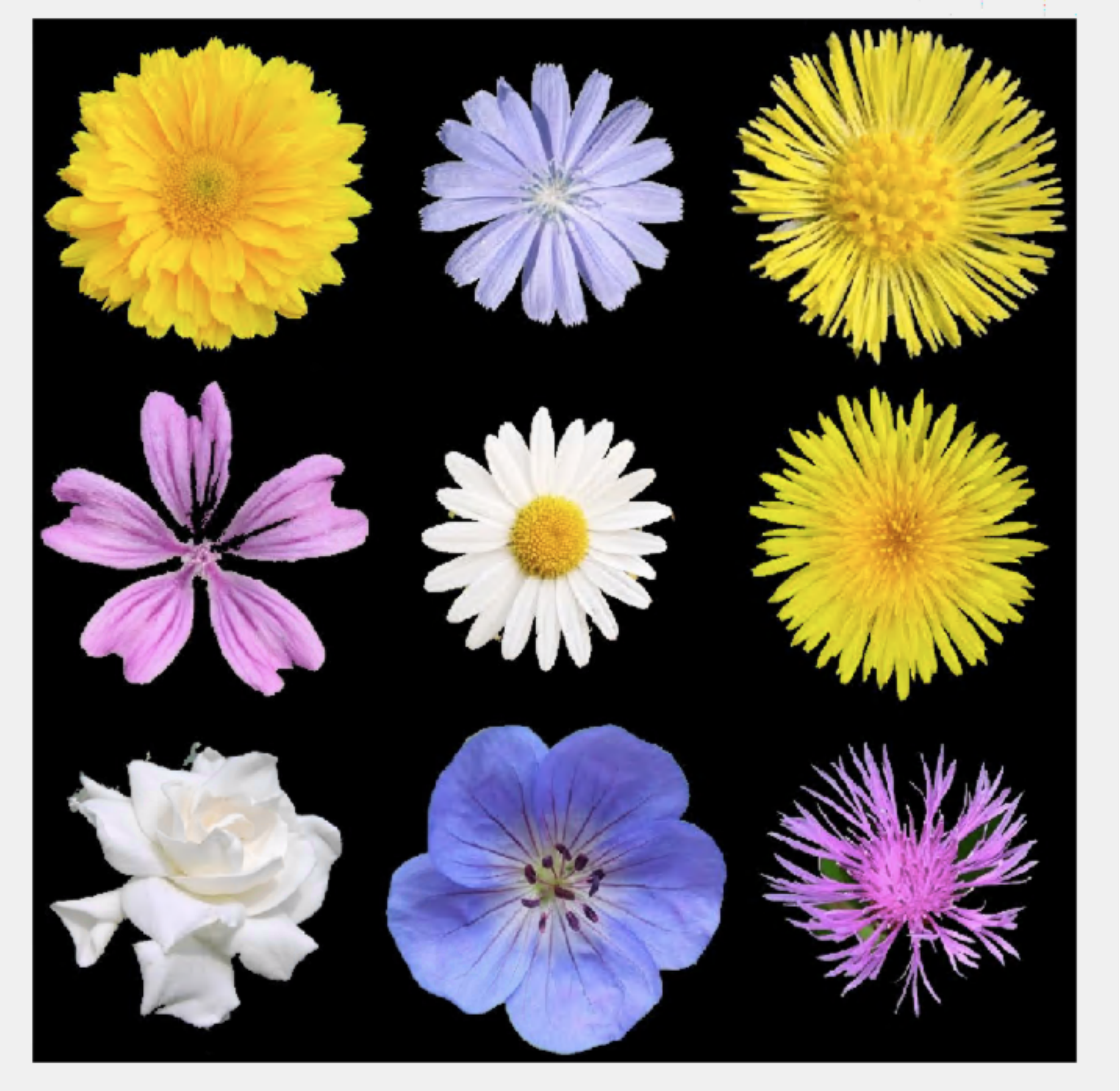


Figure 3.6 Seven-Class Dataset Sample Segmented Images.
From Left-to-Right 1st Row: Calendula, Chicory, Coltsfoot
From Left-to-Right 2nd Row: Common Mallow, Daisy, Dandelion
From Left-to-Right 3rd Row: Gardenia, Geranium, and Knapweed

Feature Extraction with Histogram of Gradients (HOG)

Texture features provide measurements for visual patterns in an image. Specific textures in an image can be calculated by modeling texture as a two-dimensional gray-level variation within a segmented region. Zaidah Ibrahim et. Al verifies in their research that histogram of gradients (HOG) as a texture feature extraction method performs most optimal with a one-to-one SMV classifier model. As such, this analysis extracts texture features using HOG with various parameter sizes. HOG is a feature descriptor that counts occurrences of gradient orientation in a localized portion of an image. HOG focuses on the structure or shape of the object. HOG is similar in approach to edge orientation histograms. Edge orientation histograms builds histograms with the directions of the gradients as the edges which becomes the borders or contours. HOG is considered a more optimal method than all other edge descriptors as it uses magnitude as well as angel of the gradient to compute features. To extract HOG features, first the HOG parameter size must be determined. HOG cell size can range from any m by n matrix size. For this analysis, first the HOG cell size is applied on a sample training image. Then HOG features are extracted at each cell size on the same image. Lastly, the HOG features for each cell size are plotted and visualize to evaluate the number of features extracted at each cell size and determine if a shape and object is created in the resulting visualization.

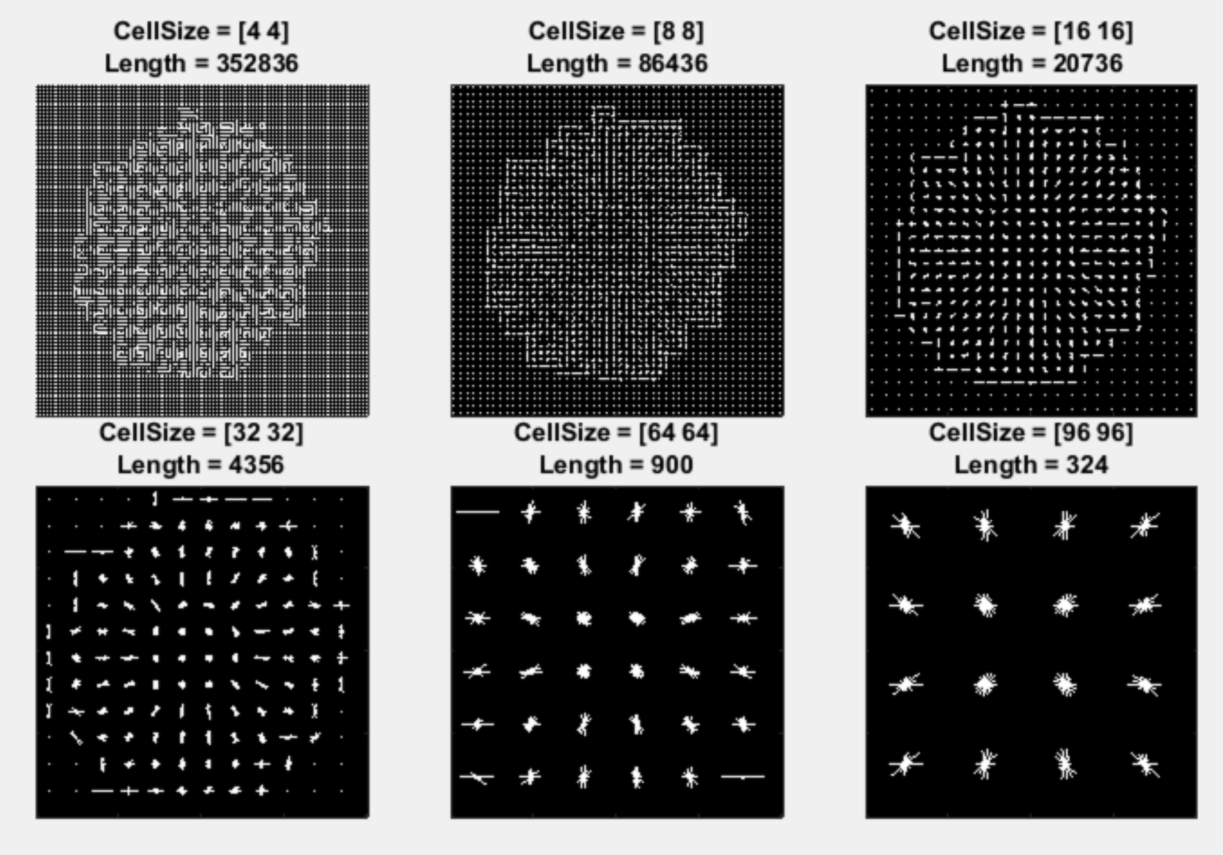


Figure 4.1 Visualization of extracted HOG features at various HOG Parameter Sizes

Figure 4.1 shows the visualization of the HOG features on a sample training image at various HOG parameter sizes. Selecting the most optimal cell size depends on which cell size can best capture the spatial information with the lowest dimensionality. Cell Size 8 by 8 displays the shape and details of the object but the feature length is very large, and the dimensionality would be too high. Cell Size 96 by 96 no longer displays the object shape and feature length may be too small. Cell Size 32 by 32 and 64 by 64 still retains the object shape. Cell Size 32 by 32 still has a large feature size whereas Cell Size 64 by 64 has lower feature size. The optimal cell size is determined by trial and error. In the analysis, three different cell sizes will be used and compared with the results of the SVM classifier to determine which cell size provided the most optimal performance.

Classification with Support Vector Machines (SVM)

Support vector machines (SMV) are supervised learning models that map training samples to points in a hyperspace to maximize the boundaries between classes. Test samples are then mapped into the same hyperspace and predicted to belong to a class based on which side of the boundary the sample falls in. This analysis implements a one-to-one encoding scheme for multi-class classification. The one-to-one approach allows for points to be split at each class at each level which is optimal for the classification of datasets with multiple classes. Since SVM is originally a binary classification algorithm, the one-to-one approach allows for multiple binary SMV learners and has been shown as an effective approach for multi-class classification. This approach is the same approach used in Sonali Agarwal et. Al and Zaidah Ibrahim et Al's research and will be implemented in this analysis to evaluate model performance. The SVM classifier models will be built with 32 by 32, 64 by 64, and 96 by 96 HOG feature parameters for the 5-class dataset, 7-class dataset, and 9-class dataset. Three models for each dataset for a total of nine models. For each dataset first the HOG features are extracted from the training set for the specified cell size. Then, the SVM model is fit using the training set features and the training set labels. Next, the HOG features are extracted from the testing set using the same specified cell size. Then, the SVM model is used to predict the testing set features. Next, a confusion matrix is calculated to cross-tabulate the predicted versus the actual labels. A confusion matrix is a common method to describe the performance of a classification model on a set of test data for which the true values are known. The confusion matrix provides accurate insight into how correctly the model has classified the classes or how the classes are misclassified. The process is repeated for each HOG parameter cell size and for each dataset.

Model Evaluation

The 5-class, 7-class, and 9-class dataset will each have three SVM classifier models. The three models using different HOG parameter sizes will be evaluated to determine which HOG parameter size had the most optimal performance. Then, the best model from each of the datasets will be compared to determine which SVM classifier model performed the best in terms of accuracy. Accuracy will be determined by the confusion matrix. From the confusion matrix, accuracy can be calculated by divided diagonal of the confusion matrix by the total number of attributes. To evaluate the models, k-fold cross validation will be implemented, and the estimated loss will be calculated using cross-validation. The loss function provides the estimated error which performs a 10-fold cross validation error estimate on a specified model and provides the misclassification rate. The estimated accuracy from cross-validation can be calculated by subtracting 1 from the error estimate. The estimated accuracy from cross-validation may or may not be different from the accuracy from the confusion matrix. If accuracy is similar, the model is stable and likely the same accuracy results can be achieved with each new implementation.

Results

First, the 5-class dataset with daisy, dandelion, gardenia, geranium, and knapweed had an accuracy of 92%. The estimated error using cross-validation was 11%. The estimated accuracy with cross-validation was 89%.

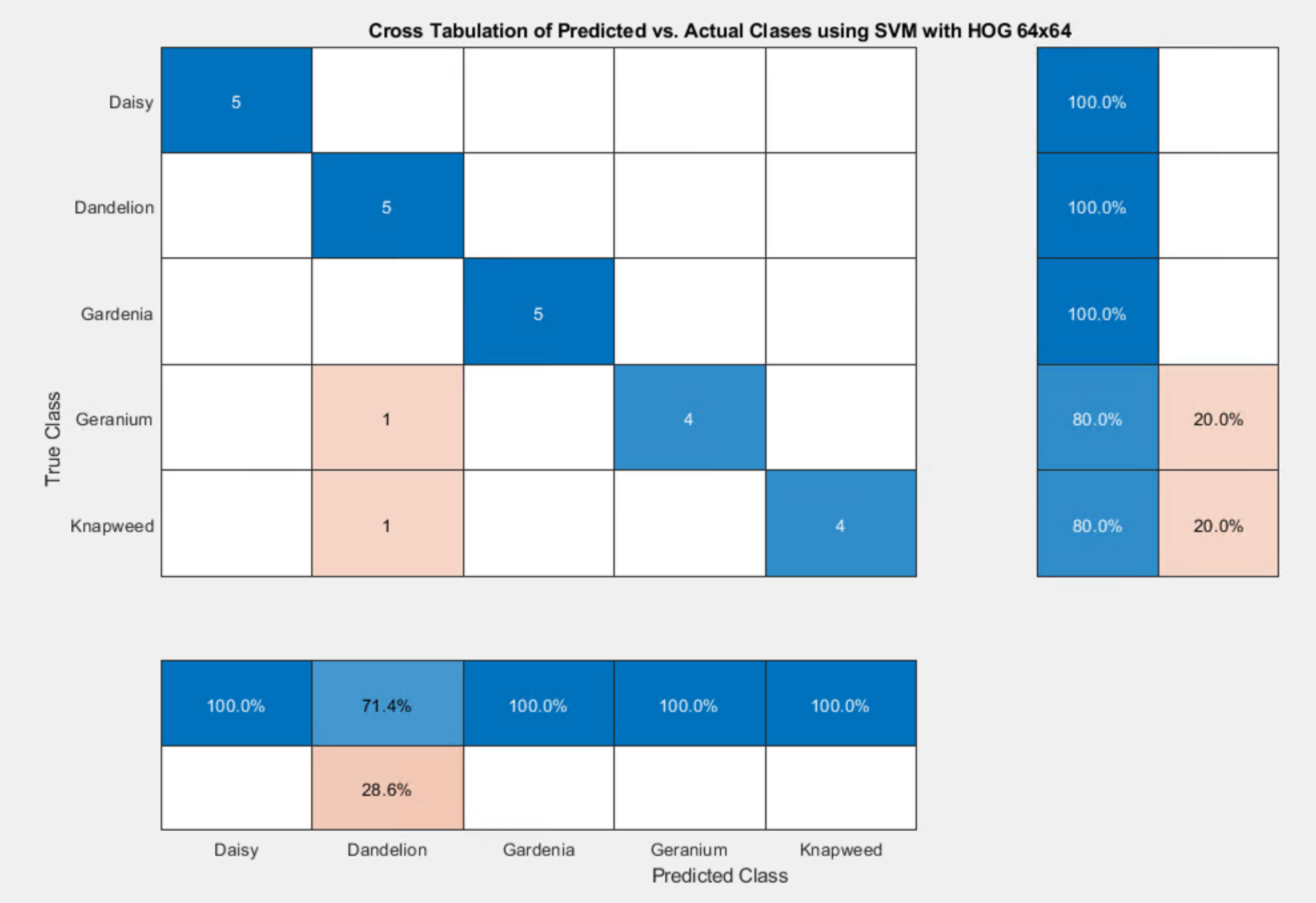


Figure 5.1 Confusion Matrix for 5-Class SVM Model with HOG [64 64]

HOG Feature Size	Model Accuracy	Estimated Error with Cross-Validation	Estimated Accuracy with Cross-Validation
64 x 64	92%	11%	89%
96 x 96	88%	17%	83%
32 x 32	84%	10%	90%

Figure 5.2 5-Class Model Accuracy with HOG Feature Size Comparison

The 5-class SVM classifier performed well. The model inaccurately predicted dandelion but was able to predict all other classes. The estimated accuracy from cross-validation has a similar accuracy score which shows that this model is an optimal and stable model. The HOG parameter size that had the best optimal performance was a cell size of 64 by 64. At 64 by 64, accuracy was the highest. However, interestingly, 32 by 32 although had a lower model accuracy, had a lower estimated error with cross-validation compared to all other cell sizes.

Second, the 7-class dataset with calendula, chicory, daisy, dandelion, gardenia, geranium, and knapweed had an accuracy of 91.43%. The estimated error using cross-validation was 22.86%. The estimated accuracy with cross-validation was 77.14%.

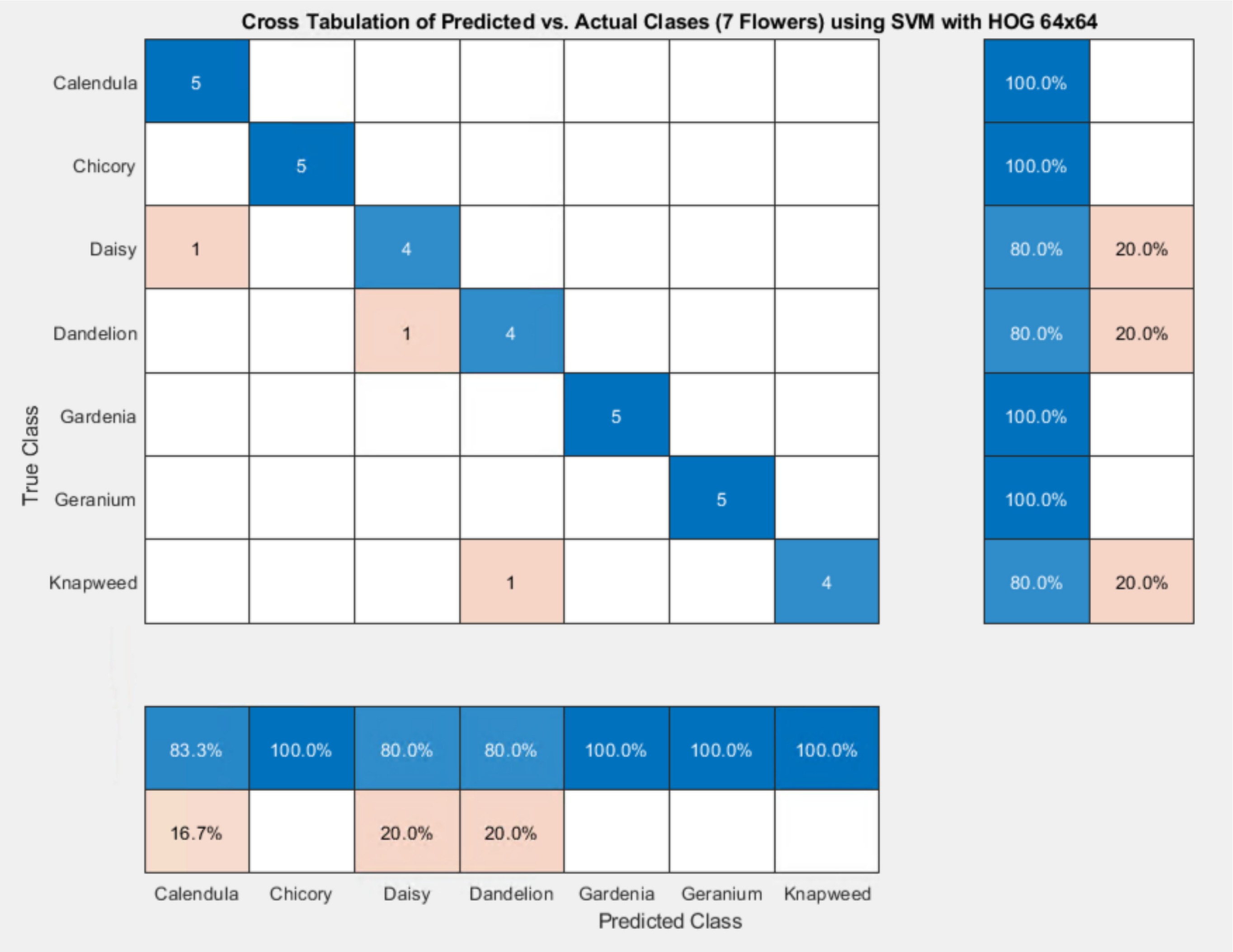


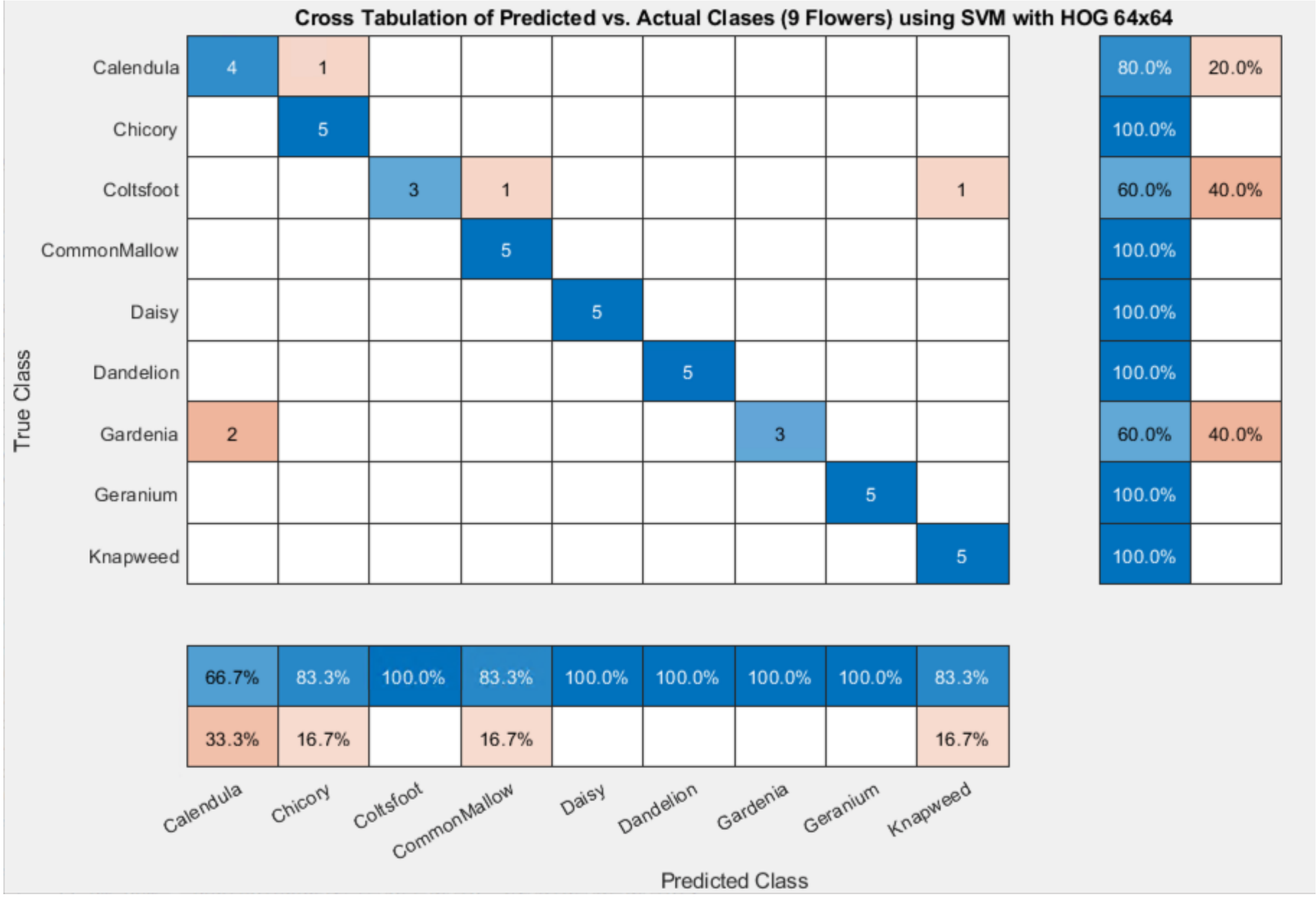
Figure 5.3 Confusion Matrix for 7-Class SVM Model with HOG [64 64]

HOG Feature Size	Model Accuracy	Estimated Error with Cross-Validation	Estimated Accuracy with Cross-Validation
64 x 64	91.43%	22.86%	77.14%
96 x 96	82.86%	30%	70%
32 x 32	82.86%	30.71%	62.29%

Figure 5.4 7-Class Model Accuracy with HOG Feature Size Comparison

The 7-class SVM classifier performed equally well as the 5-class model. The HOG parameter size that had the best optimal performance was a cell size of 64 by 64. At 64 by 64, accuracy was the highest. The 7-class model has a much higher estimated error from cross-validation compared to the 5-class model. The model inaccurately predicted the yellow flowers which were introduced in this dataset as similar colored flowers. The estimated accuracy from cross-validation has been much lower than the model accuracy which indicates that the model may not predict with precision compared to the 5-class model and results may not be repeated as the results shown above. The HOG feature size of 32 by 32 and 96 by 96 performed fairly the same for this model with little difference.

Third, the 9-class dataset with calendula, chicory, coltsfoot, common mallow, daisy, dandelion, gardenia, geranium, and knapweed had an accuracy of 88.89%. The estimated error using cross-validation was 31.11%. The estimated accuracy with cross-validation was 68.89%.



Cross Tabulation of Predicted vs. Actual Clases (9 Flowers) using SVM with HOG 64x64

True Class \ Predicted Class	Calendula	Chicory	Coltsfoot	CommonMallow	Daisy	Dandelion	Gardenia	Geranium	Knapweed		
Calendula	4	1								80.0%	20.0%
Chicory		5								100.0%	
Coltsfoot			3	1					1	60.0%	40.0%
CommonMallow				5						100.0%	
Daisy					5					100.0%	
Dandelion						5				100.0%	
Gardenia	2						3			60.0%	40.0%
Geranium								5		100.0%	
Knapweed									5	100.0%	
	66.7%	83.3%	100.0%	83.3%	100.0%	100.0%	100.0%	100.0%	83.3%		
	33.3%	16.7%		16.7%					16.7%		

Figure 5.4 Confusion Matrix for 9-Class SVM Model with HOG [64 64]

HOG Feature Size	Model Accuracy	Estimated Error with Cross-Validation	Estimated Accuracy with Cross-Validation
64 x 64	89.89%	31.11%	68.89%
96 x 96	77.78%	36.11%	63.89%
32 x 32	80%	33.89%	66.11%

Figure 5.4 9-Class Model Accuracy with HOG Feature Size Comparison

The 9-class SVM classifier performed the weakest compared to the 7-class and 5-class model. The HOG parameter size that had the best optimal performance was a cell size of 64 by 64. At 64 by 64, accuracy was the highest. The 9-class model also has the highest estimated error from cross validation and the lowest estimated accuracy score. The model inaccurately predicted the yellow, violet, and pink flowers added as additions to this dataset which are all flowers with similar color and shape. This model shows that although the SVM classifier performs will with multiclass classification, the higher the number of classes, the lower the model accuracy and the higher the estimated error. Compared to previous models, this model at 32 by 32 cell size performed better than at 96 by 96 cell size. In the 7-calss and 5-class models, 96 by 96 cell size outperformed 32 by 32.

Lastly, the three best models using the 64 by 64 HOG feature size from the 5-class, 7-class, and 9-class are compared. The best model for flower classification is the 5-class model which had the highest model accuracy and lowest estimates error. The 7-class model still performed equally as well as the 5-class model in terms of model accuracy, but the estimated error was double compared to the 5-class model. The 9-class model had the lowest performance with the lowest model accuracy, highest estimated error, and lowest estimated accuracy from cross-validation. The results some that as the number of classes increases, model accuracy decreases. The SVM classifier can classify flowers with similar colors but may not be able to accuracy classify flowers with both similar color and similar shape or texture. The images that were most misclassified were images of flowers that had similar color and shape to other flowers in the respective dataset.

Conclusion

This analysis performed classification on wild edible flowers using color segmentation and texture features extracted using histogram of gradients (HOG). The analysis evaluated different HOG feature sizes and created classifier models using support vector (SVM) machines for each feature size. The analysis focused on comparing different number of classes and evaluated the SVM models to determine model performance for 5-class, 7-class, and 9-class datasets. The results confirm that SVM using a one-to-one approach for multi-class classification performed well for the classification of flowers using texture features. The best model accuracy was 92% for 5-class flower classification. HOG feature extraction can be used successfully to classify flowers. The best HOG feature size for this analysis was 64 by 64. Determining the best feature size depends on the input image and the best feature size will retain enough spatial information to recognize an object shape when visualizing the HOG extracted features but with lowest dimensionality. The accuracy of the SVM classifier models decrease as the number of classes increases. With more classes in the model, the estimated error or loss increases. Although the SVM model was able to classify flowers of similar color, the model was less successful when the flowers are both of similar color and of similar shape and texture.

This analysis is limited in several ways. The sample size has a relatively low number of images. The images were all close-up photos of flowers. The flowers selected read well in a red or blue color channel. The flower images had backgrounds that were dark, blurred, or mostly of greenery which may have segmented well using color segmentation due to these constraints. Future exportation could explore a smaller number of classes but with all flowers being the same color and shape or explore training a larger number of training samples so that the classifier has more data to better optimize for classification.

Works Cited

Agarwal, Sonali, Anand Singh Jalal, and Mohd Khan. "Plant Identification using Leaf Image Analysis." *Proceedings of 3rd International Conference on Internet of Things and Connected Technologies (ICIoTCT)*. 2018.

Haque, Farhana, and Safwana Haque. "Plant Recognition System Using Leaf Shape Features And Minimum Euclidean Distance." ICTACT Journal on Image & Video Processing 9.2 (2018).

Ibrahim, Zaidah, Nurbaity Sabri, and Nur Nabilah Abu Mangshor. "Leaf recognition using texture features for herbal plant identification." *Indonesian Journal of Electrical Engineering and Computer Science* 9.1 (2018): 152-156.

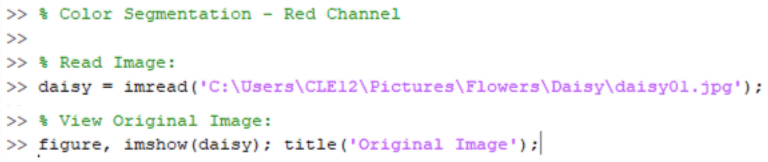
Kho, Soon Jye, et al. "Automated plant identification using artificial neural network and support vector machine." *Frontiers in Life Science* 10.1 (2017): 98-107.

Munisami, Trishen, et al. "Plant leaf recognition using shape features and colour histogram with K-nearest neighbour classifiers." *Procedia Computer Science* 58 (2015): 740-747.

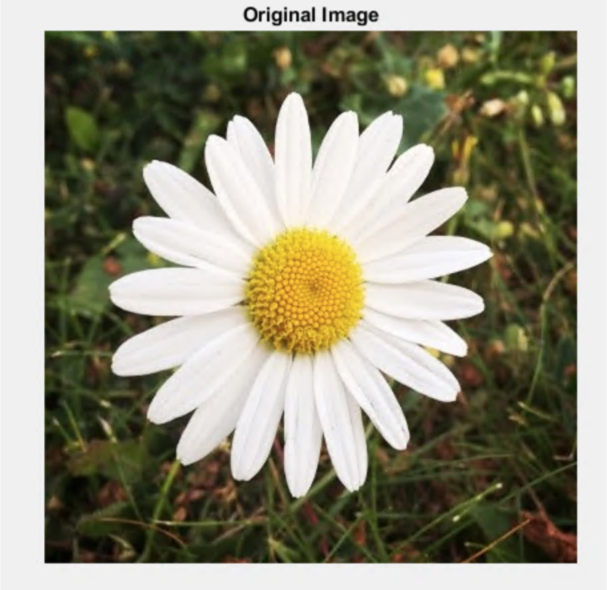
Shaparia, Riddhi, Narendra Patel, and Zankhana Shah. "Flower classification using texture and color features." *Kalpa Publications in Computing* 2 (2017): 113-118.

Source Code: Pre-Processing and Segmentation

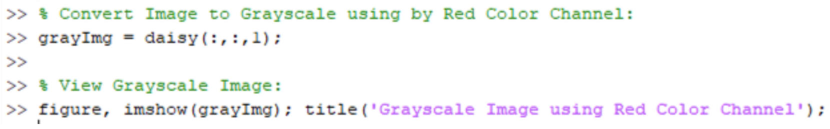
Color Segmentation – Red Channel



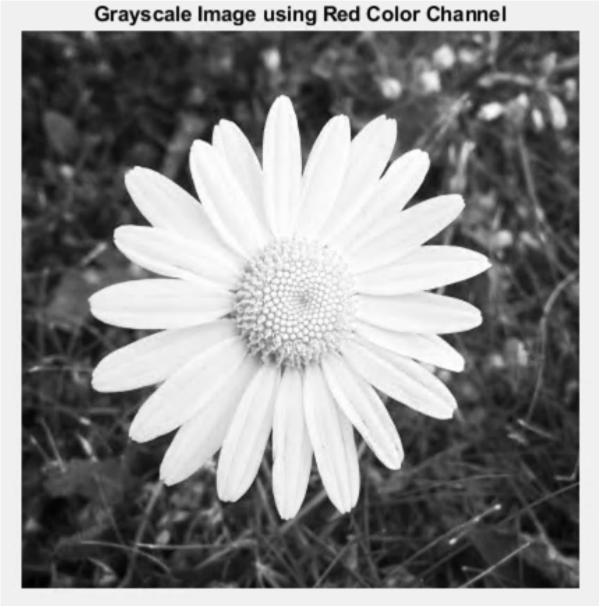
```
>> % Color Segmentation - Red Channel
>>
>> % Read Image:
>> daisy = imread('C:\Users\CLE12\Pictures\Flowers\Daisy\daisy01.jpg');
>> % View Original Image:
>> figure, imshow(daisy); title('Original Image');
```



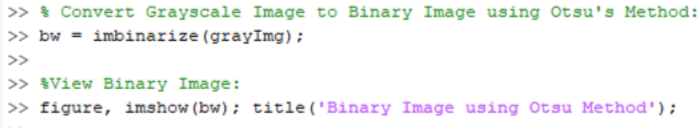
Original Image



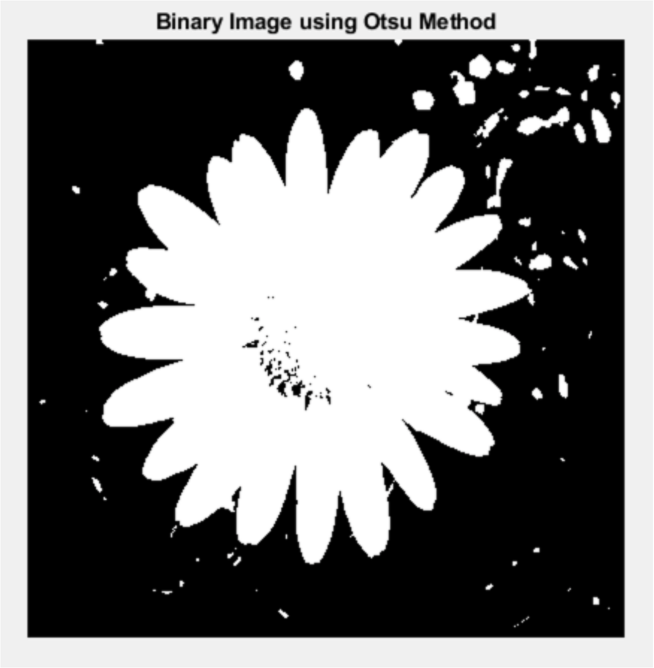
```
>> % Convert Image to Grayscale using by Red Color Channel:
>> grayImg = daisy(:,:,1);
>>
>> % View Grayscale Image:
>> figure, imshow(grayImg); title('Grayscale Image using Red Color Channel');
```

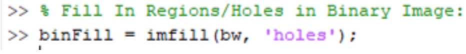


Grayscale Image using Red Color Channel

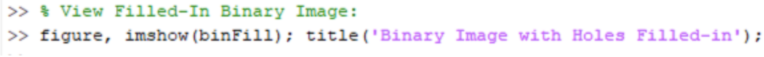


```
>> % Convert Grayscale Image to Binary Image using Otsu's Method:
>> bw = imbinarize(grayImg);
>>
>> %View Binary Image:
>> figure, imshow(bw); title('Binary Image using Otsu Method');
```

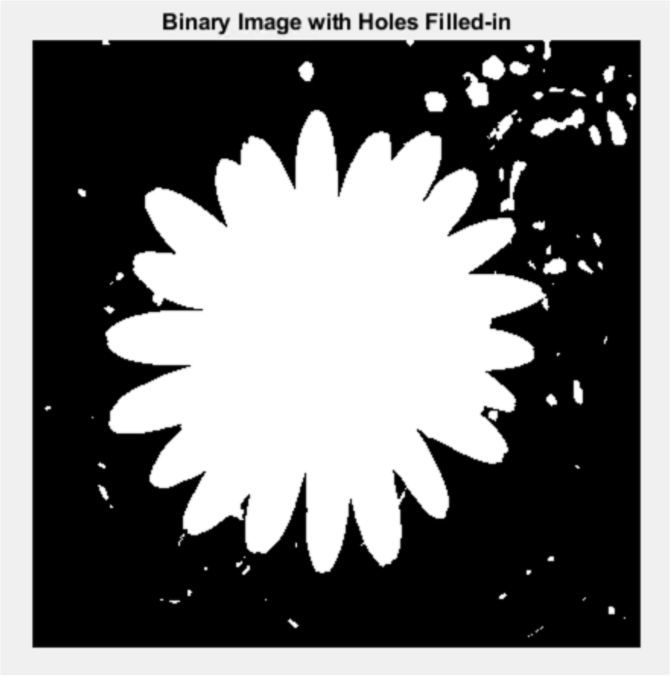


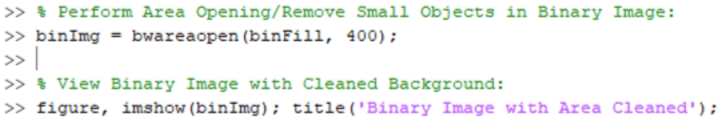


```
>> % Fill In Regions/Holes in Binary Image:
>> binFill = imfill(bw, 'holes');
```

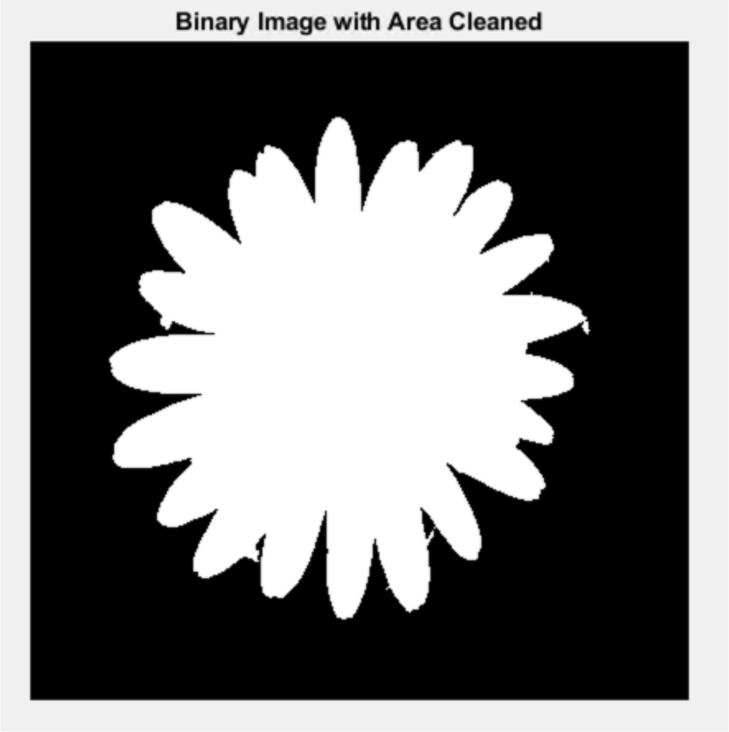


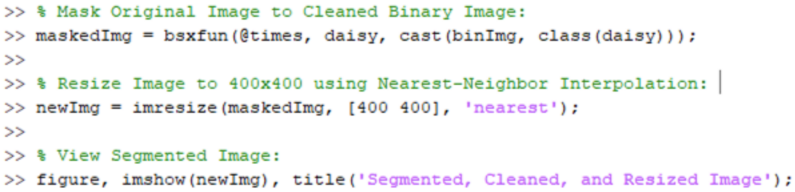
```
>> % View Filled-In Binary Image:
>> figure, imshow(binFill); title('Binary Image with Holes Filled-in');
```



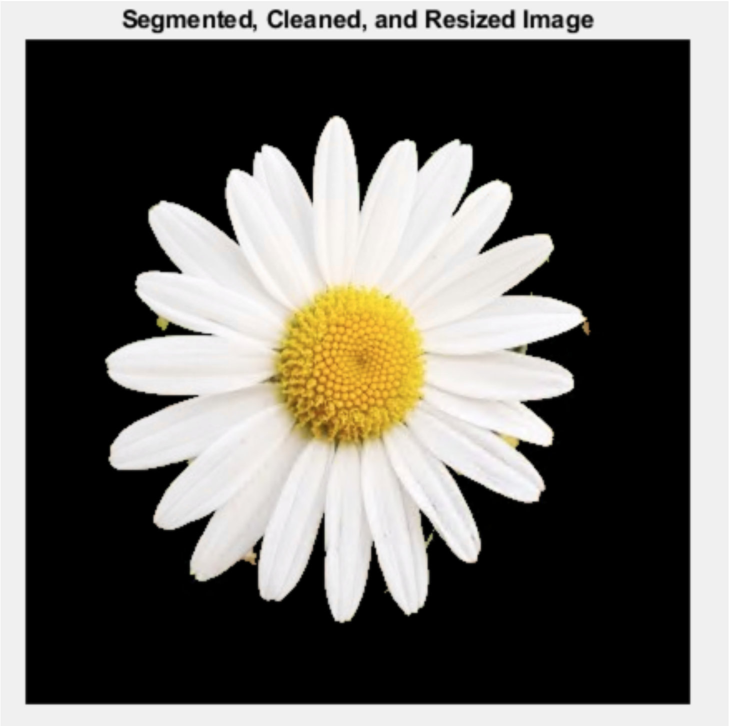


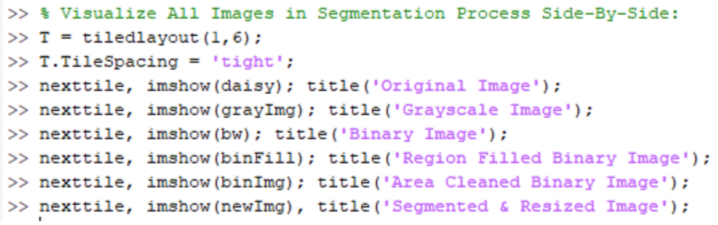
```
>> % Perform Area Opening/Remove Small Objects in Binary Image:
>> binImg = bwareaopen(binFill, 400);
>> 
>> % View Binary Image with Cleaned Background:
>> figure, imshow(binImg); title('Binary Image with Area Cleaned');
```



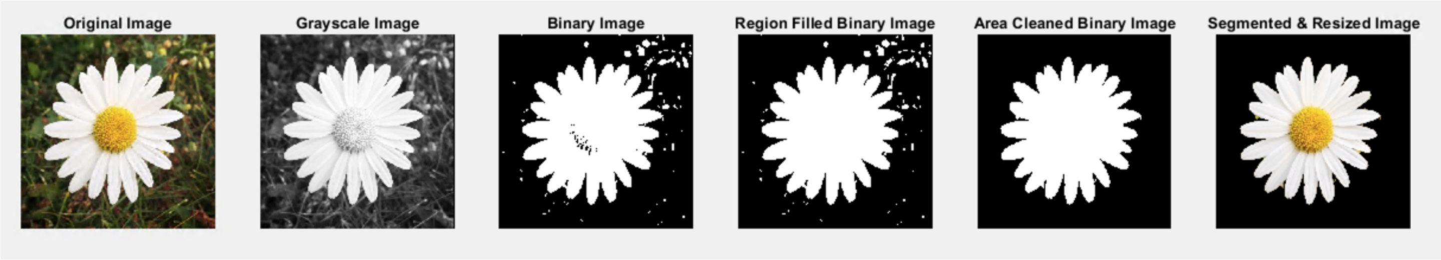


```
>> % Mask Original Image to Cleaned Binary Image:
>> maskedImg = bsxfun(@times, daisy, cast(binImg, class(daisy)));
>> 
>> % Resize Image to 400x400 using Nearest-Neighbor Interpolation: 
>> newImg = imresize(maskedImg, [400 400], 'nearest');
>> 
>> % View Segmented Image:
>> figure, imshow(newImg), title('Segmented, Cleaned, and Resized Image');
```

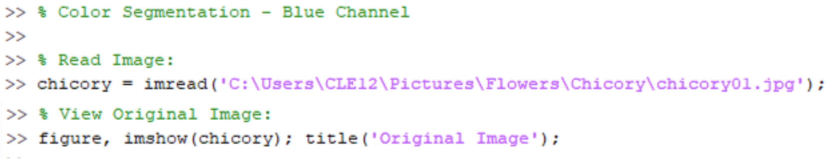




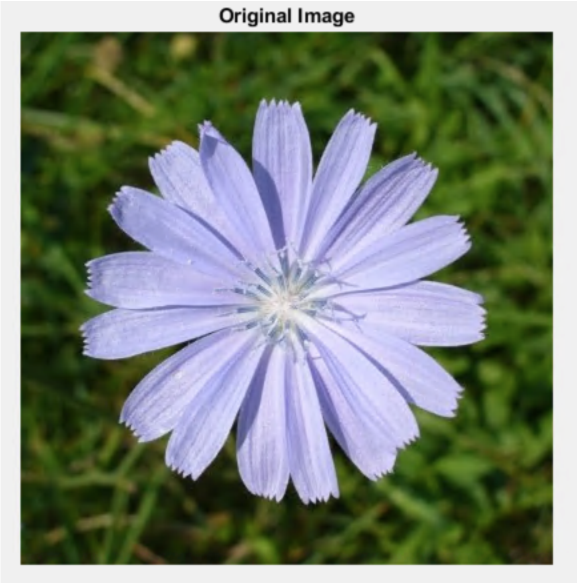
```
>> % Visualize All Images in Segmentation Process Side-By-Side:
>> T = tiledlayout(1,6);
>> T.TileSpacing = 'tight';
>> nexttile, imshow(daisy); title('Original Image');
>> nexttile, imshow(grayImg); title('Grayscale Image');
>> nexttile, imshow(bw); title('Binary Image');
>> nexttile, imshow(binFill); title('Region Filled Binary Image');
>> nexttile, imshow(binImg); title('Area Cleaned Binary Image');
>> nexttile, imshow(newImg), title('Segmented & Resized Image');
```

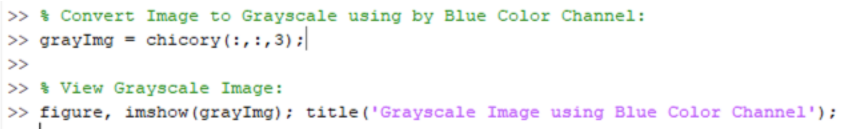

Original Image
Grayscale Image
Binary Image
Region Filled Binary Image
Area Cleaned Binary Image
Segmented & Resized Image

Color Segmentation – Blue Channel

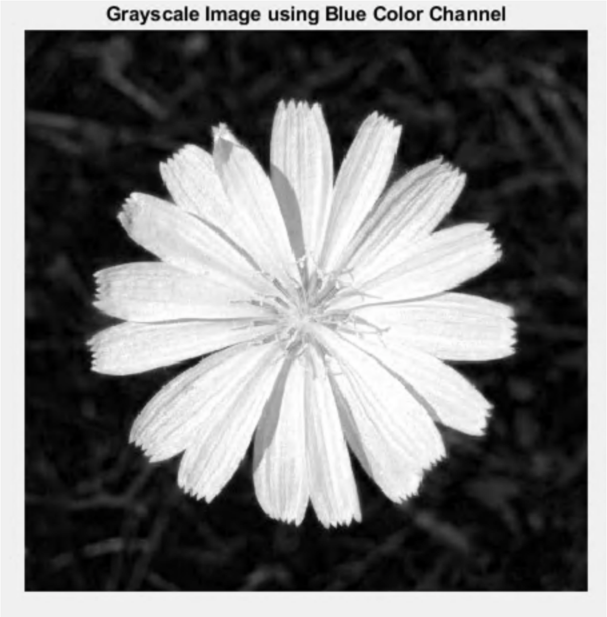


```
>> % Color Segmentation - Blue Channel
>>
>> % Read Image:
>> chicory = imread('C:\Users\CLE12\Pictures\Flowers\Chicory\chicory01.jpg');
>> % View Original Image:
>> figure, imshow(chicory); title('Original Image');
```

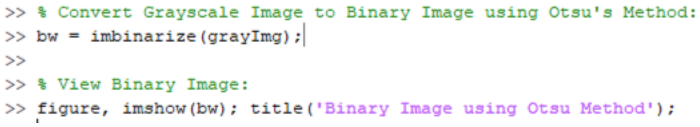




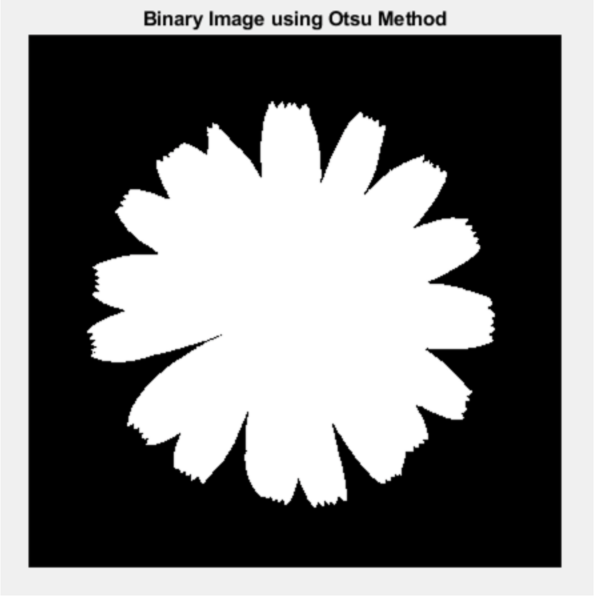
```
>> % Convert Image to Grayscale using by Blue Color Channel:
>> grayImg = chicory(:,:,3);
>>
>> % View Grayscale Image:
>> figure, imshow(grayImg); title('Grayscale Image using Blue Color Channel');
```



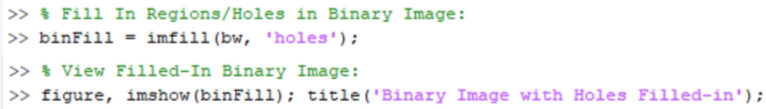
Grayscale Image using Blue Color Channel



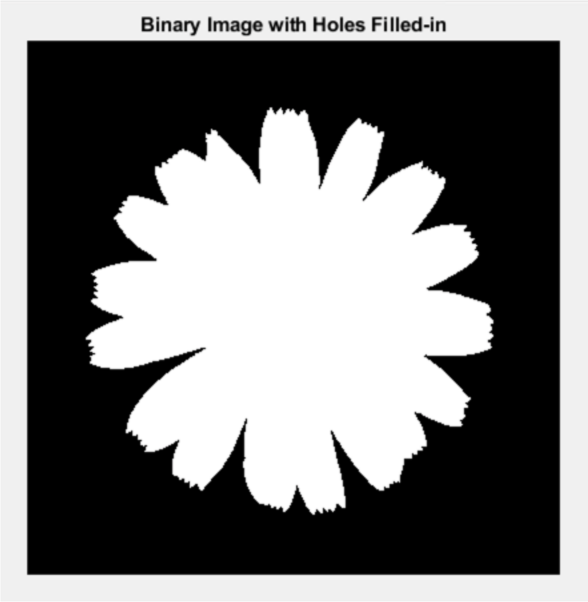
```
>> % Convert Grayscale Image to Binary Image using Otsu's Method:
>> bw = imbinarize(grayImg);
>>
>> % View Binary Image:
>> figure, imshow(bw); title('Binary Image using Otsu Method');
```



Binary Image using Otsu Method

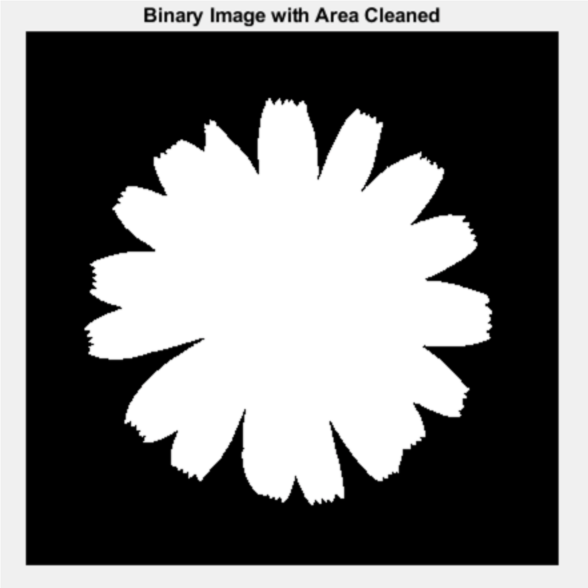


```
>> % Fill In Regions/Holes in Binary Image:
>> binFill = imfill(bw, 'holes');
>> % View Filled-In Binary Image:
>> figure, imshow(binFill); title('Binary Image with Holes Filled-in');
```



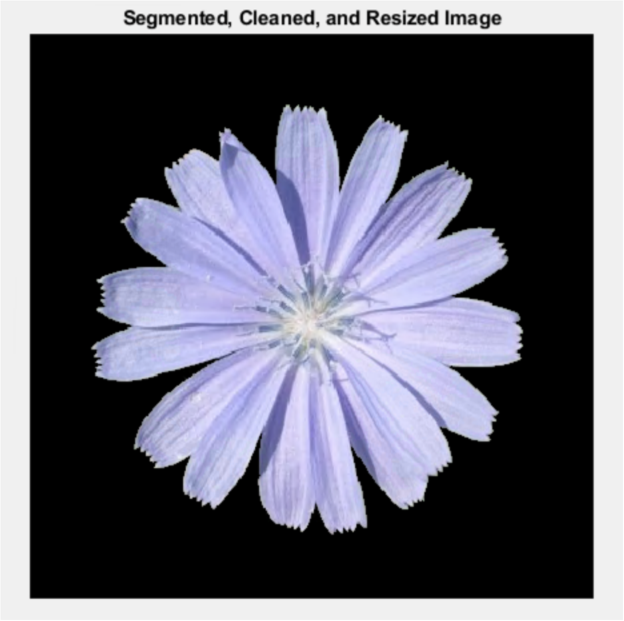
Binary Image with Holes Filled-in

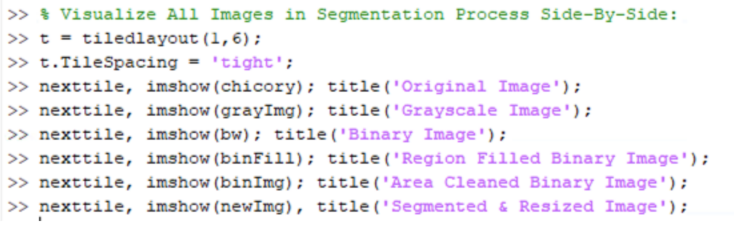
```
>> % Perform Area Opening/Remove Small Objects in Binary Image:
>> binImg = bwareaopen(binFill, 400);
>>
>> % View Binary Image with Cleaned Background:
>> figure, imshow(binImg); title('Binary Image with Area Cleaned');
```



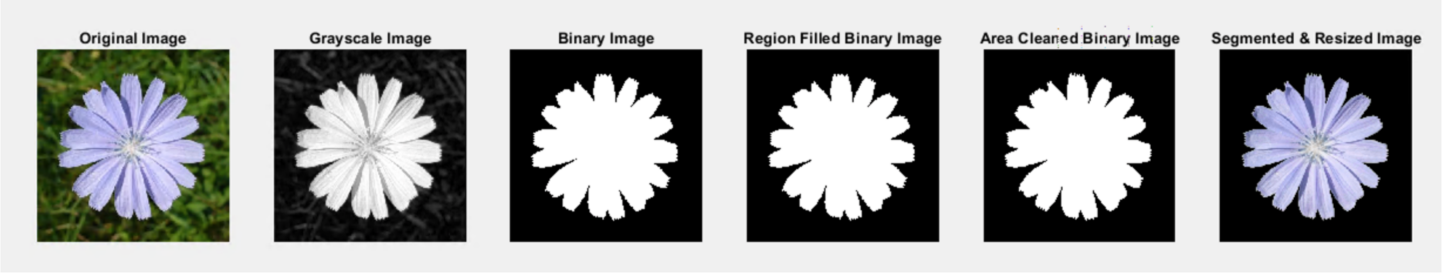
Binary Image with Area Cleaned

```
>> % Mask Original Image to Cleaned Binary Image:
>> maskedImg = bsxfun(@times, chicory, cast(binImg, class(chicory)));
>>
>> % Resize Image to 400x400 using Nearest-Neighbor Interpolation:
>> newImg = imresize(maskedImg, [400 400], 'nearest');
>>
>> % View Segmented Image:
>> figure, imshow(newImg), title('Segmented, Cleaned, and Resized Image');
```





```
>> % Visualize All Images in Segmentation Process Side-By-Side:
>> t = tiledlayout(1,6);
>> t.TileSpacing = 'tight';
>> nexttile, imshow(chicory); title('Original Image');
>> nexttile, imshow(grayImg); title('Grayscale Image');
>> nexttile, imshow(bw); title('Binary Image');
>> nexttile, imshow(binFill); title('Region Filled Binary Image');
>> nexttile, imshow(binImg); title('Area Cleaned Binary Image');
>> nexttile, imshow(newImg), title('Segmented & Resized Image');
```

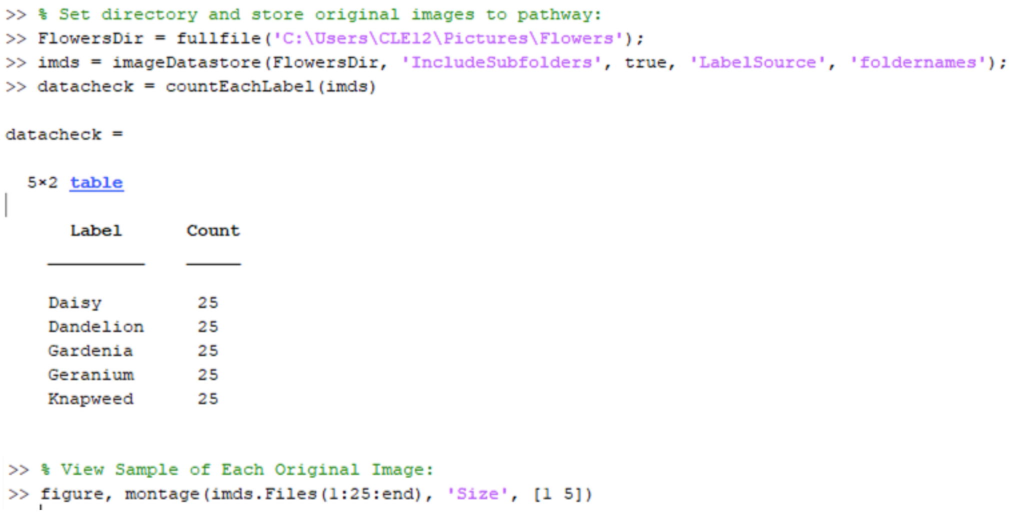

Original Image
Grayscale Image
Binary Image
Region Filled Binary Image
Area Cleaned Binary Image
Segmented & Resized Image

Code used for bulk-preprocessing in MATLAB:

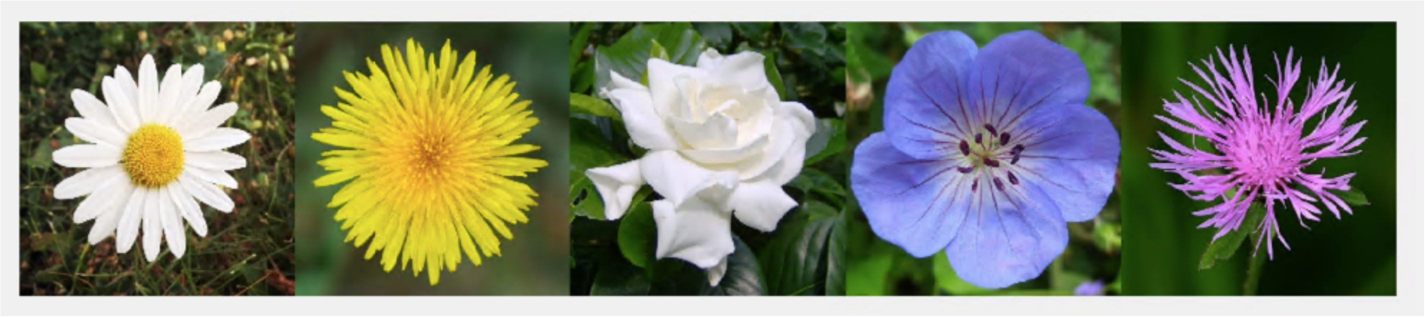
```
Images = dir(‘C:\folder_pathway\*.jpg’)
outDir = ‘C\new_folder_pathway\’;
mkdir(OutDir);
for i = 1:length(Images)
  ImgName = strcat(‘C:\folder_pathway\’, Images(i).name);
  Img = imread(ImgName);
  grayImg = Img(:, :, 1);
  bw = imbinarize(grayImg);
  binImg = imfill(bw, ‘holes’);
  binImg = bwareaopen(binImg, 400);
  maskedImg = bsxfun(@times, Img, cast(binImg, class(Img)));
  newImg = imresize(maskedImg, [400 400], ‘nearest’);
  imwrite(newImg, strcat(outDir, Images(i).name));
end
```

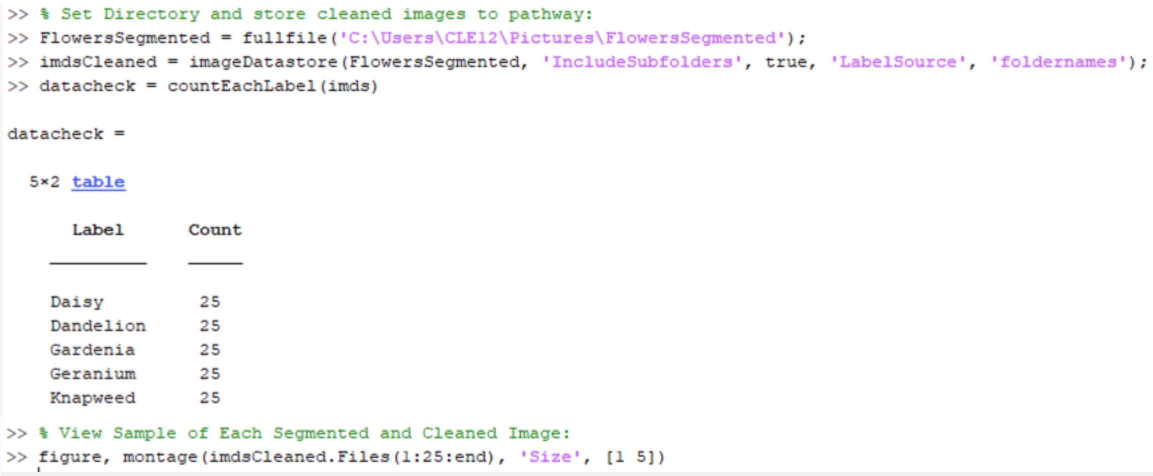
Source Code: Classification

5-Class: Daisy, Dandelion, Gardenia, Geranium, Knapweed

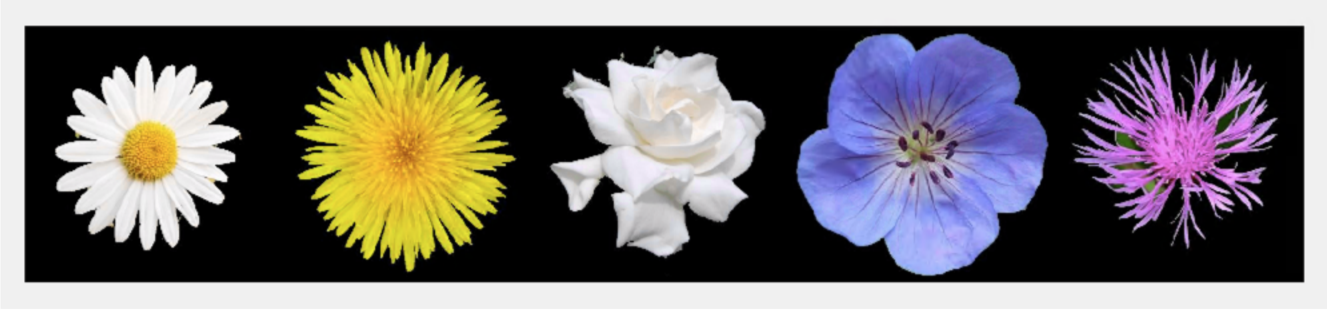


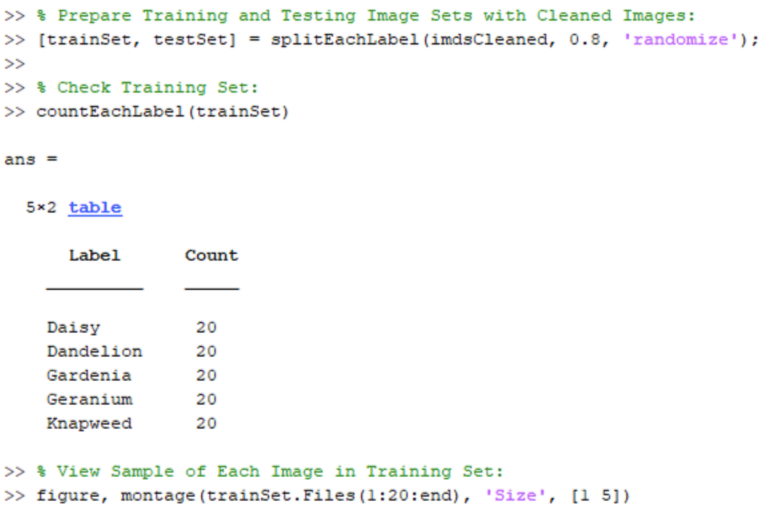
```
>> % Set directory and store original images to pathway:
>> FlowersDir = fullfile('C:\Users\CLE12\Pictures\Flowers');
>> imds = imageDatastore(FlowersDir, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
>> datacheck = countEachLabel(imds)

datacheck =

  5×2 table

      Label       Count
    _________     _____

    Daisy          25
    Dandelion      25
    Gardenia       25
    Geranium       25
    Knapweed       25


>> % View Sample of Each Original Image:
>> figure, montage(imds.Files(1:25:end), 'Size', [1 5])
```



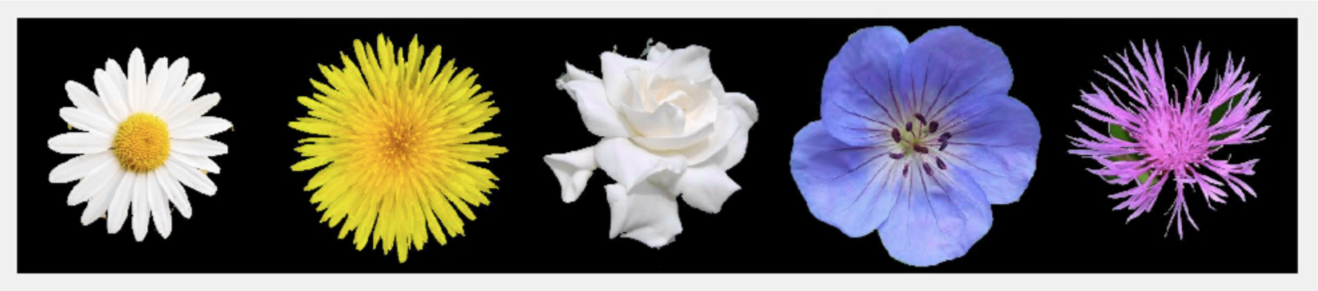


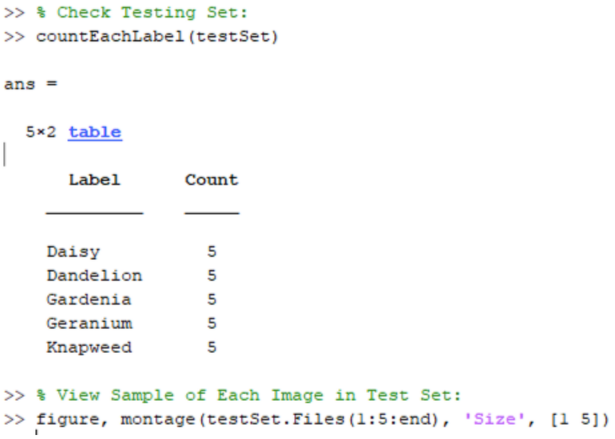
```
>> % Set Directory and store cleaned images to pathway:
>> FlowersSegmented = fullfile('C:\Users\CLE12\Pictures\FlowersSegmented');
>> imdsCleaned = imageDatastore(FlowersSegmented, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
>> datacheck = countEachLabel(imds)

datacheck =

  5×2 table

      Label       Count
    _________     _____

    Daisy          25
    Dandelion      25
    Gardenia       25
    Geranium       25
    Knapweed       25
>> % View Sample of Each Segmented and Cleaned Image:
>> figure, montage(imdsCleaned.Files(1:25:end), 'Size', [1 5])
```



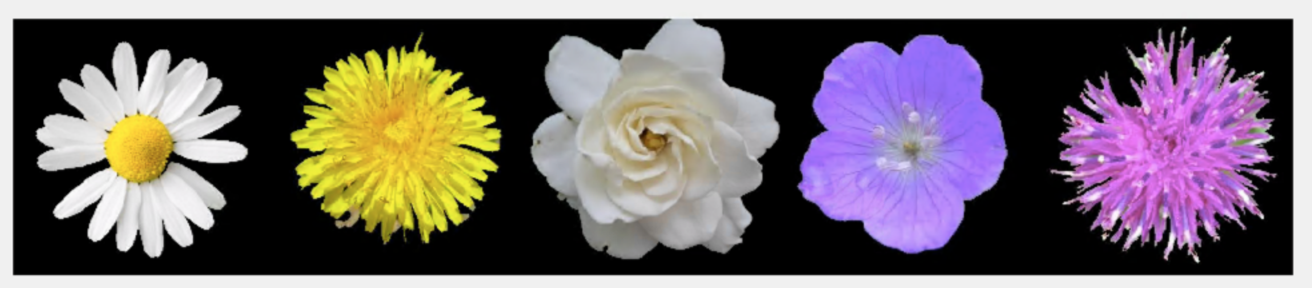


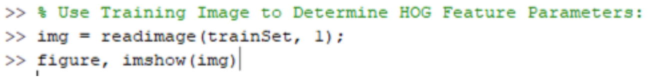
```
>> % Prepare Training and Testing Image Sets with Cleaned Images:
>> [trainSet, testSet] = splitEachLabel(imdsCleaned, 0.8, 'randomize');
>>
>> % Check Training Set:
>> countEachLabel(trainSet)

ans =

  5×2 table

      Label       Count
    _________     _____

    Daisy          20
    Dandelion      20
    Gardenia       20
    Geranium       20
    Knapweed       20

>> % View Sample of Each Image in Training Set:
>> figure, montage(trainSet.Files(1:20:end), 'Size', [1 5])
```



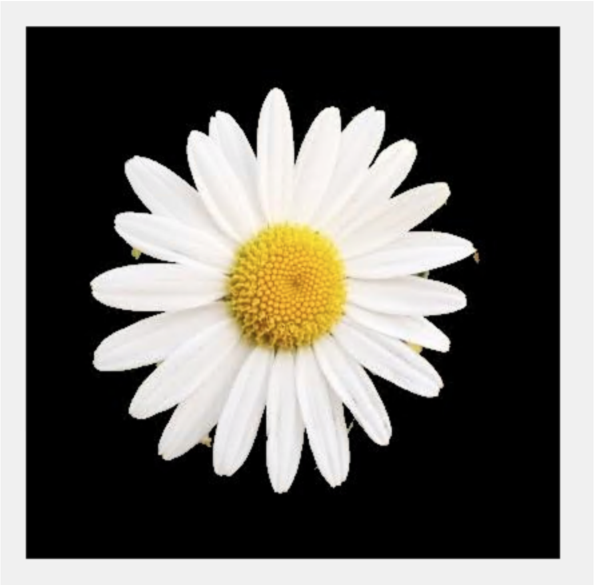


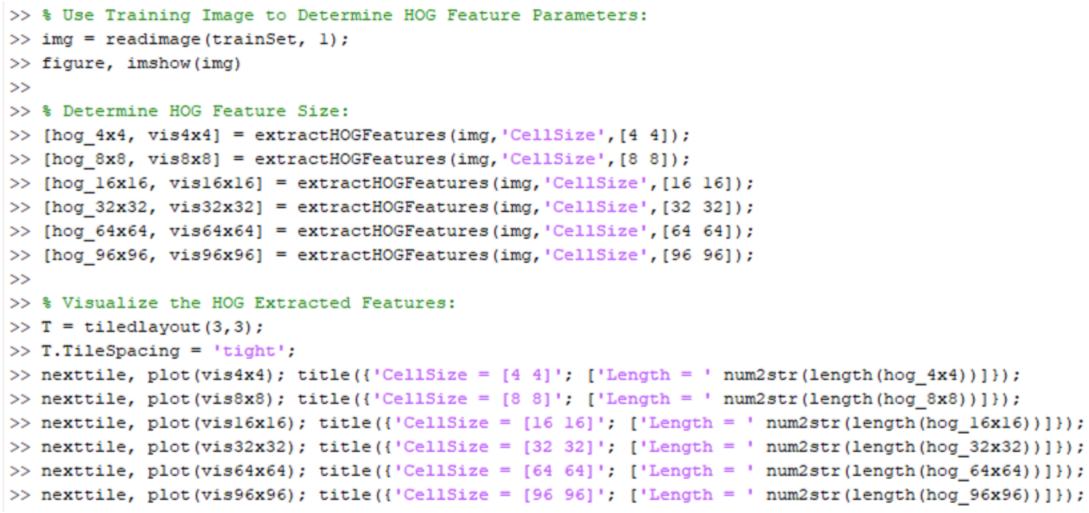
```
>> % Check Testing Set:
>> countEachLabel(testSet)

ans =

  5×2 table

      Label       Count
    _________     _____

    Daisy           5
    Dandelion       5
    Gardenia        5
    Geranium        5
    Knapweed        5

>> % View Sample of Each Image in Test Set:
>> figure, montage(testSet.Files(1:5:end), 'Size', [1 5])
```



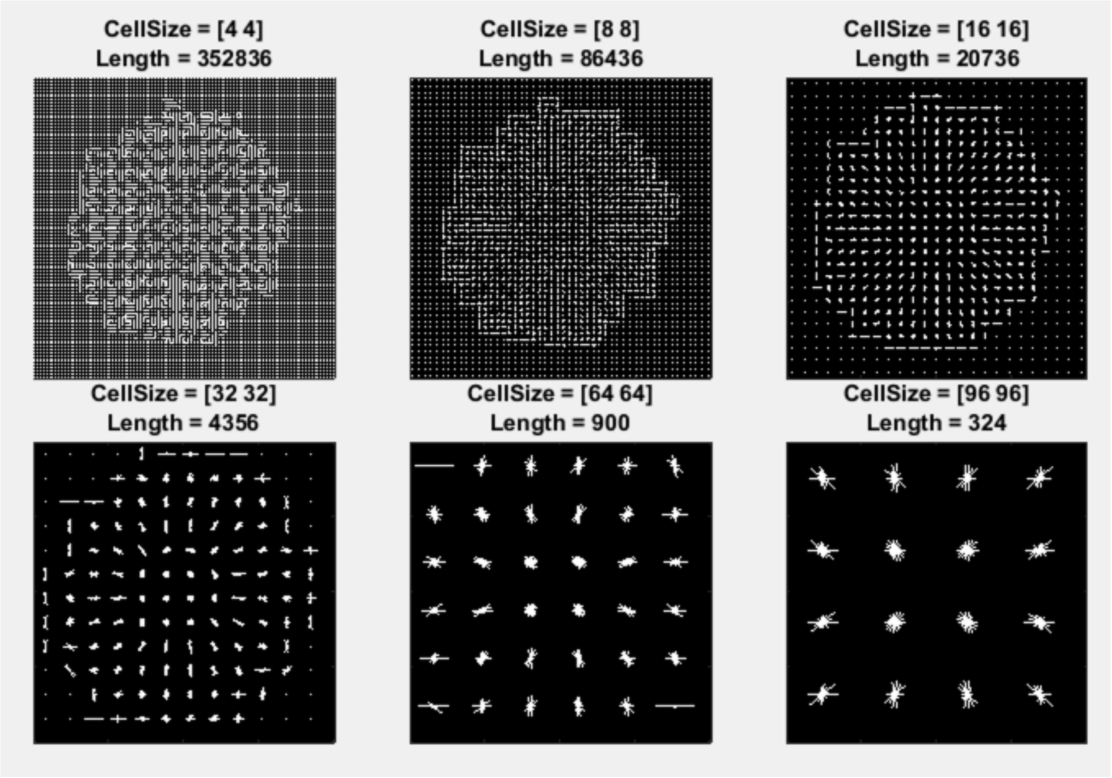


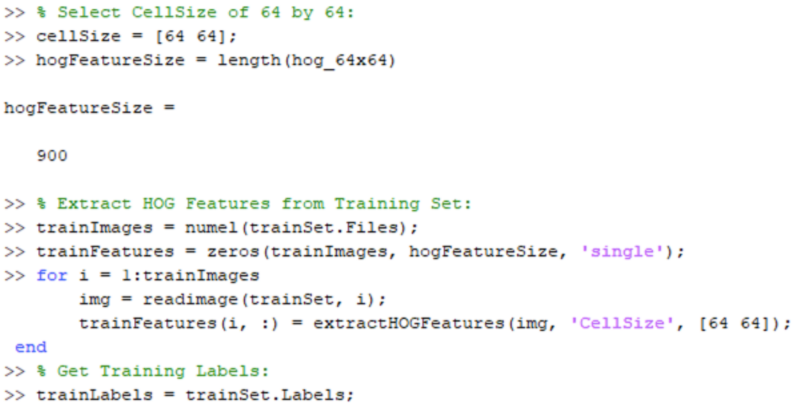
```
>> % Use Training Image to Determine HOG Feature Parameters:
>> img = readimage(trainSet, 1);
>> figure, imshow(img)
```



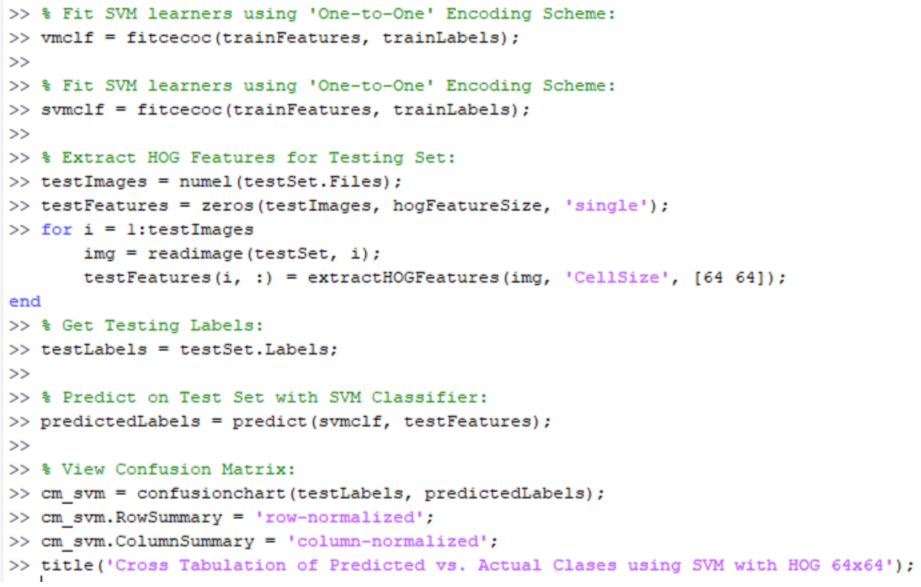


```
>> % Use Training Image to Determine HOG Feature Parameters:
>> img = readimage(trainSet, 1);
>> figure, imshow(img)
>>
>> % Determine HOG Feature Size:
>> [hog_4x4, vis4x4] = extractHOGFeatures(img,'CellSize',[4 4]);
>> [hog_8x8, vis8x8] = extractHOGFeatures(img,'CellSize',[8 8]);
>> [hog_16x16, vis16x16] = extractHOGFeatures(img,'CellSize',[16 16]);
>> [hog_32x32, vis32x32] = extractHOGFeatures(img,'CellSize',[32 32]);
>> [hog_64x64, vis64x64] = extractHOGFeatures(img,'CellSize',[64 64]);
>> [hog_96x96, vis96x96] = extractHOGFeatures(img,'CellSize',[96 96]);
>>
>> % Visualize the HOG Extracted Features:
>> T = tiledlayout(3,3);
>> T.TileSpacing = 'tight';
>> nexttile, plot(vis4x4); title({'CellSize = [4 4]'; ['Length = ' num2str(length(hog_4x4))]});
>> nexttile, plot(vis8x8); title({'CellSize = [8 8]'; ['Length = ' num2str(length(hog_8x8))]});
>> nexttile, plot(vis16x16); title({'CellSize = [16 16]'; ['Length = ' num2str(length(hog_16x16))]});
>> nexttile, plot(vis32x32); title({'CellSize = [32 32]'; ['Length = ' num2str(length(hog_32x32))]});
>> nexttile, plot(vis64x64); title({'CellSize = [64 64]'; ['Length = ' num2str(length(hog_64x64))]});
>> nexttile, plot(vis96x96); title({'CellSize = [96 96]'; ['Length = ' num2str(length(hog_96x96))]});
```

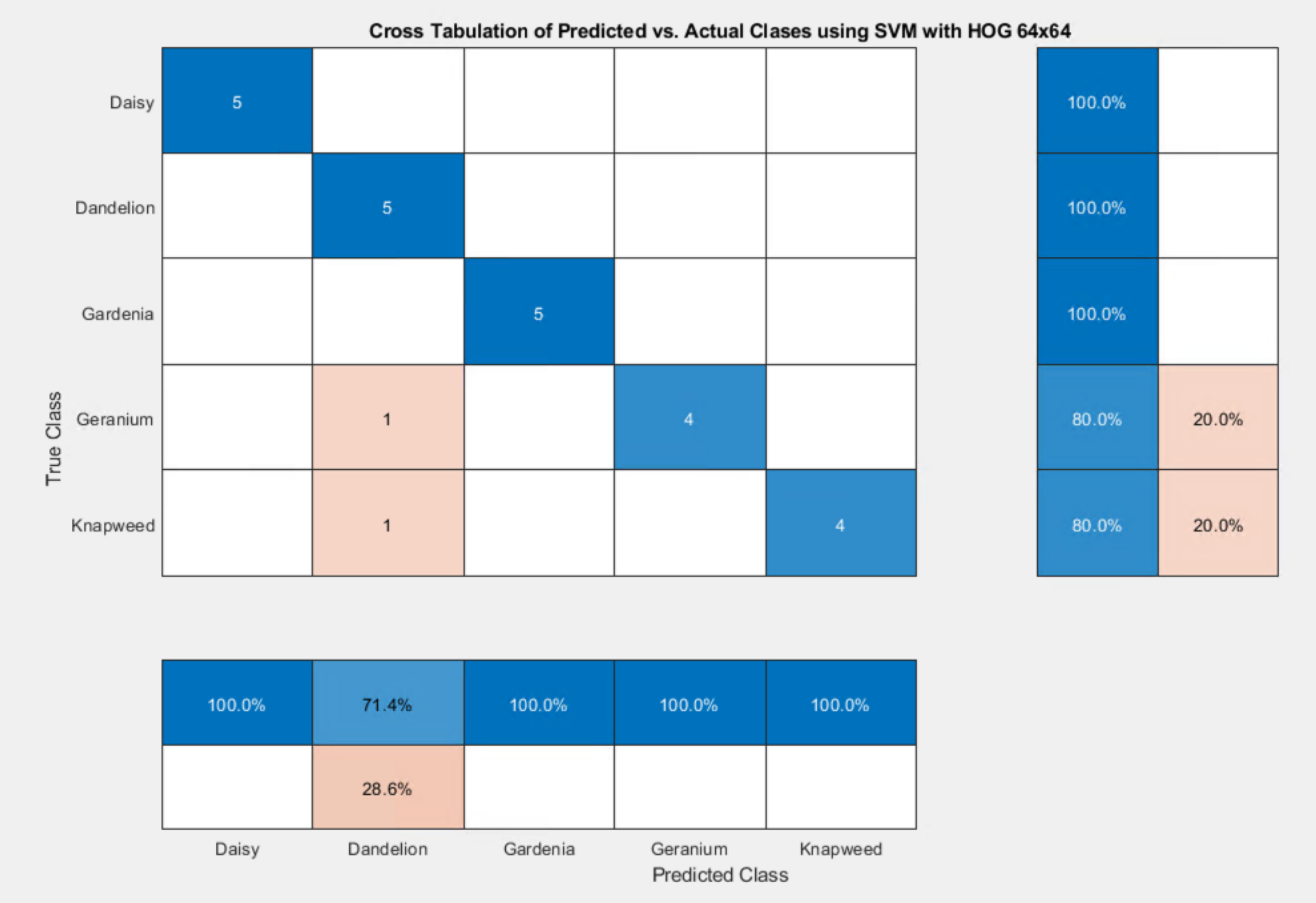




```
>> % Select CellSize of 64 by 64:
>> cellSize = [64 64];
>> hogFeatureSize = length(hog_64x64)

hogFeatureSize =

   900

>> % Extract HOG Features from Training Set:
>> trainImages = numel(trainSet.Files);
>> trainFeatures = zeros(trainImages, hogFeatureSize, 'single');
>> for i = 1:trainImages
       img = readimage(trainSet, i);
       trainFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [64 64]);
 end
>> % Get Training Labels:
>> trainLabels = trainSet.Labels;
```



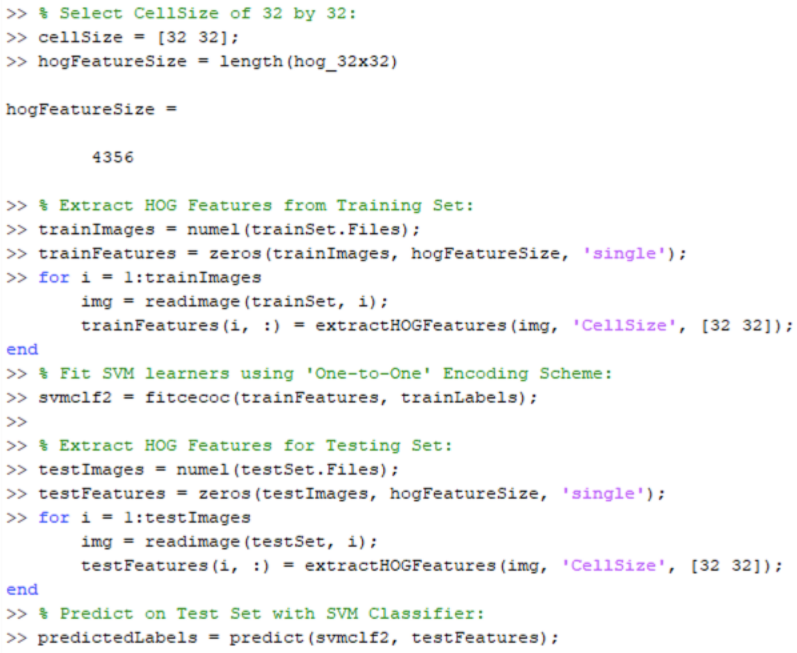
```
>> % Fit SVM learners using 'One-to-One' Encoding Scheme:
>> vmclf = fitcecoc(trainFeatures, trainLabels);
>>
>> % Fit SVM learners using 'One-to-One' Encoding Scheme:
>> svmclf = fitcecoc(trainFeatures, trainLabels);
>>
>> % Extract HOG Features for Testing Set:
>> testImages = numel(testSet.Files);
>> testFeatures = zeros(testImages, hogFeatureSize, 'single');
>> for i = 1:testImages
       img = readimage(testSet, i);
       testFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [64 64]);
end
>> % Get Testing Labels:
>> testLabels = testSet.Labels;
>>
>> % Predict on Test Set with SVM Classifier:
>> predictedLabels = predict(svmclf, testFeatures);
>>
>> % View Confusion Matrix:
>> cm_svm = confusionchart(testLabels, predictedLabels);
>> cm_svm.RowSummary = 'row-normalized';
>> cm_svm.ColumnSummary = 'column-normalized';
>> title('Cross Tabulation of Predicted vs. Actual Clases using SVM with HOG 64x64');
```



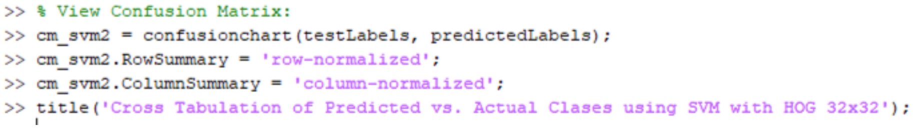
Cross Tabulation of Predicted vs. Actual Clases using SVM with HOG 64x64

Daisy	5					100.0%	
Dandelion		5				100.0%	
Gardenia			5			100.0%	
Geranium		1		4		80.0%	20.0%
Knapweed		1			4	80.0%	20.0%
	100.0%	71.4%	100.0%	100.0%	100.0%		
		28.6%					
	Daisy	Dandelion	Gardenia	Geranium	Knapweed		

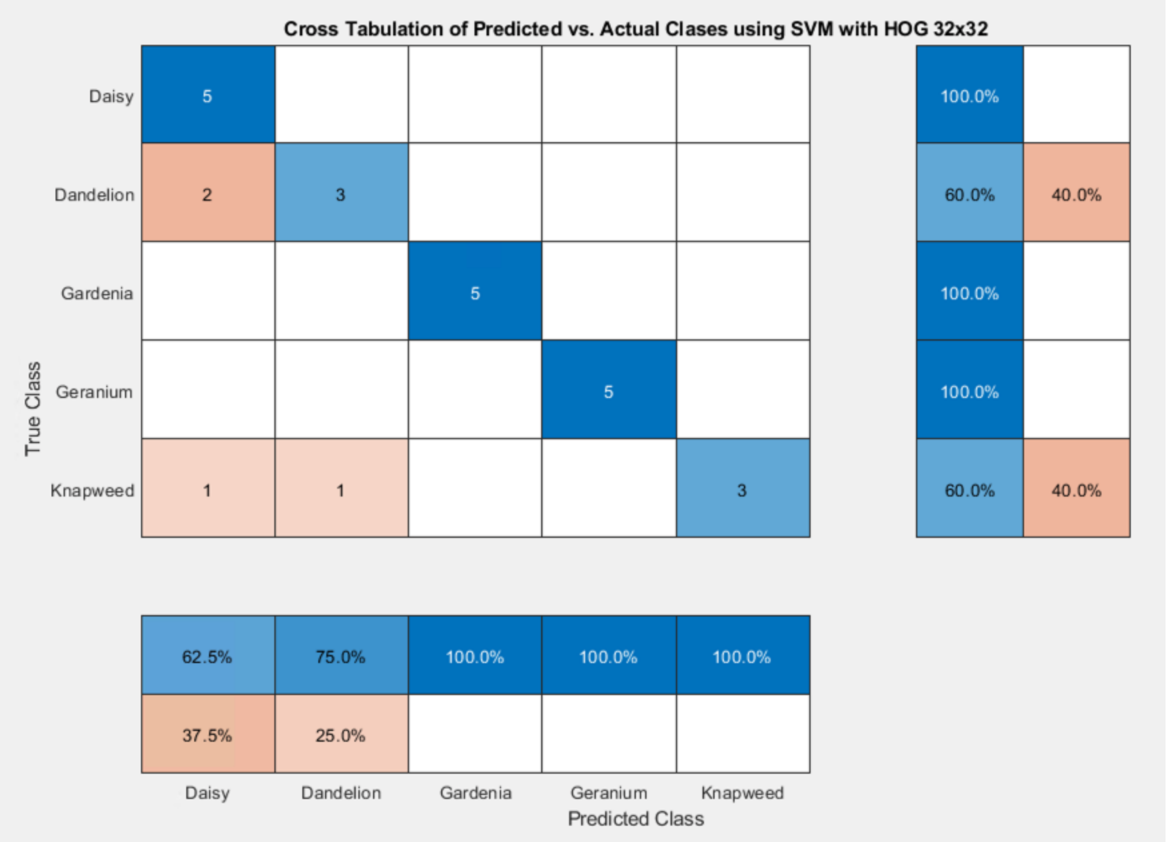
```
>> % Calculate Accuracy using Confusion Matrix:
>> conMat = confusionmat(testLabels, predictedLabels);
>> svm_acc = sum(diag(conMat))/length(testLabels)

svm_acc =

    0.9200

>> % Estimate Loss Using Cross-Validation:
>> CVSVMdl = crossval(svmclf);
>> error = kfoldLoss(CVSVMdl)

error =

    0.1100

>>
>> % Accuracy Score from Cross-Validation:
>> cv_accu = 1 - error

cv_accu =

    0.8900
```



```
>> % Select CellSize of 32 by 32:
>> cellSize = [32 32];
>> hogFeatureSize = length(hog_32x32)

hogFeatureSize =

        4356

>> % Extract HOG Features from Training Set:
>> trainImages = numel(trainSet.Files);
>> trainFeatures = zeros(trainImages, hogFeatureSize, 'single');
>> for i = 1:trainImages
       img = readimage(trainSet, i);
       trainFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [32 32]);
end
>> % Fit SVM learners using 'One-to-One' Encoding Scheme:
>> svmclf2 = fitcecoc(trainFeatures, trainLabels);
>>
>> % Extract HOG Features for Testing Set:
>> testImages = numel(testSet.Files);
>> testFeatures = zeros(testImages, hogFeatureSize, 'single');
>> for i = 1:testImages
       img = readimage(testSet, i);
       testFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [32 32]);
end
>> % Predict on Test Set with SVM Classifier:
>> predictedLabels = predict(svmclf2, testFeatures);
```

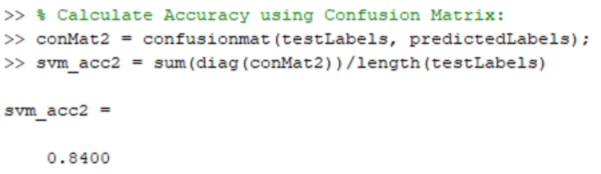


```
>> % View Confusion Matrix:
>> cm_svm2 = confusionchart(testLabels, predictedLabels);
>> cm_svm2.RowSummary = 'row-normalized';
>> cm_svm2.ColumnSummary = 'column-normalized';
>> title('Cross Tabulation of Predicted vs. Actual Clases using SVM with HOG 32x32');
```

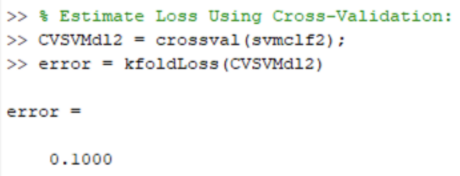


Cross Tabulation of Predicted vs. Actual Clases using SVM with HOG 32x32

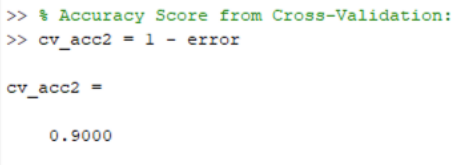
True Class	Daisy	Dandelion	Gardenia	Geranium	Knapweed		
Daisy	5					100.0%	
Dandelion	2	3				60.0%	40.0%
Gardenia			5			100.0%	
Geranium				5		100.0%	
Knapweed	1	1			3	60.0%	40.0%
	62.5%	75.0%	100.0%	100.0%	100.0%		
	37.5%	25.0%					



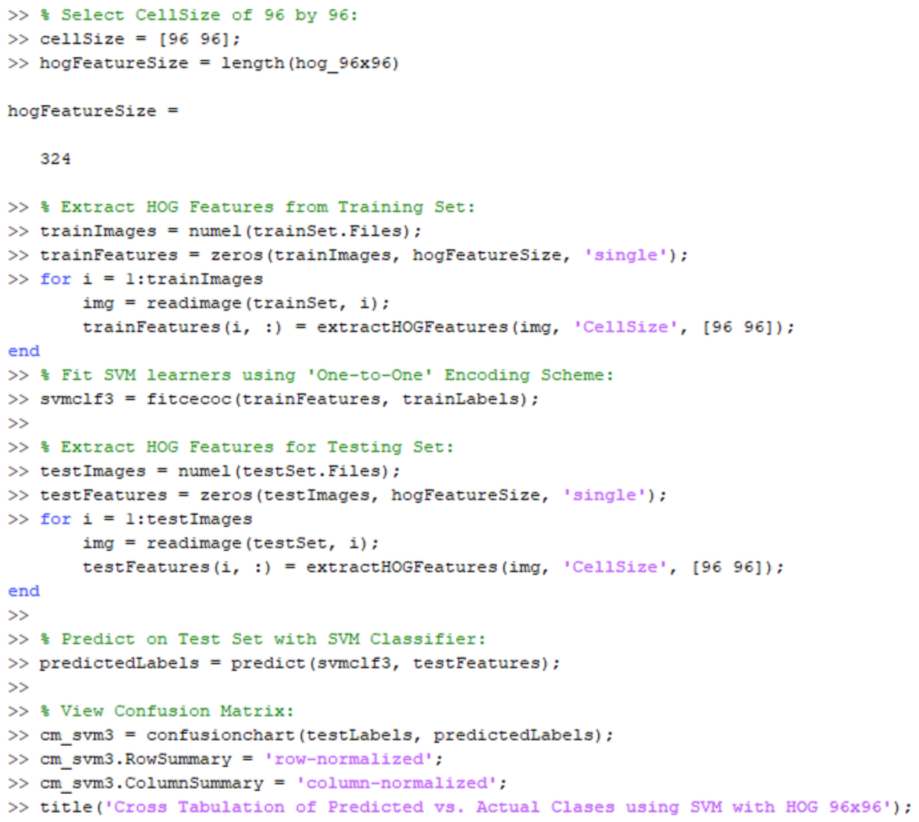
```
>> % Calculate Accuracy using Confusion Matrix:
>> conMat2 = confusionmat(testLabels, predictedLabels);
>> svm_acc2 = sum(diag(conMat2))/length(testLabels)

svm_acc2 =

    0.8400
```



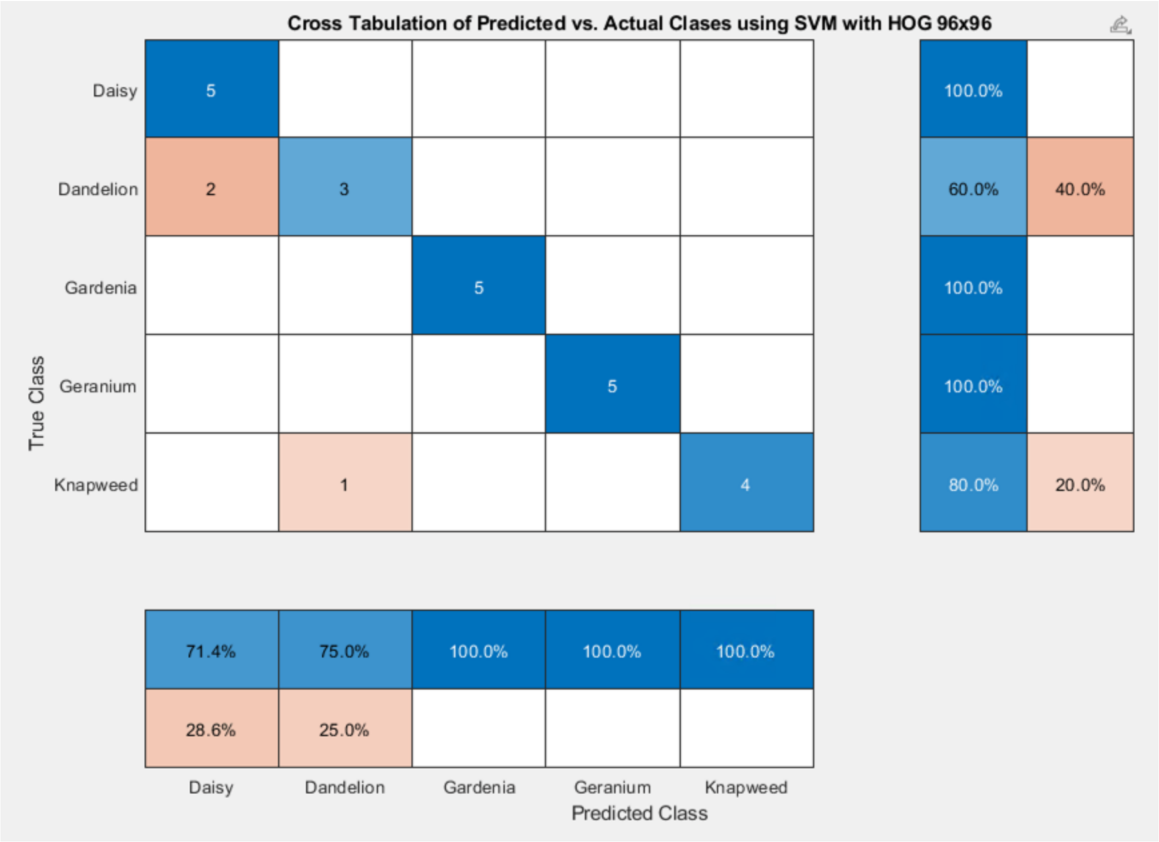
```
>> % Estimate Loss Using Cross-Validation:
>> CVSVMdl2 = crossval(svmclf2);
>> error = kfoldLoss(CVSVMdl2)

error =

    0.1000
```

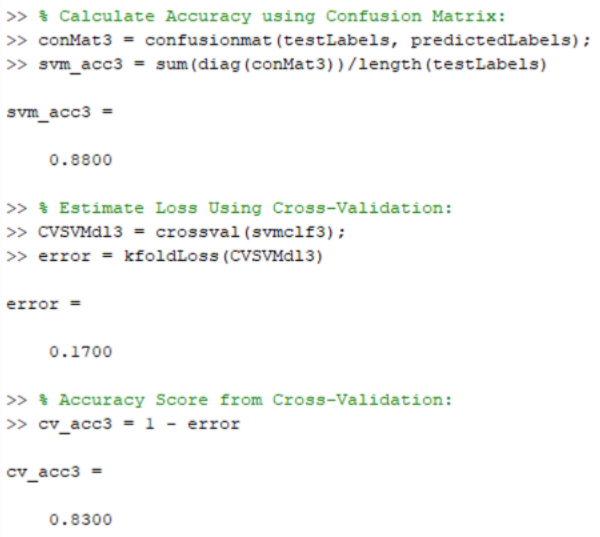


```
>> % Accuracy Score from Cross-Validation:
>> cv_acc2 = 1 - error

cv_acc2 =

    0.9000
```



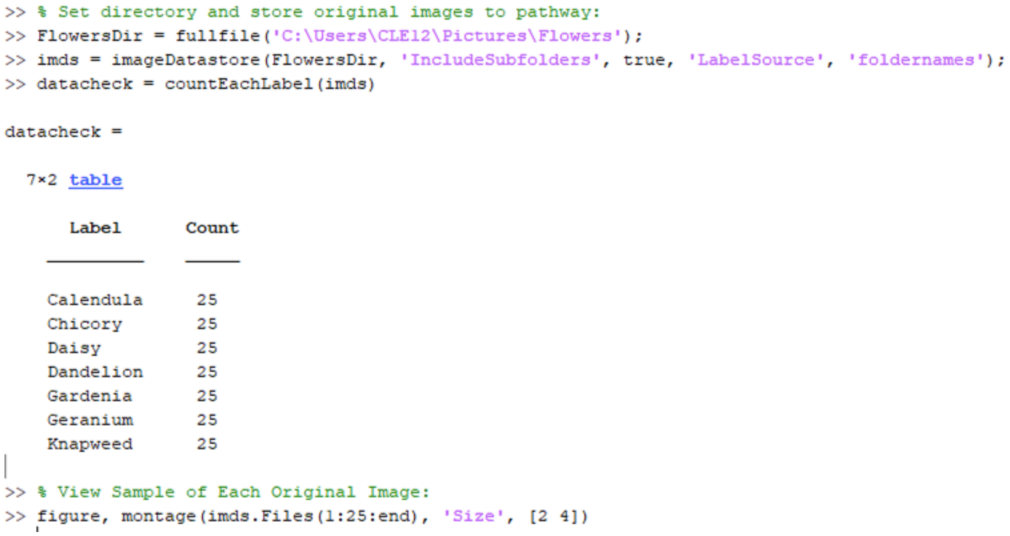
```
>> % Select CellSize of 96 by 96:
>> cellSize = [96 96];
>> hogFeatureSize = length(hog_96x96)

hogFeatureSize =

   324

>> % Extract HOG Features from Training Set:
>> trainImages = numel(trainSet.Files);
>> trainFeatures = zeros(trainImages, hogFeatureSize, 'single');
>> for i = 1:trainImages
       img = readimage(trainSet, i);
       trainFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [96 96]);
end
>> % Fit SVM learners using 'One-to-One' Encoding Scheme:
>> svmclf3 = fitcecoc(trainFeatures, trainLabels);
>>
>> % Extract HOG Features for Testing Set:
>> testImages = numel(testSet.Files);
>> testFeatures = zeros(testImages, hogFeatureSize, 'single');
>> for i = 1:testImages
       img = readimage(testSet, i);
       testFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [96 96]);
end
>>
>> % Predict on Test Set with SVM Classifier:
>> predictedLabels = predict(svmclf3, testFeatures);
>>
>> % View Confusion Matrix:
>> cm_svm3 = confusionchart(testLabels, predictedLabels);
>> cm_svm3.RowSummary = 'row-normalized';
>> cm_svm3.ColumnSummary = 'column-normalized';
>> title('Cross Tabulation of Predicted vs. Actual Clases using SVM with HOG 96x96');
```



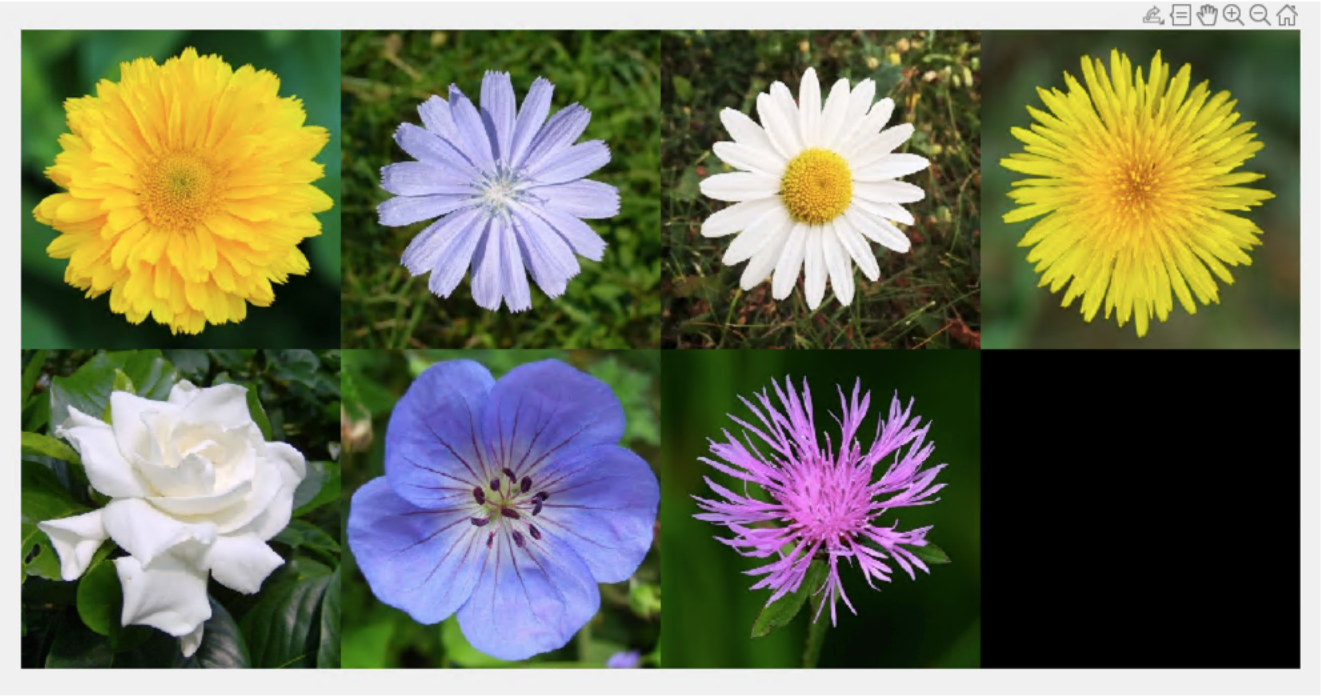


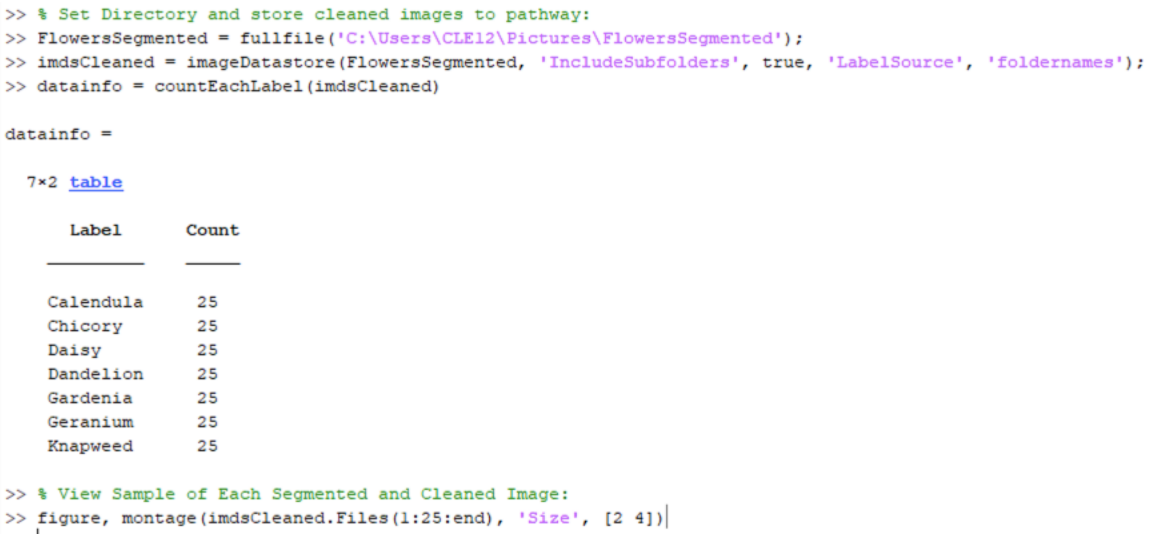
```
>> % Calculate Accuracy using Confusion Matrix:
>> conMat3 = confusionmat(testLabels, predictedLabels);
>> svm_acc3 = sum(diag(conMat3))/length(testLabels)

svm_acc3 =

    0.8800

>> % Estimate Loss Using Cross-Validation:
>> CVSVMdl3 = crossval(svmclf3);
>> error = kfoldLoss(CVSVMdl3)

error =

    0.1700

>> % Accuracy Score from Cross-Validation:
>> cv_acc3 = 1 - error

cv_acc3 =

    0.8300
```

7-Class: Calendula, Chicory, Daisy, Dandelion, Gardenia, Geranium, and Knapweed

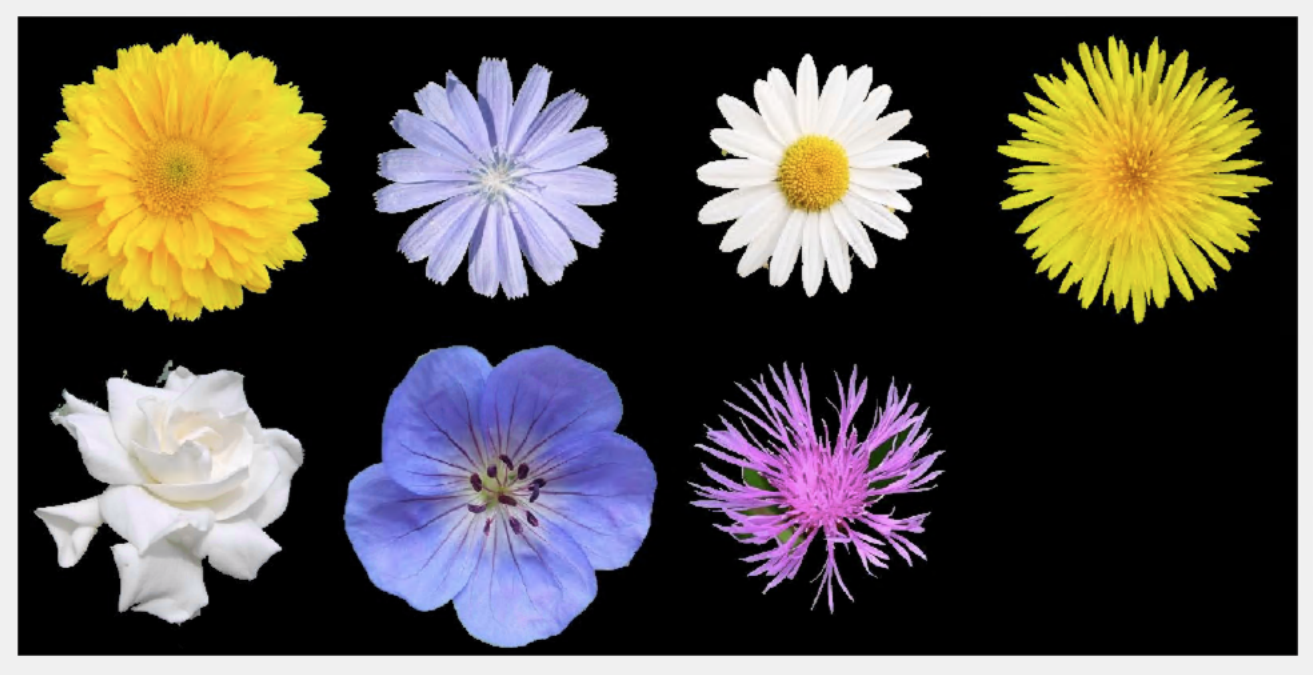


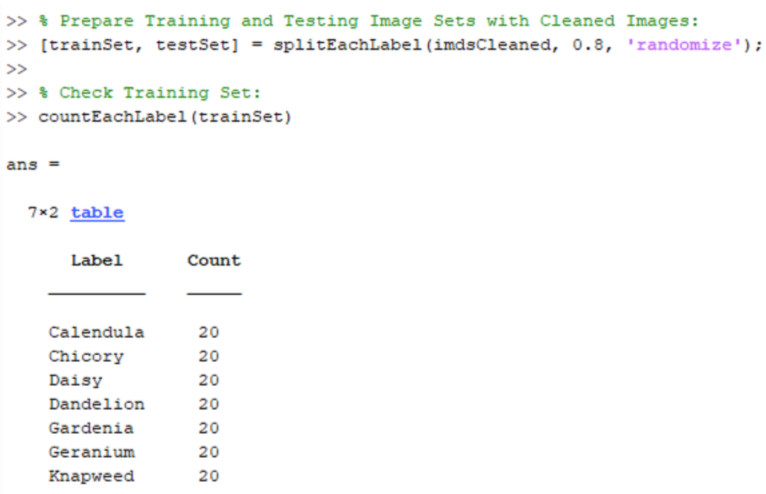
```
>> % Set directory and store original images to pathway:
>> FlowersDir = fullfile('C:\Users\CLE12\Pictures\Flowers');
>> imds = imageDatastore(FlowersDir, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
>> datacheck = countEachLabel(imds)

datacheck =

  7×2 table

      Label       Count
    __________    _____

    Calendula      25
    Chicory        25
    Daisy          25
    Dandelion      25
    Gardenia       25
    Geranium       25
    Knapweed       25

>> % View Sample of Each Original Image:
>> figure, montage(imds.Files(1:25:end), 'Size', [2 4])
```



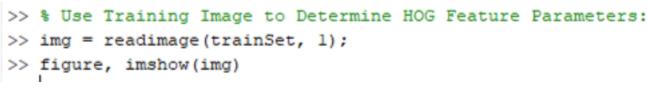


```
>> % Set Directory and store cleaned images to pathway:
>> FlowersSegmented = fullfile('C:\Users\CLE12\Pictures\FlowersSegmented');
>> imdsCleaned = imageDatastore(FlowersSegmented, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
>> datainfo = countEachLabel(imdsCleaned)

datainfo =

  7×2 table

      Label      Count
    _________    _____

    Calendula     25
    Chicory       25
    Daisy         25
    Dandelion     25
    Gardenia      25
    Geranium      25
    Knapweed      25

>> % View Sample of Each Segmented and Cleaned Image:
>> figure, montage(imdsCleaned.Files(1:25:end), 'Size', [2 4])
```

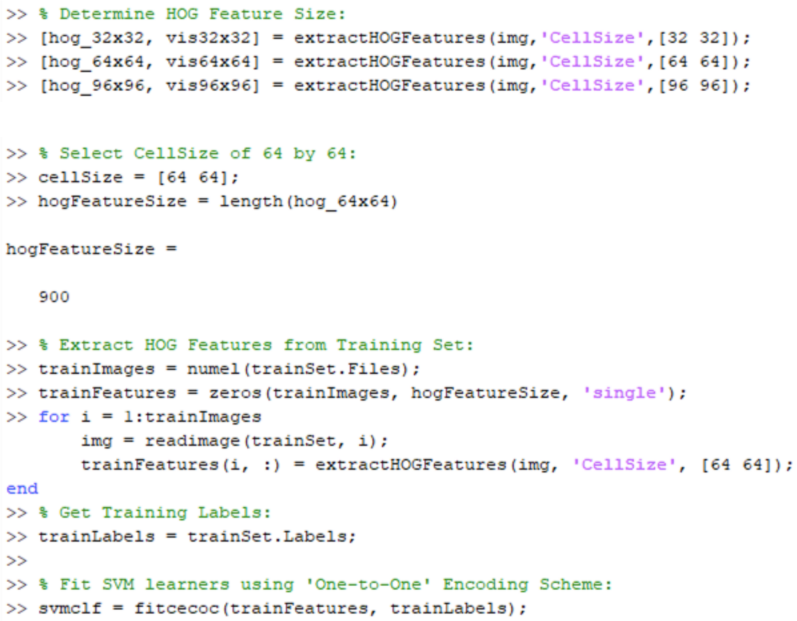




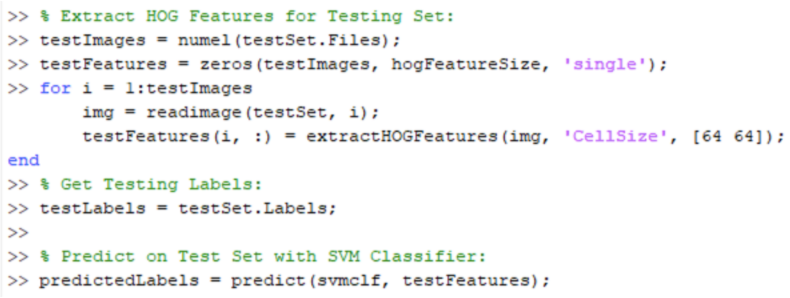
```
>> % Prepare Training and Testing Image Sets with Cleaned Images:
>> [trainSet, testSet] = splitEachLabel(imdsCleaned, 0.8, 'randomize');
>>
>> % Check Training Set:
>> countEachLabel(trainSet)

ans =

  7×2 table

      Label      Count
    _________    _____

    Calendula     20
    Chicory       20
    Daisy         20
    Dandelion     20
    Gardenia      20
    Geranium      20
    Knapweed      20
```



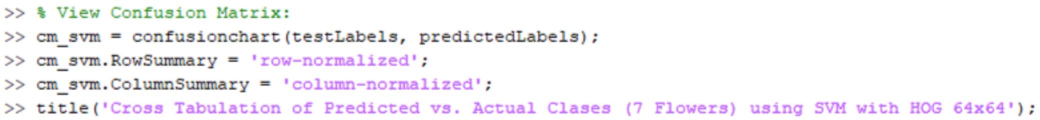
```
>> % Use Training Image to Determine HOG Feature Parameters:
>> img = readimage(trainSet, 1);
>> figure, imshow(img)
```



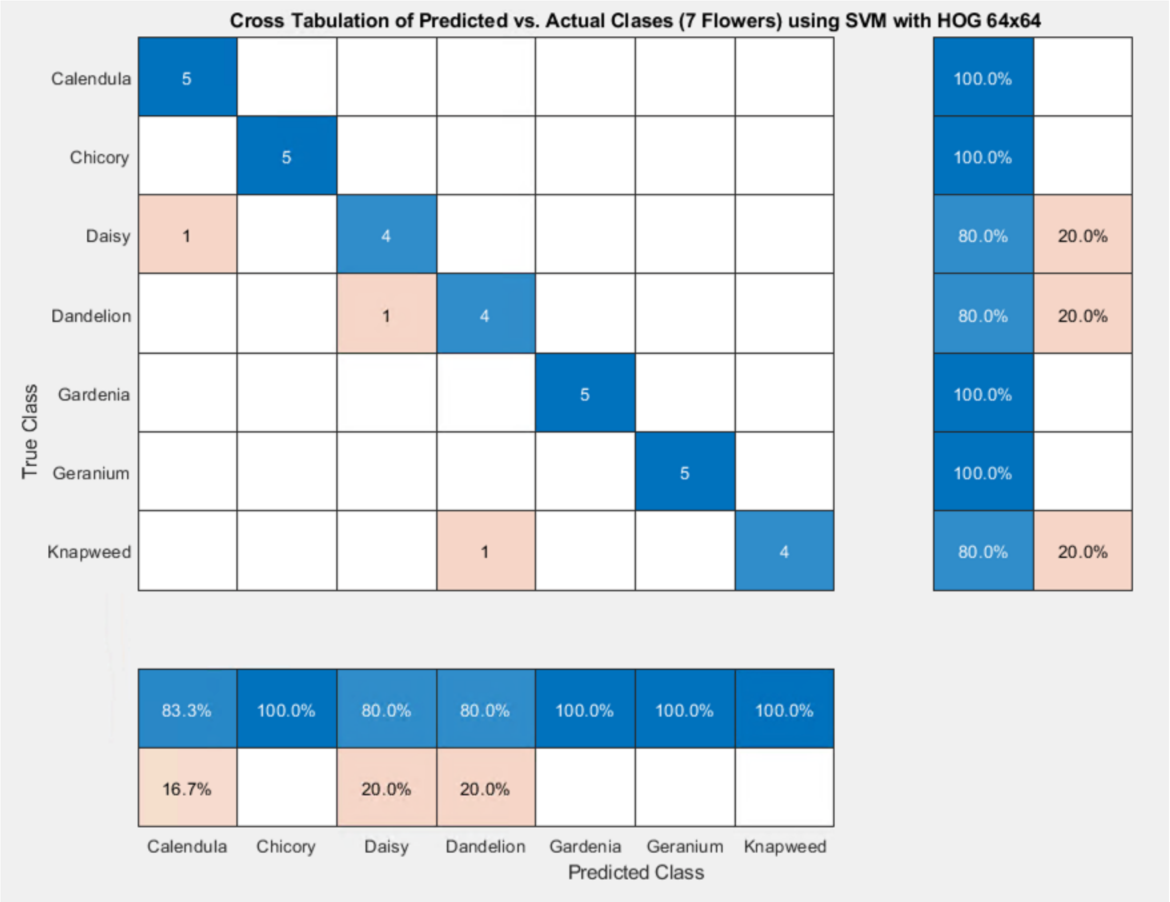
```
>> % Determine HOG Feature Size:
>> [hog_32x32, vis32x32] = extractHOGFeatures(img,'CellSize',[32 32]);
>> [hog_64x64, vis64x64] = extractHOGFeatures(img,'CellSize',[64 64]);
>> [hog_96x96, vis96x96] = extractHOGFeatures(img,'CellSize',[96 96]);


>> % Select CellSize of 64 by 64:
>> cellSize = [64 64];
>> hogFeatureSize = length(hog_64x64)

hogFeatureSize =

   900

>> % Extract HOG Features from Training Set:
>> trainImages = numel(trainSet.Files);
>> trainFeatures = zeros(trainImages, hogFeatureSize, 'single');
>> for i = 1:trainImages
      img = readimage(trainSet, i);
      trainFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [64 64]);
end
>> % Get Training Labels:
>> trainLabels = trainSet.Labels;
>>
>> % Fit SVM learners using 'One-to-One' Encoding Scheme:
>> svmclf = fitcecoc(trainFeatures, trainLabels);
```



```
>> % Extract HOG Features for Testing Set:
>> testImages = numel(testSet.Files);
>> testFeatures = zeros(testImages, hogFeatureSize, 'single');
>> for i = 1:testImages
      img = readimage(testSet, i);
      testFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [64 64]);
end
>> % Get Testing Labels:
>> testLabels = testSet.Labels;
>>
>> % Predict on Test Set with SVM Classifier:
>> predictedLabels = predict(svmclf, testFeatures);
```



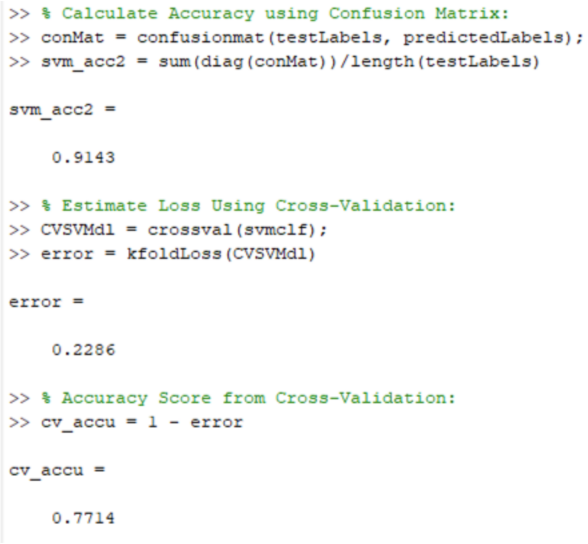
```
>> % View Confusion Matrix:
>> cm_svm = confusionchart(testLabels, predictedLabels);
>> cm_svm.RowSummary = 'row-normalized';
>> cm_svm.ColumnSummary = 'column-normalized';
>> title('Cross Tabulation of Predicted vs. Actual Clases (7 Flowers) using SVM with HOG 64x64');
```



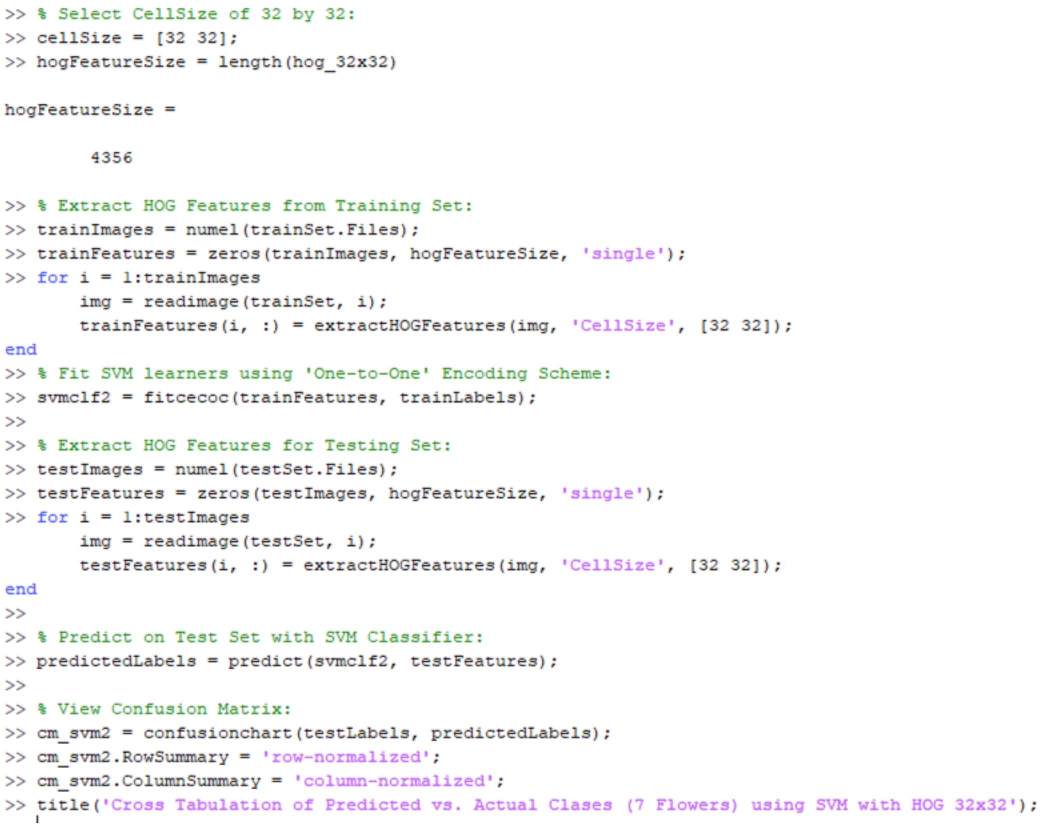
Cross Tabulation of Predicted vs. Actual Clases (7 Flowers) using SVM with HOG 64x64

True Class	Calendula	Chicory	Daisy	Dandelion	Gardenia	Geranium	Knapweed		
Calendula	5							100.0%	
Chicory		5						100.0%	
Daisy	1		4					80.0%	20.0%
Dandelion			1	4				80.0%	20.0%
Gardenia					5			100.0%	
Geranium						5		100.0%	
Knapweed				1			4	80.0%	20.0%
	83.3%	100.0%	80.0%	80.0%	100.0%	100.0%	100.0%		
	16.7%		20.0%	20.0%					

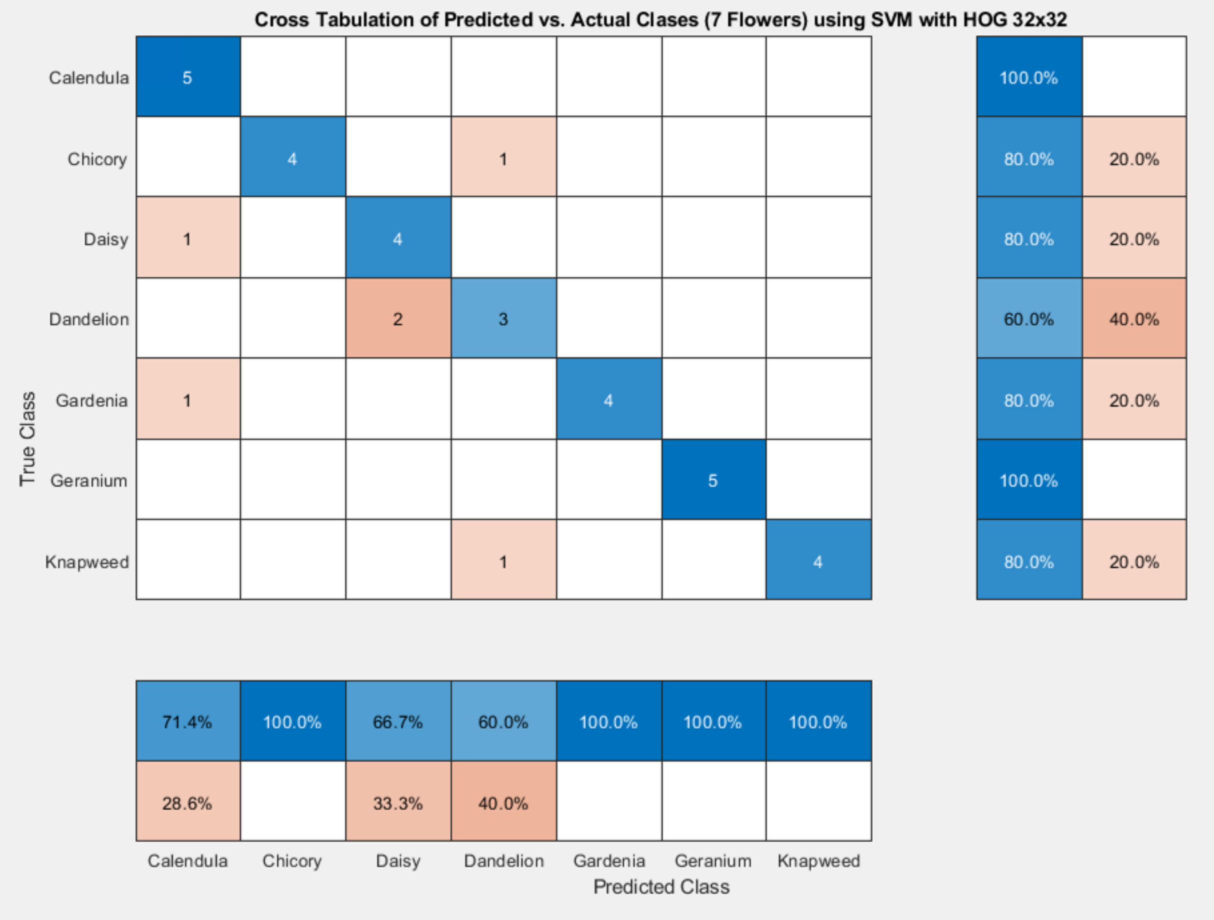
Predicted Class



```
>> % Calculate Accuracy using Confusion Matrix:
>> conMat = confusionmat(testLabels, predictedLabels);
>> svm_acc2 = sum(diag(conMat))/length(testLabels)

svm_acc2 =

    0.9143

>> % Estimate Loss Using Cross-Validation:
>> CVSVMdl = crossval(svmclf);
>> error = kfoldLoss(CVSVMdl)

error =

    0.2286

>> % Accuracy Score from Cross-Validation:
>> cv_accu = 1 - error

cv_accu =

    0.7714
```



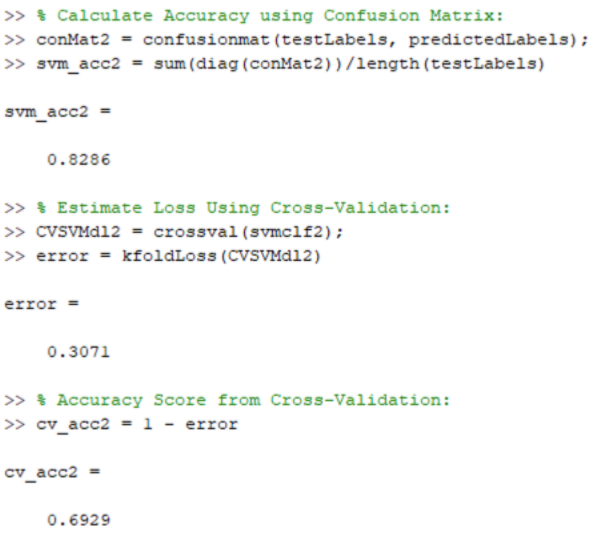
```
>> % Select CellSize of 32 by 32:
>> cellSize = [32 32];
>> hogFeatureSize = length(hog_32x32)

hogFeatureSize =

        4356

>> % Extract HOG Features from Training Set:
>> trainImages = numel(trainSet.Files);
>> trainFeatures = zeros(trainImages, hogFeatureSize, 'single');
>> for i = 1:trainImages
       img = readimage(trainSet, i);
       trainFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [32 32]);
end
>> % Fit SVM learners using 'One-to-One' Encoding Scheme:
>> svmclf2 = fitcecoc(trainFeatures, trainLabels);
>>
>> % Extract HOG Features for Testing Set:
>> testImages = numel(testSet.Files);
>> testFeatures = zeros(testImages, hogFeatureSize, 'single');
>> for i = 1:testImages
       img = readimage(testSet, i);
       testFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [32 32]);
end
>>
>> % Predict on Test Set with SVM Classifier:
>> predictedLabels = predict(svmclf2, testFeatures);
>>
>> % View Confusion Matrix:
>> cm_svm2 = confusionchart(testLabels, predictedLabels);
>> cm_svm2.RowSummary = 'row-normalized';
>> cm_svm2.ColumnSummary = 'column-normalized';
>> title('Cross Tabulation of Predicted vs. Actual Clases (7 Flowers) using SVM with HOG 32x32');
```



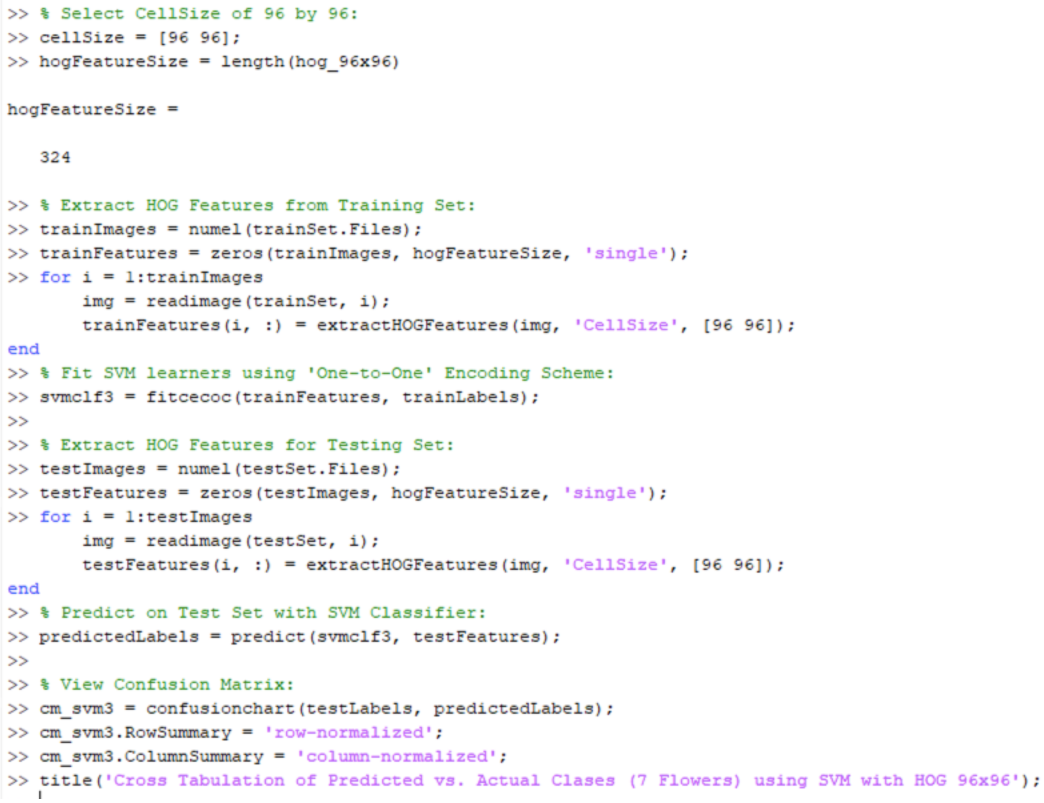
Cross Tabulation of Predicted vs. Actual Clases (7 Flowers) using SVM with HOG 32x32

True Class	Calendula	Chicory	Daisy	Dandelion	Gardenia	Geranium	Knapweed		
Calendula	5							100.0%	
Chicory		4		1				80.0%	20.0%
Daisy	1		4					80.0%	20.0%
Dandelion			2	3				60.0%	40.0%
Gardenia	1				4			80.0%	20.0%
Geranium						5		100.0%	
Knapweed				1			4	80.0%	20.0%
	71.4%	100.0%	66.7%	60.0%	100.0%	100.0%	100.0%		
	28.6%		33.3%	40.0%					

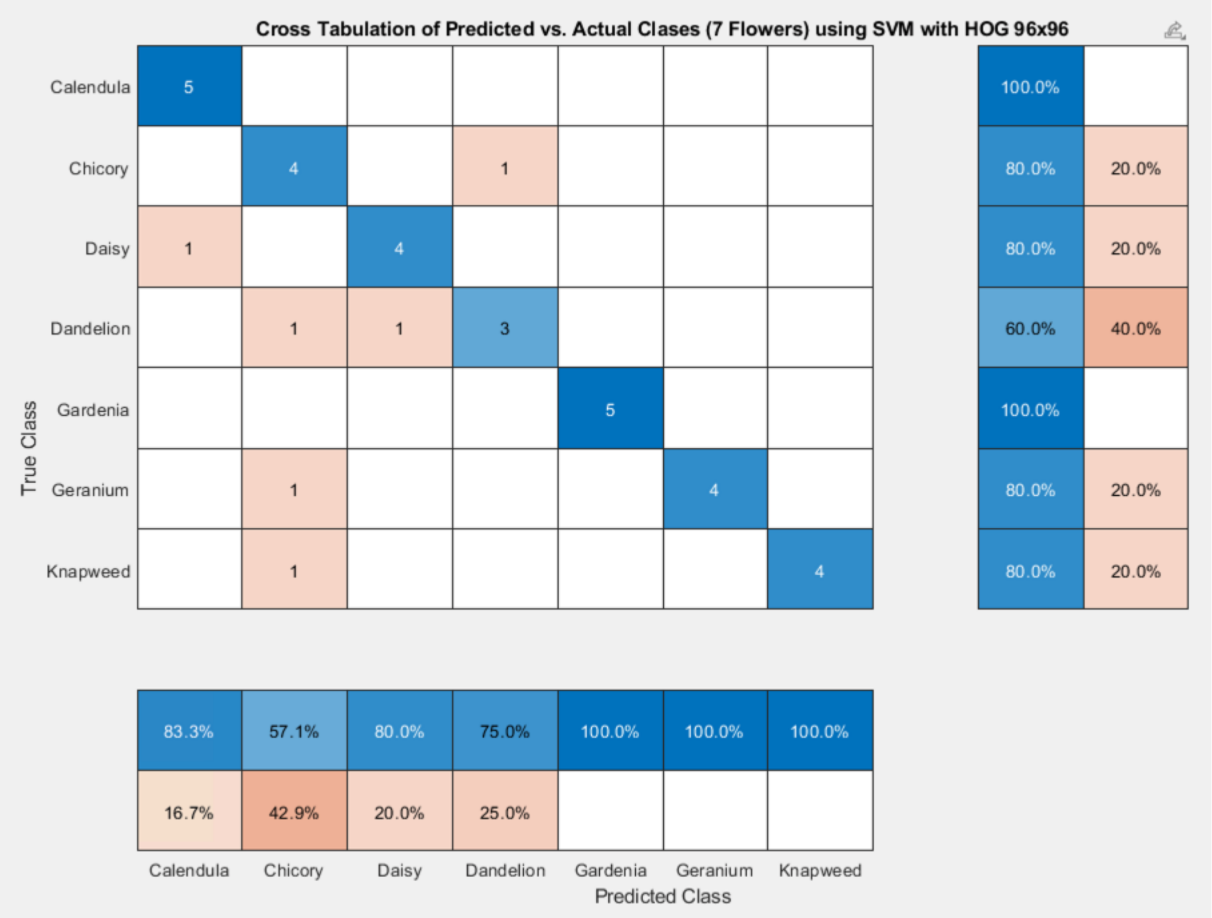
Predicted Class



```
>> % Calculate Accuracy using Confusion Matrix:
>> conMat2 = confusionmat(testLabels, predictedLabels);
>> svm_acc2 = sum(diag(conMat2))/length(testLabels)

svm_acc2 =

    0.8286

>> % Estimate Loss Using Cross-Validation:
>> CVSVMdl2 = crossval(svmclf2);
>> error = kfoldLoss(CVSVMdl2)

error =

    0.3071

>> % Accuracy Score from Cross-Validation:
>> cv_acc2 = 1 - error

cv_acc2 =

    0.6929
```



```
>> % Select CellSize of 96 by 96:
>> cellSize = [96 96];
>> hogFeatureSize = length(hog_96x96)

hogFeatureSize =

   324

>> % Extract HOG Features from Training Set:
>> trainImages = numel(trainSet.Files);
>> trainFeatures = zeros(trainImages, hogFeatureSize, 'single');
>> for i = 1:trainImages
       img = readimage(trainSet, i);
       trainFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [96 96]);
end
>> % Fit SVM learners using 'One-to-One' Encoding Scheme:
>> svmclf3 = fitcecoc(trainFeatures, trainLabels);
>> 
>> % Extract HOG Features for Testing Set:
>> testImages = numel(testSet.Files);
>> testFeatures = zeros(testImages, hogFeatureSize, 'single');
>> for i = 1:testImages
       img = readimage(testSet, i);
       testFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [96 96]);
end
>> % Predict on Test Set with SVM Classifier:
>> predictedLabels = predict(svmclf3, testFeatures);
>> 
>> % View Confusion Matrix:
>> cm_svm3 = confusionchart(testLabels, predictedLabels);
>> cm_svm3.RowSummary = 'row-normalized';
>> cm_svm3.ColumnSummary = 'column-normalized';
>> title('Cross Tabulation of Predicted vs. Actual Clases (7 Flowers) using SVM with HOG 96x96');
```

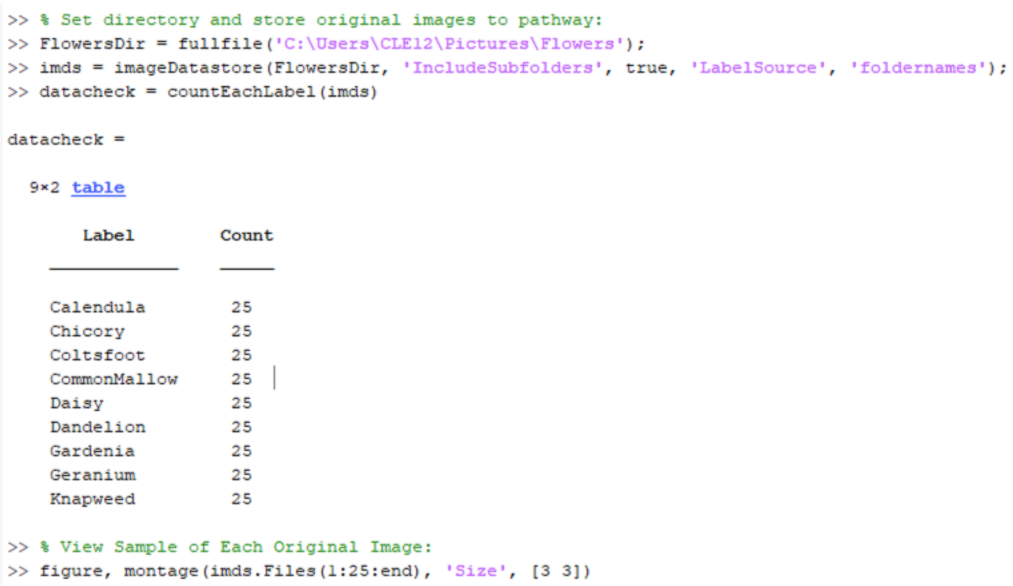


Cross Tabulation of Predicted vs. Actual Clases (7 Flowers) using SVM with HOG 96x96

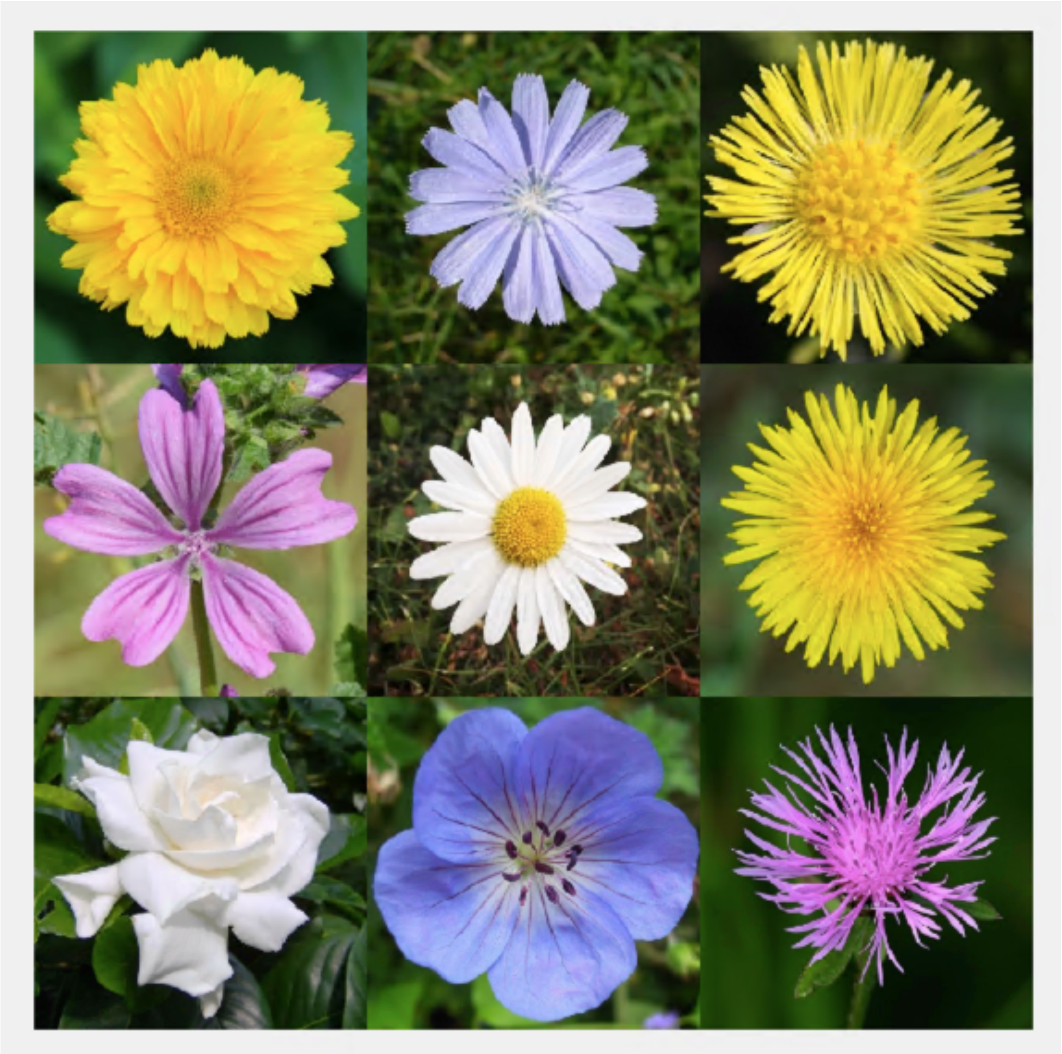
True Class \ Predicted Class	Calendula	Chicory	Daisy	Dandelion	Gardenia	Geranium	Knapweed		
Calendula	5							100.0%	
Chicory		4		1				80.0%	20.0%
Daisy	1		4					80.0%	20.0%
Dandelion		1	1	3				60.0%	40.0%
Gardenia					5			100.0%	
Geranium		1				4		80.0%	20.0%
Knapweed		1					4	80.0%	20.0%
	83.3%	57.1%	80.0%	75.0%	100.0%	100.0%	100.0%		
	16.7%	42.9%	20.0%	25.0%					

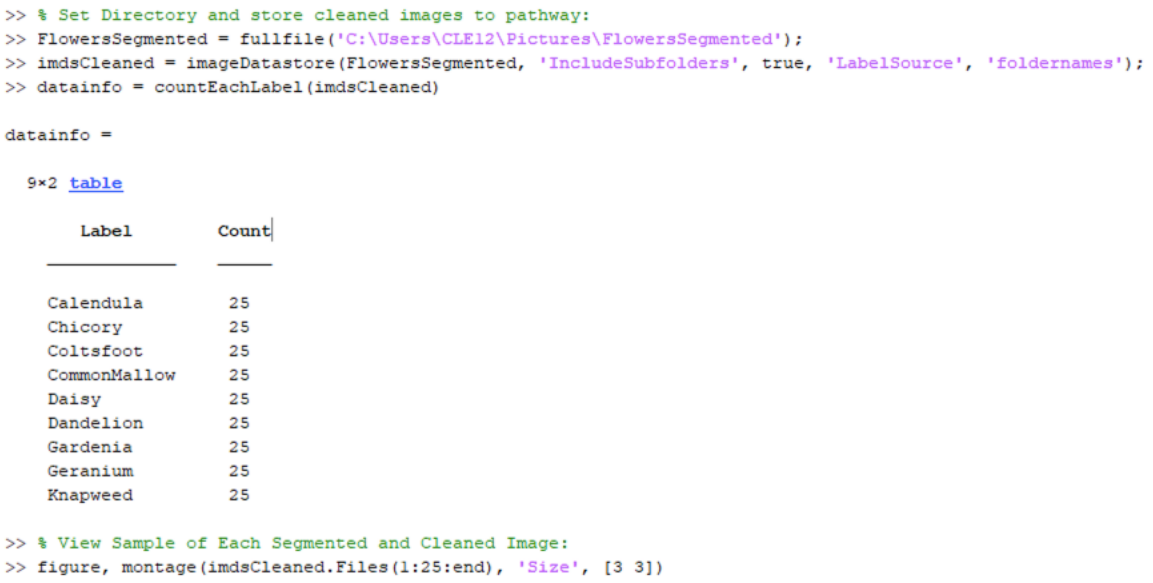
```
>> % Calculate Accuracy using Confusion Matrix:
>> conMat3 = confusionmat(testLabels, predictedLabels);
>> svm_acc3 = sum(diag(conMat3))/length(testLabels)

svm_acc3 =

    0.8286

>> % Estimate Loss Using Cross-Validation:
>> CVSVMdl3 = crossval(svmclf3);
>> error = kfoldLoss(CVSVMdl3)

error =

    0.3000

>> % Accuracy Score from Cross-Validation:
>> cv_acc3 = 1 - error

cv_acc3 =

    0.7000
```

9-Class: Calendula, Chicory, Daisy, Dandelion, Gardenia, Geranium, and Knapweed

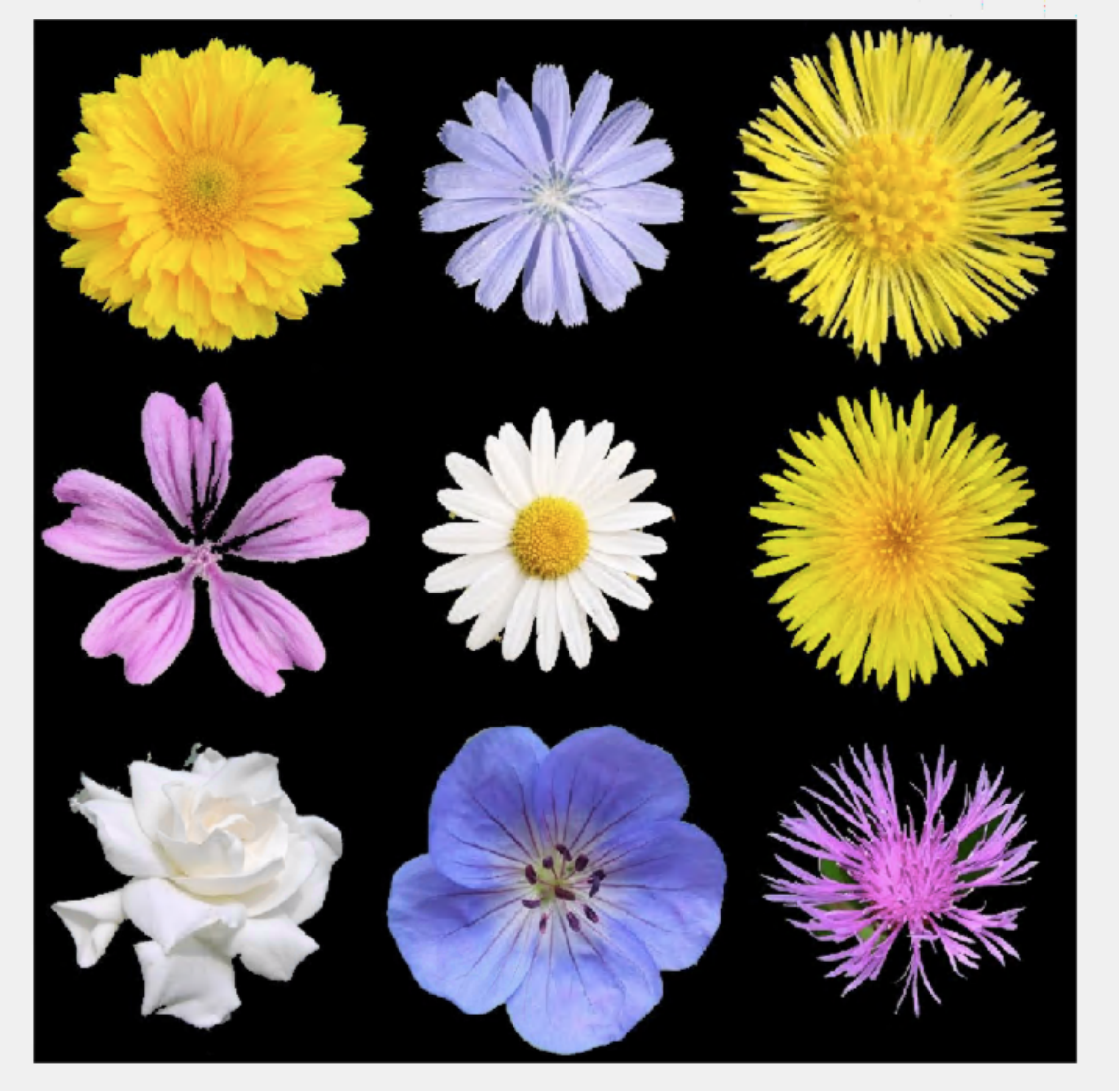


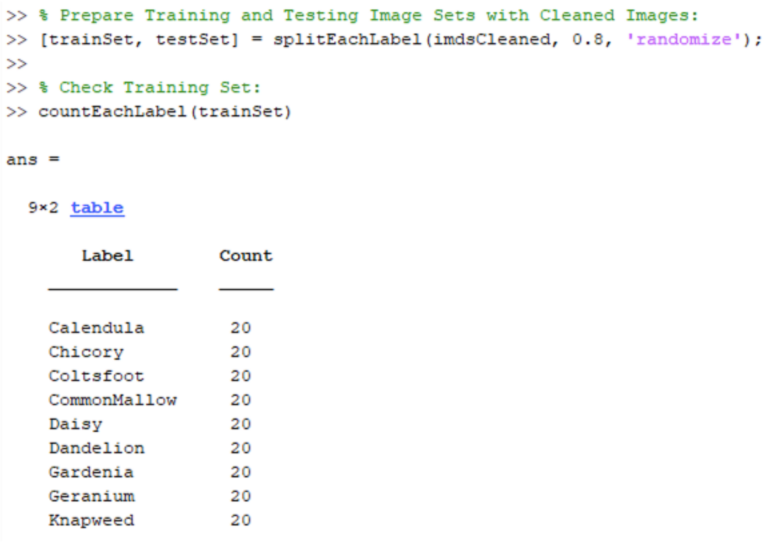
```
>> % Set directory and store original images to pathway:
>> FlowersDir = fullfile('C:\Users\CLE12\Pictures\Flowers');
>> imds = imageDatastore(FlowersDir, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
>> datacheck = countEachLabel(imds)

datacheck =

  9×2 table

       Label        Count
    ____________    _____

    Calendula        25
    Chicory          25
    Coltsfoot        25
    CommonMallow     25
    Daisy            25
    Dandelion        25
    Gardenia         25
    Geranium         25
    Knapweed         25

>> % View Sample of Each Original Image:
>> figure, montage(imds.Files(1:25:end), 'Size', [3 3])
```



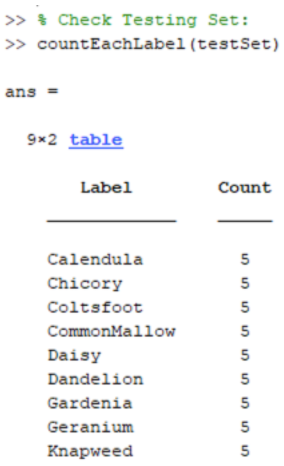


```
>> % Set Directory and store cleaned images to pathway:
>> FlowersSegmented = fullfile('C:\Users\CLE12\Pictures\FlowersSegmented');
>> imdsCleaned = imageDatastore(FlowersSegmented, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
>> datainfo = countEachLabel(imdsCleaned)

datainfo =

  9×2 table

        Label        Count
    ____________     _____

    Calendula         25
    Chicory           25
    Coltsfoot         25
    CommonMallow      25
    Daisy             25
    Dandelion         25
    Gardenia          25
    Geranium          25
    Knapweed          25

>> % View Sample of Each Segmented and Cleaned Image:
>> figure, montage(imdsCleaned.Files(1:25:end), 'Size', [3 3])
```

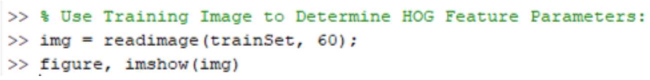




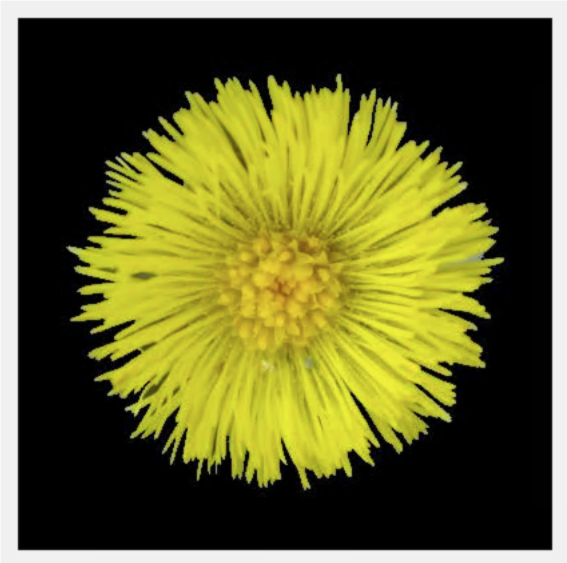
```
>> % Prepare Training and Testing Image Sets with Cleaned Images:
>> [trainSet, testSet] = splitEachLabel(imdsCleaned, 0.8, 'randomize');
>>
>> % Check Training Set:
>> countEachLabel(trainSet)

ans =

  9×2 table

       Label          Count
    ____________    _____

    Calendula         20
    Chicory           20
    Coltsfoot         20
    CommonMallow      20
    Daisy             20
    Dandelion         20
    Gardenia          20
    Geranium          20
    Knapweed          20
```

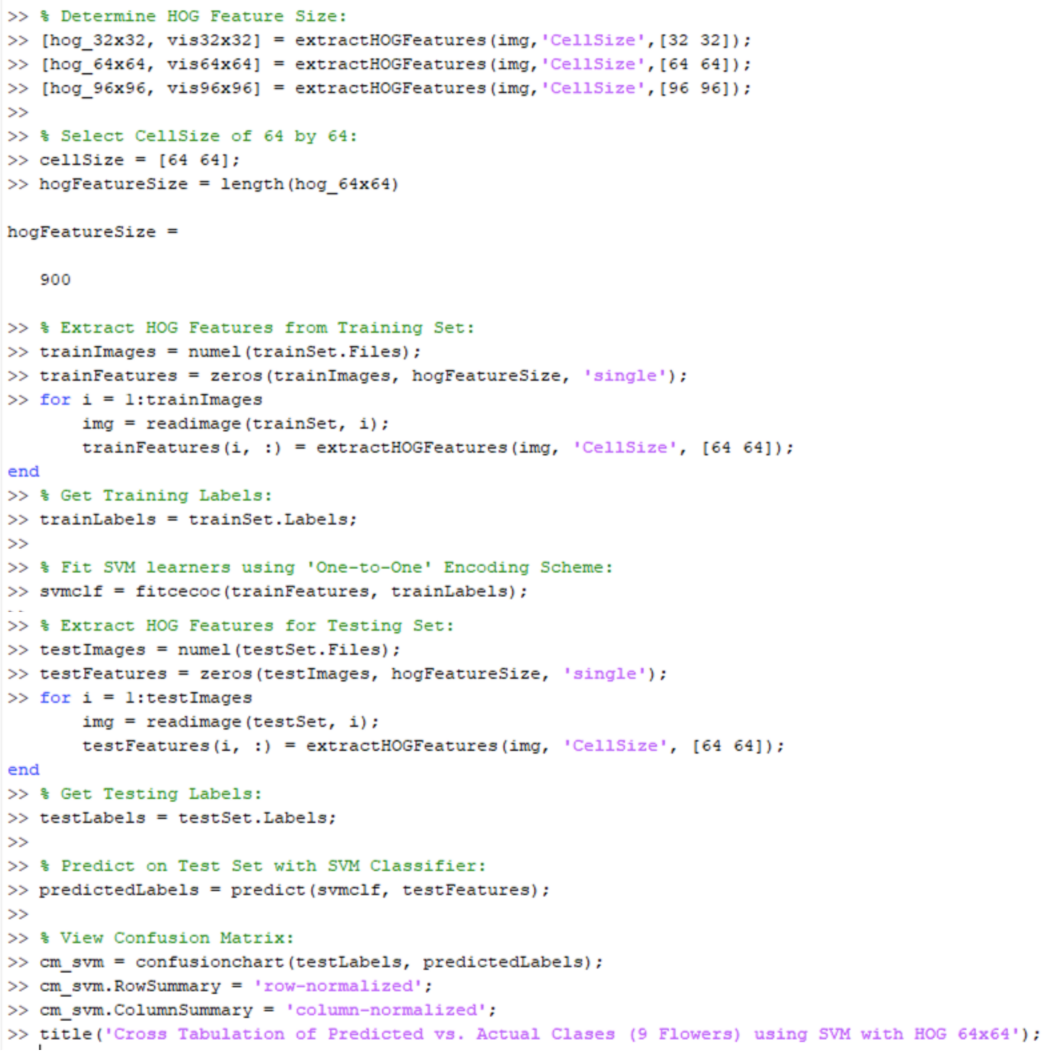


```
>> % Check Testing Set:
>> countEachLabel(testSet)

ans =

  9×2 table

       Label          Count
    ____________    _____

    Calendula         5
    Chicory           5
    Coltsfoot         5
    CommonMallow      5
    Daisy             5
    Dandelion         5
    Gardenia          5
    Geranium          5
    Knapweed          5
```

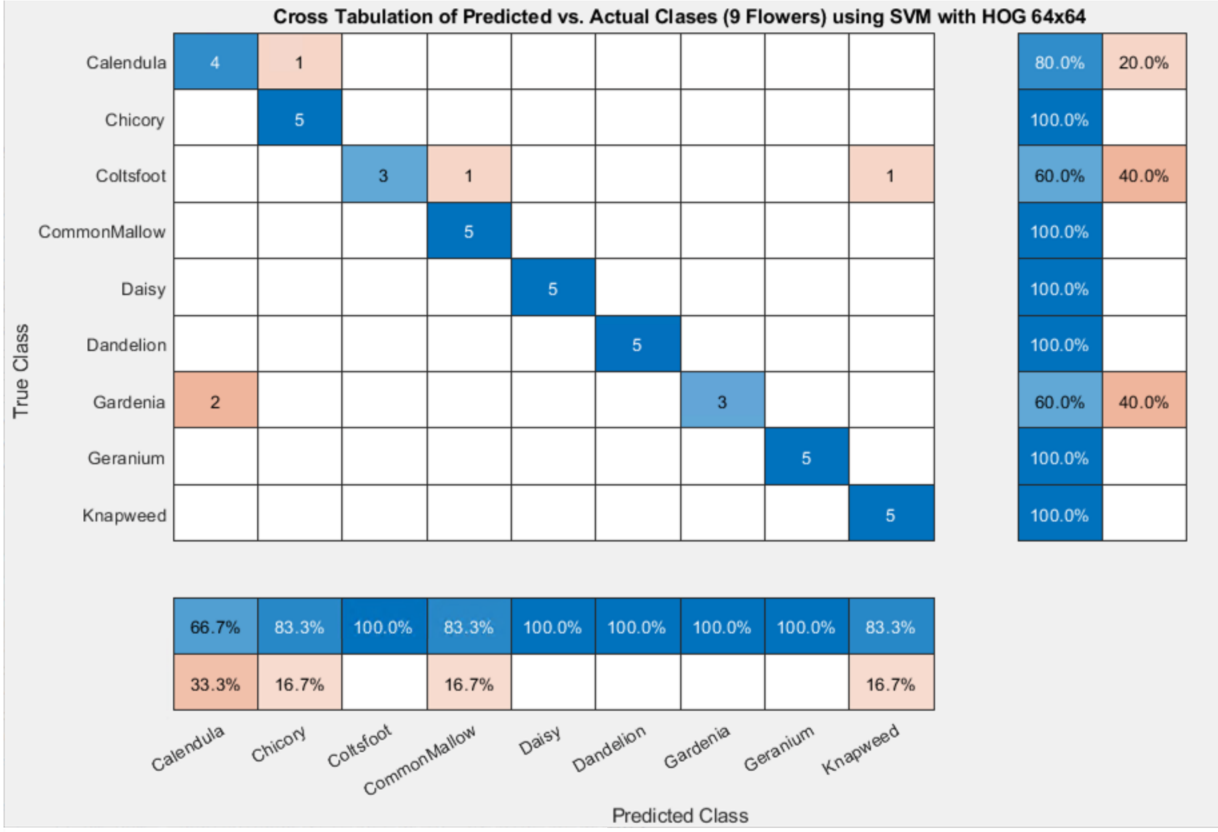


```
>> % Use Training Image to Determine HOG Feature Parameters:
>> img = readimage(trainSet, 60);
>> figure, imshow(img)
```





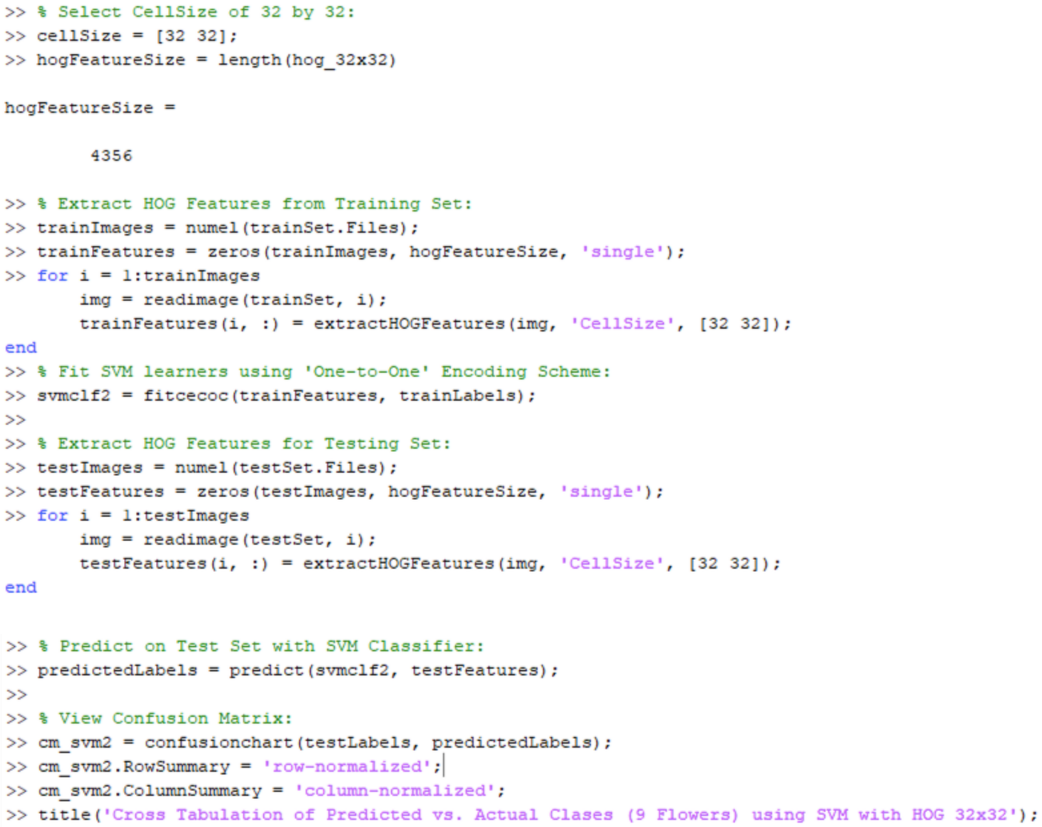
```
>> % Determine HOG Feature Size:
>> [hog_32x32, vis32x32] = extractHOGFeatures(img,'CellSize',[32 32]);
>> [hog_64x64, vis64x64] = extractHOGFeatures(img,'CellSize',[64 64]);
>> [hog_96x96, vis96x96] = extractHOGFeatures(img,'CellSize',[96 96]);
>>
>> % Select CellSize of 64 by 64:
>> cellSize = [64 64];
>> hogFeatureSize = length(hog_64x64)

hogFeatureSize =

   900

>> % Extract HOG Features from Training Set:
>> trainImages = numel(trainSet.Files);
>> trainFeatures = zeros(trainImages, hogFeatureSize, 'single');
>> for i = 1:trainImages
       img = readimage(trainSet, i);
       trainFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [64 64]);
end
>> % Get Training Labels:
>> trainLabels = trainSet.Labels;
>>
>> % Fit SVM learners using 'One-to-One' Encoding Scheme:
>> svmclf = fitcecoc(trainFeatures, trainLabels);
>> % Extract HOG Features for Testing Set:
>> testImages = numel(testSet.Files);
>> testFeatures = zeros(testImages, hogFeatureSize, 'single');
>> for i = 1:testImages
       img = readimage(testSet, i);
       testFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [64 64]);
end
>> % Get Testing Labels:
>> testLabels = testSet.Labels;
>>
>> % Predict on Test Set with SVM Classifier:
>> predictedLabels = predict(svmclf, testFeatures);
>>
>> % View Confusion Matrix:
>> cm_svm = confusionchart(testLabels, predictedLabels);
>> cm_svm.RowSummary = 'row-normalized';
>> cm_svm.ColumnSummary = 'column-normalized';
>> title('Cross Tabulation of Predicted vs. Actual Clases (9 Flowers) using SVM with HOG 64x64');
```



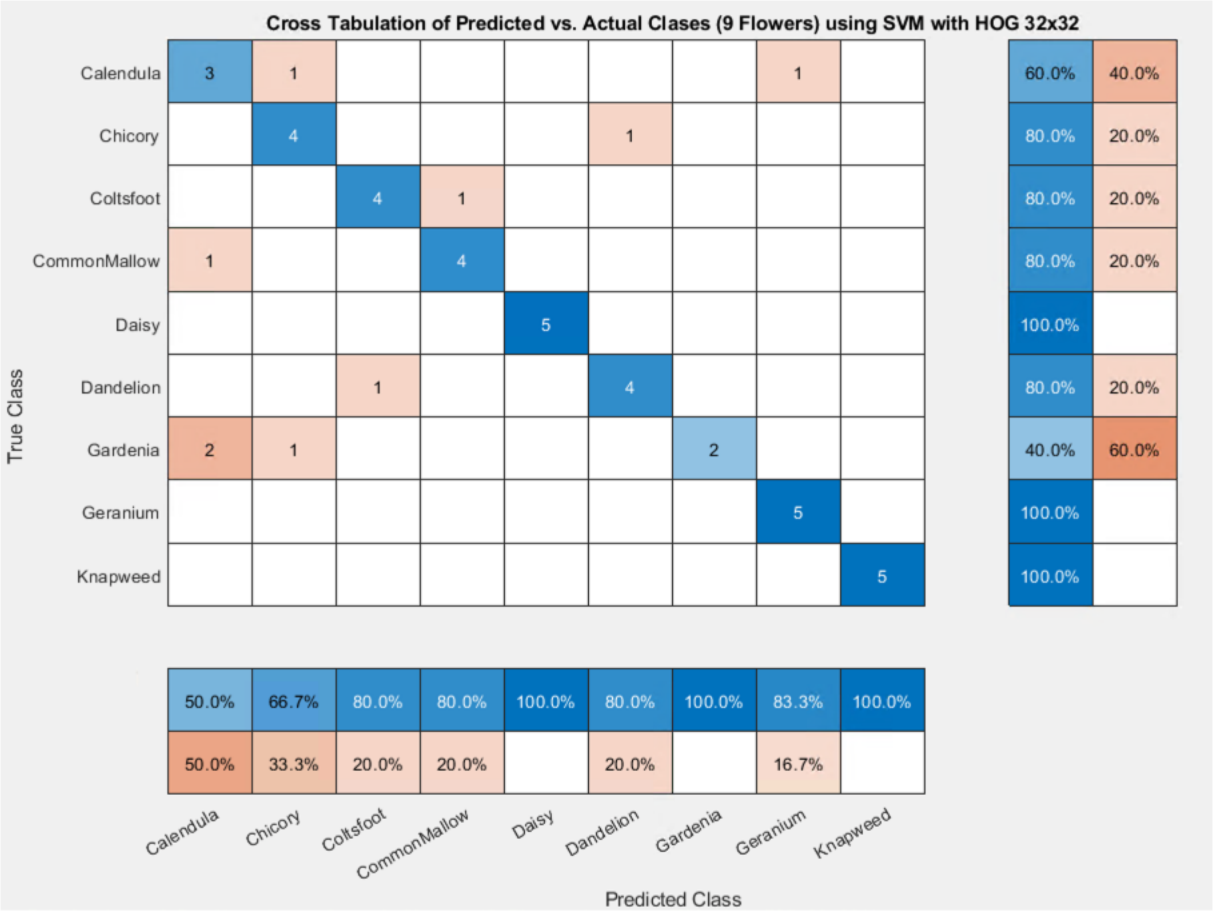
Cross Tabulation of Predicted vs. Actual Clases (9 Flowers) using SVM with HOG 64x64

	Calendula	Chicory	Coltsfoot	CommonMallow	Daisy	Dandelion	Gardenia	Geranium	Knapweed		
Calendula	4	1								80.0%	20.0%
Chicory		5								100.0%	
Coltsfoot			3	1					1	60.0%	40.0%
CommonMallow				5						100.0%	
Daisy					5					100.0%	
Dandelion						5				100.0%	
Gardenia	2						3			60.0%	40.0%
Geranium								5		100.0%	
Knapweed									5	100.0%	
	66.7%	83.3%	100.0%	83.3%	100.0%	100.0%	100.0%	100.0%	83.3%		
	33.3%	16.7%		16.7%					16.7%		

```
>> % Calculate Accuracy using Confusion Matrix:
>> conMat = confusionmat(testLabels, predictedLabels);
>> svm_acc2 = sum(diag(conMat))/length(testLabels)

svm_acc2 =

    0.8889

>> % Estimate Loss Using Cross-Validation:
>> CVSVMdl = crossval(svmclf);
>> error = kfoldLoss(CVSVMdl)

error =

    0.3111

>> % Accuracy Score from Cross-Validation:
>> cv_accu = 1 - error

cv_accu =

    0.6889
```

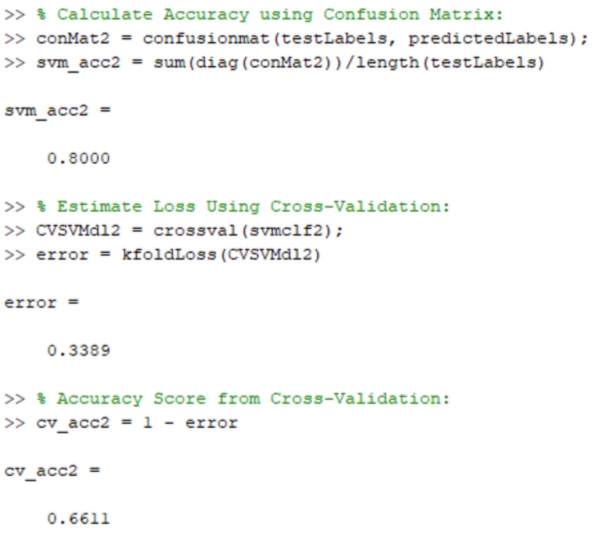


```
>> % Select CellSize of 32 by 32:
>> cellSize = [32 32];
>> hogFeatureSize = length(hog_32x32)

hogFeatureSize =

        4356

>> % Extract HOG Features from Training Set:
>> trainImages = numel(trainSet.Files);
>> trainFeatures = zeros(trainImages, hogFeatureSize, 'single');
>> for i = 1:trainImages
       img = readimage(trainSet, i);
       trainFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [32 32]);
end
>> % Fit SVM learners using 'One-to-One' Encoding Scheme:
>> svmclf2 = fitcecoc(trainFeatures, trainLabels);
>>
>> % Extract HOG Features for Testing Set:
>> testImages = numel(testSet.Files);
>> testFeatures = zeros(testImages, hogFeatureSize, 'single');
>> for i = 1:testImages
       img = readimage(testSet, i);
       testFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [32 32]);
end

>> % Predict on Test Set with SVM Classifier:
>> predictedLabels = predict(svmclf2, testFeatures);
>>
>> % View Confusion Matrix:
>> cm_svm2 = confusionchart(testLabels, predictedLabels);
>> cm_svm2.RowSummary = 'row-normalized';
>> cm_svm2.ColumnSummary = 'column-normalized';
>> title('Cross Tabulation of Predicted vs. Actual Clases (9 Flowers) using SVM with HOG 32x32');
```

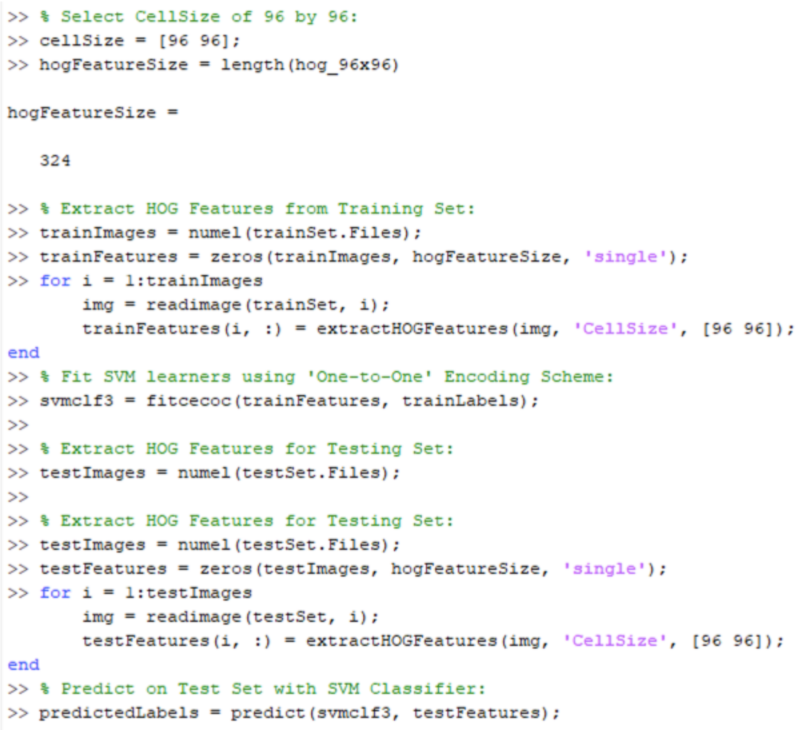


Cross Tabulation of Predicted vs. Actual Clases (9 Flowers) using SVM with HOG 32x32

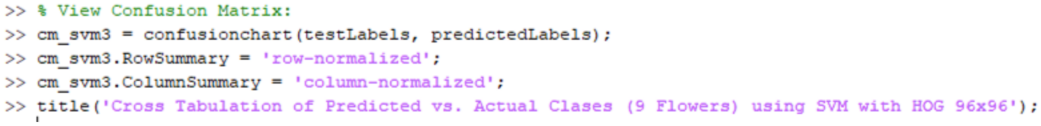
	Calendula	Chicory	Coltsfoot	CommonMallow	Daisy	Dandelion	Gardenia	Geranium	Knapweed		
Calendula	3	1						1		60.0%	40.0%
Chicory		4				1				80.0%	20.0%
Coltsfoot			4	1						80.0%	20.0%
CommonMallow	1			4						80.0%	20.0%
Daisy					5					100.0%	
Dandelion			1			4				80.0%	20.0%
Gardenia	2	1					2			40.0%	60.0%
Geranium								5		100.0%	
Knapweed									5	100.0%	
	50.0%	66.7%	80.0%	80.0%	100.0%	80.0%	100.0%	83.3%	100.0%		
	50.0%	33.3%	20.0%	20.0%		20.0%		16.7%			



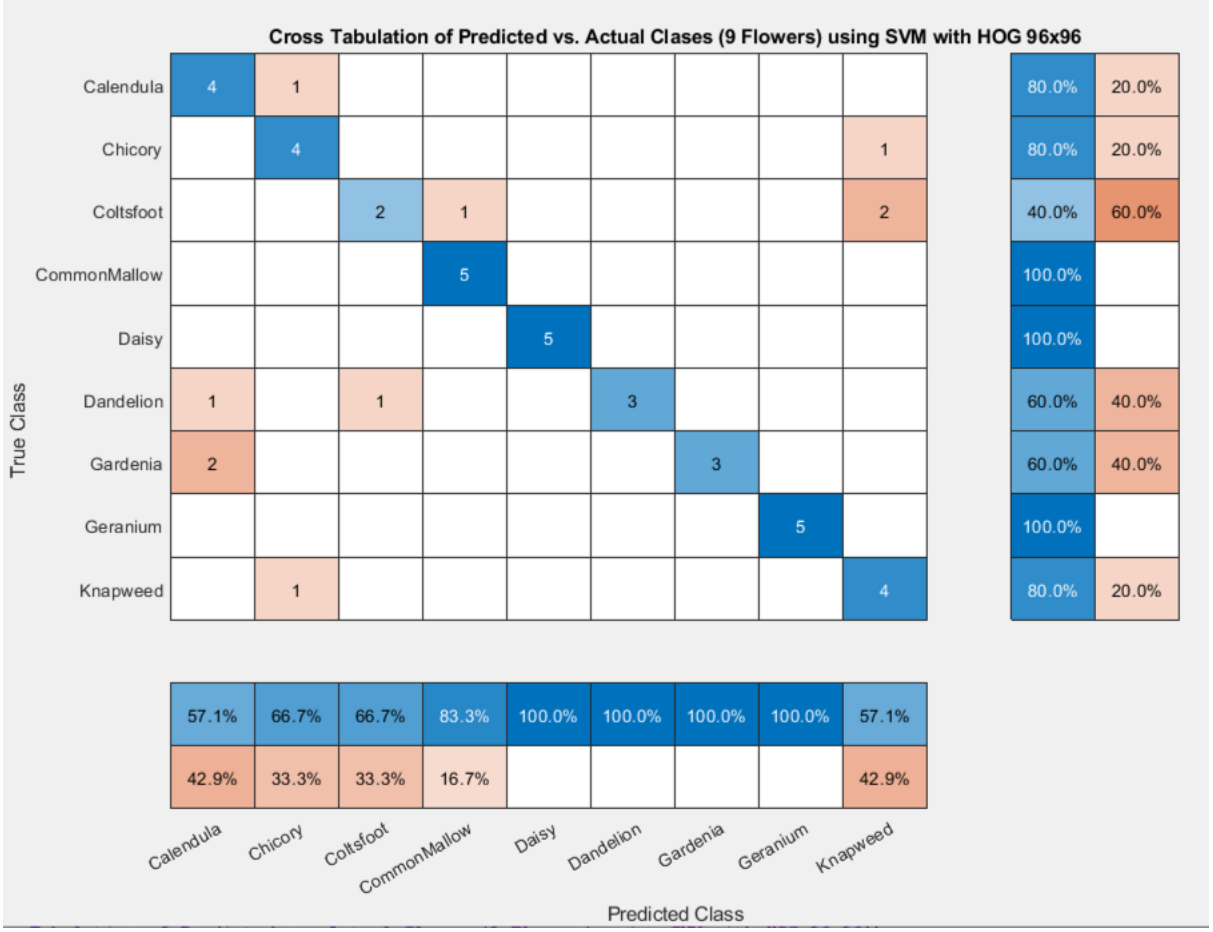
```
>> % Calculate Accuracy using Confusion Matrix:
>> conMat2 = confusionmat(testLabels, predictedLabels);
>> svm_acc2 = sum(diag(conMat2))/length(testLabels)

svm_acc2 =

    0.8000

>> % Estimate Loss Using Cross-Validation:
>> CVSVMdl2 = crossval(svmclf2);
>> error = kfoldLoss(CVSVMdl2)

error =

    0.3389

>> % Accuracy Score from Cross-Validation:
>> cv_acc2 = 1 - error

cv_acc2 =

    0.6611
```



```
>> % Select CellSize of 96 by 96:
>> cellSize = [96 96];
>> hogFeatureSize = length(hog_96x96)

hogFeatureSize =

   324

>> % Extract HOG Features from Training Set:
>> trainImages = numel(trainSet.Files);
>> trainFeatures = zeros(trainImages, hogFeatureSize, 'single');
>> for i = 1:trainImages
       img = readimage(trainSet, i);
       trainFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [96 96]);
end
>> % Fit SVM learners using 'One-to-One' Encoding Scheme:
>> svmclf3 = fitcecoc(trainFeatures, trainLabels);
>> 
>> % Extract HOG Features for Testing Set:
>> testImages = numel(testSet.Files);
>> 
>> % Extract HOG Features for Testing Set:
>> testImages = numel(testSet.Files);
>> testFeatures = zeros(testImages, hogFeatureSize, 'single');
>> for i = 1:testImages
       img = readimage(testSet, i);
       testFeatures(i, :) = extractHOGFeatures(img, 'CellSize', [96 96]);
end
>> % Predict on Test Set with SVM Classifier:
>> predictedLabels = predict(svmclf3, testFeatures);
```

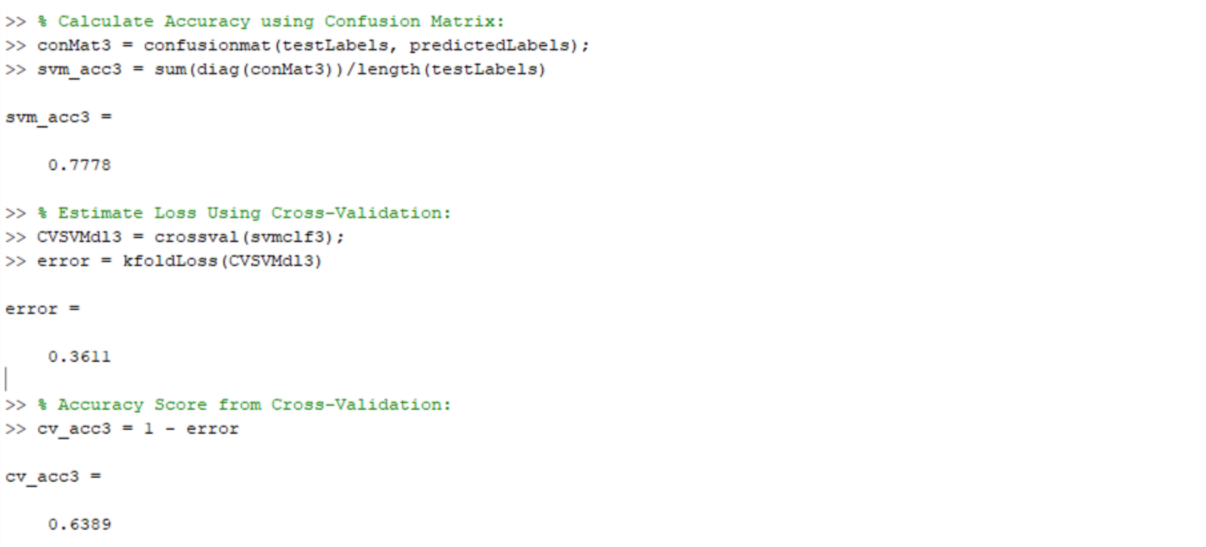


```
>> % View Confusion Matrix:
>> cm_svm3 = confusionchart(testLabels, predictedLabels);
>> cm_svm3.RowSummary = 'row-normalized';
>> cm_svm3.ColumnSummary = 'column-normalized';
>> title('Cross Tabulation of Predicted vs. Actual Clases (9 Flowers) using SVM with HOG 96x96');
```



Cross Tabulation of Predicted vs. Actual Clases (9 Flowers) using SVM with HOG 96x96

	Calendula	Chicory	Coltsfoot	CommonMallow	Daisy	Dandelion	Gardenia	Geranium	Knapweed		
Calendula	4	1								80.0%	20.0%
Chicory		4							1	80.0%	20.0%
Coltsfoot			2	1					2	40.0%	60.0%
CommonMallow				5						100.0%	
Daisy					5					100.0%	
Dandelion	1		1			3				60.0%	40.0%
Gardenia	2						3			60.0%	40.0%
Geranium								5		100.0%	
Knapweed		1							4	80.0%	20.0%
	57.1%	66.7%	66.7%	83.3%	100.0%	100.0%	100.0%	100.0%	57.1%		
	42.9%	33.3%	33.3%	16.7%					42.9%		



```
>> % Calculate Accuracy using Confusion Matrix:
>> conMat3 = confusionmat(testLabels, predictedLabels);
>> svm_acc3 = sum(diag(conMat3))/length(testLabels)

svm_acc3 =

    0.7778

>> % Estimate Loss Using Cross-Validation:
>> CVSVMdl3 = crossval(svmclf3);
>> error = kfoldLoss(CVSVMdl3)

error =

    0.3611

>> % Accuracy Score from Cross-Validation:
>> cv_acc3 = 1 - error

cv_acc3 =

    0.6389
```