DTSA 5510 - Unsupervised Algorithms in Machine Learning Final Project: Unsupervised Learning in the Case of UAV Image Classification

Israel Johnson

August 21, 2023

University of Colorado **Boulder**

PROBLEM STATEMENT

- There has been ongoing research in the area of applying artificial intelligence and machine learning (AI/ML) techniques in detecting unmanned aerial vehicles (UAVs). These applications usually relate to a combat scenario or military use case.
- Researchers still explore different ML model approaches for analyzing images and video data related to UAVs, especially when images or videos can contain other objects in the environment that can make UAV detection more challenging.

PROJECT OBJECTIVE

- The goal for this project is to apply multiple unsupervised learning model techniques for UAV classification, conduct comparative analysis to find the best unsupervised model within the scope of the project, and make final comparisons of the best unsupervised model results with the results of a supervised learning model approach (in order to determine which specific approach performs better for this use case).

- Three unsupervised learning models are applied to this project, which includes the following: a hierarchical clustering model and two anomaly detection models: Non-negative matrix factorization (NMF) and Multi-layer Perceptron Regressor (MLP). A support vector machine (SVM) classifier will serve as the supervised learning model approach for comparison purposes.

DATA SOURCES

1.) Gareth. 2022. *UAV Detection Dataset.* Retrieved from: https://www.kaggle.com/datasets/nelyg8002000/uav-detection-dataset-images

2.) Ailurophile. 2019. *200 Bird Species with 11,788 Images.* Retrieved from: https://www.kaggle.com/datasets/veeralakrishna/200-bird-species-with-11788-images

3.) Kaggle (Research Code Competition). 2019. *Generative Dog Images*. Retrieved from: https://www.kaggle.com/competitions/generative-dog-images/overview

GITHUB REPOSITORY SOURCE

- **Source:** https://github.com/IsraelsLibrary/DTSA_5510_Unsupervised_Algorithms__In_Machine_Learning

DATA CLEANING PROCESS

- For initial data processing, the software reads in three datasets, each dataset focusing on a different category of images (UAVs, Birds, and Dogs).
- Two helper functions clean the datasets by identifying and removing any corrupted images in each dataset.
- Principal Component Analysis (PCA) and flattening the image data helps in reducing dimensionality of the datasets and processing the data for model analysis.

DATA CLEANING PROCESS

```python
# Helper functions that check for any corrupted images in the selected directories.
# If present, those corrupted image files are removed.

def is_corrupted(image_path):
    try:
        Image.open(image_path).verify()
        return False
    except (IOError, SyntaxError):
        return True

def delete_corrupted_images(directory):
    return [
        filename
        for filename in os.listdir(directory)
        if is_corrupted(os.path.join(directory, filename)) and os.remove(os.path.join(directory, filename))
    ]

```

University of Colorado **Boulder**

DATA TRANSFORMATIONS

```python
# Reading in the image datasets
uav_imgs = []
birds_imgs = []
dogs_imgs = []

uav_path = "data/uavs_datasets/"
bird_path = "data/birds_datasets/"
dogs_path = "data/dogs_datasets/"
collection = {uav_path:uav_imgs, bird_path:birds_imgs, dogs_path:dogs_imgs}

# Checking and removing any corrupted images from the file directories. Storing the valid images
# into a temporary data structure, with image datasets assigned to their corresponding image categories.
for path, imgs in collection.items():
    # Remove corrupted images before loading
    corrupted_images = delete_corrupted_images(path)
    if len(corrupted_images) > 0:
        print("Corrupted images removed:", corrupted_images)
    else:
        print("Found no corrupted images in the selected directory.")

    goodies = [img for img in os.listdir(path) if img.endswith('.jpg')]
    imgs.extend([np.array(Image.open(os.path.join(path, img))) for img in goodies])
```

```python
# Reformatting the custom dataset to contain an equal number of images
# for each category (for further testing)
mod_collection = {'UAVs': collection["data/uavs_datasets/"][:5],
                  'Birds': collection["data/birds_datasets/"][:5],
                  'Dogs': collection["data/dogs_datasets/"][:5]}

df = pd.DataFrame(mod_collection)
```

University of Colorado **Boulder**

DATA TRANSFORMATIONS

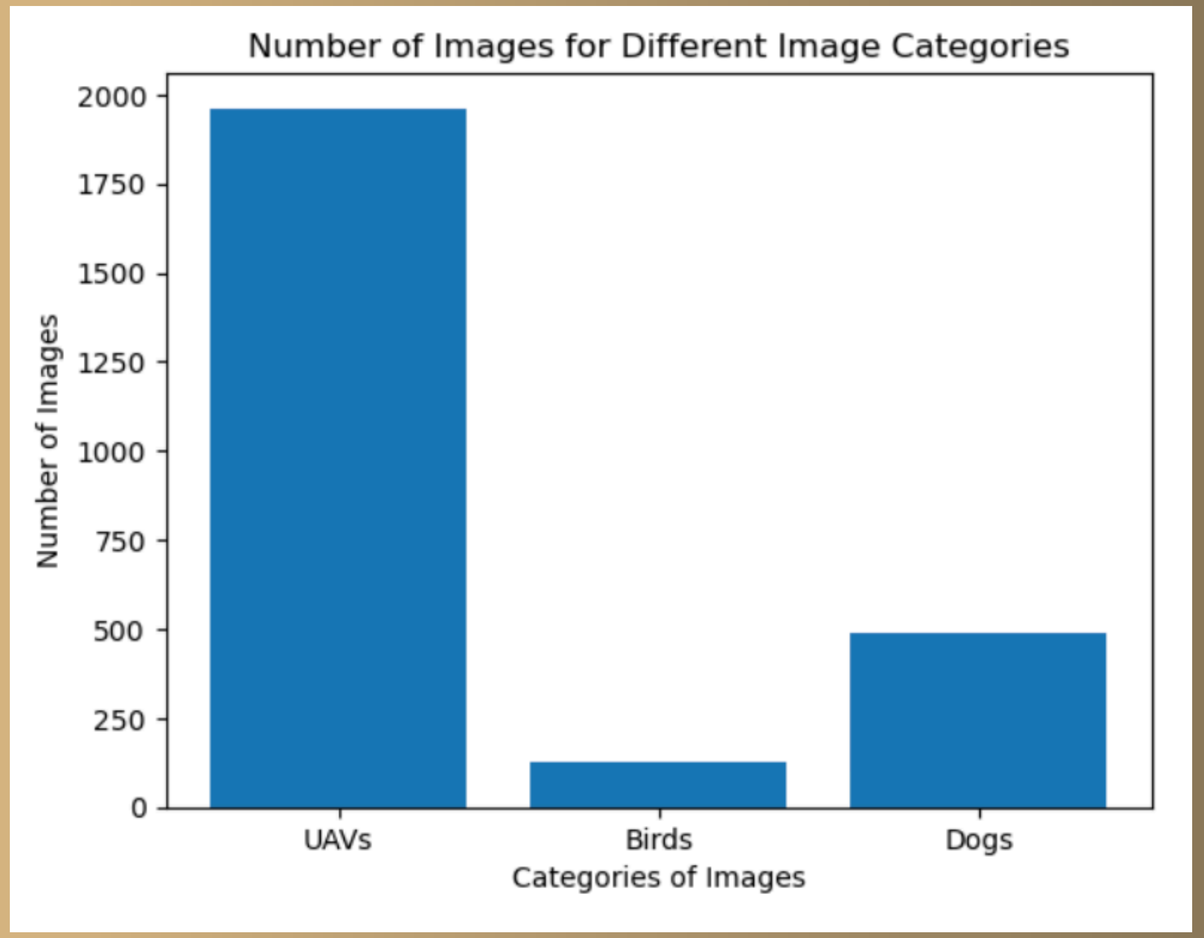
```
# Step 1: Data Preprocessing

# Processing and flattening the input matrices in preparation for model training

# Dividing the data up by categories from the custom dataframe.
uav_pixel_data = df['UAVs'].values
bird_pixel_data = df['Birds'].values
dog_pixel_data = df['Dogs'].values

# Flattening the matrix values from the datasets by category
uav_flattened_data = [pixel for image in uav_pixel_data for row in image for pixel in row]
bird_flattened_data = [pixel for image in bird_pixel_data for row in image for pixel in row]
dog_flattened_data = [pixel for image in dog_pixel_data for row in image for pixel in row]

# Converting the flattened data into feature matrices
uav_feature_matrix = np.array(uav_flattened_data)
bird_feature_matrix = np.array(bird_flattened_data)
dog_feature_matrix = np.array(dog_flattened_data)

# Applying PCA for dimensionality reduction
num_components = 3
pca = PCA(n_components=num_components)

uav_preprocessed_data = pca.fit_transform(uav_feature_matrix[:100])
bird_preprocessed_data = pca.fit_transform(bird_feature_matrix[:100])
dog_preprocessed_data = pca.fit_transform(dog_feature_matrix[:100])

preprocessed_data = np.vstack((uav_preprocessed_data,bird_preprocessed_data,dog_preprocessed_data))

```

University of Colorado **Boulder**

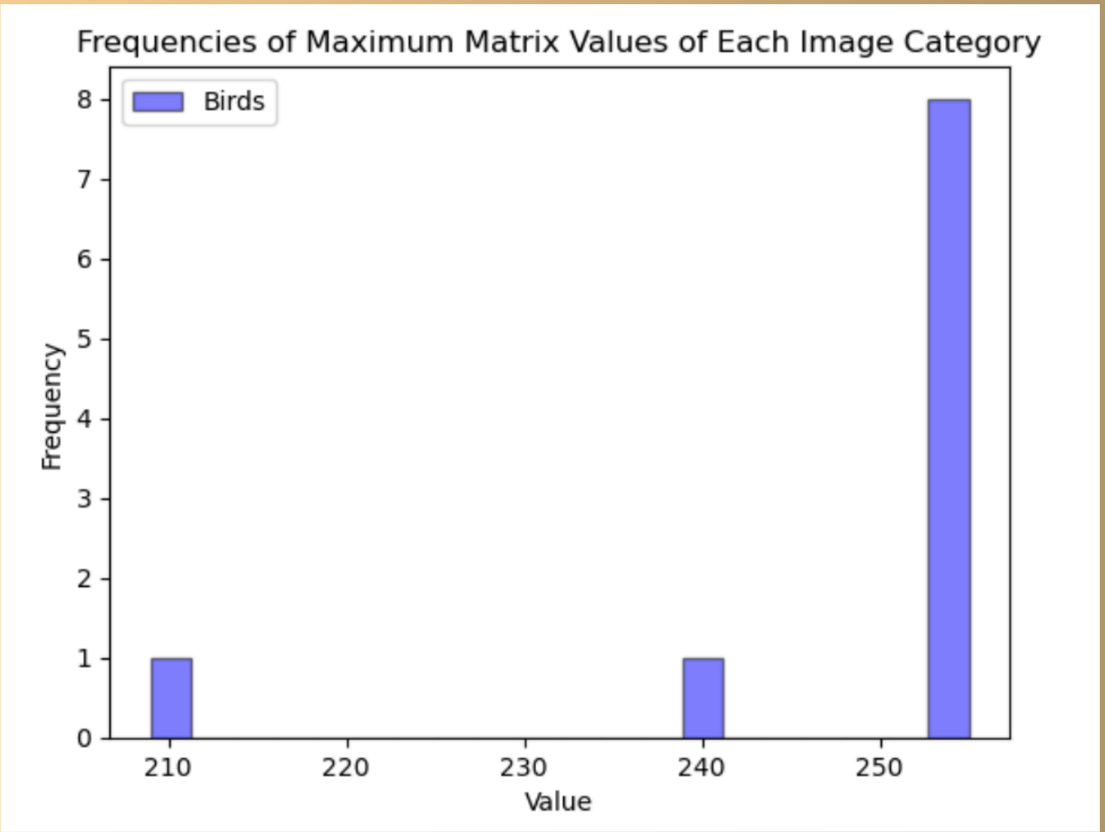
EXPLORATORY DATA ANALYSIS (EDA)

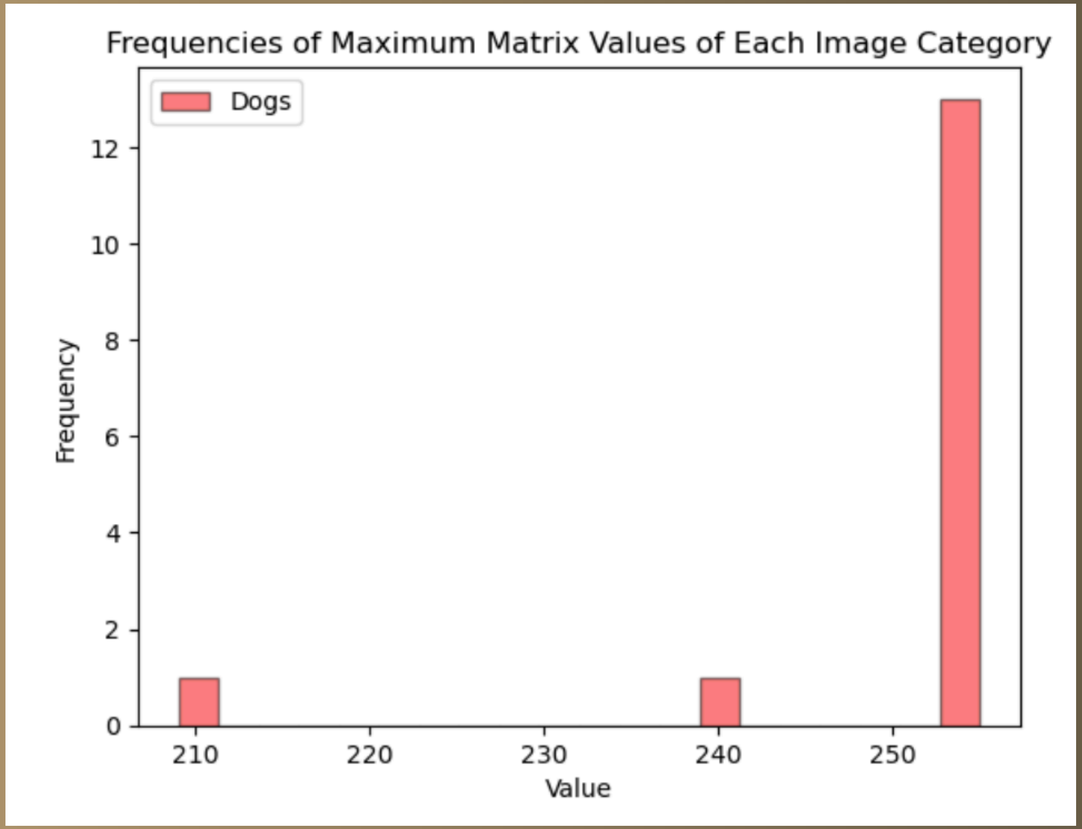
- Generated histograms to track the frequencies of the minimum values, maximum values, and average values of image matrices for each category of images.
- The histograms revealed enough differences between image data frequencies to confirm that there is a sufficient amount of project data to perform image classification.

EXPLORATORY DATA ANALYSIS

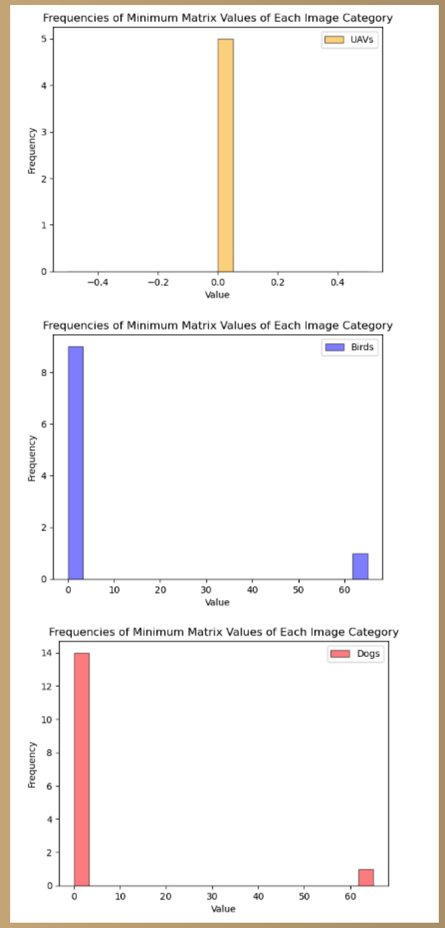


EXPLORATORY DATA ANALYSIS



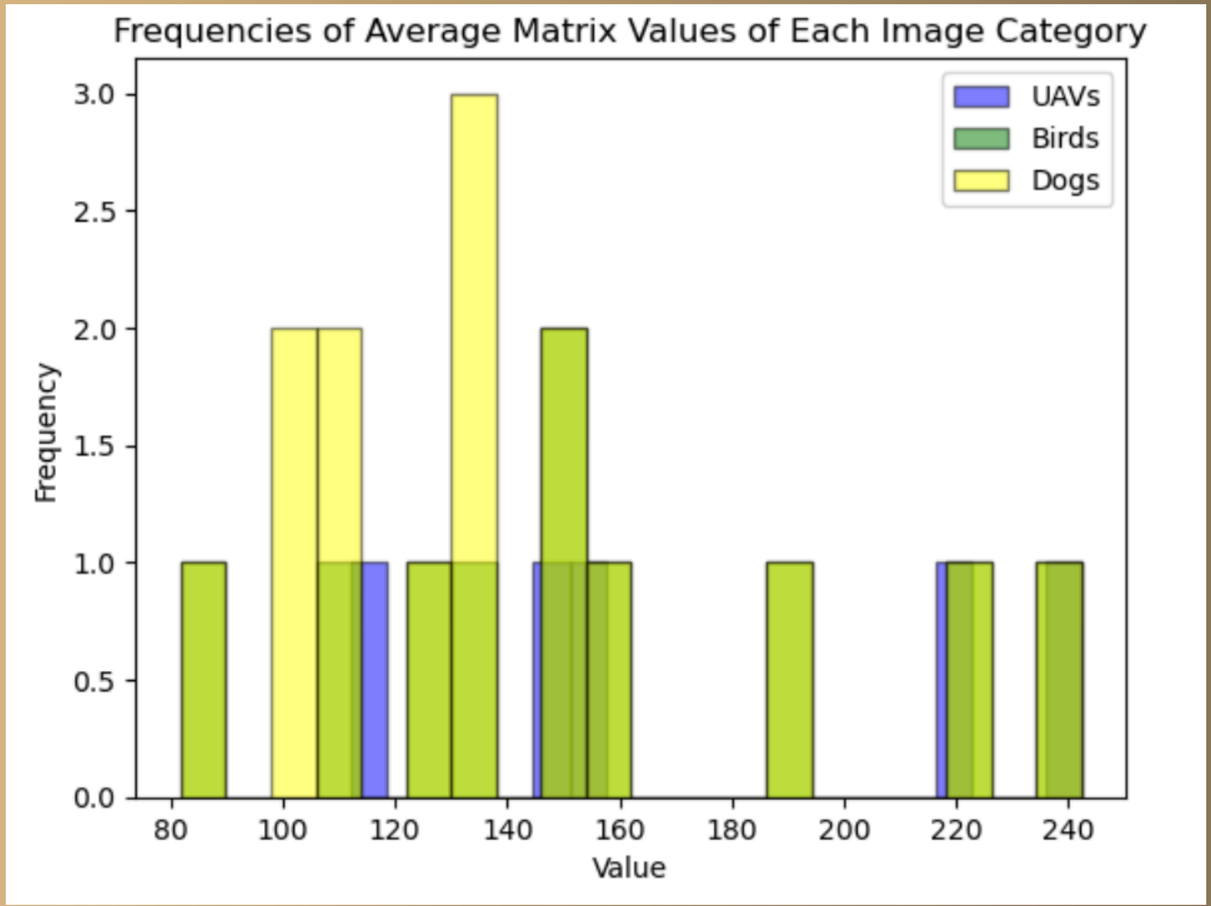


EXPLORATORY DATA ANALYSIS



University of Colorado **Boulder**

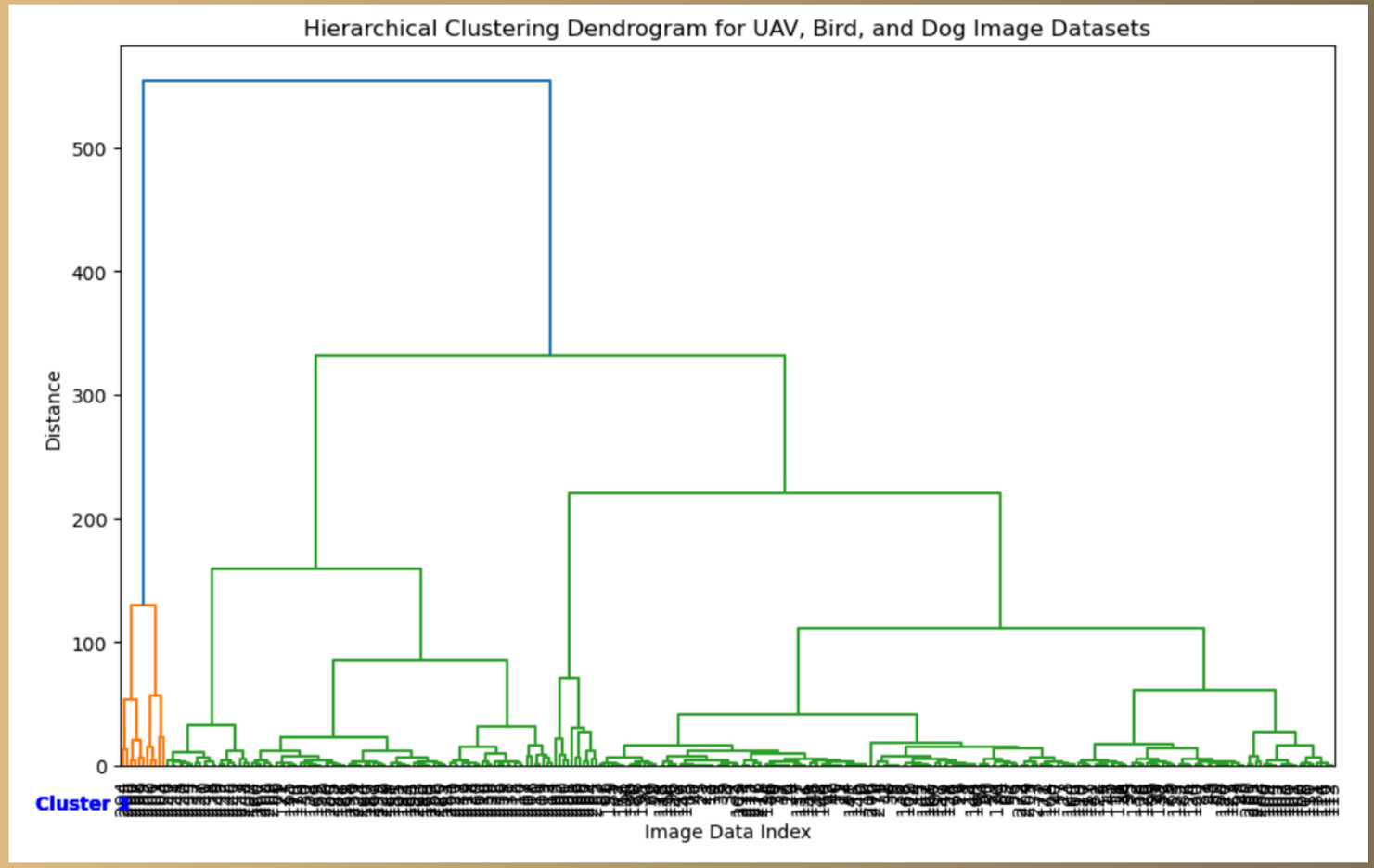
EXPLORATORY DATA ANALYSIS



MODEL APPROACH AND ANALYSIS

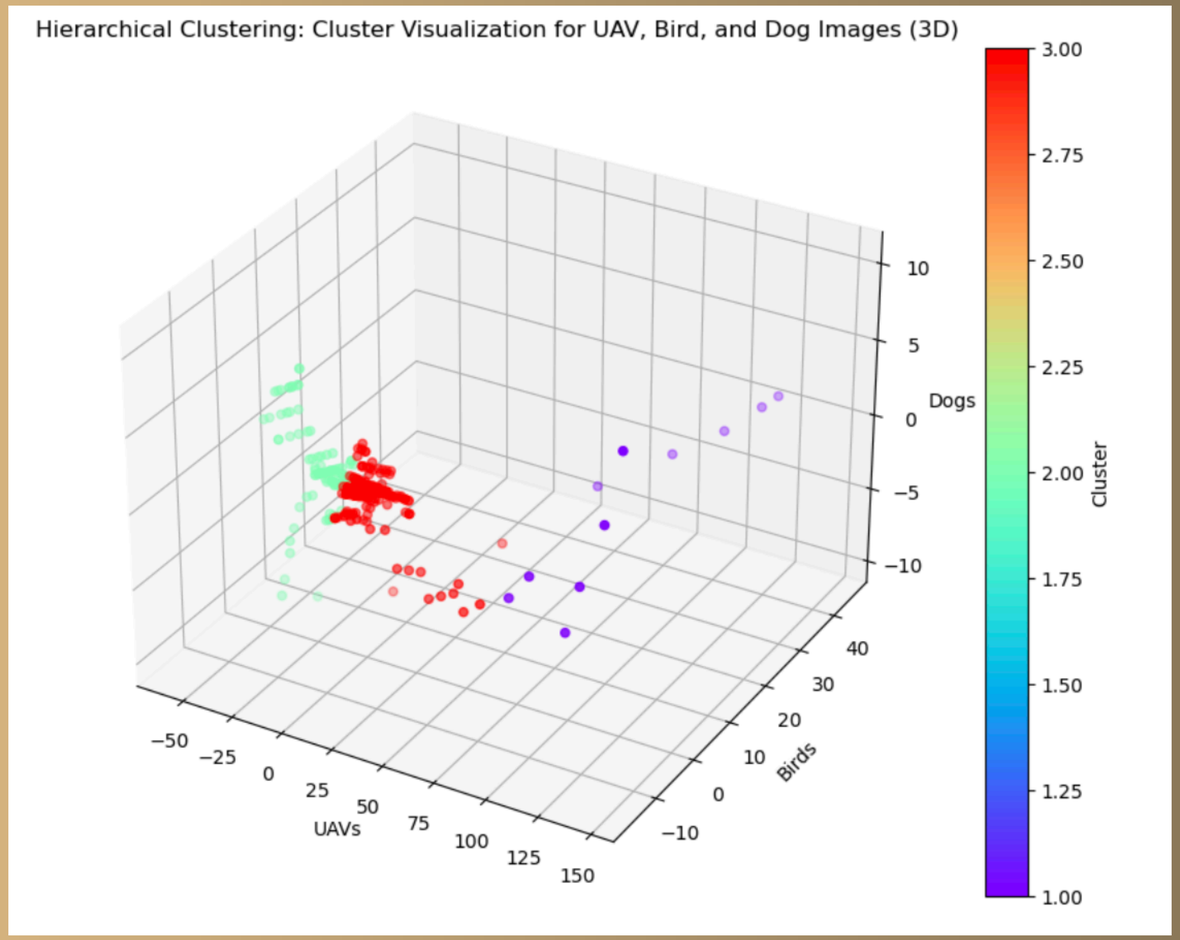
- First unsupervised model approach is a hierarchical clustering model, which creates a cluster for every given feature (image category) that is present.
- The second and third models are anomaly detection models with different base types (NMF and MLP) which are designed to treat UAV data as “anomalies” and group all other image types in a separate classification group.
- The final model is a supervised learning model approach (specifically, a support vector machine classifier) that also performs image classification.

HIERARCHICAL CLUSTERING MODEL



University of Colorado **Boulder**

HIERARCHICAL CLUSTERING MODEL



University of Colorado **Boulder**

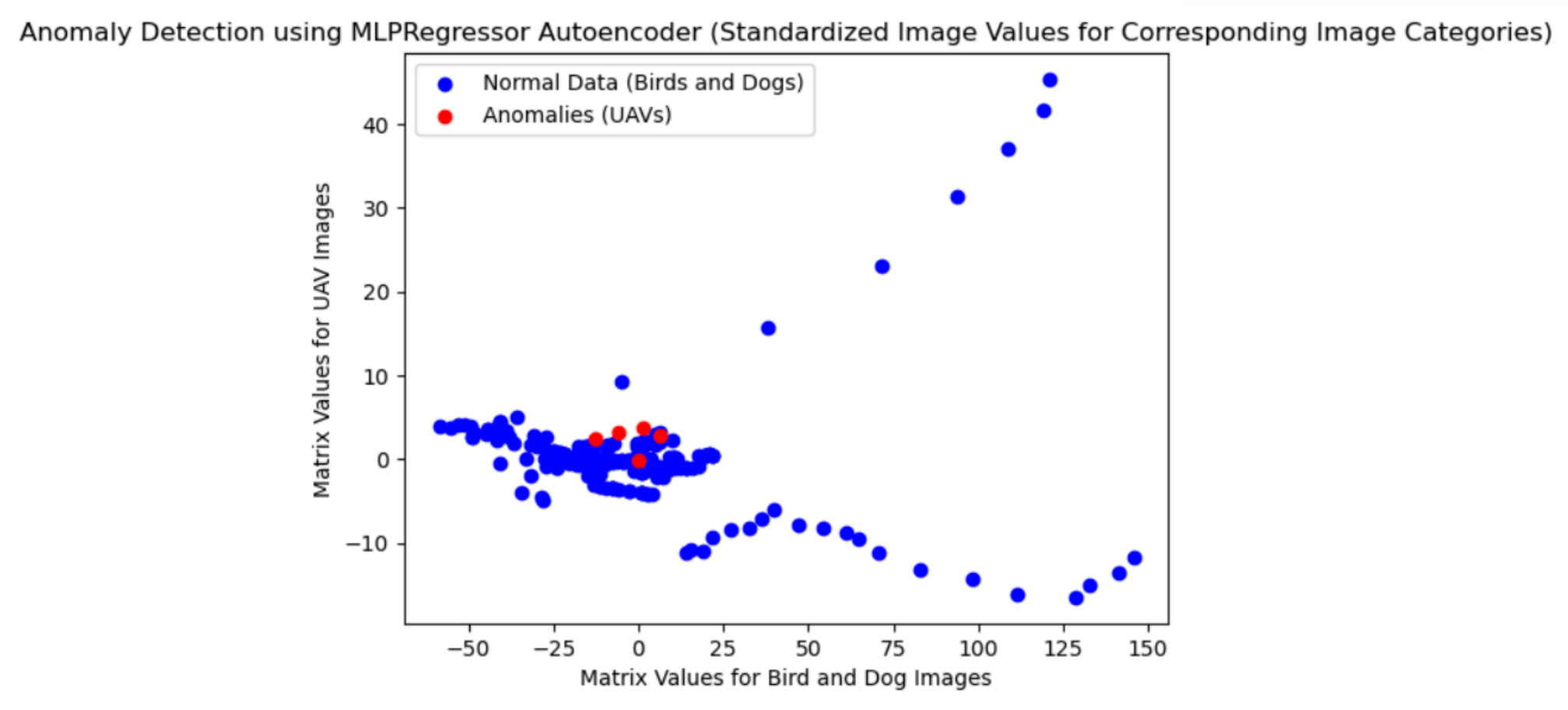
ANOMALY DETECTION MODEL - AUTO-ENCODER 01 (MLP)

```python
# Creating a second anomaly detection model, using MLPRegressor as a base to detect anomalies from
# the same dataset (treating UAV image datasets as the anomalies in this case).
from sklearn.neural_network import MLPRegressor

anomalies = uav_preprocessed_data

# Combining all other datasets (except for UAV data) for training purposes
normal_data = np.vstack((bird_preprocessed_data, dog_preprocessed_data))

# Standardizing the 'normal' data
scaler = StandardScaler()
normal_data_scaled = scaler.fit_transform(normal_data)

# Training an autoencoder on the 'normal' data
input_dim = normal_data_scaled.shape[1]
encoding_dim = 2

autoencoder = MLPRegressor(hidden_layer_sizes=[encoding_dim], activation='relu', solver='adam', max_iter=500)
autoencoder.fit(normal_data_scaled, normal_data_scaled)

# Standardizing the UAV dataset
mlp_anomalous_data_scaled = scaler.transform(anomalies)

# Retrieving the reconstruction errors for anomalous data
mlp_anomalous_data_reconstructed = autoencoder.predict(mlp_anomalous_data_scaled)
mlp_reconstruction_errors = np.mean(np.square(mlp_anomalous_data_scaled - mlp_anomalous_data_reconstructed), axis=1

# Setting a threshold for anomaly detection
threshold = np.percentile(reconstruction_errors, 95)

# Identifying anomalies in the anomalous dataset
detected_anomalies = anomalies[reconstruction_errors > threshold]

# Calculating the reconstruction errors for anomalous data
reconstruction_errors = np.mean(np.square(mlp_anomalous_data_scaled - mlp_anomalous_data_reconstructed), axis=1)

# Calculate RMSE
MLPRegress_rmse = np.sqrt(np.mean(np.square(reconstruction_errors)))

print("Detected Anomalies:")
print(detected_anomalies)
```

```
Detected Anomalies:
[[-5.72389977  3.1208072  -0.53715342]
 [ 1.17398842  3.76583042 -0.59268238]
 [ 6.47816864  2.84395983 -0.71848694]
 [ 4.7536966   2.68270403 -0.7046047 ]
 [ 3.02922455  2.52144822 -0.69072246]]
```

University of Colorado **Boulder**

ANOMALY DETECTION MODEL - AUTO-ENCODER 01 (MLP)



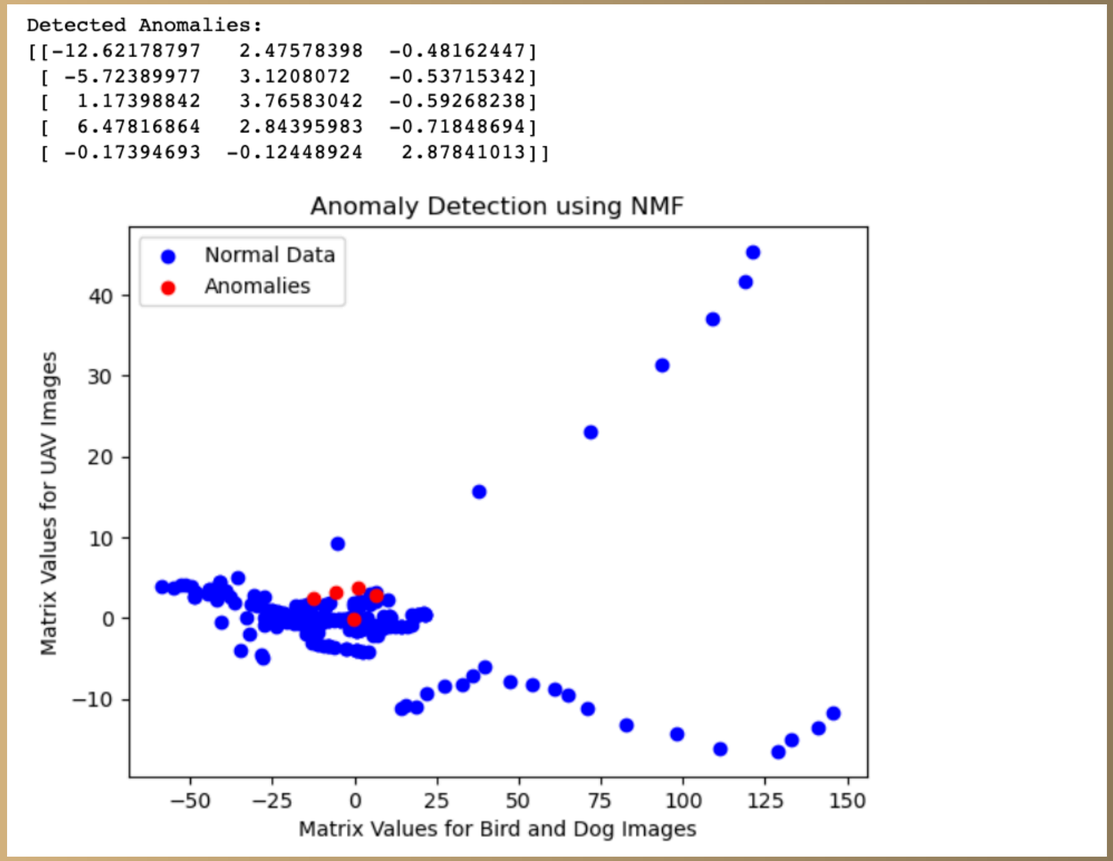
ANOMALY DETECTION MODEL - AUTO-ENCODER 02 (NMF)

```
# Training an NMF model with the 'normal' data
n_components = 2  # Number of components for NMF

nmf = NMF(n_components=n_components)
nmf.fit(normal_nmf_data_shifted)
```

University of Colorado **Boulder**

ANOMALY DETECTION MODEL - AUTO-ENCODER 02 (NMF)

```
Detected Anomalies:
[[-12.62178797   2.47578398  -0.48162447]
 [ -5.72389977   3.1208072   -0.53715342]
 [  1.17398842   3.76583042  -0.59268238]
 [  6.47816864   2.84395983  -0.71848694]
 [ -0.17394693  -0.12448924   2.87841013]]
```



SUPERVISED LEARNING MODEL - SVM CLASSIFIER

```python
# Creating a SVM classifier model and training it with the same dataset.
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Converting the list of lists of matrices into a flattened feature matrix
features = [matrix.flatten() for matrix in preprocessed_data[:150]]
feature_mat = np.array(features)
labels = preprocessed_data[150:300]

# Splitting the data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(feature_mat, labels, test_size=0.2, random_state=42)

y_train = [matrix.flatten() for matrix in y_train]
y_train = np.concatenate(y_train)
y_train = (y_train > 0.5).astype(int)

# Training an SVM classifier
svm_classifier = SVC(kernel='linear', random_state=5)  # Linear kernel for simplicity
svm_classifier.fit(X_train, y_train[:len(X_train)])

# Creating a binary representation for the true and predicted labels.
y_test = [matrix.flatten() for matrix in y_test]
y_test = np.concatenate(y_test)
y_test = (y_train > 0.5).astype(int)

y_pred = svm_classifier.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)


```

University of Colorado **Boulder**

EVALUATION METRICS

- This process involves three main comparisons to determine which is the best quality model for this use case. The first comparison is between the two anomaly detection models, based on their root mean squared error (RMSE) values.

- The second comparison involves qualitative analysis between the best performing anomaly detection model and the hierarchical clustering model. Since the hierarchical clustering model does not involve RMSE metrics, analysis will observe different metrics in the form of within-cluster sum of squares (WCSS) metrics for the hierarchical clustering model.

EVALUATION METRICS (HIERARCHICAL CLUSTERING)

```
# Computing the 'within-cluster sum of squares' values for the hierarchical clustering model
# over a range of increasing number of clusters to observe the model performance.
wcss_values = []

for num_clusters in range(1, 11):
    linkage_matrix = linkage(preprocessed_data, method='ward')
    cluster_assignments = fcluster(linkage_matrix, t=num_clusters, criterion='maxclust')

    cluster_centroids = []
    for cluster_id in np.unique(cluster_assignments):
        cluster_points = preprocessed_data[cluster_assignments == cluster_id]
        cluster_centroid = np.mean(cluster_points, axis=0)
        cluster_centroids.append(cluster_centroid)
    cluster_centroids = np.array(cluster_centroids)

    # Calculate pairwise distances between data points and their assigned cluster centroids
    distances = pairwise_distances(preprocessed_data, cluster_centroids, metric='euclidean')
    wcss = np.sum(np.min(distances, axis=1))  # Calculate WCSS
    wcss_values.append(wcss)

print('Within-cluster Sum of Squares Values for Hierarchical Clustering Model: ', wcss_values)

```

```
Within-cluster Sum of Squares Values for Hierarchical Clustering Model:  [5327.626686524979, 4247.639264672412, 3252.
270062461683, 2788.043597498734, 2285.323245154893, 2113.812307634568, 1726.6660516382951, 1501.0864706710938, 1426.5
55325684797, 1280.5615734779371]
```

University of Colorado **Boulder**

EVALUATION METRICS (ANOMALY DETECTION)

```
NMF RMSE Values:  [0.5799661121221118, 0.5799661121221118, 0.5799661121221118, 0.5799661121221118, 0.579966112122111
8, 0.5799661121221118, 0.5799661121221118, 0.5799661121221118, 0.5799661121221118, 0.5799661121221118]


MLPRegressor RMSE Values:  [0.15228291603093527, 0.18179312716755813, 0.10597459151450436, 0.08516477000726493, 0.114
8780268591905 6, 0.051718861525945445, 0.12246666051215643, 0.02257935688840208, 0.04156332181339631, 0.01449840943203
2688]
```

University of Colorado **Boulder**

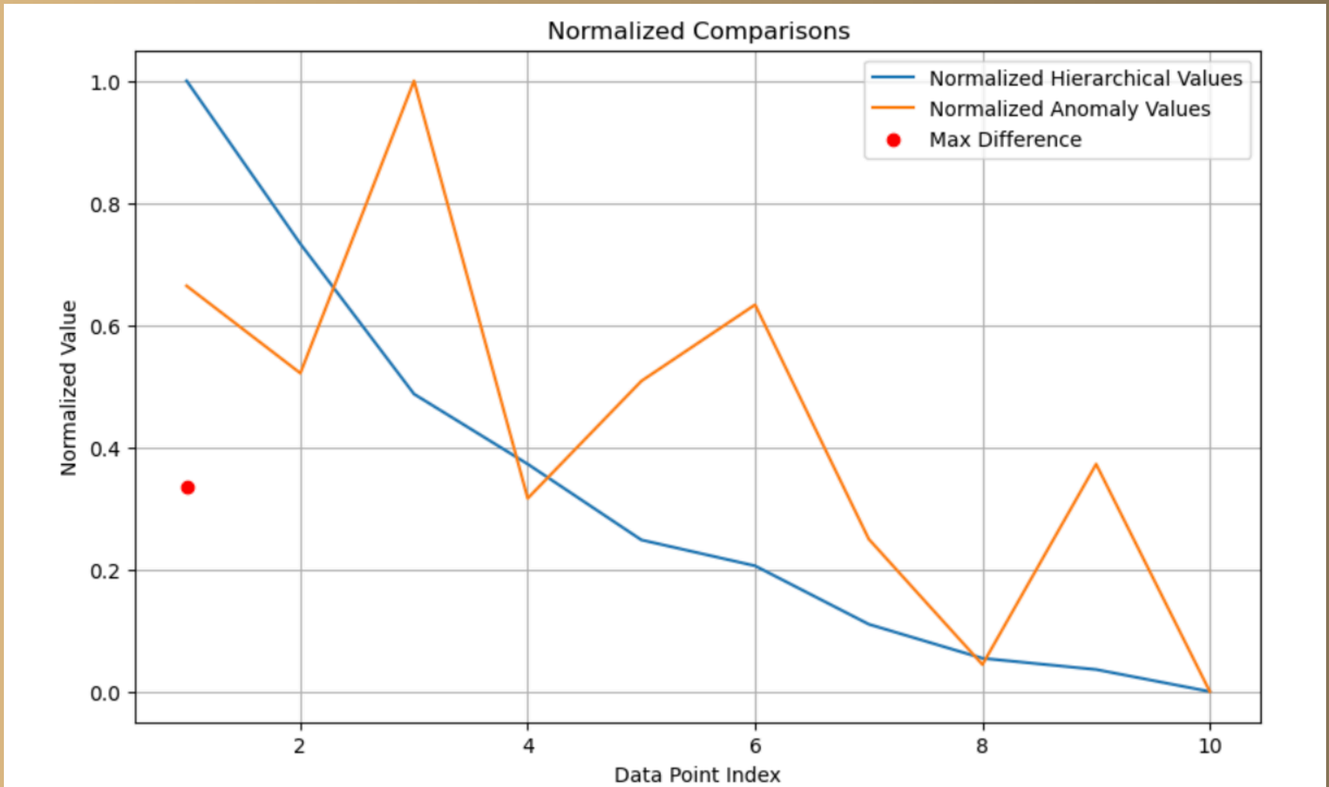
EVALUATION METRICS (SVM CLASSIFIER)

```
Accuracy: 0.7
Confusion Matrix:
[[21  0]
 [ 9  0]]
```

```
SVM Accuracies:  [0.6785714285714286, 0.5517241379310345, 0.6896551724137931, 0.6551724137931034, 0.6551724137931034,
0.7931034482758621, 0.8, 0.6333333333333333, 0.6666666666666666, 0.6333333333333333, 0.7]
```

University of Colorado **Boulder**

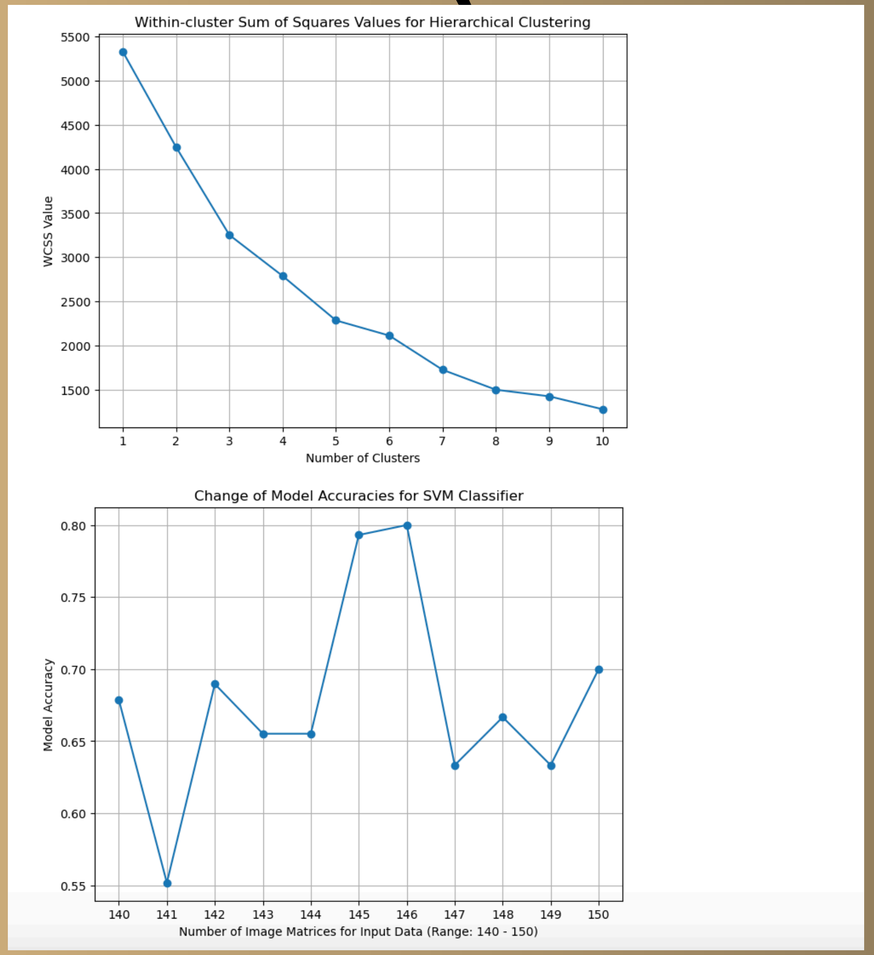
EVALUATION METRICS (FINAL COMPARISONS)




```
The list with the greatest difference is at index 0 with a difference of 0.335793451141265
```

University of Colorado **Boulder**

EVALUATION METRICS (FINAL COMPARISONS)



University of Colorado **Boulder**

RESULTS AND DISCUSSION

- Qualitative analysis was involved in the final comparisons of the models. Even though RMSE metrics could not be generated for the hierarchical clustering model, there was still enough information to receive a general perception on the overall behavior and performance of each model.

- The visualizations indicate that there are inconsistencies in the error metrics and accuracies when training the following models through multiple iterations: both anomaly detection models and the SVM classifier model. In regards to the hierarchical clustering model, this was the only model that showed consistency and gradual improvements across multiple iterations.

LESSONS LEARNED

- Data understanding and the model selection process were very crucial aspects of this project.
- Testing the same machine learning models with larger scales of data can determine robustness of the given models (but could also impact overall performance and runtime).
- Making use of different types of evaluation metrics

CONCLUSION

- In conclusion, the hierarchical clustering model was found to be the best model in terms of consistency and overall performance.
- Further tests and model optimization can improve the performance of all the other models involved.
- Incorporating other types of clustering models (such as K-means clustering) could yield different but better results in the search for best models to use for UAV image classification

THANK YOU

University of Colorado **Boulder**