DTSA 5510 - UNSUPERVISED ALGORITHMS IN MACHINE LEARNING (FINAL PROJECT)

This final project is for the following course: 'DTSA 5510 - Unsupervised Algorithms in Machine Learning', where the main goal is to complete the following objective(s):

1.) Identify an unsupervised learning problem to perform exploratory data analysis (EDA) and model analysis. This will also include evaluation metrics as well as a section for discussion and conclusions.

The report and final project contents can also be found at the following Github repository:
https://github.com/IsraelsLibrary/DTSA_5510_Unsupervised_Algorithms__In_Machine_Learning

PROBLEM STATEMENT

There has been past and ongoing research on image classification and detection of unmanned aerial vehicles (UAVs), particularly in a combat scenario or military environment. The goal of this project will be to apply unsupervised learning in the case of simple image classification of UAV images among other images within a given dataset.

For the model approach, I use clustering techniques such as hierarchical clustering and anomaly detection models to classify the different images. After further optimization, I will choose the best performing model and compare it with a supervised learning model approach to determine whether or not if an unsupervised learning approach performs better than a selected supervised learning model (within the scope of this project).

Three main datasets will be used for this project, in the form of UAV image datasets, bird image datasets, and dog image datasets. Principal Component Analysis (PCA) and further data transformations will be applied to reduce dimensionality and combine the data into one custom dataset in order to use it for training and model analysis. Sources for the datasets can be found in the following link(s):

1.) Image Datasets of UAVs: https://www.kaggle.com/datasets/nelyg8002000/uav-detection-dataset-images
2.) Image Datasets of Birds: https://www.kaggle.com/datasets/veeralakrishna/200-bird-species-with-11788-images
3.) Image Datasets of Dogs: https://www.kaggle.com/competitions/generative-dog-images/data

EXPLORATORY DATA ANALYSIS (EDA): PCA, DATA CLEANING, AND TRANSFORMATIONS

For the first stage of EDA, I read in the three main datasets that I use for analysis. The data processing includes data cleaning (consisting of two helper functions) to remove any corrupted images or null entries from the datasets. Here, the initial goal is to process the data and use it as a basis to form a new dataset that includes an equal number of images from each data source. This new dataset is used for training purposes for the unsupervised models and supervised model.

In [31]:
```
# Importing necessary libraries and software for EDA and data processing.
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import math
import itertools
from itertools import cycle
import re
from PIL import Image
import os
import base64
```

DATA CLEANING

In [32]:
```
# 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))
    ]
```

DATA PROCESSING AND TRANSFORMATIONS

In [33]:
```
# 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])
```

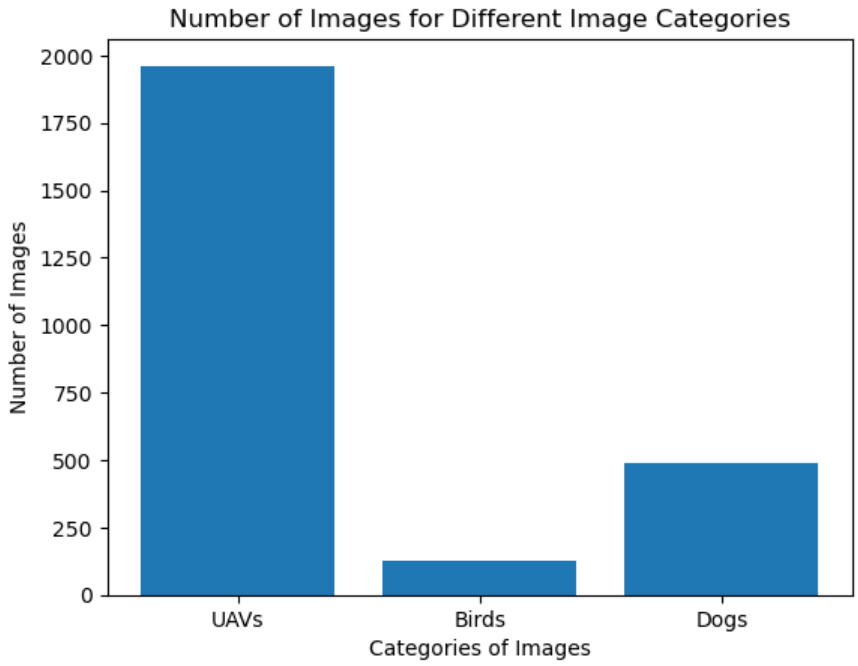
```
Found no corrupted images in the selected directory.
Found no corrupted images in the selected directory.
Found no corrupted images in the selected directory.
```

EXPLORATORY DATA ANALYSIS : DATA VISUALIZATION 01

For the first visualization of the EDA process, I am observing the number of images for each category of images (UAVs, birds, and dogs), finding out which category has the minimum number of images. The minimum number will serve as an ?equal factor? for the number of images used from each category.

In [34]:
```
# Generating a bar plot to track the total number of collected images for each category.
data_counts = {key: len(matrices) for key, matrices in collection.items()}

plt.bar(['UAVs', 'Birds', 'Dogs'], data_counts.values())

# Add labels and title
plt.xlabel('Categories of Images')
plt.ylabel('Number of Images')
plt.title('Number of Images for Different Image Categories')
```

Out[34]:
```
Text(0.5, 1.0, 'Number of Images for Different Image Categories')
```



EXPLORATORY DATA ANALYSIS : FURTHER DATA TRANSFORMATIONS

For the following series of visualizations, I observe the matrix properties of each category of images. I generate histograms to capture the frequency of maximum, minimum, and average values of each matrix from the datasets. Each set of histograms represent a different category of images. The goal for the histograms is to capture any differences of frequencies for each image categories, as confirmation that it possible to perform classification with the given dataset.

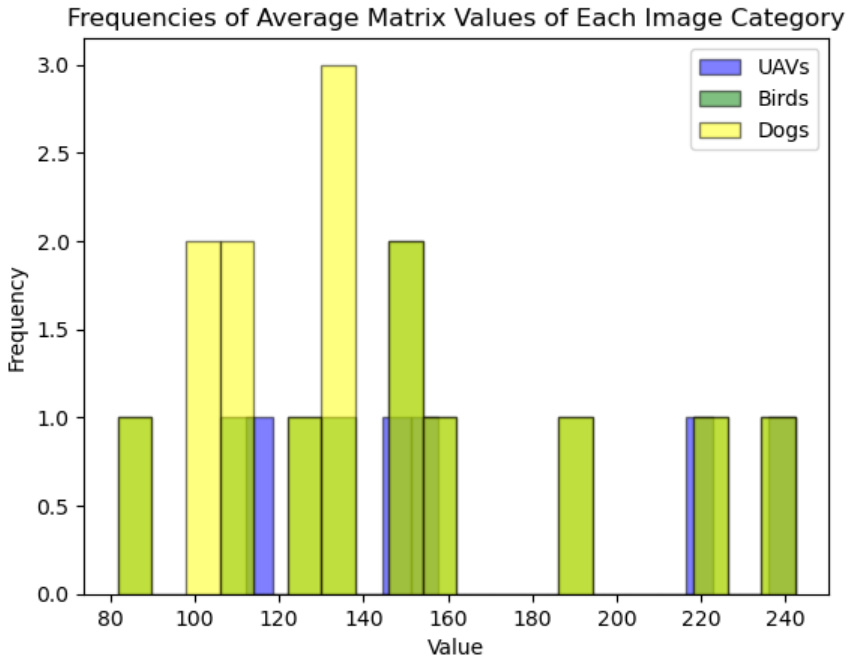
EXPLORATORY DATA ANALYSIS (DATA VISUALIZATIONS)

In [35]:
```
# 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)
```

In [36]:
```
# Capturing the maximum, minimum, and average matrix values of image data for each image category.
# This information is used for EDA visualizations to observe the differences in values for different
# image categories.

means = []
means_ = []
maxes = []
maxes_ = []
mins = []
mins_ = []
avg_data = {}
mins_data = {}
max_data = {}

for column in df.columns:
    for matrix in df[column]:
        means.append(np.mean(matrix, axis=1))
        maxes.append(np.max(matrix, axis=1))
        mins.append(np.min(matrix, axis=1))


    for mean in means:
        means_.append(mean.flatten())
    for i in range(len(means_)):
        means_[i] = np.mean(means_[i])


    avg_data[column] = means_
    means_ = []
```

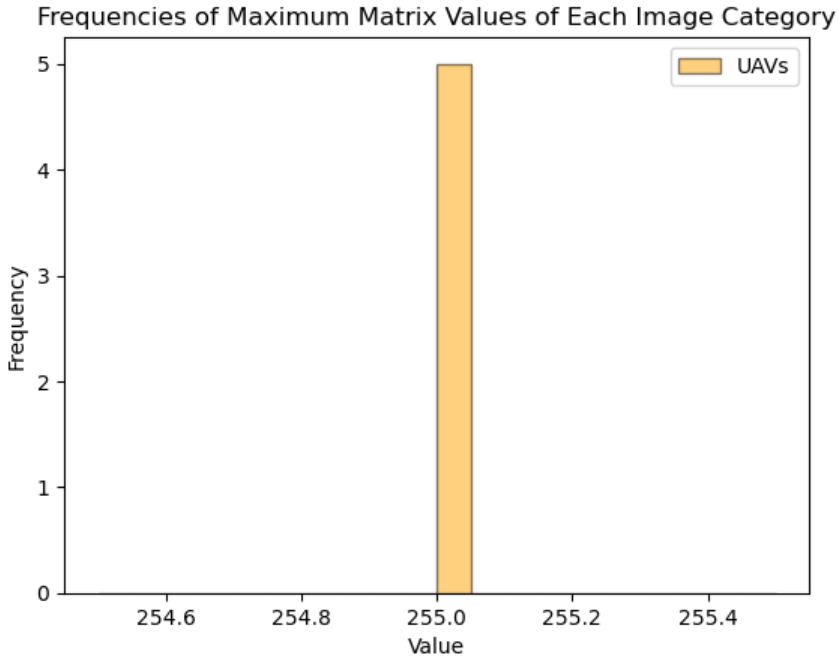
```
    for max_ in maxes:
        maxes_.append(max_.flatten())
    for j in range(len(maxes_)):
        maxes_[j] = np.max(maxes_[j])

    max_data[column] = maxes_
    maxes_ = []

    for min_ in mins:
        mins_.append(min_.flatten())
    for k in range(len(mins_)):
        mins_[k] = np.min(mins_[k])

    mins_data[column] = mins_
    mins_ = []
```

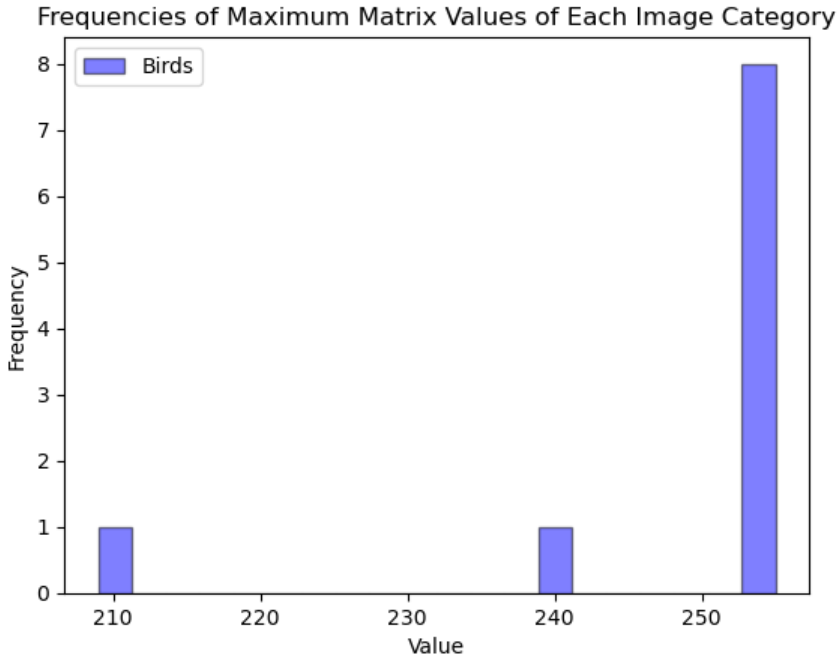
In [37]:
```
# Plotting the average matrix values for each category on a histogram. This plot will help to reveal
# any differences in average values, which will help confirm that classification with the given dataset
# is possible.

# Creating a figure and axis
colors = ['blue', 'green', 'yellow']
plt.figure()

# Plotting histograms in a loop with a different color corresponding to each image category
for ind, column in enumerate(list(avg_data.keys())):
    plt.hist(avg_data[column], bins=20, alpha=0.5, edgecolor="black", color=colors[ind], label=column)


# Adding labels and a legend
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Frequencies of Average Matrix Values of Each Image Category')

plt.legend()

# Generating the plot
plt.show()
```

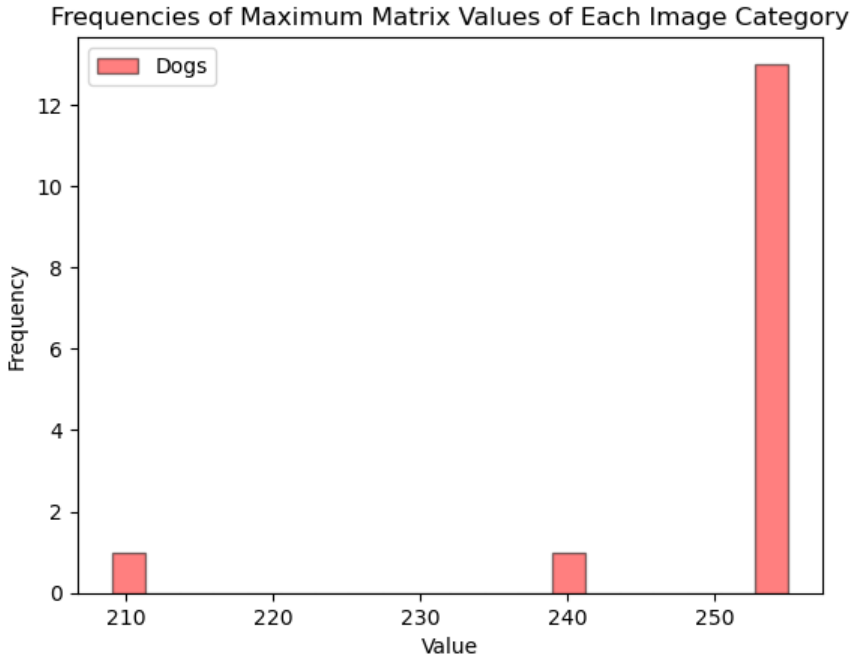


In [38]:
```
# The following code generates a series of histograms to reveal the frequency of maximum categories
# for each category


# Creating a figure and axis
colors = ['orange', 'blue', 'red']
plt.figure()

# Plotting histograms for each image category
for ind, column in enumerate(list(max_data.keys())):
    plt.hist(max_data[column], bins=20, alpha=0.5, edgecolor="black", color=colors[ind], label=column)
```

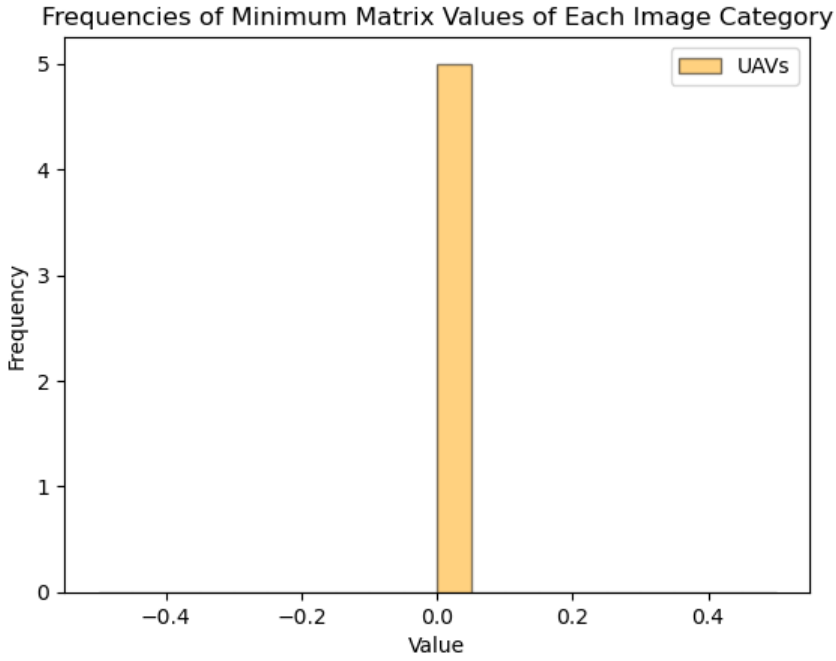
```
# Adding labels and a legend
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Frequencies of Maximum Matrix Values of Each Image Category')

plt.legend()

# Generating the plot
plt.show()
```

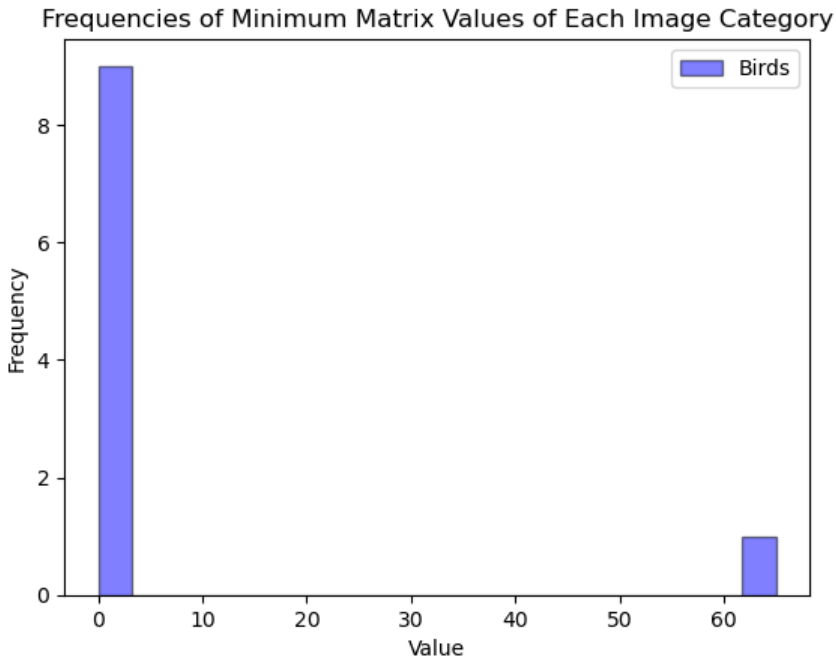


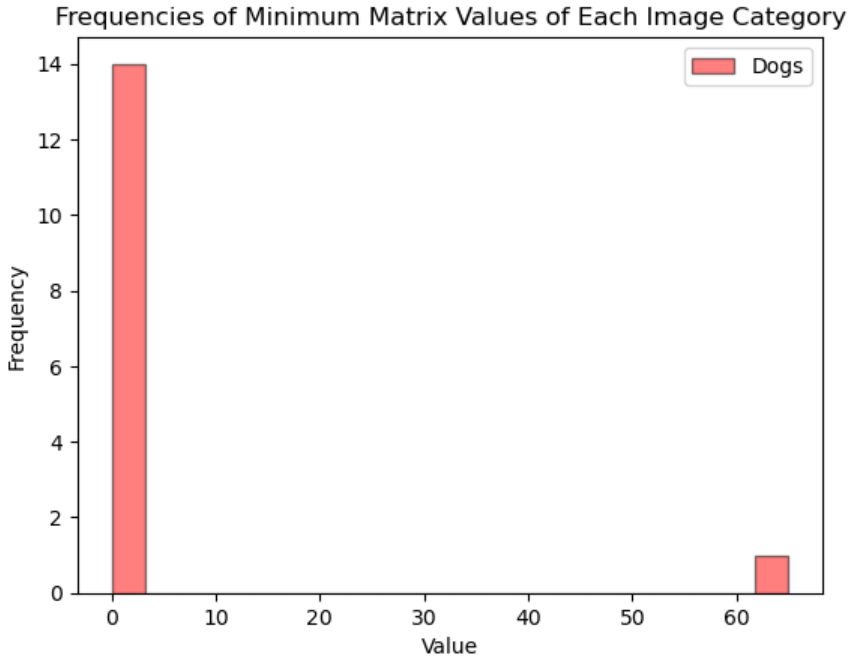




In [39]:
```
# The following code generates a series of histograms to reveal the frequency of minimum categories
# for each category

colors = ['orange', 'blue', 'red']
# Create a figure and axis
plt.figure()

# Plot histograms in a loop with random colors
for ind, column in enumerate(list(mins_data.keys())):
    plt.hist(mins_data[column], bins=20, alpha=0.5, edgecolor="black", color=colors[ind], label=column)


    # Add labels and a legend
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.title('Frequencies of Minimum Matrix Values of Each Image Category')

    plt.legend()

    # Show the plot
    plt.show()
```







EXPLORATORY DATA ANALYSIS (CONCLUSION)

From the histograms, it is noticeable that (in the case of maximum and minimum matrix values) there is not a significant difference and variety of frequencies between 'Birds' and 'Dogs' datasets. However, when comparing those frequencies with 'UAV' data (particularly the average of matrix values) there is a more noticeable difference. This indicates that it is possible to perform image classification with the given image data.

MODEL APPROACH AND ANALYSIS

For the model approach, analysis involves use of the following model types: a hierarchical clustering model, two autoencoders (non-negative matrix factorization and multi-layer perceptron regressor) and a support vector machine (SVM) classifier (for comparison purposes). At this stage of analysis, the goal is to achieve successful classification with all of the given models. Additional data processing is required for the input data in order to properly train the models.

In [40]:
```
# Importing machine learning libraries for further model generation and analysis
from sklearn import svm
from sklearn.decomposition import NMF,PCA
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.cluster import AgglomerativeClustering
from sklearn.impute import SimpleImputer
from sklearn.metrics import pairwise_distances
```

In [41]:
```
# Further 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))
```

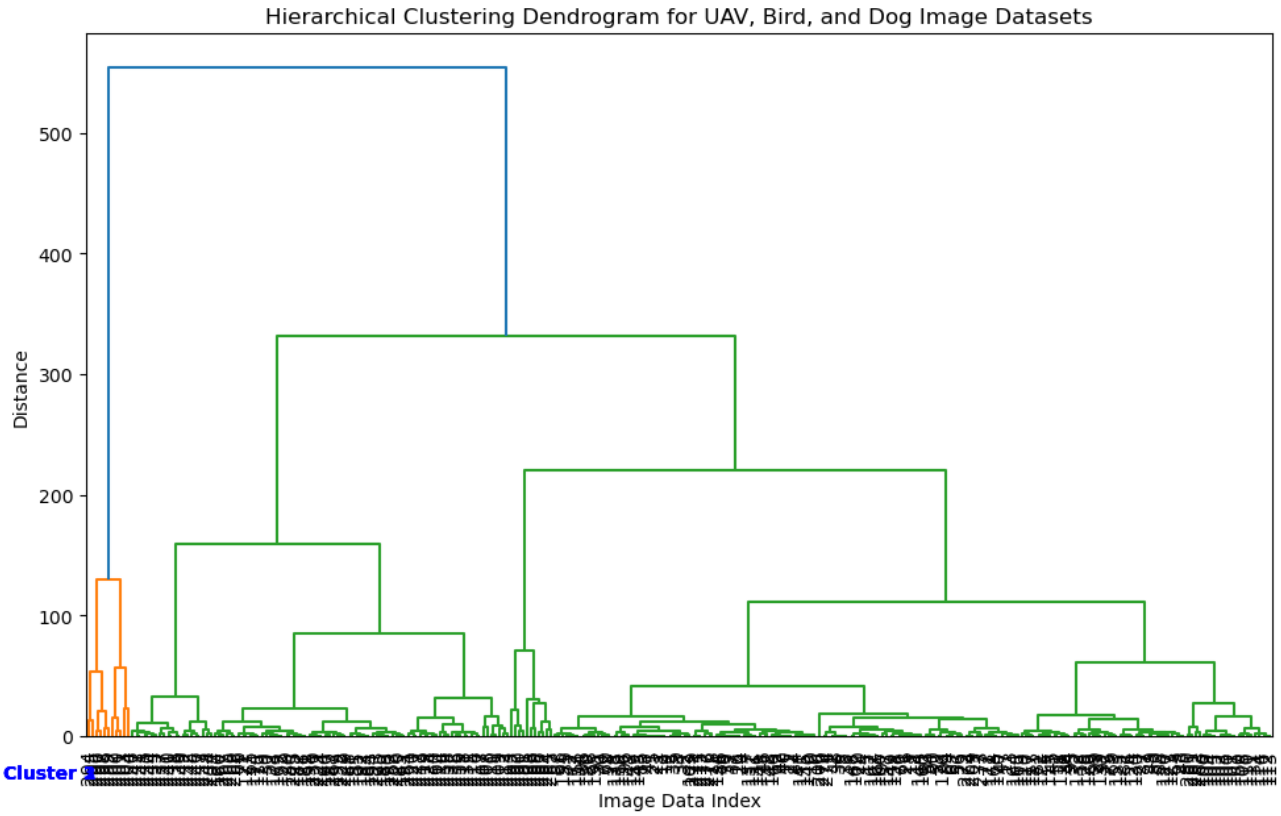
In [42]:
```
# Step 2: Handle Missing Values
# Replace NaN values with 0 using SimpleImputer
#imputer = SimpleImputer(strategy='constant', fill_value=0)
#preprocessed_data_imputed = pd.DataFrame(imputer.fit_transform(preprocessed_data))


# Filter the DataFrame to keep only columns with 'int' or 'float' data types
#filtered_data = preprocessed_data_imputed.select_dtypes(include=['int', 'float'])
```

HIERARCHICAL CLUSTERING MODEL

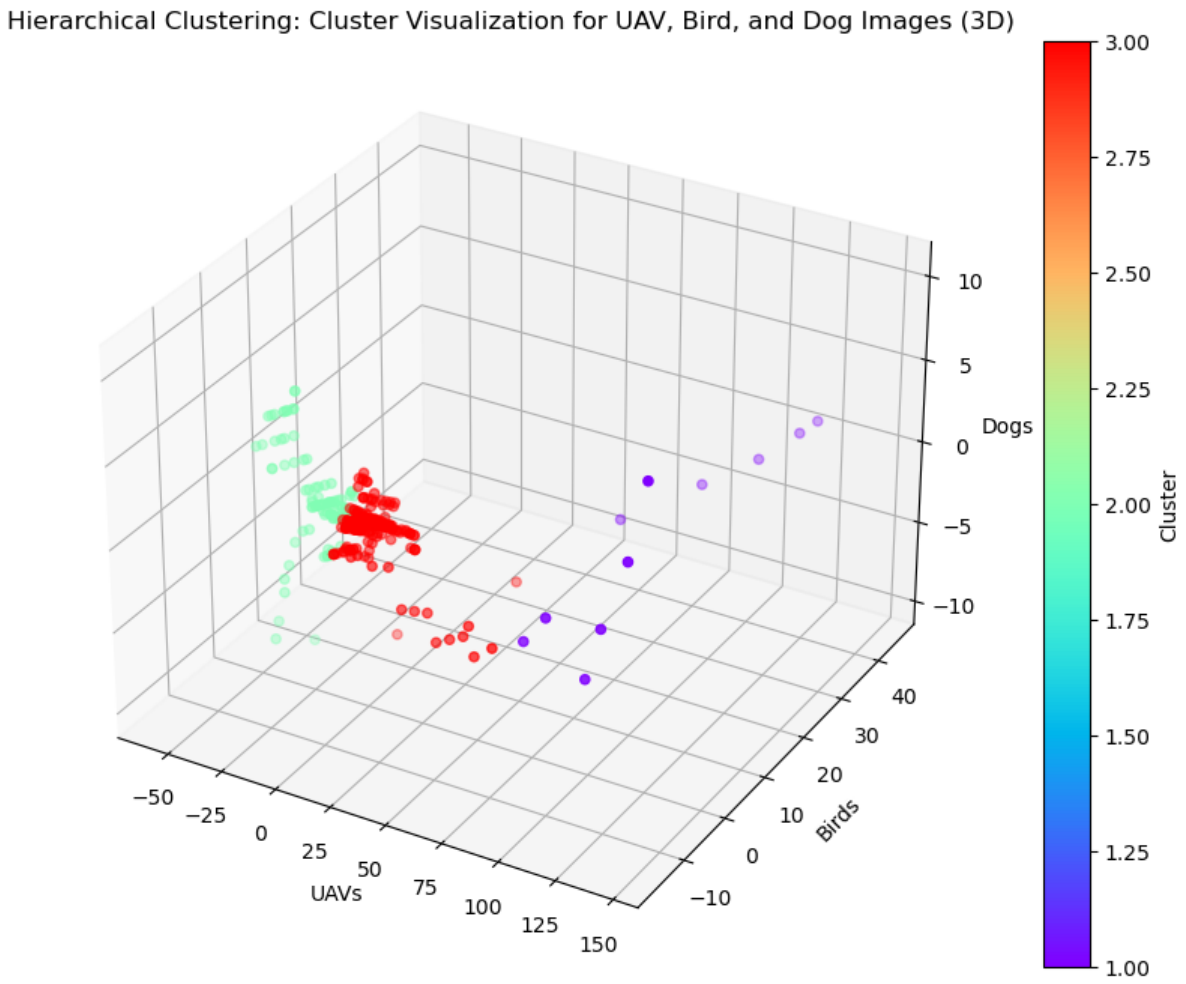
The first model presented here is the hierarchical clustering model. Classification was achieved, with the results showing an accurate number of 3 clusters, corresponding the the 3 different image categories.

In [43]:
```
# Generating a hierarchical clustering model and associated dendrogram to reveal clusters for each image categ
import numpy as np
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import fcluster

# Calculating the linkage matrix and generating the cluster assignments for the number of total clusters.
num_clusters = 3
linkage_matrix = linkage(preprocessed_data, method='ward')  # You can use other linkage methods as well
cluster_assignments = fcluster(linkage_matrix, t=num_clusters, criterion='maxclust')


# Plotting the dendrogram
plt.figure(figsize=(10, 6))
data_labels = list(df.columns)
dendrogram(linkage_matrix, leaf_font_size=10)

unique_clusters = np.unique(cluster_assignments)
annotations = [f'Cluster {cluster_id}' for cluster_id in unique_clusters]

for idx, cluster_label in enumerate(data_labels):
    plt.annotate(annotations[idx], (0, 0), textcoords="offset points", xytext=(-20, -20),
                 va='center', ha='center', color='blue', fontweight='bold')

# Adjust the x-axis labels for better readability
plt.xticks(rotation=90)
plt.tight_layout()

plt.title('Hierarchical Clustering Dendrogram for UAV, Bird, and Dog Image Datasets')
plt.xlabel('Image Data Index')
```

```
plt.ylabel('Distance')
plt.show()
```



In [44]:
```
# Generating a 3D scatterplot to show the behavior of clusters in respect to each cluster category.
from mpl_toolkits.mplot3d import Axes3D

# Specifying the distance threshold to obtain a certain number of clusters
threshold = 350

# Plotting the clusters with the image data point labels
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(preprocessed_data[:, 0], preprocessed_data[:, 1], preprocessed_data[:, 2], c=cluster_assi

# Setting labels for the axes
ax.set_xlabel('UAVs')
ax.set_ylabel('Birds')
ax.set_zlabel('Dogs')

# Adding a colorbar
colorbar = plt.colorbar(scatter)
colorbar.set_label('Cluster')
data_labels = list(df.columns)


plt.title('Hierarchical Clustering: Cluster Visualization for UAV, Bird, and Dog Images (3D)')
plt.show()
```



In [45]:
```
# Generating cluster assignments based on the number of clusters
num_clusters = 3  # You need to define the number of clusters
cluster_assignments = fcluster(linkage_matrix, t=num_clusters, criterion='maxclust')

# Calculating cluster centroids
cluster_centroids = [
    [sum(feature) / len(cluster_points) for feature in zip(*cluster_points)]
    for cluster_id in set(cluster_assignments)
    if (cluster_points := [preprocessed_data[i] for i, c in enumerate(cluster_assignments) if c == cluster_id]
]

# Calculating pairwise distances between data points and their assigned cluster centroids
distances = pairwise_distances(preprocessed_data, cluster_centroids, metric='euclidean')

# Calculating within-cluster sum of squares
wcss = np.sum(np.min(distances, axis=1))
```

ANOMALY DETECTION MODEL APPROACH (AUTO-ENCODERS)

For the second and third unsupervised learning models, both are set up as anomaly detection models. In this case, the models use non-negative matrix factorization (NMF) and multi-layer perceptron (MLP) regression. Before training the models, I treat the UAV data as the main "anomalies" and group the other datasets together as one separate dataset. As a result, the models achieve classification and are able to reveal the UAV data as anomalies.

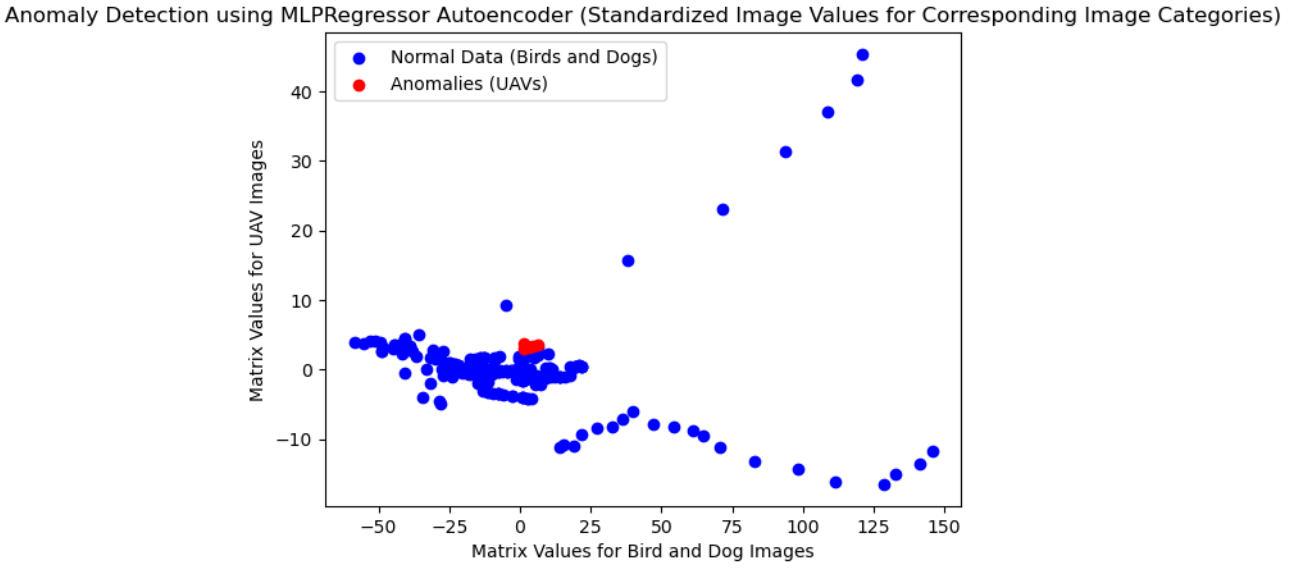
In [46]:
```
# Creating a second anomaly detection model, using MLPRegressor as a base to detect anomalies from
# the same dataset (treating UAV image datasets as the anomalies in this case).
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler

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), a

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

# Identifying anomalies in the anomalous dataset
detected_anomalies = anomalies[mlp_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=

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

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

```
Detected Anomalies:
[[ 1.17398842  3.76583042 -0.59268238]
 [ 1.25591263  2.99135514  0.58777366]
 [ 2.98038468  3.15261095  0.57389142]
 [ 6.42932878  3.47512256  0.54612695]
 [ 4.70485673  3.31386675  0.56000919]]
```

```
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
```

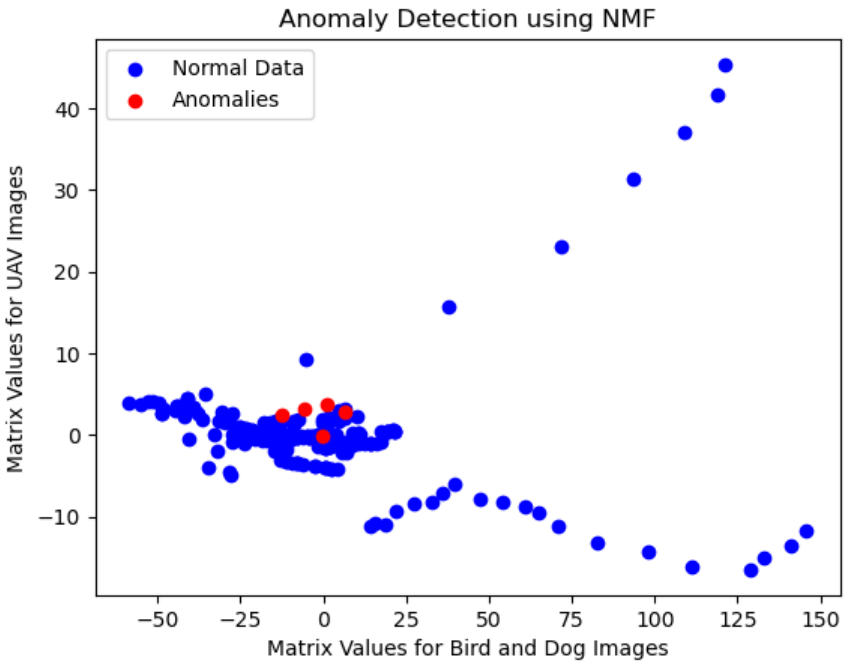
In [47]:
```
# Plotting the 'normal' data in blue and UAV 'anomalies' in red
plt.scatter(normal_data[:, 0], normal_data[:, 1], c='blue', label='Normal Data (Birds and Dogs)')
plt.scatter(detected_anomalies[:, 0], detected_anomalies[:, 1], c='red', label='Anomalies (UAVs)')
plt.xlabel('Matrix Values for Bird and Dog Images')
plt.ylabel('Matrix Values for UAV Images')
plt.legend()
plt.title('Anomaly Detection using MLPRegressor Autoencoder (Standardized Image Values for Corresponding Image
plt.show()
```



In [48]:
```
# Creating a third anomaly detection model, using Non-negative Matrix Factorization
# as a base to detect anomalies from the same dataset (treating UAV image datasets
# as the anomalies in this case).
```

```
# The code process is similar to the previous case with the MLPRegressor Autoencoder
# with some crucial changes.

from sklearn.decomposition import NMF

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

# Ensuring the data is non-negative
data_min = np.min(normal_data_scaled)
normal_nmf_data_shifted = normal_data_scaled - data_min

# 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)

# Transforming the anomalous (UAV) dataset, using NMF
nmf_anomalous_data_scaled = scaler.transform(anomalies)

# Ensuring the anomalous data is non-negative
data_min = np.min(nmf_anomalous_data_scaled)
anomalous_data_shifted = nmf_anomalous_data_scaled - data_min
anomalous_data_transformed = nmf.transform(anomalous_data_shifted)

# Reconstructing anomalous data using NMF
nmf_anomalous_data_reconstructed = np.dot(anomalous_data_transformed, nmf.components_)

# Calculating reconstruction errors for anomalous data
reconstruction_errors = np.mean(np.square(anomalous_data_shifted - nmf_anomalous_data_reconstructed), axis=1)

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

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

# Calculating reconstruction errors for anomalous data
reconstruction_errors = np.mean(np.square(anomalous_data_shifted - nmf_anomalous_data_reconstructed), axis=1)

# Calculating RMSE
NME_rmse = np.sqrt(np.mean(np.square(reconstruction_errors)))

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

# Plotting the 'normal' data in blue and UAV 'anomalies' in red
plt.scatter(normal_data[:, 0], normal_data[:, 1], c='blue', label='Normal Data')
plt.scatter(detected_anomalies[:, 0], detected_anomalies[:, 1], c='red', label='Anomalies')
plt.xlabel('Matrix Values for Bird and Dog Images')
plt.ylabel('Matrix Values for UAV Images')
plt.legend()
plt.title('Anomaly Detection using NMF')
plt.show()
```

```
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 APPROACH

As part of the supervised learning approach, the selected model is a SVM model. The goal in including this model as part of the project is to confirm whether or not if the best unsupervised learning model can perform better than the SVM model with the same dataset. For this stage, the SVM classifier achieves image classification with the given dataset.

In [49]:
```
# 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)
```

EVALUATION METRICS (SUPERVISED LEARNING APPROACH)

Initially, the SVM classifier was only achieving an accuracy of 50%. However, after increasing the size of the input data, the SVM classifier was able to achieve 70% accuracy. This model optimization is to prepare the SVM classifier for further evaluation

In [50]:
```
# Computing the accuracy, creating a confusion matrix, and
# plotting the ROC curve for the SVM classifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_recall_curve, auc, roc_curve, roc_auc_

# Calculating the accuracy
accuracy = accuracy_score(y_test[:len(y_pred)], y_pred)
print("Accuracy:", accuracy)
```

```
# Creating a confusion matrix
conf_matrix = confusion_matrix(y_test[:len(y_pred)], y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

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

EVALUATION METRICS (UNSUPERVISED LEARNING APPROACHES)

For the next step, evaluation metrics involve generating RMSE metrics for both anomaly detection models. I generate these root mean-squared error (RMSE) metrics after iterating the training process for both models after 10 cycles (with each iteration including an increase in the number of components used for both models). Further details and results are shown below.

Since we cannot generate RMSE metrics for the hierarchical clustering model, we will be looking at a different form of metrics: the within-cluster sum of squares (WCSS) metric. In general, this metric is used for evaluating the quality of clustering algorithms. If there is a low WCSS metric value, this indicates that the data points are tightly packed around their respective methods, indicating that the clusters are more well-defined. Like the anomaly detection models, the code generates its respective metrics through multiple iterations of the hierarchical clustering model, with each iteration including an increase in the number of clusters. This is mainly for observing how rapidly the WCSS values change and whether or not the clusters become more well-defined.

In [51]:
```
# Computing the RMSE values for the MLPRegressor and NMF autoencoders for different values for
# the number of components used.

nmf_rmse_values = []
mlp_rmse_values = []
for num_components in range(1, 11):
    nmf = NMF(n_components=num_components)
    mlp = MLPRegressor(hidden_layer_sizes=(num_components,), activation='relu', solver='adam', max_iter=500)

    nmf.fit(normal_nmf_data_shifted)
    mlp.fit(normal_data_scaled, normal_data_scaled)

    transformed_data = scaler.transform(anomalies)  # Transform data using NMF

    # Standardizing the anomalous dataset
    mlp_anomalous_data_scaled = scaler.transform(anomalies)
    # Get reconstruction errors for anomalous data
    mlp_anomalous_data_reconstructed = mlp.predict(mlp_anomalous_data_scaled)

    mlp_reconstruction_errors = np.mean(np.square(mlp_anomalous_data_scaled - mlp_anomalous_data_reconstructed
    nmf_reconstruction_errors = np.mean(np.square(nmf_anomalous_data_scaled - nmf_anomalous_data_reconstructed

    nmf_rmse = np.sqrt(np.mean(np.square(nmf_reconstruction_errors)))
    mlp_rmse = np.sqrt(np.mean(np.square(mlp_reconstruction_errors)))
    nmf_rmse_values.append(nmf_rmse)
    mlp_rmse_values.append(mlp_rmse)

print('NMF RMSE Values: ', nmf_rmse_values)
print('\n')
print('MLPRegressor RMSE Values: ', mlp_rmse_values)
```

```
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/decomposition/_nmf.py:1665: ConvergenceWar
ning: Maximum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/decomposition/_nmf.py:1665: ConvergenceWar
ning: Maximum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/decomposition/_nmf.py:1665: ConvergenceWar
ning: Maximum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
```

```
NMF RMSE Values:  [0.5799661121221118, 0.5799661121221118, 0.5799661121221118, 0.5799661121221118, 0.579966112
1221118, 0.5799661121221118, 0.5799661121221118, 0.5799661121221118, 0.5799661121221118, 0.5799661121221118]


MLPRegressor RMSE Values:  [0.1425245208640313, 0.24772215900467198, 0.06452324401534809, 0.02643524311850014
4, 0.04278232448570561, 0.0836313024367996, 0.037489226452916, 0.048207032959060675, 0.02851852091578941, 0.02
1212218015679275]
```

```
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/decomposition/_nmf.py:1665: ConvergenceWar
ning: Maximum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
/Users/israeljohnson/anaconda3/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't con
verged yet.
  warnings.warn(
```

In [52]:

```python
# 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))
    wcss_values.append(wcss)
```

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

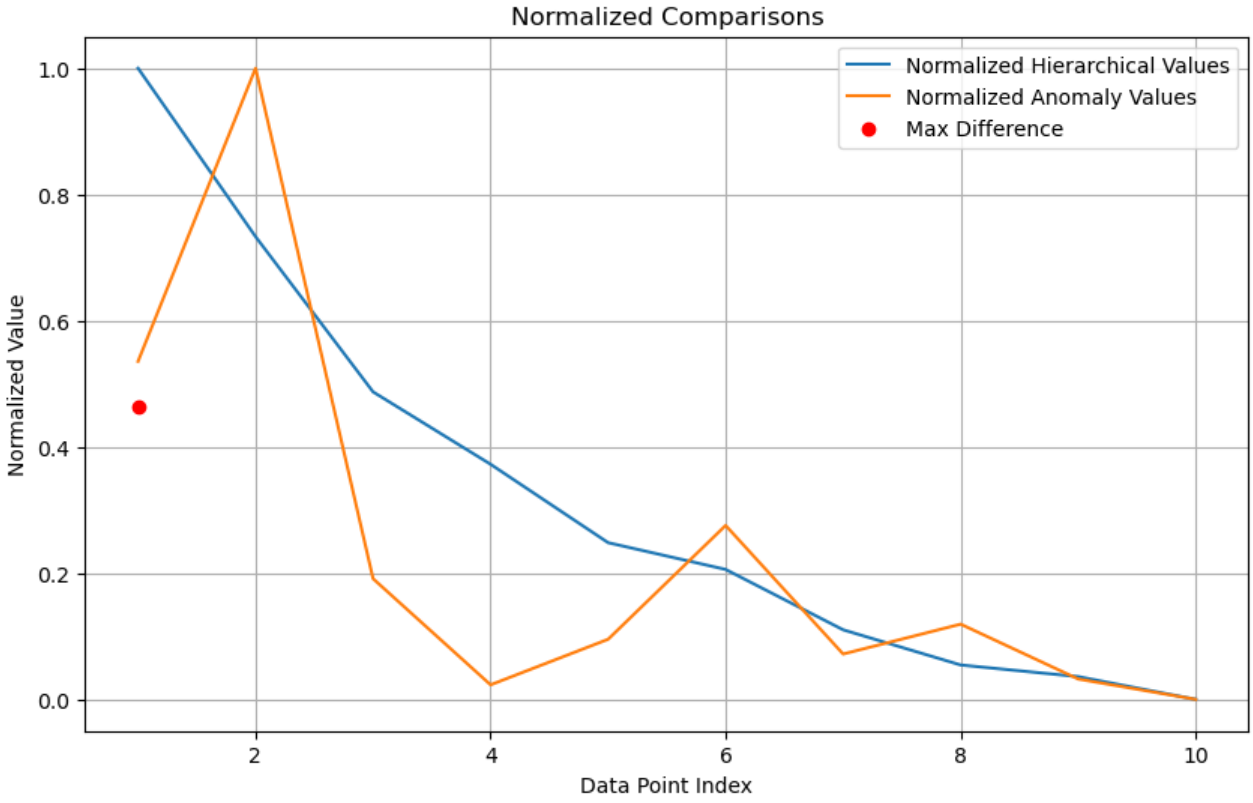
```
Within-cluster Sum of Squares Values for Hierarchical Clustering Model:  [5327.626686524979, 4247.63926467241
2, 3252.270062461683, 2788.043597498734, 2285.323245154893, 2113.812307634568, 1726.6660516382951, 1501.086470
6710938, 1426.555325684797, 1280.5615734779371]
```

EVALUATION METRICS (HYPER-TUNING AND FINAL COMPARISONS)

As the final stage of evaluation metrics, I compare the metrics of both anomaly detection models to determine which model performed better (in this case, it turned out to be the MLP model). Next, I compare the MLP metrics with the hierarchical clustering WCSS metrics. However, these are two different forms of metrics and we cannot do a direct comparison and will have to incorporate qualitative analysis. So, we will be observating the differences of metrics as the models complete their respective iterations. I normalize the metrics differences to track which model has the greatest differences in decreased metrics (the model with the greater difference will be determined as the model with better quality). In this case, the selected model with a greater difference in metrics is the hierarchical clustering model. Granted, WCSS metrics does not indicate model performance, but it does reveal that the model-related clusters become more and more well-defined over time.

With a third and final comparison, the results of the hierarchical clustering model is compared to the SVM classifier model. At this point, we know that the hierarchical clusters become more well-defined over time, so the main goal here is to observe the changes of SVM model accuracy over multiple iterations. As a result, the SVM model was found to have inconsistent accuracies over multiple iterations. Even though it does achieve high accuracies for certain iterations, the model does not maintain these accuracies for later iterations. In the end, the hierarchical clustering model is found to be the best option.

In [53]:
```
# Finding the best unsupervised learning model, based on model performance and error metrics.
anomaly_rmse = min(nmf_rmse_values, mlp_rmse_values)
hiearchical_wcss = wcss_values

# Normalizing the lists of both the hierarchical clustering model and the best anomaly detection model.
# This is to determine which model had the largest decrease in error metrics over time.
normalized_wcss = [(x - min(hiearchical_wcss)) / (max(hiearchical_wcss) - min(hiearchical_wcss)) for x in hiea
normalized_diff = [(x - min(anomaly_rmse)) / (max(anomaly_rmse) - min(anomaly_rmse)) for x in anomaly_rmse]

# Calculating the differences in normalized values
normalized_diffs = [normalized_wcss[i] - normalized_diff[i] for i in range(len(normalized_wcss))]

# Finding the index with the greatest difference
max_difference_index = normalized_diffs.index(max(normalized_diffs))


# Plotting
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(normalized_wcss) + 1), normalized_wcss, label='Normalized Hierarchical Values')
plt.plot(range(1, len(normalized_diff) + 1), normalized_diff, label='Normalized Anomaly Values')
plt.scatter(max_difference_index + 1, normalized_diffs[max_difference_index], color='red', label='Max Differen
plt.title('Normalized Comparisons')
plt.xlabel('Data Point Index')
plt.ylabel('Normalized Value')
plt.legend()
plt.grid(True)
plt.show()

print(f"The list with the greatest difference is at index {max_difference_index} with a difference of {normali
```



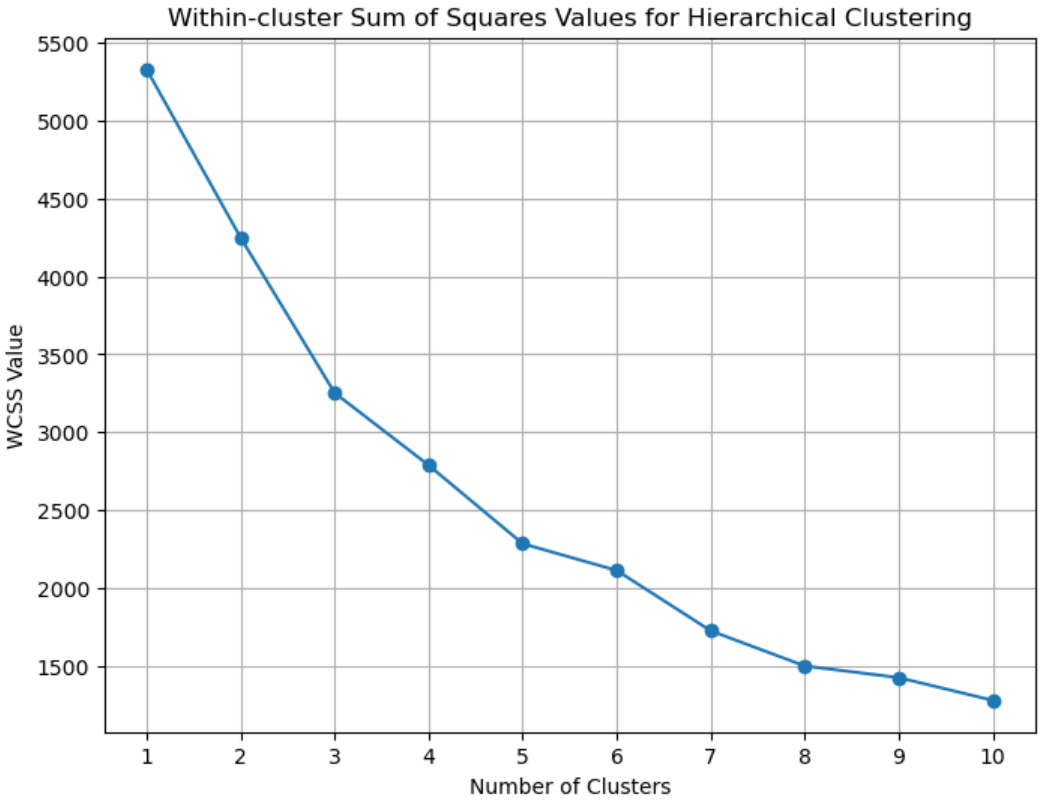

```
The list with the greatest difference is at index 0 with a difference of 0.4644283499493418
```

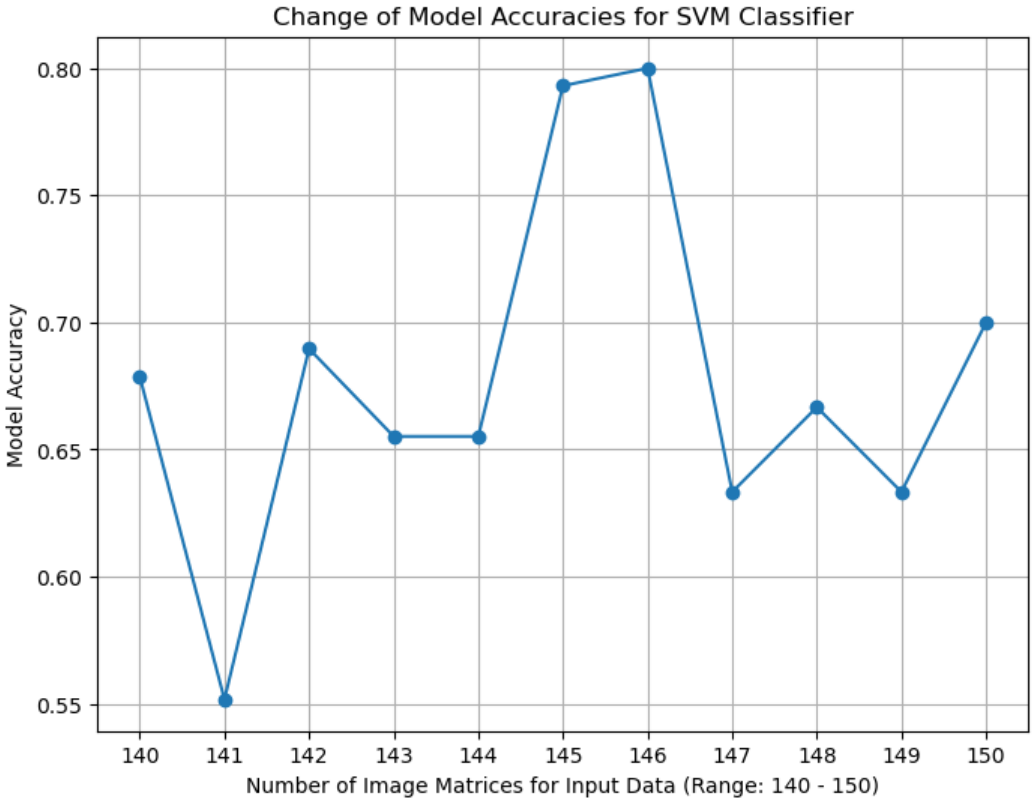
In [54]:
```
# Converting the list of lists of matrices into a flattened feature matrix

accs = []
for dimens in range(140, 151):
    features = [matrix.flatten() for matrix in preprocessed_data[:dimens]]
    feature_mat = np.array(features)
    temp = dimens * 2
    labels = preprocessed_data[dimens:temp]

    # 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)
    accuracy = accuracy_score(y_test[:len(y_pred)], y_pred)
    accs.append(accuracy)

print('SVM Accuracies: ', accs)
```

```
SVM Accuracies:  [0.6785714285714286, 0.5517241379310345, 0.6896551724137931, 0.6551724137931034, 0.6551724137
931034, 0.7931034482758621, 0.8, 0.6333333333333333, 0.6666666666666666, 0.6333333333333333, 0.7]
```

In [55]:
```
# Plotting
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(wcss_values) + 1), wcss_values, marker='o')
plt.title('Within-cluster Sum of Squares Values for Hierarchical Clustering')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS Value')
plt.xticks(range(1, len(wcss_values) + 1))
plt.grid(True)
```

```
# Plotting
plt.figure(figsize=(8, 6))
plt.plot(range(140, 151), accs, marker='o')
plt.title('Change of Model Accuracies for SVM Classifier')
plt.xlabel('Number of Image Matrices for Input Data (Range: 140 - 150)')
plt.ylabel('Model Accuracy')
plt.xticks(range(140, 151))
plt.grid(True)

plt.show()
```





DISCUSSION

Throughout the final project, there were many challenges including the following: the model selection process, dimensionality reduction, and working with different types of metrics for the evaluation stage. Initially, there were plans to implement a K-means clustering model as an additional model approach. However, many errors occurred in processing the input data into a proper format for the K-means model, so the input image data had to be processed differently in order to train the alternative hierarchical clustering model.

PCA proved to be a very crucial method for the data transformations process. There were some initial dimensionality issues, so PCA resolved this issue so that that input data can be used for training purposes for all the other given models.

Another challenge came during the evaluation metrics process. Since RMSE metrics could not be generated for all involved models, I had to rely on other forms of metrics as well as qualitative analysis. Even though I could not directly compare the performances of all models, I could observe how well the clusters were defined in cases where I could not generate RMSE metrics. As a result, I was still able to make a general determination on how effective certain models were when compared to others. With additional time, evaluation metrics is one of the major areas where I would investigate further and conduct further quantitative analysis.

CONCLUSION

In the end, I was able to show through evaluation metrics and qualitative analysis that the hierarchical clustering model was the best model for this project scope of UAV image classification. For future work on this project, other types of clustering models (including K-means clustering) would be added as options to test against the hierarchical clustering model. As another change, more data would be included, to see how well the clustering models maintain performance over time with larger volumes of data.

CITATIONS

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