University of Tehran

School of Electrical and Computer Engineering

Real or Not? Using Machine Learning to Distinguish Synthetic and Real Images

Machine Learning Course - Spring 2023

Authors:	*Student Numbers:*
Alireza Javid	810198375
Hesam Asadollahzadeh	810198346
Masoud Tahmasbi Fard	810198429
Amir Mahdi Ansaripour	810198358

Contents

1 Introduction

1.1 Problem Statement

Nowadays, generative models have become very widespread and intelligent, and visually differentiating between the images generated by them and reality has become increasingly difficult. Therefore, it is essential to use machine learning methods whose task is to classify fake and real classes. This would help to address the challenges posed by the proliferation of these sophisticated image-generating models.

Classifying images into different categories using machine learning algorithms is a common task in computer vision. In this particular project, the aim is to classify synthetically generated images from real ones into three categories: sea, jungle, and mountain. The images are generated using generative models such as DALL·E 2 [1], Stable Diffusion [2], and other similar models.

1.2 Challenges

1.2.1 Data Collection Challenges

The first challenge in this project is collecting high-quality data for both synthetic and real images. In the case of real images, it is difficult to find a large number of images that accurately represent the three categories. Images may be captured from different angles, under different lighting conditions, and have different resolutions, which can make it difficult for the model to classify them accurately. Moreover, the availability of relevant datasets for this specific task is limited.

In the case of synthetic images, the challenge is to generate images that accurately represent the three categories. Generative models such as DALL·E 2 and Stable Diffusion can generate high-quality images, but the images may not always accurately represent the desired category. For instance, a synthetic image generated using DALL·E 2 might depict a mountain that looks like a jungle or vice versa. Hence, collecting a diverse set of images representing the categories accurately is crucial.

In addition, the task of identifying precise prompts to serve as inputs for the text-to-image model presents a considerable level of difficulty. This is because the prompts need to capture the desired image features and details while also being concise and informative enough for the model to generate high-quality images. The selection of appropriate prompts may require a deep understanding of the image domain, which can be a time-consuming and complex process. Furthermore, inaccurate or inadequate prompts may result in the model generating low-quality or irrelevant images, making the task of prompt selection a critical aspect of the text-to-image generation process.

1.2.2 Problem-Solving Challenges

After collecting the data, the next challenge is to develop an accurate machine learning model that can classify the images into the three categories accurately. This is a challenging task because the generated images may not always accurately represent the

desired category, and the real images may have variations in lighting, angle, and resolution.

Another problem-solving challenge is choosing an appropriate machine learning algorithm for classification. Different algorithms such as Convolutional Neural Networks (CNNs), Support Vector Machines (SVMs) [3], and Random Forests (RFs) can be used for this task. However, selecting the best algorithm that can classify images accurately and efficiently can be difficult.

Furthermore, choosing the appropriate hyperparameters for the selected algorithm can also be a challenge. Hyperparameters are parameters that are set before training the model and can affect the performance of the model. Selecting the right set of hyperparameters for the algorithm is crucial to obtain an accurate model.

More specifically, classification models that aim to differentiate real from fake images must exhibit a high degree of robustness when handling variations in the input data. Essentially, the model's ability to detect real or fake images should not be influenced by extraneous factors such as noise or changes in spurious features like image style. Meeting this requirement can be exceedingly difficult, particularly since features such as image style can alter the RGB ratio of an image, potentially impairing the detection accuracy. Therefore, it is imperative to develop classification models that can withstand these types of variations, in order to guarantee consistent and precise detection outcomes.

Thus, extracting features for these models is a challenging task. For example, in images with a sea theme (whether real or fake), the dominant pixels are usually blue, while in jungle-themed images, they are mostly green. Therefore, when determining features, it is essential to note that only one component, such as color, cannot help us distinguish between real and fake images. Consequently, developing robust feature extraction techniques that can capture the diverse characteristics of different image themes is crucial to accurately detect artificially generated images.

1.3 Data Collection

The initial stage in undertaking any machine learning project is to gather suitable data for the problem at hand. Without a robust and comprehensive dataset, the effectiveness and accuracy of the machine learning model will be severely limited. Thus, it is crucial to allocate sufficient resources and effort into collecting relevant and diverse data that adequately captures the intricacies and nuances of the problem domain. Only then can the machine learning algorithm effectively analyze and learn from the data to generate valuable insights and predictions.

During the data collection phase, a combination of real and synthetic samples was used to create a diverse and representative dataset. Specifically, our team utilized the DreamStudio, Midjourney, DALL·E 2, and Stable Diffusion models to generate 120 synthetic images with labels of mountains, sea, and jungle, each of which incorporates various styles, including hyper-realistic, detailed surrealism, and realistic designs. To ensure a sufficient level of generalization, real images with different backgrounds and themes were also collected, bringing the total number of real and synthetic images to 240. The inclusion of a wide range of styles and backgrounds will enable the resulting

model to accurately classify both synthetic and real images, ultimately leading to more robust and reliable predictions.

1.4 Background Research and Promising Strategies

1.4.1 Histogram of Oriented Gradients (HOG)

The Histogram of Oriented Gradients (HOG) [4] is a feature extraction method for images that describes the distribution of edge orientations in an image. The HOG method works by first dividing the image into small cells, and then calculating the gradient (i.e., the direction and magnitude of change) within each cell. The gradient is then used to compute a histogram of edge orientations within each cell.

To further improve the discriminative power of the HOG features, neighboring cells are combined into larger blocks, and the histograms within each block are normalized. This normalization helps to reduce the effects of variations in lighting and contrast, which can cause the same image to appear differently in different conditions.

The resulting HOG feature vector represents the distribution of edge orientations within the image and can be used for various computer vision tasks. The HOG feature method has been widely used in computer vision due to its simplicity, effectiveness, and robustness.

1.4.2 Scale-Invariant Feature Transform (SIFT)

Scale-Invariant Feature Transform (SIFT) [5] is a feature detection and description method used in computer vision to extract distinctive features from images that are invariant to scale, rotation, and illumination changes.

The SIFT algorithm works by first identifying key points in an image using a difference of Gaussian function, which is convolved with the image at multiple scales. Next, the algorithm computes local image gradients and orientation histograms around each key point, which are used to generate a descriptor vector that describes the key point's appearance. The descriptor is invariant to scale, orientation, and affine distortion, which makes it useful for object recognition and image-matching applications. Finally, the descriptors are normalized and thresholded to eliminate noise and improve the robustness of the algorithm.

SIFT has been widely used in a variety of computer vision applications, including object recognition, image stitching, and video tracking. However, it has been largely surpassed by more recent methods such as SURF and ORB, which are faster and more efficient.

1.4.3 Speeded Up Robust Features (SURF)

Speeded Up Robust Features (SURF) [6] is a popular feature extraction method used in computer vision to detect and describe interest points in images. The key idea behind SURF is to identify distinctive local features that are invariant to image scaling, rotation,

and lighting conditions. SURF achieves this by computing a set of scale and rotation invariant features known as *descriptors*.

SURF features are based on the concept of scale-invariant feature transform (SIFT) and are designed to be robust to scale, rotation, and changes in illumination. The method involves detecting interest points (keypoints) in an image and extracting feature descriptors around each keypoint. The keypoint detection process involves identifying areas of the image with high local contrast and high curvature. This is achieved by convolving the image with a set of Gaussian filters at different scales and detecting extrema in the difference of Gaussian (DoG) responses.

Once keypoints are detected, SURF generates feature descriptors by computing a Haar wavelet response in x and y directions at different scales and orientations around each keypoint. The resulting feature vectors are 64-dimensional and are designed to be highly distinctive and robust to noise and geometric transformations.

SURF is known for its efficiency and speed, making it a popular method for real-time applications such as object recognition, tracking, and 3D reconstruction.

1.4.4 Fake Images Discriminator (FID)

The article "Detection of GAN-Synthesized Image Based on Discrete Wavelet Transform [7]" proposes a novel approach to identifying synthesized images using a combination of mathematical techniques. The authors suggest plotting the distribution of each RGB color of an image to obtain the random variable FNCC, which is a measure of the correlation of component distributions. As FNCC itself is a random variable, it has several key statistical properties such as mean, standard deviation, skewness, and kurtosis. The authors note that the kurtosis of FNCC shows a discernable difference between synthetic and real images, with synthetic images tending to have a higher kurtosis. This finding makes it a promising feature to use for classification purposes, highlighting the importance of utilizing advanced mathematical tools to improve the accuracy of image classification models.

2 Feature Extraction

First of all, we extract simple and naive statistical features of the images. We define a function called ‘extract_RGB_features‘ that takes the path to a directory containing image files as input. It aims to extract RGB features from the images present in the specified directory. The function uses the OpenCV library to read each image file, calculates various statistics such as the mean and standard deviation of the RGB channels, and computes percentiles of the pixel values. These statistics are then collected into a list called ‘RGB_features‘ for each image that meets certain criteria. The progress of the feature extraction is displayed using print statements.

RGB features are important in image processing and computer vision tasks because they provide valuable information about the color composition and distribution within an image. By calculating the mean and standard deviation of the RGB channels, we can gain insights into the average color intensity and the variation in color values, respectively. This information can be useful for tasks such as image classification, where different objects or scenes may have distinct color characteristics.

Additionally, by computing percentiles of the pixel values, the code captures information about the distribution of colors across the image. This can help in detecting patterns or identifying specific regions with unique color properties. Percentiles provide a way to understand the distribution of colors beyond just the mean and standard deviation. For instance, they can reveal if certain color values dominate the image or if there is a wide range of colors present.

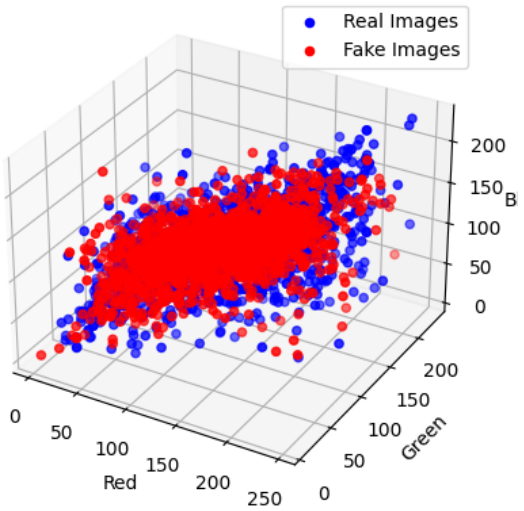


Figure 1: Distribution of Mean Values of RGB Channels

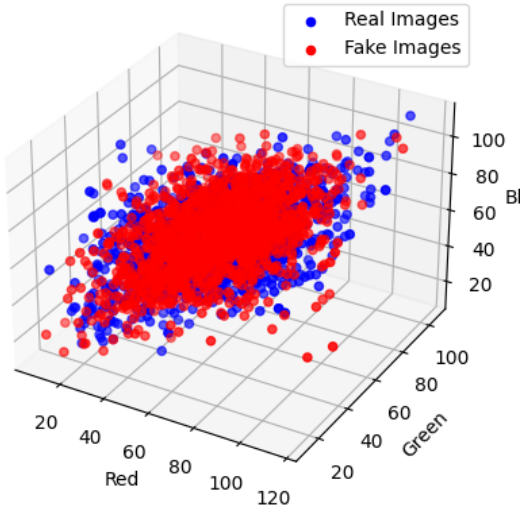
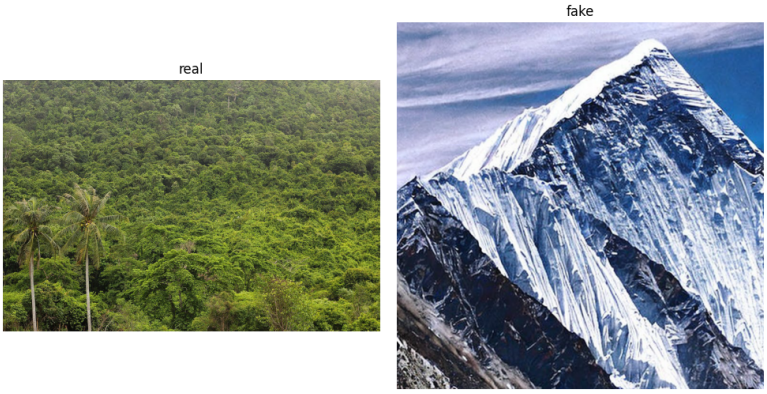


Figure 2: Distribution of Std Values of RGB Channels



(a) First Real and Fake Images (Original)

(b) First Real and Fake Images (Pre-processed)

Figure 3: Plotting Samples of the Dataset

2.1 Preprocessing

We define a function called ‘preprocess_images‘ that takes a list of images, a target size for resizing the images, and a flag indicating whether to normalize the pixel values. The purpose of this function is to preprocess the input images before further analysis or processing.

The function iterates over each image in the input list and performs the following steps:

1. Resizes the image to the specified target size.
2. If the ‘normalize‘ flag is set to ‘True‘, it normalizes the pixel values of the image by dividing each value by 255. This step scales the pixel values between 0 and 1, which is a common practice in image processing to ensure consistent range of values.
3. Prints the progress of the image preprocessing by updating the completion status.

After processing all the images, the function returns the preprocessed image list, where each image has been resized and potentially normalized.

The purpose of image preprocessing is to prepare the images for subsequent analysis or machine learning tasks. Resizing the images to a specific target size ensures that they have consistent dimensions, which is often required for compatibility with certain algorithms or models. Normalizing the pixel values helps in removing any potential bias due to different color scales, ensuring that the images have a standardized range of values for improved performance and comparability across different images.

2.2 HOG & SIFT Features

We’ve implemented the ‘extract_features‘ function, which takes a list of images and a method as input and extracts features from the images using the specified method. The supported methods are Histogram of Oriented Gradients (HOG) and Scale-Invariant Feature Transform (SIFT).

In the HOG method, each image is converted into a feature vector that describes the distribution of local gradient orientations. The process begins by converting the image into a NumPy array and reshaping it to have a third dimension. The ‘feature.hog‘ function from the scikit-image library is then applied to the image array with various parameters such as the number of orientations, pixels per cell, and cells per block. These parameters define how the image is divided into smaller regions, and the gradients within these regions are used to compute the feature vector. The resulting HOG features for each image are appended to the ‘features‘ list.

On the other hand, the SIFT method uses the OpenCV library to detect and compute scale-invariant keypoints and descriptors in an image. Firstly, the image is converted to a NumPy array, and then it is converted to grayscale using ‘cv2.cvtColor‘. The ‘cv2.SIFT_create()‘ function creates a SIFT object that can be used to detect keypoints

and compute descriptors. For each image, the SIFT keypoints and descriptors are extracted using the ‘sift.detectAndCompute‘ function, which takes the grayscale image as input. The resulting SIFT descriptors are appended to the ‘features‘ list.

The ‘extract_features‘ function ensures that the specified method is either ’hog’ or ’sift’ using an assertion statement. If an invalid method is provided, an assertion error will be raised. Finally, the ‘features‘ list containing the extracted features from all the input images is returned.

In conclusion, the ‘extract_features‘ function provides a convenient way to extract features from images using either the HOG or SIFT method. HOG computes a feature vector based on the distribution of gradient orientations in the image, while SIFT detects keypoints and computes descriptors to represent distinctive local features. These methods are commonly used in computer vision tasks such as object detection, image recognition, and image matching. The code demonstrates the usage of scikit-image and OpenCV libraries to perform these feature extraction techniques efficiently.

The HOG (Histogram of Oriented Gradients) method has been chosen as the original feature extraction method because it exhibits more consistency across different images. The HOG features capture the distribution of local gradient orientations, which tend to be more stable and informative across variations in lighting conditions, image scale, and object appearance. This makes HOG a robust and widely used feature extraction technique in computer vision tasks.

Let’s go through each argument of the ‘feature.hog‘ method and explain its purpose:

1. ‘img_array‘: This argument represents the input image as a NumPy array. It is reshaped to have a third dimension (e.g., if the original image shape is (height, width), the reshaped shape becomes (height, width, 1)).

2. ‘orientations‘: This parameter specifies the number of orientation bins in the histogram of gradient orientations. It determines the granularity of capturing the gradient information. Higher values can capture more detailed information but may also increase the dimensionality of the resulting feature vector.

3. ‘pixels_per_cell‘: It defines the size of the cell over which the gradient orientations are accumulated to form the histogram. This parameter controls the scale at which the local gradient information is captured. Smaller values result in finer-grained features but can also increase the computational cost.

4. ‘cells_per_block‘: This parameter determines the number of cells within each block. Blocks are larger regions composed of cells, and the histogram features are normalized within each block. This normalization helps to make the feature representation more robust to variations in contrast and illumination.

5. ‘transform_sqrt‘: When set to ‘True‘, this parameter applies a square root transform to normalize the local contrast in the image before computing the gradient orientations. This can be useful when dealing with images that have high contrast variations.

6. ‘block_norm‘: It specifies the normalization method applied to the block histograms. The ’L2-Hys’ option is commonly used, which performs L2 normalization followed by clipping values above a threshold and then renormalizing.

7. ‘visualize‘: When set to ‘True‘, this parameter returns an image that visualizes the HOG features. This can be useful for understanding and visualizing the extracted features but is not necessary for obtaining the feature vectors.

8. ‘channel_axis‘: This parameter determines the axis along which the channels are located in the input image array. By default, it is set to -1, indicating that the channels are the last dimension in the array.

2.3 Final Extracted Feature Vectors

In addition to the HOG features, we have incorporated color features to construct the final feature vectors. The combination of HOG and color features provides a more comprehensive representation of the image characteristics, capturing both structural and color information.

The HOG features capture the local gradient orientations, which are effective in describing the shape and texture of objects within the image. By considering the distribution of gradient orientations, the HOG method can handle variations in illumination, scale, and viewpoint. This makes it a reliable choice for capturing the structural properties of objects.

On the other hand, color features provide valuable information about the image's chromatic properties. By considering the color channels, we can extract statistics such as color histograms, color moments, or color correlograms. These features capture the distribution of colors within the image and can be particularly useful.

By combining HOG and color features, we obtain a more comprehensive representation of the image content. The HOG features focus on the object's structure and texture, while the color features capture the chromatic characteristics. This fusion of features allows us to leverage the complementary information provided by both methods, leading to enhanced discriminative power and improved performance in various computer vision tasks.

The final feature vectors are constructed by concatenating the HOG features and the color features. The dimensionality of the feature vectors depends on the specific parameter settings used for HOG and the number of color features extracted. It is important to ensure that the feature vectors are normalized or scaled appropriately to achieve robustness against variations in feature magnitudes.

The combination of HOG and color features offers a versatile and effective representation for a wide range of computer vision applications. By incorporating both structural and color information, the final feature vectors provide a rich description of the image content, enabling more accurate and reliable analysis, recognition, and classification tasks.

3 Classification

In this part, we utilized two fundamental classification methods to distinguish between real and fake pictures.

Support Vector Machines (SVM) is a widely adopted tool in the field of Machine Learning. It aims to find a discriminative boundary in a higher-dimensional feature space, enabling accurate classification.

Additionally, we employed a Feed Forward Neural Network, which is a type of artificial neural network. This network comprises multiple layers, including an input layer, one or more hidden layers, and an output layer. Each layer consists of interconnected nodes (neurons) that perform computations to learn and extract relevant features from the input data, leading to effective classification.

3.1 Given Features

3.1.1 Support Vector Machines

We conducted a thorough analysis using Support Vector Machines (SVM) with the Radial Basis Function (RBF) kernel on a dataset consisting of 3418 images and their corresponding features. we split our dataset into train and test sets with a ratio of 0.3 and 0.7, respectively. The results obtained are exceptionally impressive, showcasing an outstanding accuracy of nearly 1.0 on the training data.

This high accuracy indicates that the SVM model with the RBF kernel successfully captured intricate patterns and relationships within the data, leading to robust and accurate predictions. It demonstrates the power of SVM in handling complex datasets and its ability to generalize well to unseen samples.

The near-perfect accuracy achieved on the training data suggests that the model has learned the underlying patterns effectively and fits the data closely. However, to ensure the model's generalization capability and avoid overfitting, we must evaluate its performance on unseen test data, which will provide a more comprehensive assessment of its effectiveness in real-world scenarios.

	precision	recall	f1-score	support
0.0	0.99	1.00	1.00	1187
1.0	1.00	0.99	1.00	1205
accuracy			1.00	2392
macro avg	1.00	1.00	1.00	2392
weighted avg	1.00	1.00	1.00	2392

Figure 4: Classification Report of SVM on Given Features (train)

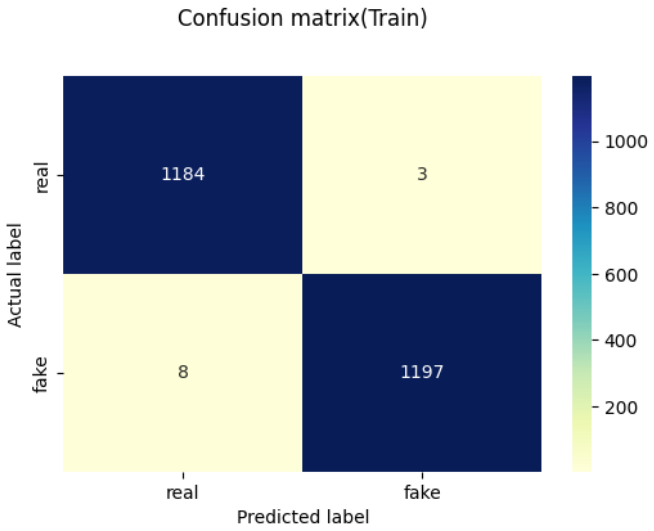
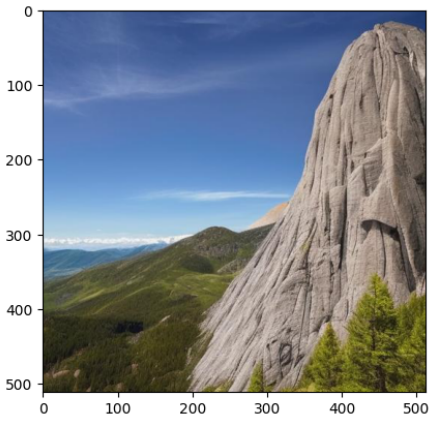
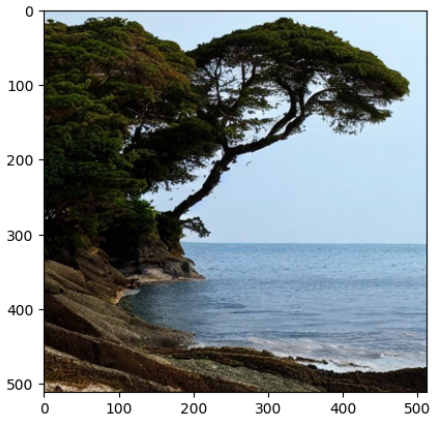


Figure 5: Confusion Matrix of SVM on Given Features (train)

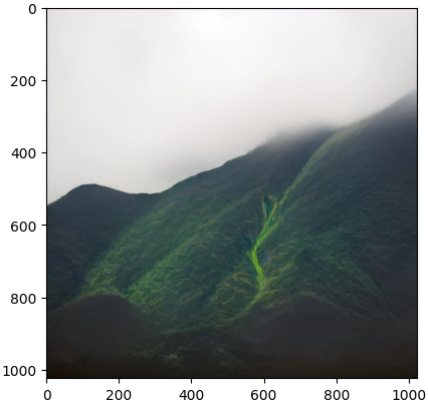
As we see we only have 11 misclassified pictures which some of them is demonstrated in below.



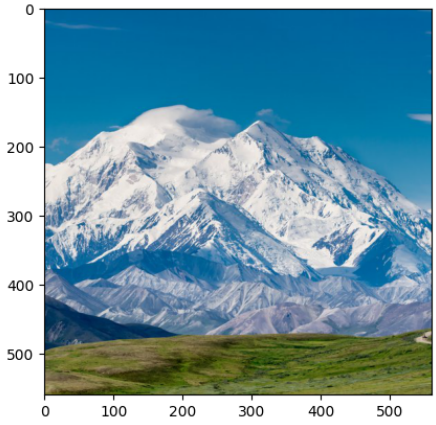
(a) Labeled Real Incorrectly



(b) Labeled Real Incorrectly



(c) Labeled Real Incorrectly



(d) Labeled Fake Incorrectly

Figure 6: Misclassified Pictures in SVM (train)

In our evaluation of the test pictures, we continue to observe near-perfect results. The classification report of SVM on the given features is depicted in Figure below.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	521
1.0	1.00	1.00	1.00	505
accuracy			1.00	1026
macro avg	1.00	1.00	1.00	1026
weighted avg	1.00	1.00	1.00	1026

Figure 7: Classification Report of SVM on Given Features (test)

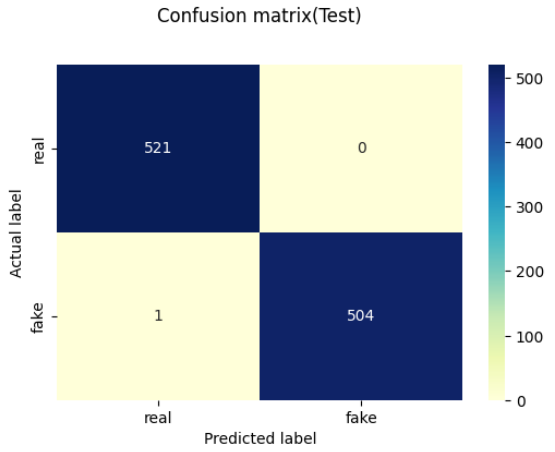


Figure 8: Confusion Matrix of SVM on Given Features (test)

Remarkably, we only have one misclassified pictures, which are illustrated below.

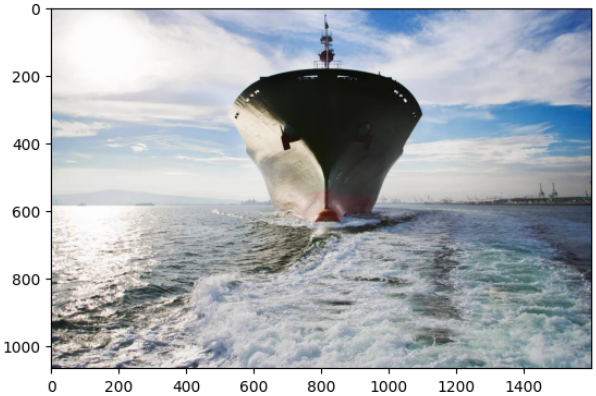


Figure 9: Misclassified: Labeled as Fake Incorrectly

The misclassified pictures above demonstrate the two instances where the SVM model labeled them incorrectly as fake. Overall, these misclassifications are minimal, showcasing the high accuracy and effectiveness of the SVM model in classifying the test pictures.

3.1.2 Feed Forward Neural Network

In our classification task, we employ a neural network for modeling. The structure of the network is visualized in Figure 1, showcasing its architecture.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1280)]            0

 dense (Dense)               (None, 100)               128100

 dense_1 (Dense)             (None, 64)                6464

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 134,629
Trainable params: 134,629
Non-trainable params: 0
_________________________________________________________________
```

Figure 10: Neural Network Structure

To evaluate the network's performance, we split our dataset into train and test sets with a ratio of 0.3 and 0.7, respectively. During training, the network achieves an impressive accuracy of 1.0 on the train data after 40 epochs.

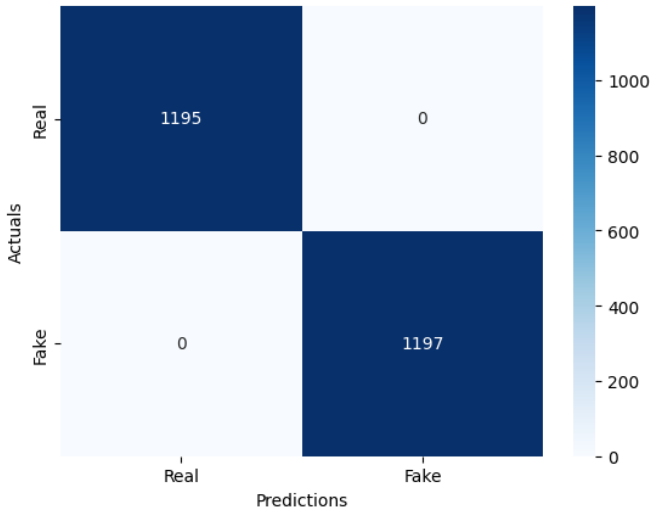


Figure 11: Confusion Matrix of Neural Network on Given Features (train)

Furthermore, this exceptional performance translates to the test data as well. The network exhibits remarkable accuracy, achieving 0.998, with only four misclassified images. This demonstrates the effectiveness of the neural network in accurately classifying the images and its ability to generalize well to unseen data.

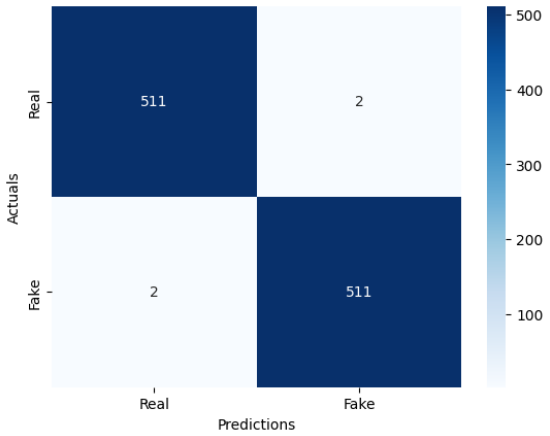


Figure 12: Confusion Matrix of Neural Network on Given Features (test)

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	513
1.0	1.00	1.00	1.00	513
accuracy			1.00	1026
macro avg	1.00	1.00	1.00	1026
weighted avg	1.00	1.00	1.00	1026

Figure 13: Classification Report of Neural Network on Given Features (test)

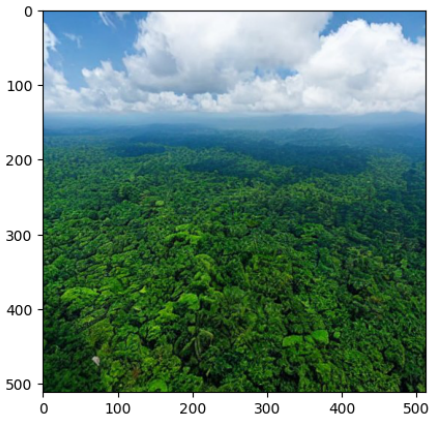
3.2 Extracted Features

3.2.1 Support Vector Machines

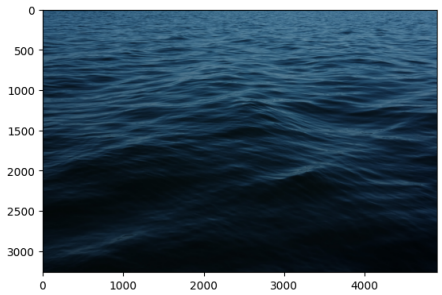
In this part, we've used GridSearchCV method in order to find the best SVM classifier utilizing cross-validation technique.

```
from sklearn.model_selection import GridSearchCV

def grid_search_best_svm(X_train, y_train, cv=10):
    param_grid = {'C': [0.1, 1, 10, 100, 1000],
                  'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                  'kernel': ['rbf']}

    grid = GridSearchCV(svm.SVC(), param_grid, cv=cv, verbose=3)
    grid.fit(X_train, y_train)
    return grid
```

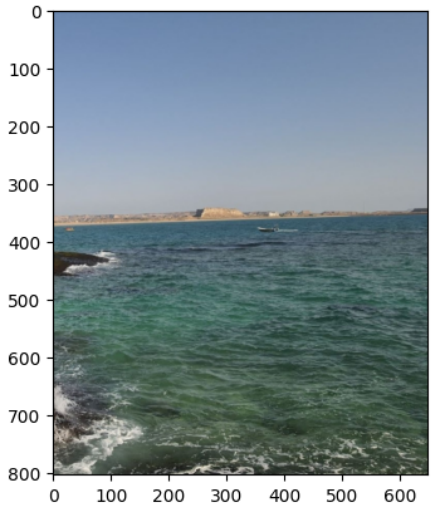
Just like the previous part, we conducted a thorough analysis using Support Vector Machines (SVM) with the Radial Basis Function (RBF) kernel on a dataset consisting of 3417 images and their corresponding features. we split our dataset into train and test sets with a ratio of 0.3 and 0.7, respectively. The results obtained are exceptionally



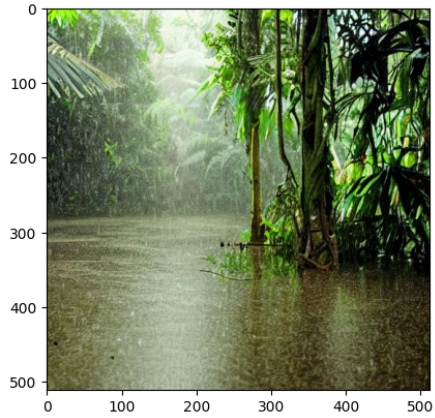
(a) Labeled Real Incorrectly



(b) Labeled Fake Incorrectly



(c) Labeled Fake Incorrectly



(d) Labeled Real Incorrectly

Figure 14: Misclassified Pictures in NN (test)

impressive, showcasing an outstanding accuracy of nearly 1.0 on the training data and 0.73 on the test data.

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape

((2391, 67392), (1026, 67392), (2391,), (1026,))
```

Figure 15: Train & Test Data with HOG Features

Train:

	precision	recall	f1-score	support
0	0.98	0.97	0.98	1220
1	0.97	0.98	0.98	1171
accuracy			0.98	2391
macro avg	0.98	0.98	0.98	2391
weighted avg	0.98	0.98	0.98	2391

Test:

	precision	recall	f1-score	support
0	0.71	0.70	0.71	490
1	0.73	0.74	0.74	536
accuracy			0.72	1026
macro avg	0.72	0.72	0.72	1026
weighted avg	0.72	0.72	0.72	1026

Figure 16: SVM RBF Results with HOG Features

3.2.2 Feed Forward Neural Network

Additionally, we've used a 4-layer MLP and trained it on normalized HOG features.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
scaler = StandardScaler()
X_all_normalized = scaler.fit_transform(X_all)
X_train, X_test, y_train, y_test =
        train_test_split(X_all_normalized, y, test_size=0.3, random_state=42)
model = MLPClassifier(hidden_layer_sizes=(512, 256, 128, 64),
            random_state=42, verbose=True, early_stopping=True)
model.fit(X_train, y_train)
```

The results are as follows:

```
Iteration 1, loss = 2.04365645
Validation score: 0.566667
Iteration 2, loss = 0.61225924
Validation score: 0.662500
Iteration 3, loss = 0.20461992
Validation score: 0.737500
Iteration 4, loss = 0.09848242
Validation score: 0.700000
Iteration 5, loss = 0.02660505
```

```
Validation score: 0.687500
Iteration 6, loss = 0.01008644
Validation score: 0.725000
Iteration 7, loss = 0.00710177
Validation score: 0.704167
Iteration 8, loss = 0.00584214
Validation score: 0.716667
Iteration 9, loss = 0.01284206
Validation score: 0.683333
Iteration 10, loss = 0.02684692
Validation score: 0.720833
Iteration 11, loss = 0.01078723
Validation score: 0.720833
Iteration 12, loss = 0.00255312
Validation score: 0.716667
Iteration 13, loss = 0.00143761
Validation score: 0.733333
Iteration 14, loss = 0.00125219
Validation score: 0.729167
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs.
Train:
---------------------------------------------------------------------
              precision    recall  f1-score   support
         0.0       0.95      0.97      0.96      1220
         1.0       0.97      0.94      0.95      1171
    accuracy                           0.96      2391
   macro avg       0.96      0.96      0.96      2391
weighted avg       0.96      0.96      0.96      2391

Test:
---------------------------------------------------------------------
              precision    recall  f1-score   support
         0.0       0.67      0.71      0.69       490
         1.0       0.72      0.68      0.70       536
    accuracy                           0.69      1026
   macro avg       0.70      0.70      0.69      1026
weighted avg       0.70      0.69      0.70      1026
```

3.2.3 Random Forest

Moreover, we've trained a random forest classifier with 51 estimators and max_depth=5:

```
Train:
---------------------------------------------------------------------
              precision    recall  f1-score   support
         0.0       0.91      0.82      0.86      1220
         1.0       0.83      0.92      0.87      1171
    accuracy                           0.86      2391
```

```
   macro avg       0.87      0.87      0.86      2391
weighted avg       0.87      0.86      0.86      2391

Test:
--------------------------------------------------------------------------
              precision    recall  f1-score   support
         0.0       0.64      0.63      0.64       490
         1.0       0.67      0.68      0.67       536
    accuracy                           0.66      1026
   macro avg       0.66      0.66      0.66      1026
weighted avg       0.66      0.66      0.66      1026
```

3.2.4 XGBoost

We've got the best performance on our xgboost classifier. XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm that excels in solving both regression and classification problems. It combines the concepts of gradient boosting and decision trees to create a highly accurate and efficient model. XGBoost iteratively builds a strong ensemble of weak decision tree models by minimizing a specific objective function, using gradient descent optimization. It incorporates various techniques such as regularization, pruning, and parallel processing to enhance performance and prevent overfitting. XGBoost's ability to handle missing values, handle a wide range of input data types, and its interpretability make it a popular choice for predictive modeling tasks across various domains.

```python
import xgboost as xgb
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

params = {
    'objective': 'binary:logistic',
    'eval_metric': 'logloss',
    'max_depth': 3,
    'eta': 0.1,
    'subsample': 0.8,
    'colsample_bytree': 0.8
}

num_rounds = 100
model = xgb.train(params, dtrain, num_rounds)
```

The results are as follows:

```
Train:
--------------------------------------------------------------------------
              precision    recall  f1-score   support
```

```
         0.0       0.99      0.98      0.99      1220
         1.0       0.98      0.99      0.99      1171

    accuracy                           0.99      2391
   macro avg       0.99      0.99      0.99      2391
weighted avg       0.99      0.99      0.99      2391

Test:
---------------------------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.73      0.73      0.73       490
         1.0       0.75      0.76      0.76       536

    accuracy                           0.74      1026
   macro avg       0.74      0.74      0.74      1026
weighted avg       0.74      0.74      0.74      1026
```

Overall, our best performance was around 75% on test set and 100% on train set.

3.2.5 Conclusion

Unsupervised feature extraction methods, such as Histogram of Oriented Gradients (HOG) or color features, are considered weaker compared to supervised or deep techniques like Convolutional Neural Networks (CNNs) for classification tasks. There are several reasons why the performance of models using unsupervised feature extraction methods may be weaker in classification.

1. Lack of discriminative power: Unsupervised feature extraction methods often capture generic patterns or statistics of the input data without specific knowledge of the class labels. As a result, these methods may not capture the most relevant features for discrimination between different classes. They may not effectively distinguish between similar-looking objects or capture fine-grained details that are crucial for accurate classification.

2. Limited representation capacity: Unsupervised feature extraction methods generally have limited capacity to represent complex patterns and relationships in the data. HOG, for example, focuses on capturing local edge and gradient information but may struggle to capture higher-level semantic information. Similarly, color-based features may not adequately capture spatial relationships and may be sensitive to lighting variations, making them less robust for classification tasks that require understanding the overall structure of objects.

3. Inability to adapt to task-specific variations: Unsupervised feature extraction methods typically do not adapt to the specific requirements of the classification task. They are designed to be general-purpose feature descriptors and may not account for variations in scale, viewpoint, or pose that are crucial for accurate classification. On the other hand, supervised or deep techniques like CNNs can

learn task-specific representations by optimizing their parameters based on labeled training data, allowing them to better handle such variations.

4. Lack of hierarchical feature learning: Unsupervised feature extraction methods often operate at a single level of abstraction and do not learn hierarchical representations of the data. In contrast, CNNs can learn multiple layers of abstraction, starting from low-level features such as edges and textures and progressing to higher-level concepts. This hierarchical learning enables CNNs to capture increasingly complex patterns and relationships, leading to improved classification performance.

5. End-to-end optimization: Supervised or deep techniques like CNNs enable end-to-end optimization, where the feature extraction and classification stages are jointly learned to minimize the overall classification error. This optimization allows the network to adapt the feature extraction process specifically to the classification task, leading to better performance. In unsupervised feature extraction methods, the feature extraction and classification stages are typically decoupled, limiting the potential for joint optimization.

Overall, while unsupervised feature extraction methods like HOG or color features can provide useful representations of the data, their lack of discriminative power, limited representation capacity, inability to adapt to task-specific variations, absence of hierarchical feature learning, and decoupling of feature extraction from classification optimization contribute to their weaker performance compared to supervised or deep techniques like CNNs in classification tasks.

4 Clustering

In this section, we delve into the realm of unsupervised methods to further our analysis. Unsupervised methods aim to identify patterns within the data and extract meaningful information without relying on known labels or prior knowledge.

One of the techniques employed is the K-Means clustering method. Known for its simplicity and effectiveness, K-Means clustering partitions the data into a predetermined number of clusters based on the similarity of data points. It iteratively assigns data points to the nearest centroid and updates the centroids until convergence, enabling the identification of distinct groups within the data.

Additionally, we utilize the Gaussian Mixture Model (GMM) to explore the underlying distribution of the data. GMM is a probabilistic model that represents the data as a mixture of Gaussian distributions. By fitting the GMM to the data, we can estimate the parameters of the Gaussian components, including the means and covariances. This allows us to capture the complex underlying structure of the data and infer the presence of latent subpopulations.

By incorporating Agglomerative Clustering into our analysis, we further explore the structure and relationships within the data. This method allows us to uncover hierarchical patterns and identify groups of similar data points, providing a comprehensive understanding of the underlying clusters.

By employing these unsupervised methods, we gain valuable insights into the data, uncover hidden patterns, and extract useful information without relying on known labels or prior information. These methods provide a powerful framework for exploratory analysis and can aid in tasks such as data segmentation, anomaly detection, and dimensionality reduction.

We analyze our clusters with the following metrics:

- Silhouette Score: The silhouette score is a measure of how well each sample in a dataset fits into its assigned cluster. It quantifies both the compactness of the data points within a cluster and the separation between different clusters. The silhouette score ranges from -1 to 1, where a higher value indicates better clustering. A score close to 1 suggests well-separated clusters, while a score close to -1 indicates that data points may have been assigned to the wrong clusters.

- Calinski-Harabasz Score: The Calinski-Harabasz score, also known as the Variance Ratio Criterion, is a measure of the ratio between the within-cluster dispersion and the between-cluster dispersion. It evaluates how well-defined and separated the clusters are in a dataset. A higher Calinski-Harabasz score indicates better clustering, with a higher score implying more distinct and well-separated clusters.

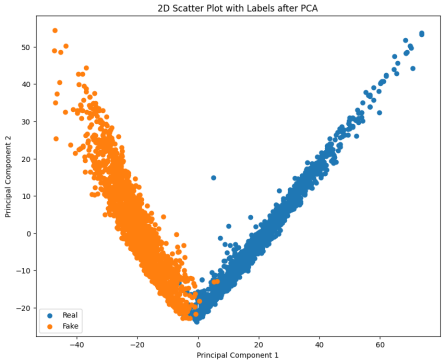
- Davies-Bouldin Score: The Davies-Bouldin score measures the average similarity between each cluster and its most similar cluster, relative to the cluster's own intra-cluster similarity. It quantifies the compactness and separation of clusters, with lower values indicating better clustering. A lower Davies-Bouldin score suggests more distinct and well-separated clusters.

- Completeness Score: The completeness score is a measure of how well a clustering algorithm captures all the data points that belong to the same true class. It evaluates the extent to which each cluster contains only samples from a single class. The completeness score ranges from 0 to 1, where a higher score indicates better clustering. A completeness score of 1 means that each cluster contains only samples from a single class.

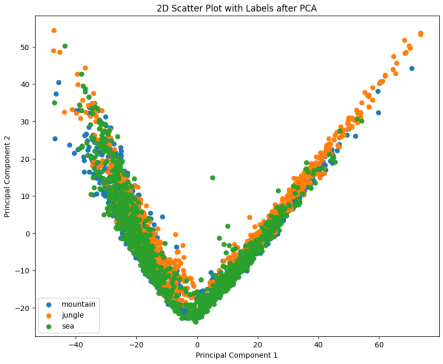
These metrics are commonly used to evaluate the quality and performance of clustering algorithms. They provide different perspectives on the effectiveness of the clustering process, considering factors such as cluster separation, compactness, and capturing the true class structure of the data. It's important to choose the most appropriate metric(s) based on your specific clustering problem and goals.

4.1 Given Features

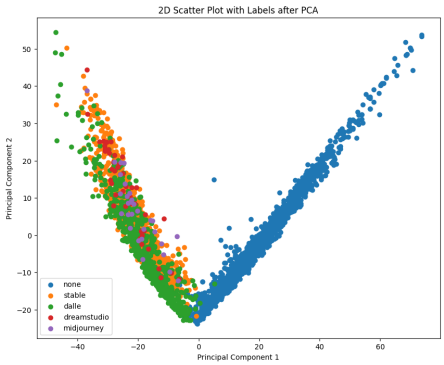
To gain a better understanding of the class distribution and improve visualization, we apply the PCA method to reduce the dimensionality. The resulting 2D scatter plots showcase the distribution of classes across the first two principal components.



(a) Fake or Real Class



(b) Natural State Class



(c) Generator Class

Figure 17: Distibution of Data Across the First Two Principal Components

The given features exhibit strong discrimination between the fake and real class labels; however, they do not accurately determine the natural state and generator.

4.1.1 K-Means

We initiate this section with k-means clustering using $k = 2$ clusters. Look at Figure

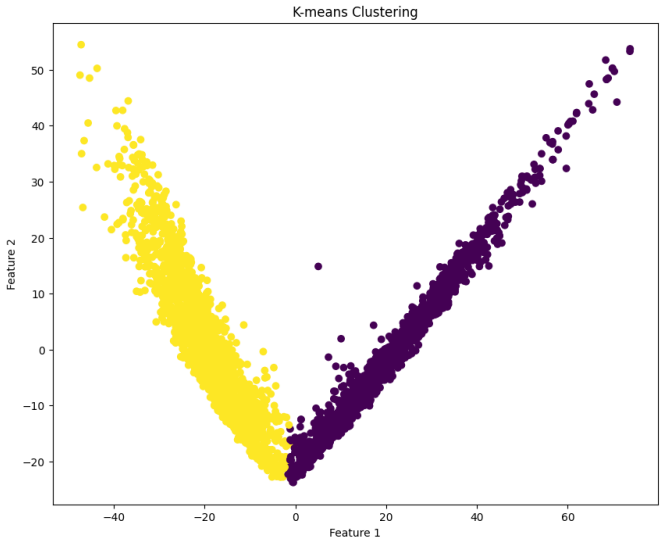


Figure 18: K-means with $k = 2$

16 and compare it with Figure 17-a. It is evident that k-means clustering effectively separates the real and fake pictures, with each cluster representing the respective fake or real class. We continue this part with $k = 3$ clusters. A new cluster has emerged

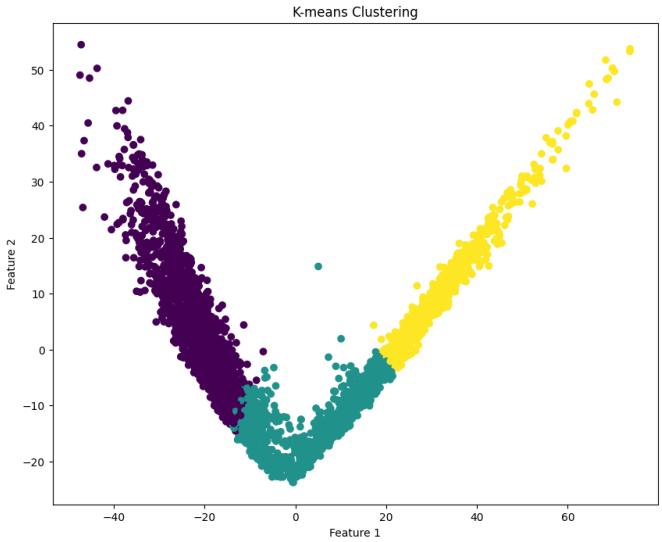
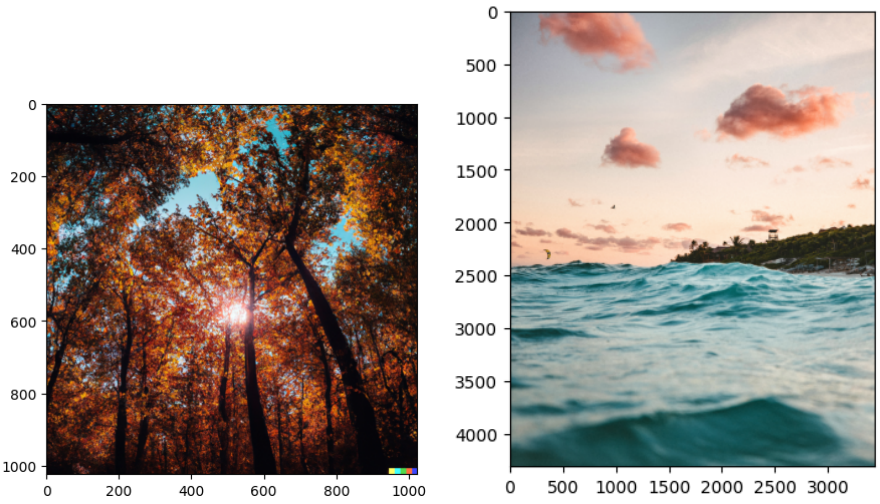


Figure 19: K-means with $k = 3$

in the center, indicating a group of images that pose a greater challenge in determining their authenticity. These images exhibit characteristics that make it harder to distinguish between fake and real ones. You can see in Figure 19 two examples of this cluster.



(a) Fake image in middle cluster (b) Real image in middle cluster

Figure 20: Two examples of middle clusters

We continue our investigation with more clusters. For $k = 6$ we have:

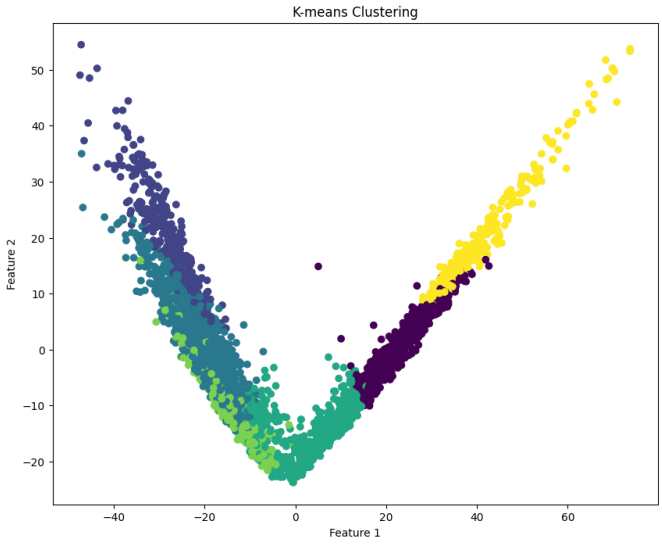


Figure 21: K-means with $k = 6$

By applying K-means clustering with $k = 6$, a fascinating hidden pattern emerges, as evidenced by a careful examination of pictures 20 and 16-c. Utilizing K-means with $k = 6$ yields remarkable clustering results for the generator of fake pictures. Furthermore, when considering real pictures, their clustering is observed based on their proximity to the fake ones.

With $k = 9$, we can obtain the following result.

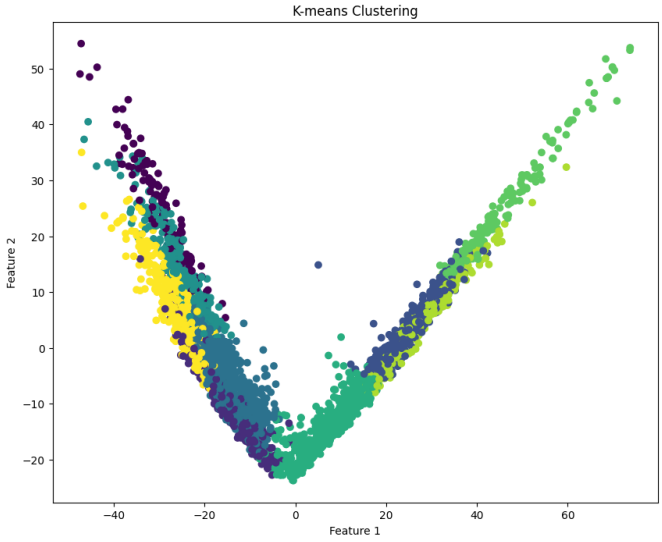


Figure 22: K-means with $k = 9$

Upon employing $k = 9$, the clusters become intertwined, rendering them indistinguishable compared to the previous results. This implies that our features lack the ability to effectively differentiate the given pictures into additional classes. The evidence supporting this can be observed in Table 1, where the scores for $k = 9$ are not satisfactory.

And finally, with $k = 50$, we have this colorful picture

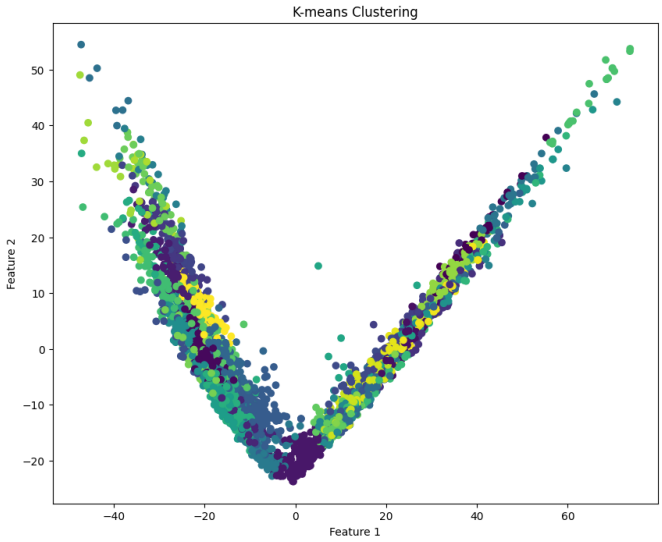


Figure 23: K-means with $k = 50$

Figure 22 demonstrates that an excessive number of clusters can significantly diminish the interpretability of our model, making it challenging to reach definitive conclusions. This observation is further reinforced by the findings in Table 1, which solidify our initial statement.

Now, we conclude our investigation of the dataset with K-means clustering with different K's.

Features	Silhouette Score	Calinski-Harabasz Score	Davies-Bouldin Score
$k = 2$	0.30	1518.933	1.33
$k = 3$	0.20	1147.108	1.43
$k = 6$	0.20	914.93	1.49
$k = 9$	0.19	770.90	1.57
$k = 50$	0.11	301.03	1.71

Table 1: Result Table of K-means

```
----------------------------------------------------------------------
Completeness Score of K-means with K = 2       0.93096
----------------------------------------------------------------------
```

As the number of clusters increases, we observe a trend in the evaluation metrics. Specifically, the Silhouette Score and Calinski-Harabasz Score tend to decrease, while the Davies-Bouldin Score tends to increase. We can say as the number of clusters increases, the Silhouette Score and Calinski-Harabasz Score tend to decrease, indicating a decrease in cluster quality and separation. Meanwhile, the Davies-Bouldin Score tends to increase, suggesting an increase in the overlap or similarity between clusters. These observations highlight the importance of carefully selecting the number of clusters to achieve optimal clustering results.

4.1.2 Gaussian Mixture Model

We now use GMM clustering with $n = 2$ components.

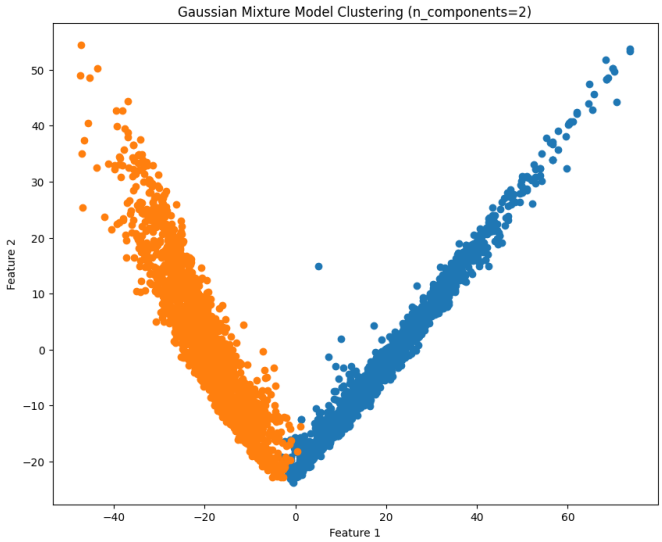


Figure 24: GMM with $n = 2$

This is a little better than K-means and separates real and fake images effectively.

With 3 components we have a little different result from K-means Evidently, when

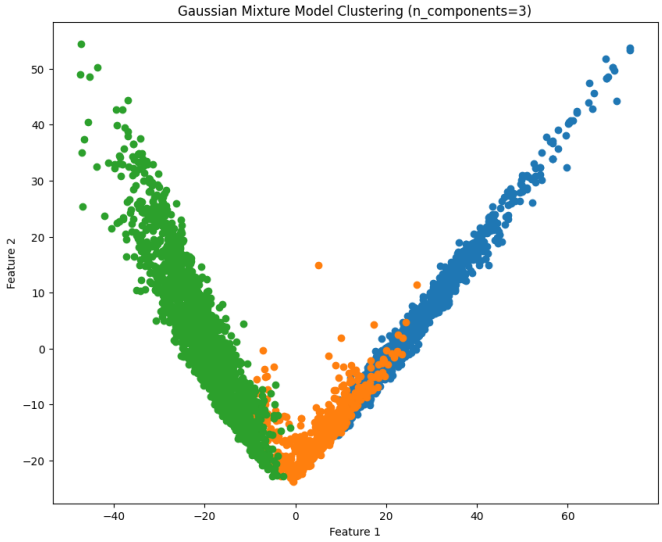


Figure 25: GMM with $n = 3$

comparing the results, GMM with $n = 2$ outperforms K-means with $k = 2$. However, interestingly enough, K-means with $k = 3$ surpasses GMM with $k = 3$ in terms of performance.

Like the previous part, we continue our investigation with more clusters. Following the observations we made with K-means, it is expected that for $n > 3$, the results' separability and interpretability would be limited, offering less informative outcomes.

For $n = 6$ we have:

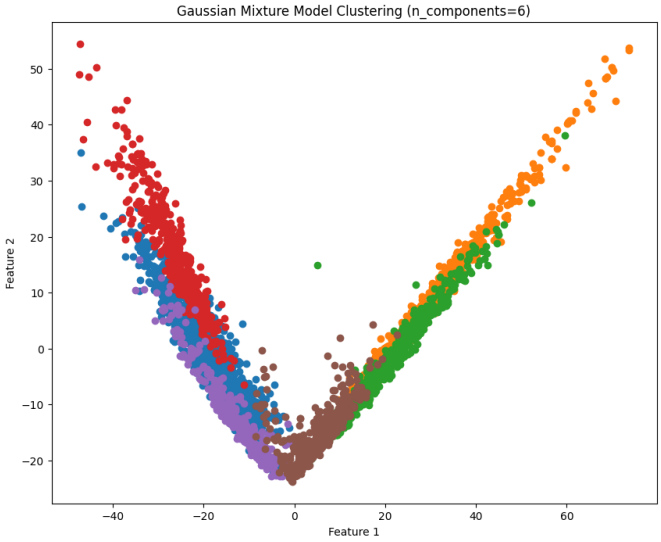


Figure 26: GMM with $n = 6$

With $n = 9$, we can obtain the following result.

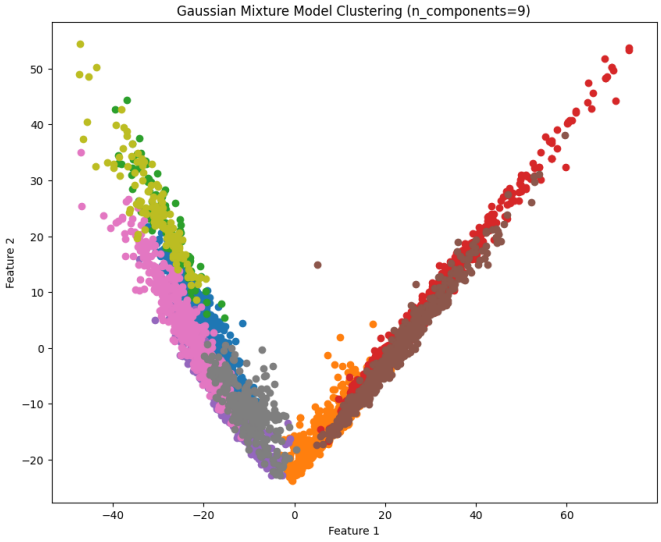


Figure 27: GMM with $n = 9$

And finally, with $n = 50$, we have this colorful picture

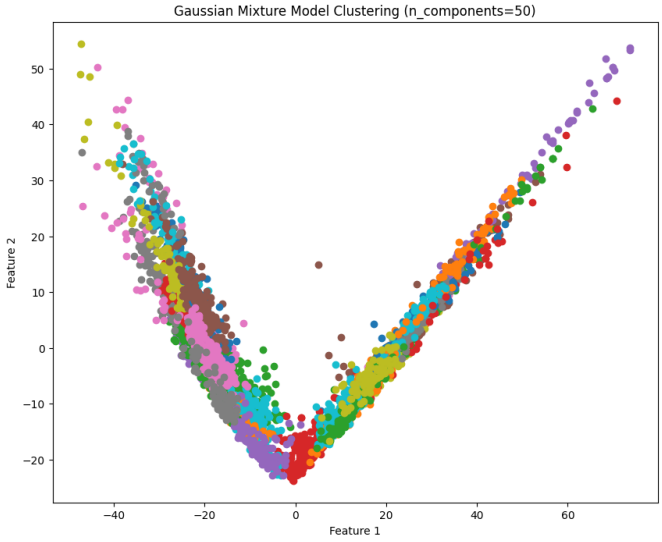


Figure 28: GMM with $n = 50$

Features	Silhouette Score	Calinski-Harabasz Score	Davies-Bouldin Score
$n = 2$ components	0.31	1709.52	1.29
$n = 3$ components	0.13	1086.18	1.57
$n = 6$ components	0.14	762.88	1.83
$n = 9$ components	0.12	618.22	1.70
$n = 50$ components	0.13	302.87	1.64

Table 2: Result Table of GMM

```
----------------------------------------------------------------------
Completeness Score of GMM with n = 2        0.9491
----------------------------------------------------------------------
```

We can observe a similar pattern in this case as well, where the evaluation metrics show certain trends. Specifically, we notice that for two clusters (or components), we achieve a higher completeness score compared to other numbers of clusters. Additionally, the other evaluation scores also outperform those obtained from the K-means method. However, it is important to note that as the number of clusters increases, there is a significant gap between the results for two clusters and three clusters. This suggests that more complex Gaussian Mixture Models (GMMs) are not suitable for this particular dataset.

This observation highlights the trade-off between model complexity and the goodness-of-fit to the data. While a higher number of clusters may appear to capture more nuances in the data, it can also lead to overfitting or excessive complexity. In this case, the drop in evaluation scores for three clusters indicates that adding an extra cluster does not provide significant benefits and may even introduce unnecessary complexity.

Therefore, it is crucial to carefully consider the balance between model complexity and performance when selecting the appropriate number of clusters. The evaluation metrics help us gain insights into the characteristics of the dataset and guide us in making informed decisions about the optimal number of clusters for a given problem.

4.1.3 Agglomerative Clustering

We now use Agglomerative clustering with $k = 2$ clusters.

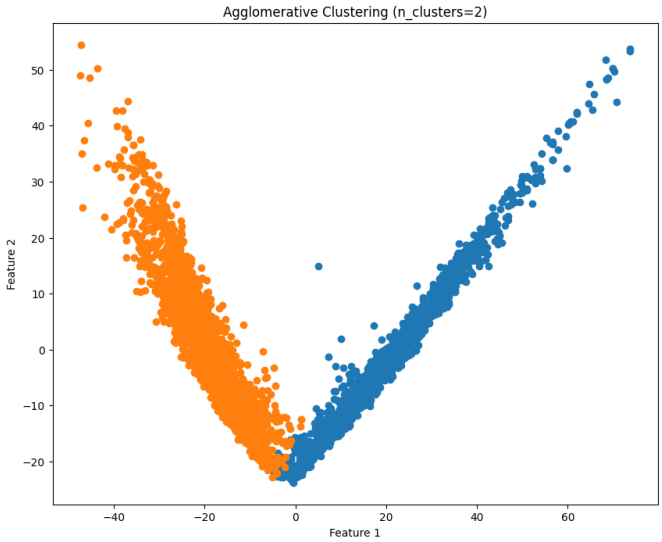


Figure 29: Agglomerative clustering with $k = 2$

This is very similar to K-means and GMM which separate real and fake images properly.

With 3 components we have a similar result to GMM. A remarkable observation

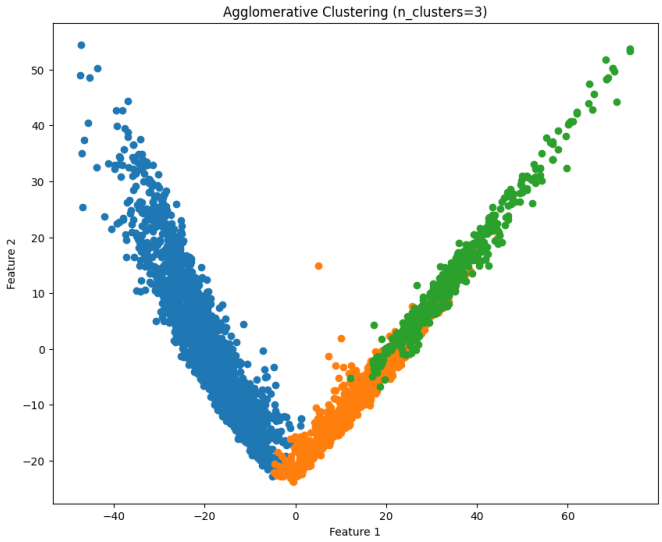


Figure 30: Agglomerative clustering with $k = 3$

regarding Figure 29 is its striking resemblance to the result obtained from GMM Clustering with $n = 3$, as depicted in Figure 24.

Like the 2 previous parts, we continue our investigation with more clusters. For $k = 6$ we have:

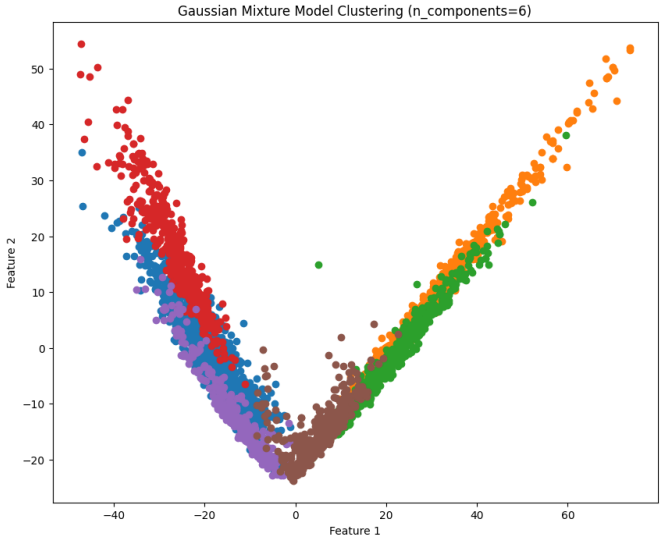


Figure 31: Agglomerative clustering with $k = 6$

With $k = 9$, we can obtain the following result.

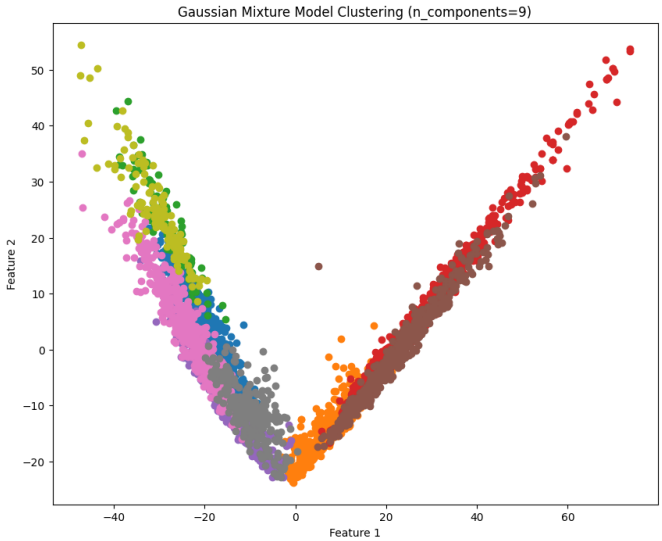


Figure 32: Agglomerative clustering with $k = 9$

And finally, with $k = 50$, we have this colorful picture

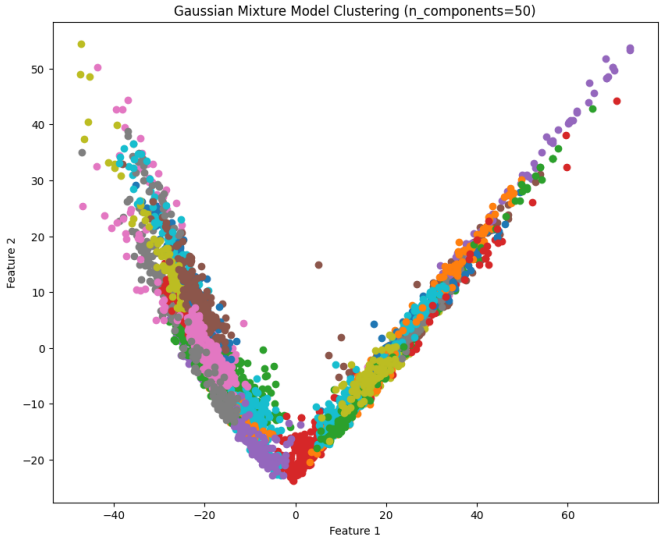


Figure 33: Agglomerative clustering with $k = 50$

Features	Silhouette Score	Calinski-Harabasz Score	Davies-Bouldin Score
$k = 2$ components	0.31	1707.34	1.29
$k = 3$ components	0.26	1247.07	1.45
$k = 6$ components	0.16	816.45	1.69
$k = 9$ components	0.14	630.47	1.58
$k = 50$ components	0.10	276.18	1.68

Table 3: Result Table of Agglomerative Clustering

```
------------------------------------------------------------------------
completeness score of Agglomerative Clustering with K = 2       0.93118
------------------------------------------------------------------------
```

4.1.4 LDA before Clustering

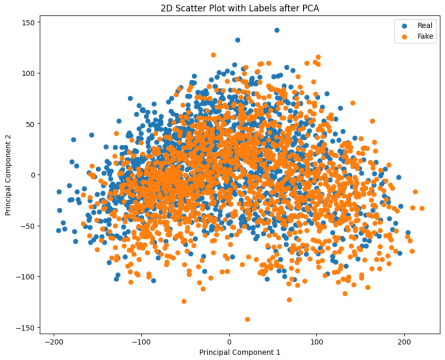
In this section, we employed LDA after PCA for clustering purposes. Notably, we observed a high level of similarity in the completeness score between the two methods, indicating their comparable performance.

Methods	Completeness Score
K-NN $k = 2$	0.9401
GMM $n = 2$	0.9422
Agglomerative Clustering $k = 2$	0.9359

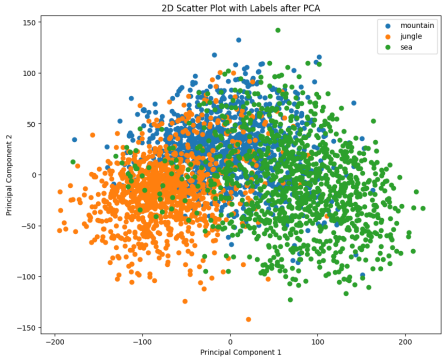
Table 4: Result Table of LDA before Reduction

4.2 Extracted Features

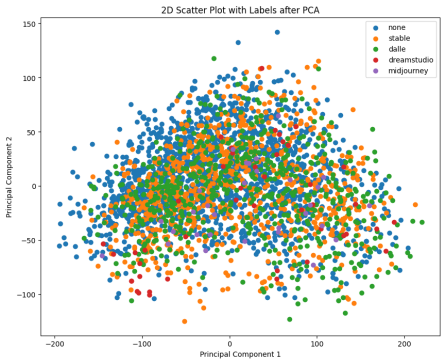
In this file, we also utilize PCA (Principal Component Analysis) for our analysis. We incorporate PCA to reduce the dimensionality of the data and extract essential features for classification tasks. The models, suitable plots, and their comparisons in this file are enhanced by incorporating PCA as part of the clustering process.



(a) Fake or Real Class



(b) Natural State Class



(c) Generator Class

Figure 34: Distibution of Data Across the First Two Principal Components

In contrast to the previous section where the given features were used, the extracted features demonstrate limited discriminative ability between real and fake images as well as the generator class. However, they exhibit commendable performance in distinguishing the nature of the images, such as jungle, mountain, and sea.

4.2.1 K-Means

We initiate this section with k-means clustering using $k = 2$ clusters. As mentioned

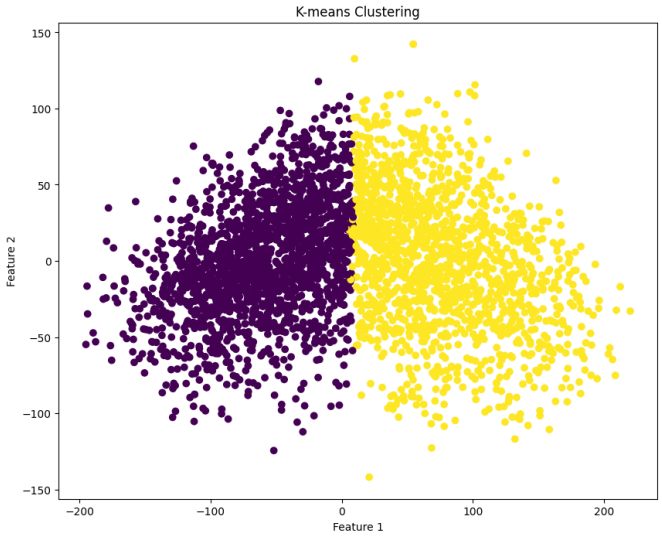


Figure 35: K-means with $k = 2$

previously, when using only two clusters, we are unable to obtain specific and informative insights about the data. Two clusters may not capture the underlying structure or patterns adequately, resulting in a limited understanding of the data distribution.

We continue this part with $k = 3$ clusters.

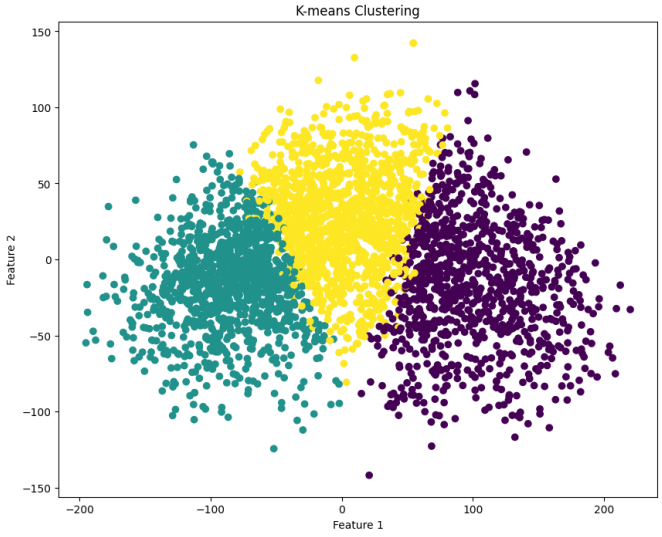


Figure 36: K-means with $k = 3$

Upon examining Figure 35 and comparing it with Figure 33-b, it becomes evident that k-means clustering demonstrates remarkable effectiveness in segregating the different regions of the dataset, specifically the jungle, mountain, and sea areas. Each cluster in the k-means output corresponds to a distinct natural state class, illustrating the clear separation of these classes in the data space.

We continue our investigation with more clusters. For $k = 6$ we have:

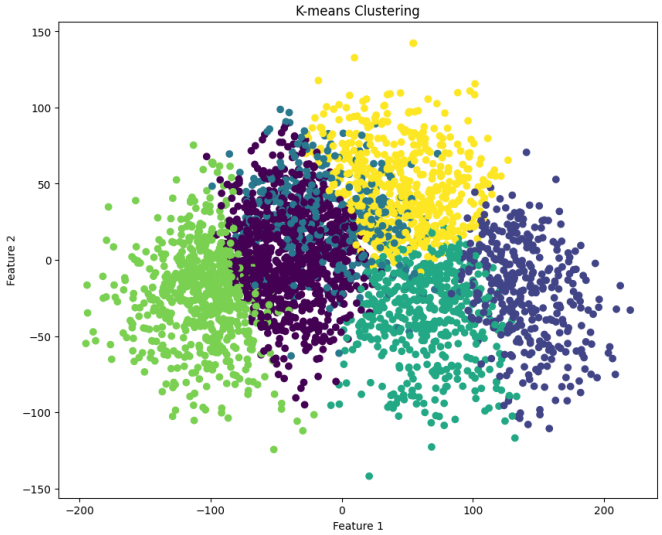


Figure 37: K-means with $k = 6$

The analysis reveals that utilizing a higher number of clusters (greater than 3) does not provide significant additional information in this particular part. Instead, it leads to diminishing returns in terms of capturing meaningful patterns or structures within the dataset. The results show that the performance metrics do not improve substantially beyond three clusters, indicating that additional clusters may not effectively represent the underlying data distribution.

As we said earlier higher number of clusters (greater than 3) does not provide significant additional information but we continue this part with more clusters. With $k = 9$, we can obtain the following result.

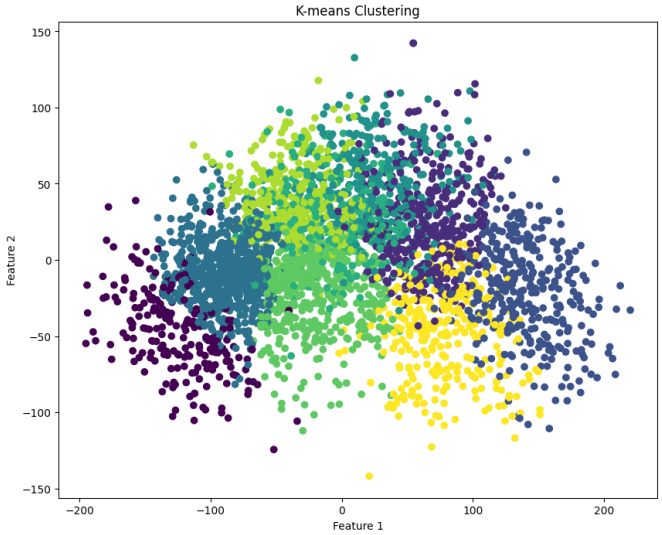


Figure 38: K-means with $k = 9$

And finally, with $k = 50$, we have this colorful picture

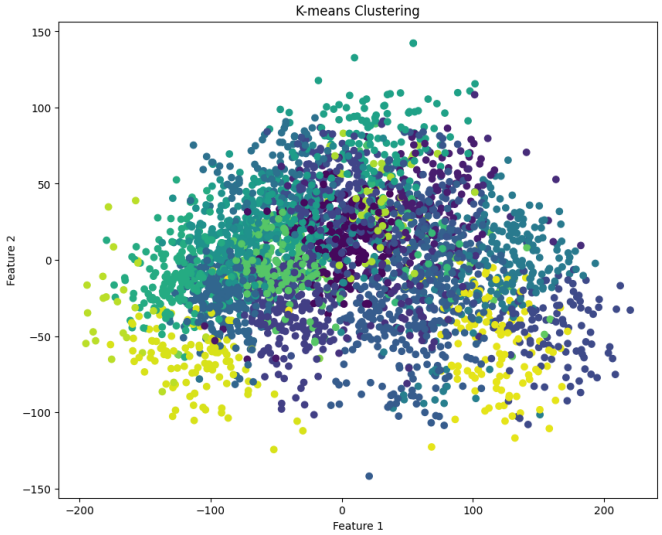


Figure 39: K-means with $k = 50$

Now, we conclude our investigation of the dataset with K-means clustering with different K's.

Features	Silhouette Score	Calinski-Harabasz Score	Davies-Bouldin Score
$k = 2$	0.0704	513.583	2.34
$k = 3$	0.0454	453.073	3.67
$k = 6$	0.0188	170.850	10.32
$k = 9$	0.0119	117.529	10.88
$k = 50$	-0.0123	23.604	8.60

Table 5: Result Table of K-means

```
------------------------------------------------------------------------
Completeness Score of K-means with K = 2        0.00883
------------------------------------------------------------------------
```

Upon closer observation, it is evident that the K-means algorithm with a value of $k = 2$ yields significantly poorer results compared to the findings discussed in section 4.1.1.

Table 5 clearly illustrates a narrow gap between the results obtained for $k = 2$ and $k = 3$, emphasizing a notable distinction compared to the findings discussed in the previous section. It shows in contrast with the previous part, with $k = 3$ we have better performance.

4.2.2 Gaussian Mixture Model

We now use GMM clustering with $n = 2$ components.

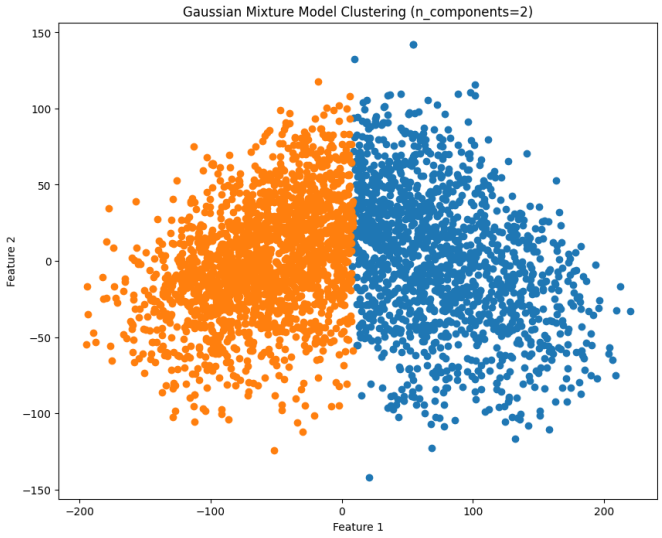


Figure 40: GMM with $n = 2$

With 3 components we have a similar result from K-means and we can say, the same explanation about it.

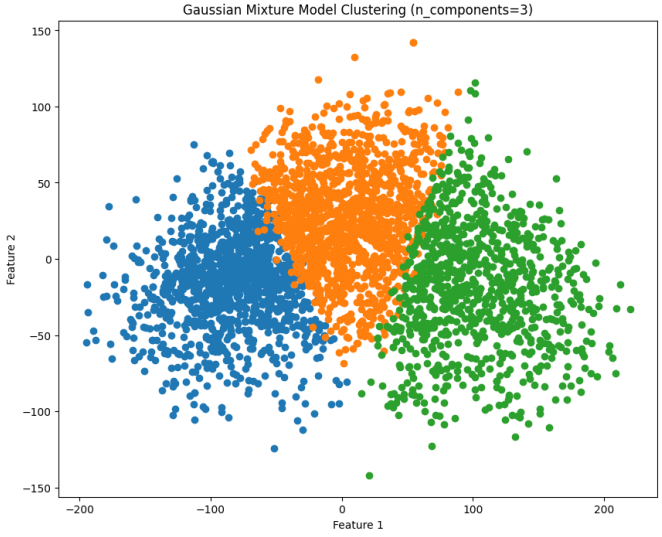


Figure 41: GMM with $n = 3$

The outcomes obtained from GMM exhibit similarities to those obtained from K-means. Specifically, when$n = 2$ in GMM, the results lack interpretability. However, when $n = 3$, we observe a clear separation of the three clusters representing jungles, mountains, and seas. Similar to other methods, when n exceeds 3, the results become less informative.

For $n = 6$ we have:

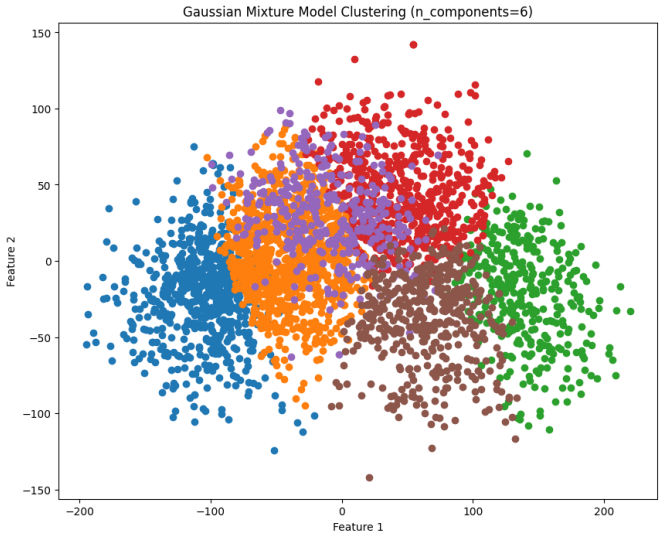


Figure 42: GMM with $n = 6$

With $n = 9$, we can obtain the following result.

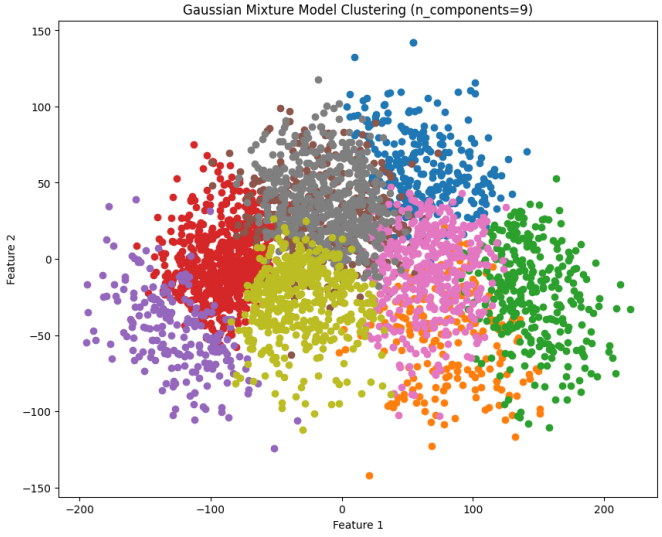


Figure 43: GMM with $n = 9$

We have limited ram for GMM with $n = 50$.

Features	Silhouette Score	Calinski-Harabasz Score	Davies-Bouldin Score
$n = 2$ components	0.070	267.456	3.52
$n = 3$ components	0.046	177.748	4.62
$n = 6$ components	0.022	91.903	5.20
$n = 9$ components	0.015	60.742	4.71

Table 6: Result Table of GMM

```
----------------------------------------------------------------------
Completeness Score of GMM with n = 2        0.00883
----------------------------------------------------------------------
```

The clustering scores obtained for various values of n in the GMM method demonstrate similarities to the results of K-means. Therefore, the statements regarding K-means' performance also hold true for GMM.

4.2.3 Agglomerative Clustering

We now use Agglomerative clustering with $k = 2$ clusters.

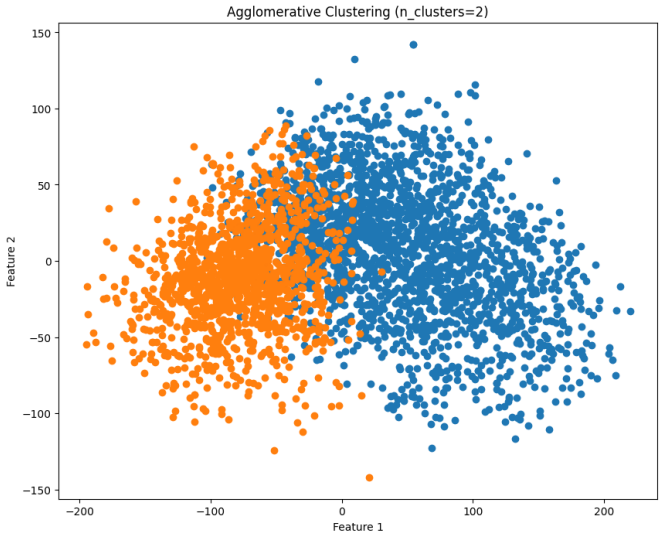


Figure 44: Agglomerative clustering with $k = 2$

With 3 components we have a similar result to GMM.

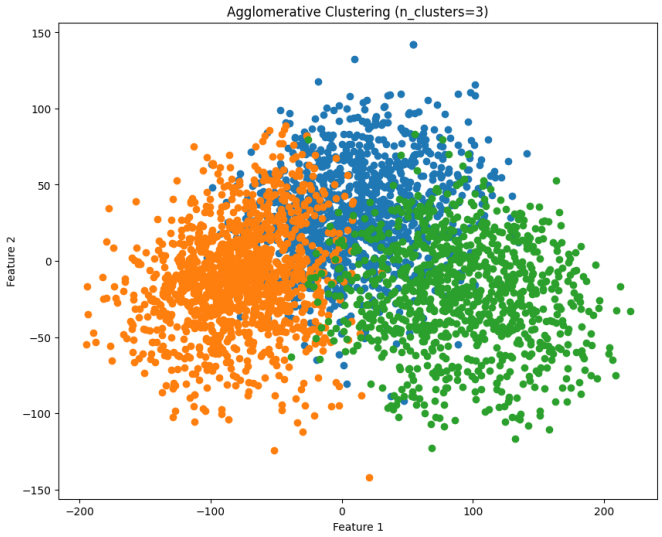


Figure 45: Agglomerative clustering with $k = 3$

Like the 2 previous parts, we continue our investigation with more clusters. For $k = 6$ we have:

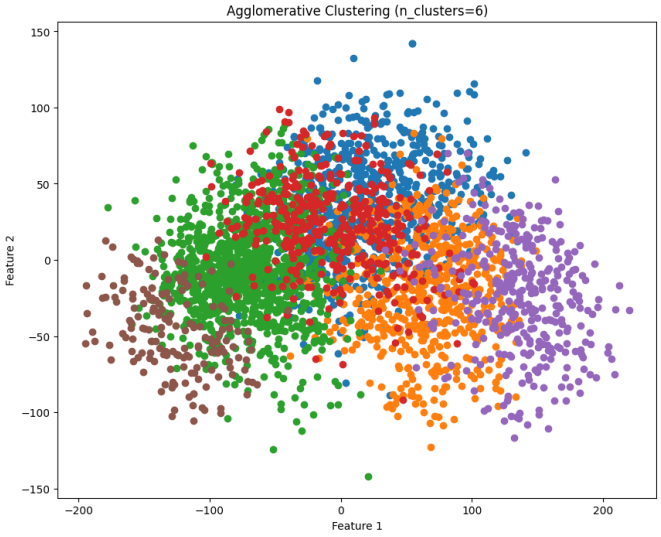


Figure 46: Agglomerative clustering with $k = 6$

With $k = 9$, we can obtain the following result.

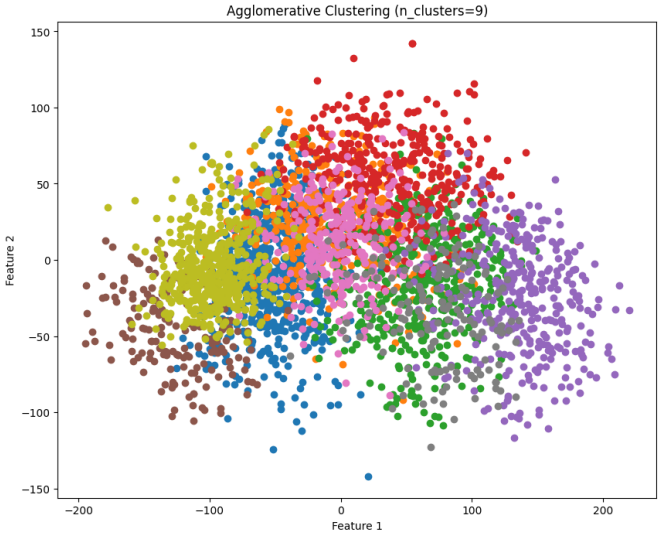


Figure 47: Agglomerative clustering with $k = 9$

And finally, with $k = 50$, we have this colorful picture

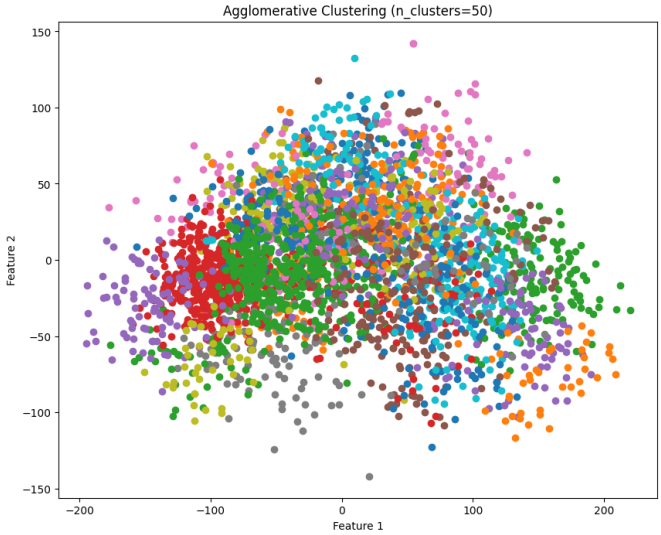


Figure 48: Agglomerative clustering with $k = 50$

Features	Silhouette Score	Calinski-Harabasz Score	Davies-Bouldin Score
$k = 2$ components	0.057	224.72	3.74
$k = 3$ components	0.043	159.65	4.81
$k = 6$ components	0.015	83.08	5.44
$k = 9$ components	0.006	58.22	6.49
$k = 50$ components	-0.0366	14.27	5.05

Table 7: Result Table of Agglomerative Clustering

```
------------------------------------------------------------------------
completeness score of Agglomerative Clustering with K = 2        0.0017
------------------------------------------------------------------------
```

Agglomerative Clustering yields results that are comparable to those of K-means and GMM, with the exception of $k = 3$. Notably, for $k = 3$, Agglomerative Clustering outperforms the other methods, and its outcome aligns more closely with Figure 33.b, which represents the natural state of data points.

4.2.4 LDA before Clustering

In this section, we employed LDA instead of PCA for clustering purposes. Notably, we observed a high level of similarity in the completeness score between the two methods, indicating their comparable performance.

Methods	Completeness Score
K-NN $k = 2$	0.9138
GMM $n = 2$	0.9120
Agglomerative Clustering $k = 2$	0.8712

Table 8: Result Table of LDA before Reduction

Let's examine our remarkable results. In our analysis, we initially applied PCA (Principal Component Analysis) to reduce the dimensionality of the dataset to two features. However, when we performed clustering using these two PCA-transformed features, we obtained a very poor completeness score, approximately 0.01, for all clustering methods.

This result can be attributed to the limitations of PCA in capturing class-specific information. PCA focuses solely on maximizing variance and may not preserve the discriminative information necessary for clustering. As a result, the clusters formed using only the PCA-transformed features were not able to capture the complete information about the underlying classes in the dataset.

To address this issue, we further utilized LDA (Linear Discriminant Analysis) after the PCA transformation. LDA aims to find linear projections that maximize class separability, making it more suitable for clustering tasks. By applying LDA after PCA, we were able to incorporate class-specific information into the clustering process.

Remarkably, after incorporating LDA, the completeness score significantly improved to 0.92. This substantial improvement indicates that the combination of PCA and LDA was effective in capturing the underlying class structure and achieving more accurate clustering results.

This result highlights the importance of utilizing techniques that consider class discriminability, such as LDA, when performing clustering tasks. By combining the dimensionality reduction capability of PCA with the discriminative power of LDA, we were able to overcome the limitations of PCA and achieve higher clustering performance.

These findings demonstrate the significance of selecting appropriate feature extraction methods and highlight the potential benefits of incorporating LDA into the clustering pipeline, particularly when working with high-dimensional datasets.

5 Attached Files

The code details of this project are attached to this report.

- *feature_extraction_data_analysis.ipynb*: In this file, suitable plots have been included regarding the features, data distribution, preprocessing, and data cleaning tasks.
- *Classification.ipynb*: In this file, model designs, appropriate plots, and their comparisons related to classification have been included.
- *Clustering.ipynb*: In this file, model designs, appropriate plots, and their comparisons related to clustering have been included.

References

[1] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," 2022.

[2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022.

[3] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893, 2005.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[6] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," Berlin, Heidelberg, pp. 404–417, 2006.

[7] G. Tang, L. Sun, X. Mao, S. Guo, H. Zhang, and X. Wang, "Detection of gan-synthesized image based on discrete wavelet transform," p. 5511435, 2021.