Tehran Institute for Advanced Studies

Department of Computer Engineering

Image Classification and Clustering

by

Armin Tourajmehr

Emran Shahbazi

Project report for Machine Learning course

Spring, 2023

Contents

1 Problem Statement

The objective of this project is to perform feature extraction, classification, and clustering on a dataset comprising both real and fake images collected from different classes of mountains, seas, and jungles. The dataset is obtained by gathering 30 real images and 30 fake images which are produced by generative models such as stable diffusion, midjourney, and dall.E. The collected data is consolidated from multiple students.

The project entails three main tasks: feature extraction, image classification, and image clustering. Firstly, feature extraction will be performed on the collected images to identify and extract relevant characteristics and patterns. This process aims to capture the distinctive features that differentiate the various classes of mountain, sea, and jungle images, as well as the discrepancies between real and fake images within each class.

Once the features are extracted, the project will proceed to image classification. The goal here is to develop a classification model that can accurately distinguish between the different classes of images (mountain, sea, and jungle), based on the extracted features. This task involves training a machine learning or deep learning algorithm on the labeled dataset to create a predictive model capable of assigning new images to their appropriate class.

Lastly, the project involves clustering the images. Clustering is an unsupervised learning technique that aims to group similar images together based on their intrinsic characteristics. By applying clustering algorithms to the collected dataset, the project aims to discover patterns and similarities within and across different classes. This step can provide valuable insights into the relationships and similarities among the real and fake images within each class and potentially uncover any anomalies or outliers.

The successful completion of these tasks will contribute to a comprehensive analysis of the collected image dataset. It will provide valuable knowledge and insights into the distinguishing features, classification accuracy, and underlying patterns present in the dataset, thereby enhancing our understanding of image classification and clustering techniques in the context of real and fake images from different natural environments.

2 Pre Processing

Preprocessing is an important step in image processing and computer vision tasks. It involves applying various techniques to enhance the quality of images, remove noise, normalize pixel values, and prepare the data for further analysis or feature extraction. Here are some common preprocessing techniques used for images:

1. Resizing: Images may need to be resized to a specific resolution to standardize their dimensions or to match the requirements of a particular algorithm or application. Resizing can be done by interpolation techniques such as nearest neighbor, bilinear, or bicubic.

2. Grayscale Conversion: In some cases, color information may not be necessary or could introduce unnecessary complexity. Converting images to grayscale simplifies the data representation by reducing the image to a single channel. This can be achieved by taking the average of the RGB channels or using specific weighted averages.

3. Cropping: Cropping involves removing unwanted parts of an image to focus on the region of interest. It can help eliminate irrelevant background or noise and highlight the important features or objects within the image.

4. Denoising: Images captured in real-world scenarios may contain noise or artifacts due to various factors such as low light conditions, compression, or sensor limitations. Denoising techniques such as Gaussian blur, median filtering, or wavelet denoising can be applied to reduce noise and improve the overall quality of the image.

5. Contrast Enhancement: Adjusting the contrast of an image can improve the visibility of details and make the features more distinguishable. Techniques like histogram equalization, contrast stretching, or adaptive histogram equalization can be used to enhance the contrast.

6. Normalization: Pixel normalization ensures that the pixel values are scaled to a specific range or distribution. Common normalization techniques include min-max normalization (scaling pixel values between 0 and 1), z-score normalization (subtracting mean and dividing by standard deviation), or mapping pixel values to a specific range (e.g., 0 to 255).

7. Gamma Correction: Gamma correction is used to adjust the overall brightness of an image to match human perception. It corrects the nonlinear relationship between pixel intensities and

the actual brightness. Applying a gamma correction curve can enhance the visibility of details in darker or brighter regions of an image.

8. Data Augmentation: In machine learning tasks, data augmentation techniques are often applied to increase the diversity and size of the training dataset. These techniques include random rotations, flips, translations, or adding random noise to the images. Data augmentation can help improve the generalization and robustness of machine learning models.

These preprocessing techniques are often applied in combination or based on the specific requirements of the task at hand. The goal is to improve the quality, remove noise, and normalize the images to make them suitable for subsequent analysis, feature extraction, or machine learning algorithms.

Preprocessing is used for images to achieve several important goals:

1. Noise Reduction: Images captured in real-world scenarios often contain noise or artifacts due to various factors such as low light conditions, sensor limitations, or compression. Preprocessing techniques such as denoising filters can help remove or reduce noise, improving the clarity and quality of the image.

2. Data Normalization: Images can have varying pixel value ranges or distributions. Normalizing the pixel values ensures consistency and comparability across different images. By scaling the pixel values to a specific range or distribution, such as between 0 and 1 or using z-scores, preprocessing facilitates proper data handling and analysis.

3. Standardization: Preprocessing techniques like resizing and cropping are used to standardize the dimensions and aspect ratios of images. This is important when working with datasets that require consistent image sizes for training machine learning models or applying algorithms that have specific input requirements.

4. Feature Extraction: Preprocessing plays a crucial role in feature extraction. By removing unnecessary noise, enhancing important details, and standardizing the images, preprocessing techniques enable more accurate and reliable feature extraction algorithms. These algorithms can then capture and represent meaningful information for tasks such as object recognition, edge detection, or texture analysis.

Overall, preprocessing is essential to improve the quality, standardization, and suitability of

images for various computer vision tasks. It helps enhance the information content of images, reduces noise, and prepares the data for effective analysis, feature extraction, and machine learning algorithms.

For the pre-processing part, we resized the photos and made them in 224 x 224 pixels format. The reason for this was that we need photos of 224 x 224 size for resnet input.

We also check if the image is P or RGBA, it means that the image is transparent or in palette mode. In such cases, the code converts the image to RGB using image.convert('RGB').

The purpose of this conversion is to ensure that the image is in standard RGB format. Some image processing operations or libraries may require images to be in RGB mode to work properly. By converting the image to RGB, any transparency or palette information is removed, and the resulting image is a full-color RGB image.

This step is included to handle specific image modes and ensure consistent processing across different image types.

Finally we store all images in JPG format in file that is called 'resized'.

3 Feature Extraction

Feature extraction is a critical step in image processing and computer vision tasks. It involves extracting meaningful and informative characteristics or patterns from images to represent them in a more compact and representative form. These extracted features can then be used for various applications such as image classification, object detection, image retrieval, and more.

The goal of feature extraction is to capture the essential visual information present in the images while reducing the dimensionality of the data. By representing images with a set of relevant features, we can effectively analyze and compare them, enabling algorithms and models to make accurate predictions or decisions.

There are various methods for feature extraction with images. Here are a few commonly used techniques:

1. Pixel-based features: These methods directly operate on the pixel values of images. They can include basic statistics like mean, variance, or histogram-based representations of color or intensity distributions.

2. Local feature descriptors: These techniques focus on extracting distinctive and repeatable features from local image regions. Examples include Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), and Oriented FAST and Rotated BRIEF (ORB). These algorithms detect and describe keypoints or interest points in the image, capturing information about local textures, corners, or edges.

3. Deep learning-based features: With the emergence of deep learning, convolutional neural networks (CNNs) have become a powerful tool for feature extraction. CNNs automatically learn hierarchical representations from images, starting with low-level features (edges, textures) and progressing to high-level features (objects, scenes). Features can be extracted from intermediate layers of pre-trained CNN models or by fine-tuning the network for specific tasks.

4. Histogram of Oriented Gradients (HOG): HOG is a popular method for extracting shape and edge features from images. It computes the distribution of gradient orientations in local image regions, capturing shape information that can be useful for object detection and recognition tasks.

These are just a few examples of feature extraction techniques. The choice of method depends on the specific task, the nature of the images, and the available computational resources. Ultimately, the goal is to represent images in a way that captures the most discriminative and relevant information for subsequent analysis and decision-making processes.

We used HOG algorithm for feature extraction in our project.

explanation of the Histogram of Oriented Gradients (HOG) feature extraction algorithm:

1. Image Preprocessing: - Convert the input image to grayscale: This is often done to simplify the computation and remove color information that may not be relevant for certain tasks. - Apply normalization or enhancement techniques: This can include methods such as gamma correction, histogram equalization, or contrast normalization. These techniques help improve the robustness of the feature extraction process.

2. Gradient Computation: - Compute the gradients of the grayscale image: The gradients capture the local intensity variations in the image. The most common approach is to calculate the gradients using the Sobel operator, which estimates the intensity changes in the horizontal and vertical directions.

3. Cell Formation: - Divide the image into small cells: Cells are square or rectangular regions that cover the entire image. The size of the cells determines the spatial resolution of the computed features. Smaller cell sizes capture fine details but may increase computational complexity.

4. Orientation Binning: - For each pixel within a cell, calculate the gradient orientation: This is done by considering the gradient direction and magnitude at that pixel. Common methods include computing the gradient angle using the arctangent function and the gradient magnitude using the Euclidean norm. - Bin the gradient orientations: The gradient orientations are divided into predefined angular bins. For example, the range of 0-180 degrees can be divided into multiple bins, such as 9 or 12. Each pixel's gradient orientation is assigned to the corresponding bin based on its angle.

5. Histogram Calculation: - Construct a histogram for each cell: Within each cell, a histogram is created by accumulating the gradient orientations into the corresponding bins. Each bin represents the frequency or strength of gradients in that particular orientation within the cell. The histogram summarizes the distribution of gradient orientations in the cell.

6. Block Normalization: - Group adjacent cells to form blocks: This helps capture spatial information and improve robustness against local variations. - Concatenate histograms from cells within each block: The histograms from the cells within a block are concatenated to form a feature vector for that block. The block can overlap with neighboring blocks to capture information from overlapping regions. - Perform normalization on the concatenated histogram: To enhance the robustness against variations in lighting conditions and contrast changes, normalization techniques are applied to the concatenated histogram. Common normalization methods include L1 or L2 normalization, which normalize the histogram values to have a unit sum or a specific norm.

7. Feature Vector: - Concatenate the normalized block histograms: The resulting normalized block histograms are concatenated to form the final feature vector for the image. This feature vector represents the distribution of local gradient orientations and magnitudes across different regions of the image. The length of the feature vector depends on the number of cells, blocks, and the size of the histogram bins used in the process.

8.Classification: -The extracted feature vector can be fed into a machine learning algorithm (e.g., SVM, neural network) for object detection or recognition tasks. The classifier is trained on a dataset with labeled examples to learn the patterns and discriminate between different objects or classes.

The resulting HOG feature vector can be used as input to machine learning algorithms or other computer vision techniques for tasks such as object detection, classification, or recognition. By capturing the distribution of gradients in different orientations, HOG is able to represent shape and edge information in images, making it effective for various visual recognition tasks.

In out code, every feature will be normalized then append to a list that. the reason we do normalize is that the normalization of feature values is done to ensure that the features have a consistent scale and distribution. By subtracting the mean and dividing by the standard deviation of the feature values, the resulting features have a mean of zero and a standard deviation of one. This normalization is important for various machine learning algorithms as it helps in equalizing the importance of different features and prevents features with larger values from dominating the learning process. It also helps in handling differences in the range and distribution of feature values across different samples or datasets. At the end we store all features into csv file.

3.1 SIFT

The SIFT (Scale-Invariant Feature Transform) algorithm is a widely used technique in computer vision for extracting distinctive features from images. It was introduced by David Lowe in 1999 and has since become a fundamental tool for various applications like image matching, object recognition, and 3D reconstruction.

The key idea behind SIFT is to find invariant features that are robust to changes in scale, rotation, and affine transformations. These features can be used to match and align images despite differences in viewpoint or appearance.

Here's an overview of the main steps involved in the SIFT algorithm:

1. Scale-space extrema detection: The algorithm begins by constructing a scale-space representation of the input image. This is achieved by convolving the image with Gaussian filters at different scales. The scale-space representation allows for the detection of features

at multiple scales. At each scale, the algorithm identifies potential interest points by finding extrema in the difference of Gaussian (DoG) images.

2. Keypoint localization: Once the potential interest points are identified, the algorithm refines their locations to improve their accuracy. It eliminates points with low contrast or points that are poorly localized along edges.

3. Orientation assignment: SIFT computes the dominant orientation for each keypoint. This step is important for achieving invariance to image rotation. A local histogram of gradient orientations is created in the region around each keypoint, and the highest peak in the histogram corresponds to the dominant orientation.

4. Keypoint descriptor computation: The algorithm constructs a descriptor for each keypoint to capture its local appearance. The descriptor is computed based on the gradient magnitude and orientation in the surrounding region. SIFT uses a 16x16 grid around the keypoint and creates a histogram of gradient orientations in each subregion. The final descriptor is a vector concatenating all the histogram values.

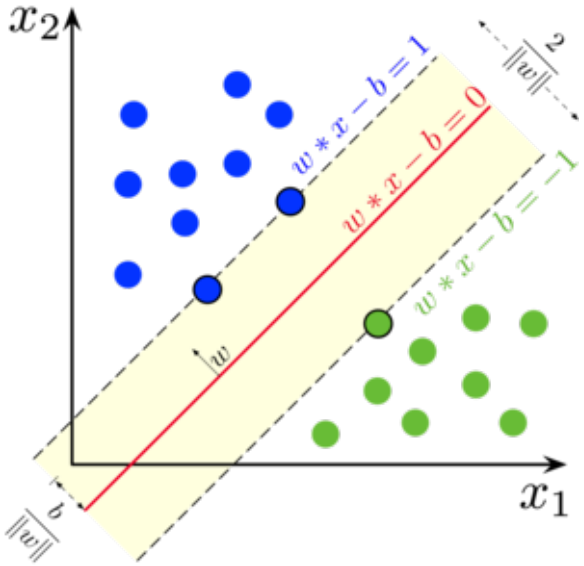
5. Feature matching: With the descriptors computed for keypoints in multiple images, the SIFT algorithm can perform feature matching. It compares the descriptors of keypoints between different images and establishes correspondences based on similarity measures like Euclidean distance or the ratio of the nearest neighbor distance to the second nearest neighbor distance.

6. Robustness and filtering: To improve robustness against outliers and mismatches, additional steps like RANSAC (Random Sample Consensus) can be used to estimate and refine geometric transformations between matched keypoints.

The SIFT algorithm has proven to be effective in various computer vision tasks due to its ability to extract and match distinctive features invariant to scale, rotation, and affine transformations. It provides a reliable way to find correspondences between images and has been widely adopted in applications such as image stitching, object recognition, and image retrieval.

4 Classification

4.1 SVM



Support Vector Machines (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. SVMs are particularly effective in handling high-dimensional data and can provide good generalization performance. Here's a comprehensive explanation of SVM classification:

Basic Concept:

SVM aims to find an optimal hyperplane that separates different classes in the feature space. In a binary classification scenario, the hyperplane is a decision boundary that maximizes the margin between the closest data points of different classes. Support vectors are the data points that lie on the margin or are misclassified. They play a crucial role in defining the decision boundary.

Linear SVM:

Linear SVM constructs a linear decision boundary in the feature space. The decision bound-

ary is represented by a hyperplane defined by a weight vector and a bias term. The optimization objective is to maximize the margin while minimizing the classification error.
Kernel Trick and Non-Linear SVM:

Linear SVM may not be sufficient when the classes are not linearly separable in the original feature space. The kernel trick allows SVM to implicitly map the data into a higher-dimensional space, where linear separation might be possible. Commonly used kernel functions include the linear kernel, polynomial kernel, Gaussian RBF kernel, and sigmoid kernel. The choice of the kernel depends on the characteristics of the data and the desired decision boundary shape.
SVM Training:

The training process involves finding the optimal hyperplane parameters that minimize a loss function while maximizing the margin. The loss function captures the trade-off between maximizing the margin and minimizing the misclassification errors. The optimization is often formulated as a quadratic programming problem, which can be solved using optimization techniques such as the Sequential Minimal Optimization (SMO) algorithm.
Regularization and Soft Margin:

SVM supports the concept of regularization, which controls the balance between maximizing the margin and allowing some misclassifications. In the case of a perfectly separable dataset, a hard margin SVM enforces strict separation without misclassifications. Soft margin SVM allows for a certain degree of misclassification, using a penalty parameter (C) to control the trade-off between margin maximization and error tolerance.
Multiclass Classification:

SVM is inherently a binary classifier, but various strategies exist to extend it to multiclass problems. One-versus-One (OvO) classification trains a separate SVM for each pair of classes and uses voting or ranking to determine the final class. One-versus-All (OvA) classification trains one SVM for each class to distinguish it from the rest.

Model Evaluation and Prediction:

SVM models are evaluated using metrics such as accuracy, precision, recall, F1 score, or area under the Receiver Operating Characteristic (ROC) curve. To make predictions on unseen data, SVM uses the learned hyperplane to classify new instances based on their feature representations.

Strengths and Weaknesses:

SVMs are effective in high-dimensional spaces, even when the number of features is greater than the number of samples. They can handle both linearly separable and non-linearly separable data through the use of appropriate kernels. SVMs are less affected by overfitting due to the margin maximization objective. However, SVMs can be sensitive to noise and outliers, and their performance may degrade with large datasets or imbalanced class distributions. The selection of kernel and regularization parameters requires careful tuning, which can be time-consuming. SVMs have been widely used for various applications, including text categorization, image classification, bioinformatics, and anomaly detection. Their ability to handle complex decision boundaries and their theoretical foundations make them a popular choice in many machine learning tasks.

Support Vector Machines (SVM) are often used in combination with Histogram of Oriented Gradients (HOG) for feature extraction and classification tasks. Here are some reasons why SVM is useful for HOG features and feature extraction:

1. Non-linear Feature Space: HOG features capture local gradient orientations and magnitudes, which are essential for representing shape and edge information in an image. However, the relationship between these features and the class labels may not be linear in the original feature space. SVM with the kernel trick allows for non-linear mappings of the features, enabling the SVM to find non-linear decision boundaries in the higher-dimensional space.

2. Handling High-Dimensional Data: HOG features can result in high-dimensional feature

vectors, especially when considering multiple orientations, cells, and blocks. SVM can handle high-dimensional data efficiently and effectively by using support vectors that lie on or near the decision boundary. The decision boundary is defined by a subset of the training data, which reduces the computational burden associated with high-dimensional feature spaces.

3. Robustness to Outliers: SVM is less sensitive to outliers compared to some other classification algorithms. The use of a margin in SVM allows for a more robust decision boundary. Outliers that fall within the margin or are misclassified are considered as support vectors and have a higher influence on the decision boundary. This robustness helps to reduce the impact of noisy or mislabeled training data on the classification performance.

4. Margin Maximization: The objective of SVM is to maximize the margin between the decision boundary and the support vectors. This margin maximization process results in a wider separation between different classes, leading to better generalization to unseen data. By maximizing the margin, SVM tends to provide a good balance between overfitting and underfitting, which helps in achieving good classification performance.

5. Effective Binary Classification: SVM is a binary classifier by nature. HOG features are often used as input to SVM for binary classification tasks, where the goal is to separate two classes. SVM's ability to find an optimal hyperplane with maximal margin between classes makes it well-suited for binary classification scenarios.

6. Flexibility with Kernel Functions: SVM allows the use of different kernel functions, such as the linear, polynomial, Gaussian RBF, or sigmoid kernels. This flexibility enables the SVM to capture complex decision boundaries and adapt to different data distributions. The choice of the kernel function depends on the specific problem and the underlying characteristics of the data.

Overall, SVM complements the feature extraction capabilities of HOG by providing a robust and effective classification framework. By leveraging SVM's ability to handle non-linear rela-

tionships, high-dimensional data, and robustness to outliers, HOG features can be effectively utilized for tasks such as object detection, pedestrian recognition, and image classification.

we use SVM for classification and get accuracy, f1-score, recall, precision too. First we explain what they are metrics and after that show our result.

Here are explanations of commonly used evaluation metrics in classification tasks:

1. Accuracy:

- Accuracy measures the overall correctness of the predictions made by a classifier.
- It is calculated as the ratio of the number of correct predictions to the total number of predictions.
- Formula: Accuracy = (TP + TN) / (TP + TN + FP + FN)
- TP: True Positives (correctly predicted positive instances)
- TN: True Negatives (correctly predicted negative instances)
- FP: False Positives (incorrectly predicted positive instances)
- FN: False Negatives (incorrectly predicted negative instances)
- Accuracy provides a general measure of how well the classifier performs, but it may not be suitable in cases of imbalanced class distributions.

2. Precision:

- Precision measures the ability of a classifier to correctly identify positive instances.
- It is calculated as the ratio of true positives to the sum of true positives and false positives.
- Formula: Precision = TP / (TP + FP)
- Precision focuses on minimizing false positives, making it useful when the cost of false positives is high.

3. Recall (Sensitivity or True Positive Rate):

- Recall measures the ability of a classifier to correctly identify positive instances out of all the actual positive instances.
- It is calculated as the ratio of true positives to the sum of true positives and false negatives.

- Formula: Recall = TP / (TP + FN)
- Recall focuses on minimizing false negatives, making it useful when the cost of missing positive instances is high.

4. F1-Score:

- The F1-score is the harmonic mean of precision and recall, providing a balanced measure of a classifier's performance.
- It combines both precision and recall into a single metric.
- Formula: F1-Score = 2 * (Precision * Recall) / (Precision + Recall)
- The F1-score considers both false positives and false negatives and is useful when both precision and recall are important.

5. Error Rate:

- The error rate, also known as the misclassification rate, measures the proportion of misclassified instances.
- It is calculated as the ratio of false positives and false negatives to the total number of instances.
- Formula: Error Rate = (FP + FN) / (TP + TN + FP + FN)
- The error rate provides a measure of the overall classification error made by the classifier.

These evaluation metrics provide insights into the performance of a classification model. Accuracy, precision, recall, and the F1-score are commonly used to assess the classifier's effectiveness, while the error rate provides a measure of the misclassification rate. The appropriate metric to use depends on the specific problem and the relative importance of false positives and false negatives in the given context.

4.2 AdaBoost

Certainly! Here's an explanation of the AdaBoost algorithm, including its mathematical formulation and how it works with images:

The AdaBoost algorithm, short for Adaptive Boosting, is a popular machine learning algorithm that combines multiple weak classifiers to create a strong ensemble classifier. It is particularly effective in handling complex classification problems.

Mathematical Formulation:

Let's consider a binary classification problem, where we have a dataset with labeled examples. Each example consists of a set of features (denoted as X) and a corresponding class label (denoted as y).

The AdaBoost algorithm assigns weights (denoted as w) to each training example. Initially, all examples have equal weights. The algorithm iteratively trains weak classifiers on the weighted data, with each weak classifier attempting to minimize the classification error on the weighted examples.

At each iteration, the weak classifier is selected based on its performance in classifying the weighted examples. The classifier's performance is evaluated using an error measure (e.g., misclassification error or weighted error rate). The algorithm gives higher importance to the misclassified examples by increasing their weights, and then adjusts the weights of the subsequent weak classifiers accordingly.

The final strong classifier is obtained by combining the weighted predictions of the weak classifiers. The weight assigned to each weak classifier's prediction is determined based on its performance during training.

Working with Images:

In the context of image classification using AdaBoost, the algorithm operates on image features extracted from the dataset. These features can be obtained using various techniques, such as SIFT, which identifies key points in an image and generates descriptors to represent those points.

The AdaBoost algorithm iteratively selects weak classifiers that focus on specific image features or patterns. For example, a weak classifier might detect certain edges, textures, or shapes. During training, the weak classifiers learn to recognize these features and their importance based on the weighted examples.

As the algorithm progresses, it assigns higher weights to misclassified images, emphasizing

their importance in subsequent iterations. This adaptability allows AdaBoost to focus on difficult-to-classify images and learn from its mistakes, gradually improving its performance.

Ultimately, the ensemble of weak classifiers combines their individual predictions using weighted voting to create a strong classifier that offers enhanced accuracy in image classification tasks.

It's important to note that while AdaBoost is a powerful algorithm, its full implementation and optimization can be complex. The algorithm's mathematical formulation and implementation details may vary depending on the specific library or framework being used.

4.3 SVM Result

In the context of using Support Vector Machines (SVM) for feature extraction with Histogram of Oriented Gradients (HOG), the commonly used evaluation metrics (precision, recall, F1-score, accuracy, and error rate) provide insights into the performance of the SVM classifier. Here's what these metrics indicate:

1. Precision:

- Precision measures the proportion of correctly predicted positive instances (e.g., objects of interest) out of all instances predicted as positive.
- In the case of SVM with HOG feature extraction, precision indicates how well the classifier identifies the objects of interest based on the extracted HOG features.
- High precision means that the SVM classifier has a low rate of false positives, i.e., it correctly identifies positive instances and has a low tendency to misclassify negative instances as positive.

2. Recall (Sensitivity or True Positive Rate):

- Recall measures the proportion of correctly predicted positive instances out of all the actual positive instances in the dataset.
- For SVM with HOG feature extraction, recall indicates how well the classifier captures all the positive instances in the data based on the extracted HOG features.
- High recall means that the SVM classifier has a low rate of false negatives, i.e., it correctly identifies most of the positive instances and has a low tendency to miss them.

3. F1-Score:

- The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the classifier's performance.
- It combines both precision and recall into a single metric, giving equal weight to both.
- The F1-score for SVM with HOG feature extraction provides an overall assessment of the classifier's ability to correctly identify positive instances while minimizing false positives and false negatives.

4. Accuracy:

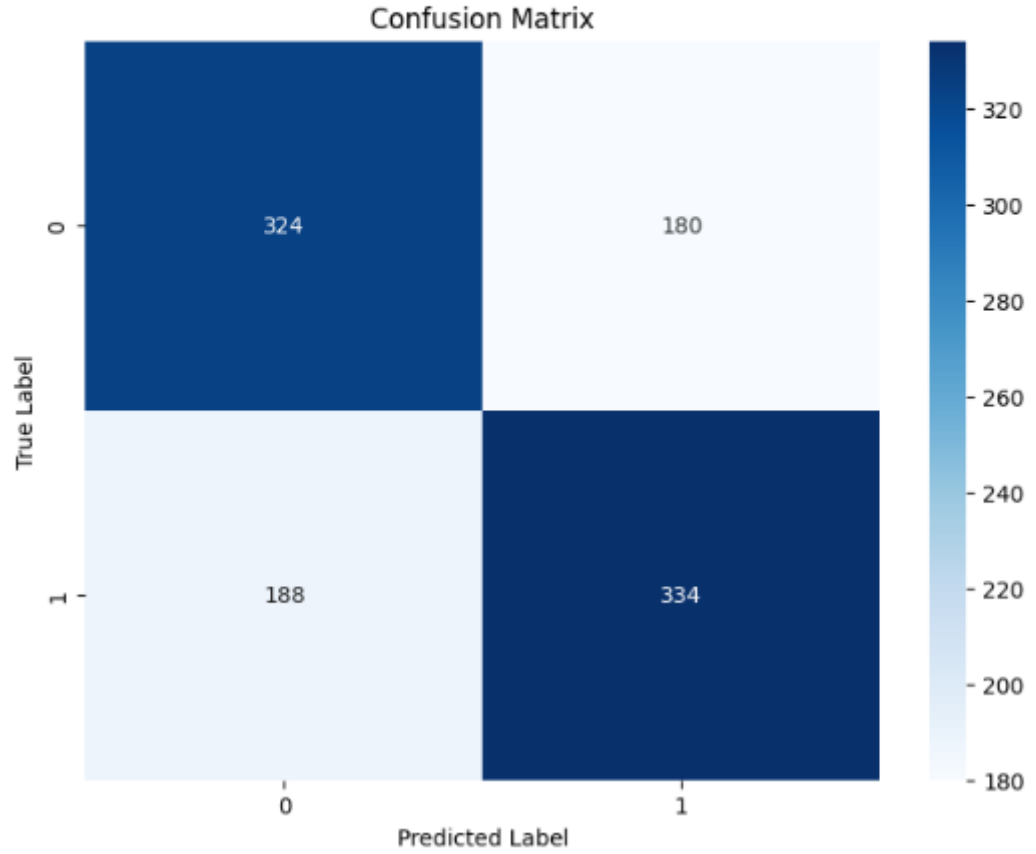
- Accuracy measures the overall correctness of the predictions made by the SVM classifier.
- It indicates the proportion of correctly classified instances, both positive and negative, out of the total instances in the dataset.
- In the context of SVM with HOG feature extraction, accuracy shows how well the SVM classifier can classify instances based on the extracted HOG features.

5. Error Rate:

- The error rate, also known as the misclassification rate, measures the proportion of misclassified instances.
- It represents the overall rate of misclassifications made by the SVM classifier, including false positives and false negatives.
- A lower error rate indicates a higher overall classification accuracy of the SVM classifier with HOG feature extraction.

These evaluation metrics help assess the performance and effectiveness of the SVM classifier using HOG feature extraction.

	precision	recall	f1-score	support
0	0.63	0.64	0.64	504
1	0.65	0.64	0.64	522
accuracy			0.64	1026
macro avg	0.64	0.64	0.64	1026
weighted avg	0.64	0.64	0.64	1026



Based on the provided information, the following evaluation metrics for an SVM classifier using HOG features:

- Accuracy: 64%
- F1-score: 64% (for both classes)
- Recall (Sensitivity): 64% (for both classes)
- Precision: 63% for class 0 and 65% for class 1

Here is a brief analysis of these metrics:

1. Accuracy: The accuracy of 64% indicates that the SVM classifier correctly predicted the class labels for 64% of the instances in the dataset. However, it is essential to consider the specific problem and the distribution of class labels in the dataset. If the classes are imbalanced, accuracy alone may not provide a comprehensive assessment of the classifier's performance.

2. F1-score: The F1-score is the harmonic mean of precision and recall. An F1-score of 64% for both classes suggests a balanced performance in terms of precision and recall. It indicates that the classifier has similar abilities to capture both true positives and avoid false positives for each class.

3. Recall: The recall value of 64% for both classes suggests that the classifier identified approximately 64% of the true positive instances for each class, while missing some positive instances (false negatives). It indicates the classifier's sensitivity in capturing positive instances.

4. Precision: The precision value of 63% for class 0 and 65% for class 1 indicates the proportion of correctly predicted positive instances out of the total instances predicted as positive. Class 1 shows slightly higher precision compared to class 0. Precision focuses on minimizing false positives, so higher precision indicates a lower rate of false positives for the respective classes.

In the confusion matrix for class 0, 324 examples are correctly predicted and 180 examples are predicted wrong. for class 1, 188 examples are predicted wrong and 334 are correctly predicted.

1. ROC Curve:

- The ROC curve is a graphical representation of the classifier's performance across different thresholds for classification.
- The x-axis represents the False Positive Rate (FPR), which is the ratio of false positives to the total number of actual negatives.

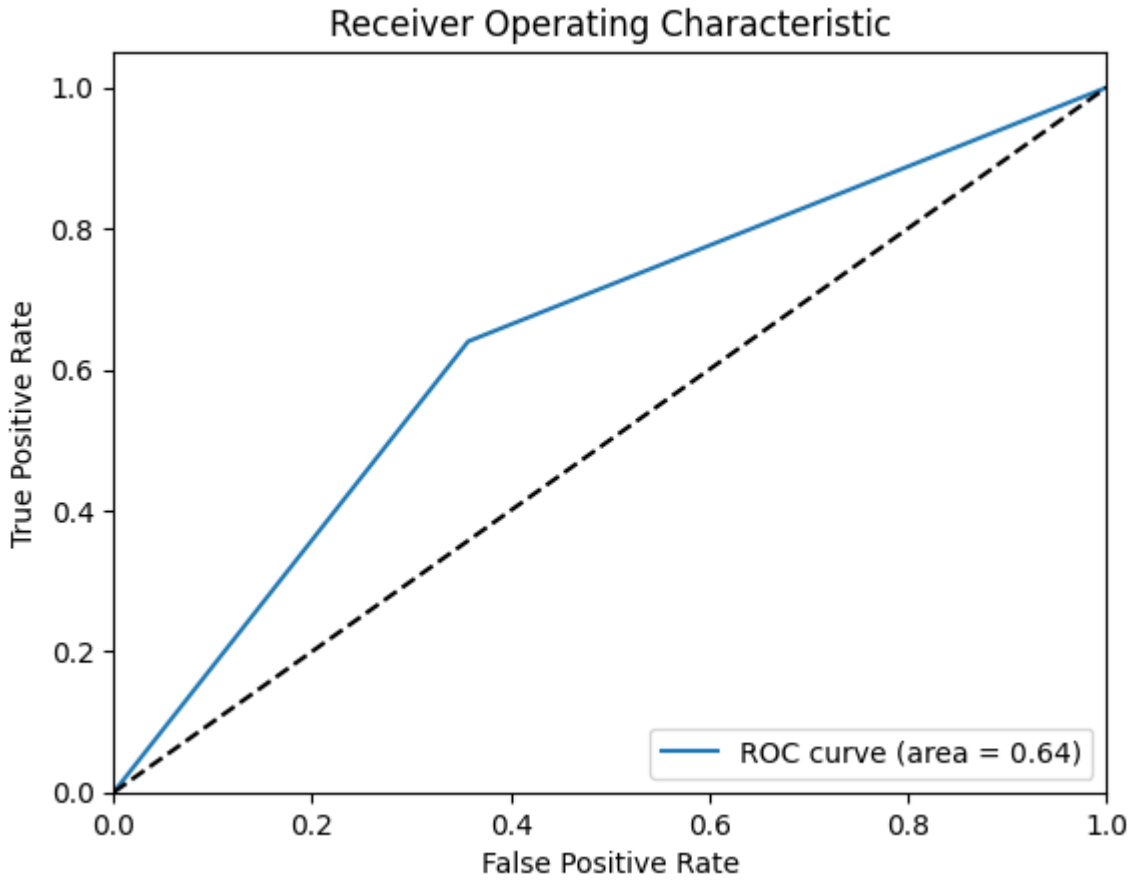
- The y-axis represents the True Positive Rate (TPR), which is the ratio of true positives to the total number of actual positives.
- Each point on the curve corresponds to a specific threshold used to classify instances.
- The curve shows how the TPR and FPR change as the classification threshold varies.

2. AUC (Area Under the Curve):

- The AUC quantifies the overall performance of the classifier represented by the ROC curve.
- It provides a single value that summarizes the classifier's ability to distinguish between positive and negative instances.
- The AUC ranges from 0 to 1, where a higher value indicates better classifier performance.
- An AUC of 1 represents a perfect classifier, while an AUC of 0.5 indicates a random classifier (no better than chance).

To analyze the ROC curve and AUC, consider the following points:

- The closer the ROC curve is to the top-left corner of the plot, the better the classifier's performance.
- A steeper ROC curve indicates a higher TPR for a given FPR, suggesting better performance.
- The AUC provides a single metric to compare different classifiers or different parameter settings for the same classifier.
- An AUC value greater than 0.5 suggests the classifier is better than random guessing.
- An AUC value closer to 1 indicates a strong classifier with high TPR and low FPR.

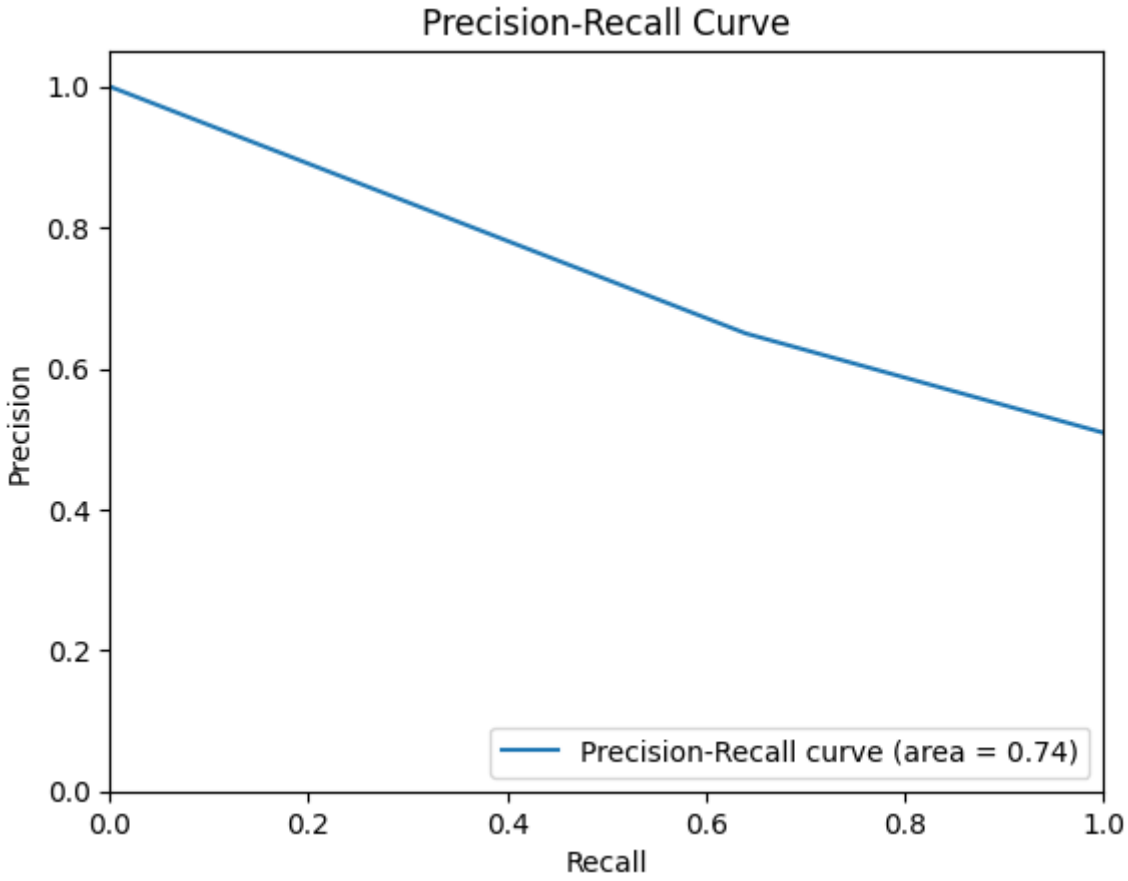


The Precision-Recall curve illustrates the trade-off between precision and recall for different classification thresholds.

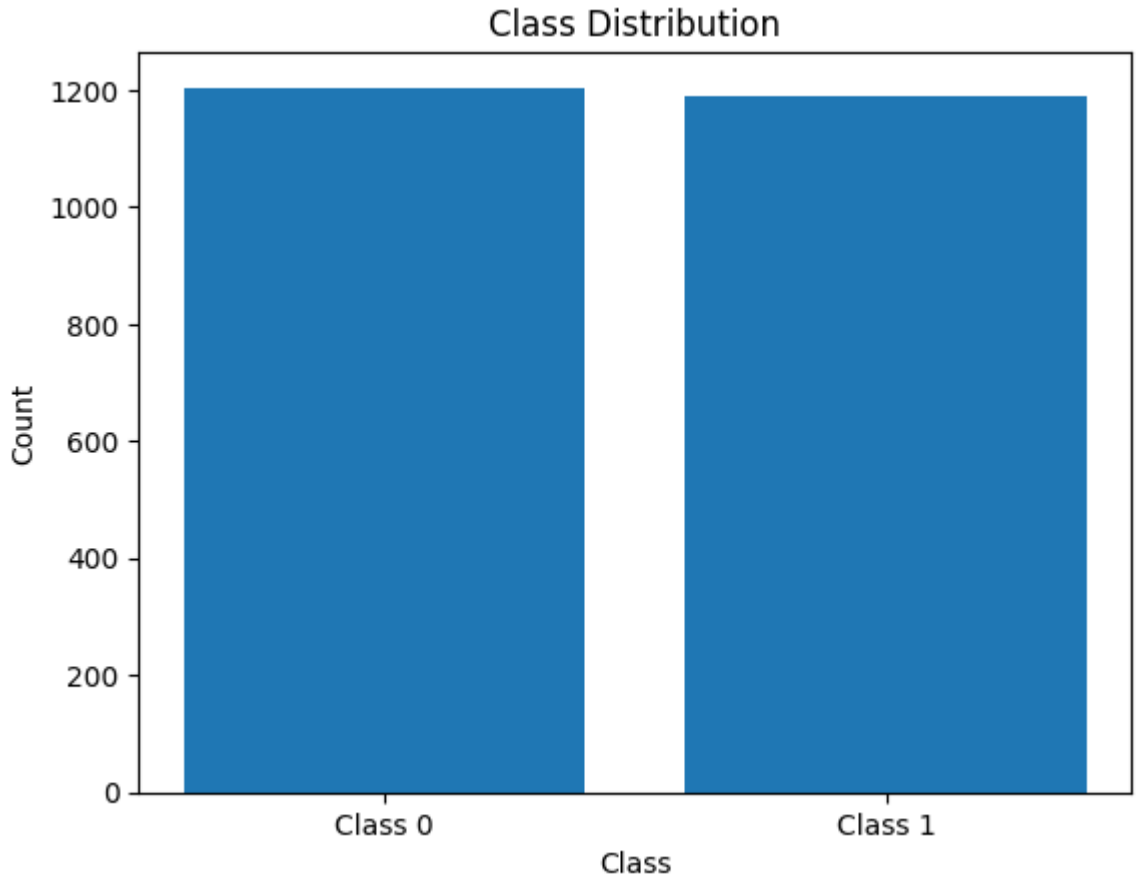
Precision is the ratio of true positives to the sum of true positives and false positives, representing the classifier's ability to correctly classify positive instances.

Recall is the ratio of true positives to the sum of true positives and false negatives, representing the classifier's ability to capture all positive instances.

The Precision-Recall curve visualizes how the precision and recall change as the classification threshold varies.



As you can see, in the best case we have, 46% of the predictions are wrong and 64% are correct.



This is Distribution of data. As you can see, the distribution of class one is a little lower than the distribution of class zero. This is because we had trouble reading a few pictures and didn't use them.

We also used PCA to try to get better accuracy.

Principal Component Analysis (PCA) is a widely used dimensionality reduction technique that aims to transform a high-dimensional dataset into a lower-dimensional representation. PCA achieves this by finding the directions (principal components) in the original feature space along which the data varies the most. Here's a more detailed explanation of PCA:

1. Dimensionality Reduction:

- PCA is used to reduce the number of features (dimensions) in a dataset while retaining as much

information as possible.
- By reducing the dimensions, PCA can simplify the data representation, remove noise, and improve computational efficiency.

2. Variance and Covariance:
- PCA is based on the variance-covariance matrix of the input features.
- Variance measures the spread or variability of a feature, while covariance measures the relationship between pairs of features.
- PCA seeks to find the directions (principal components) along which the data has the highest variance.

3. Principal Components:
- Principal components are new orthogonal axes obtained by linear combinations of the original features.
- The first principal component (PC1) captures the maximum amount of variance in the data.
- Each subsequent principal component captures the remaining variance, orthogonal to the previous components.

4. Explained Variance Ratio:
- The explained variance ratio indicates the proportion of the total variance explained by each principal component.
- It helps determine the amount of information retained by each component.
- The cumulative sum of the explained variance ratios provides an understanding of how much variance is explained as the number of components increases.

5. Data Transformation:
- PCA transforms the original dataset onto the new coordinate system defined by the principal components.
- The transformed data represents the original data projected onto a lower-dimensional subspace

spanned by the selected principal components.

6. Interpretability and Feature Importance:

- PCA does not provide direct interpretations of the original features. Instead, it gives importance to the principal components.
- The coefficients of the original features in the principal components (loadings) can provide insights into the features' contributions to the components.

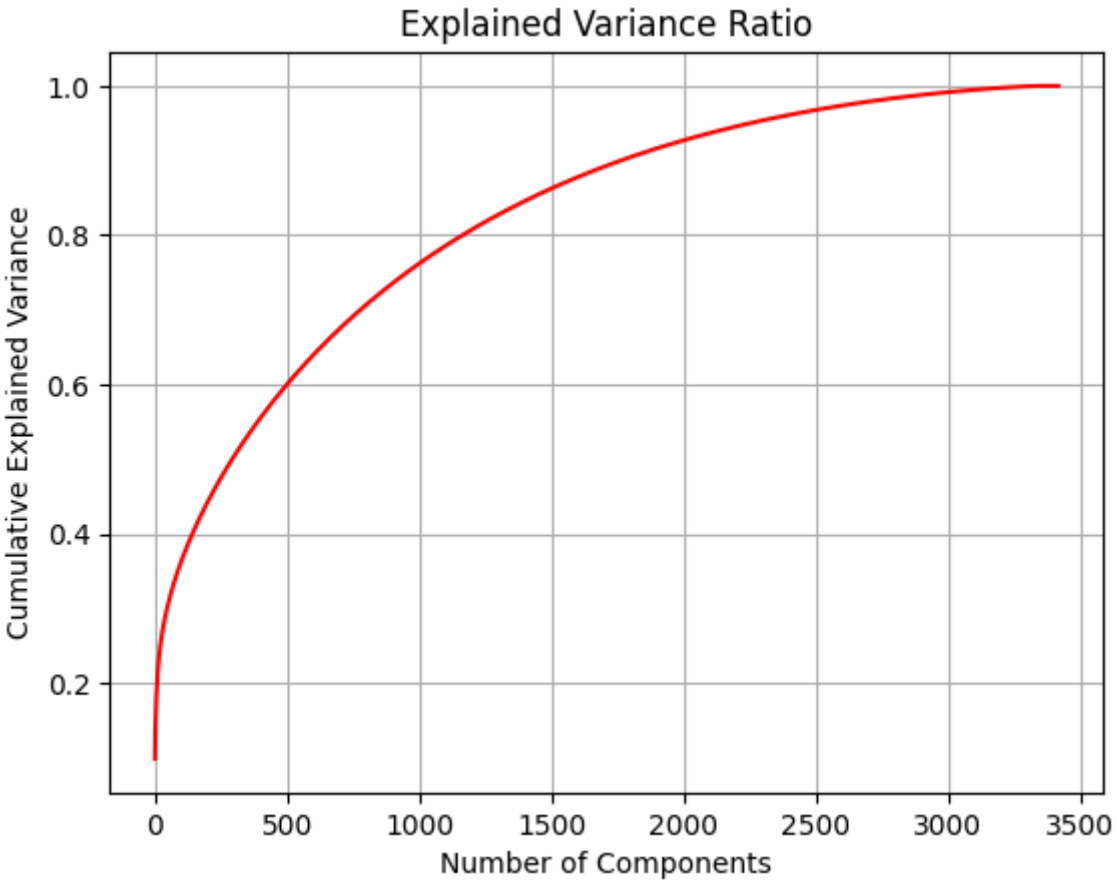
7. Reconstruction:

- PCA allows for data reconstruction by transforming the lower-dimensional representation back into the original feature space.
- The reconstructed data may not be identical to the original data but can capture the most important information.

8. Applications of PCA:

- Dimensionality reduction: PCA can simplify high-dimensional datasets and facilitate data visualization.
- Noise reduction: By discarding components with low variance, PCA can remove noise and improve signal-to-noise ratios.
- Feature extraction: PCA can transform the original features into a set of uncorrelated features, potentially enhancing subsequent analyses.
- Data compression: PCA can compress the data by selecting a subset of the most informative principal components.

PCA is widely used in various fields, including image processing, signal processing, genetics, finance, and recommendation systems. It provides a powerful tool for reducing the dimensionality of complex datasets while retaining essential information. The selection of the number of principal components depends on the desired level of information retention, computational constraints, and the specific requirements of the problem at hand.



The purpose of this code is to analyze the explained variance ratio and cumulative explained variance as the number of components increases. The resulting plot helps determine the optimal number of principal components to retain while still capturing a significant portion of the variance in the original data. The plot provides insights into how much information is retained by each additional principal component and can guide the decision on the number of components to select for dimensionality reduction.

we did for 250,500,750 and 1000 components and got accuracy for each of them

```
Performance Metrics for n_components=250:
Accuracy: 0.6530214424951267
+-------+-----------+--------+----------+---------+
| Class | Precision | Recall | F1-Score | Support |
+-------+-----------+--------+----------+---------+
|   0   |    0.66   |  0.62  |   0.64   |   504   |
|   1   |    0.65   |  0.69  |   0.67   |   522   |
+-------+-----------+--------+----------+---------+
```

```
Performance Metrics for n_components=500:
Accuracy: 0.6500974658869396
+-------+-----------+--------+----------+---------+
| Class | Precision | Recall | F1-Score | Support |
+-------+-----------+--------+----------+---------+
|   0   |    0.65   |  0.63  |   0.64   |   504   |
|   1   |    0.65   |  0.67  |   0.66   |   522   |
+-------+-----------+--------+----------+---------+

Performance Metrics for n_components=750:
Accuracy: 0.6461988304093568
+-------+-----------+--------+----------+---------+
| Class | Precision | Recall | F1-Score | Support |
+-------+-----------+--------+----------+---------+
|   0   |    0.64   |  0.62  |   0.63   |   504   |
|   1   |    0.65   |  0.67  |   0.66   |   522   |
+-------+-----------+--------+----------+---------+

Performance Metrics for n_components=1000:
Accuracy: 0.6559454191033138
+-------+-----------+--------+----------+---------+
| Class | Precision | Recall | F1-Score | Support |
+-------+-----------+--------+----------+---------+
|   0   |    0.65   |  0.64  |   0.65   |   504   |
|   1   |    0.66   |  0.67  |   0.67   |   522   |
+-------+-----------+--------+----------+---------+
```

we realized the best accuracy is for n-components = 1000. we get 65/5% accuracy.

we also do classification for 3 classes. class 0, class 1 and class 3. we use svm with One-vs-One approach for multiclass.

The SVM classifier with the One-vs-One (OvO) approach is a technique used to extend SVM for multiclass classification problems. In a binary classification scenario, SVM constructs a hyperplane to separate two classes. However, when dealing with multiple classes, the OvO approach trains a separate binary SVM classifier for each pair of classes and combines their predictions to determine the final class.

Here's how the OvO approach works:

1. Pairwise Classifiers:

- For N classes, the OvO approach constructs N*(N-1)/2 binary classifiers.
- Each binary classifier is trained on a subset of the data, containing only samples from two specific classes.
- For example, in a 3-class problem, the OvO approach trains three binary classifiers: class 1 vs. class 2, class 1 vs. class 3, and class 2 vs. class 3.

2. Training:

- Each binary classifier is trained using the data from the corresponding two classes.
- The training involves finding the optimal hyperplane that separates the samples of the two classes.
- The optimization objective is to maximize the margin and minimize the classification error for that specific pair of classes.

3. Voting/Ranking:

- To make predictions for a new unseen instance, each binary classifier is applied to the input data.
- Each classifier votes for one of the two classes it was trained on based on the position of the instance relative to its decision boundary.
- The class with the highest number of votes or the highest confidence score is selected as the predicted class for the instance.
- In some cases, the classifiers may assign equal votes to multiple classes. In such cases, a ranking mechanism is used to determine the final class.

4. Performance Evaluation:

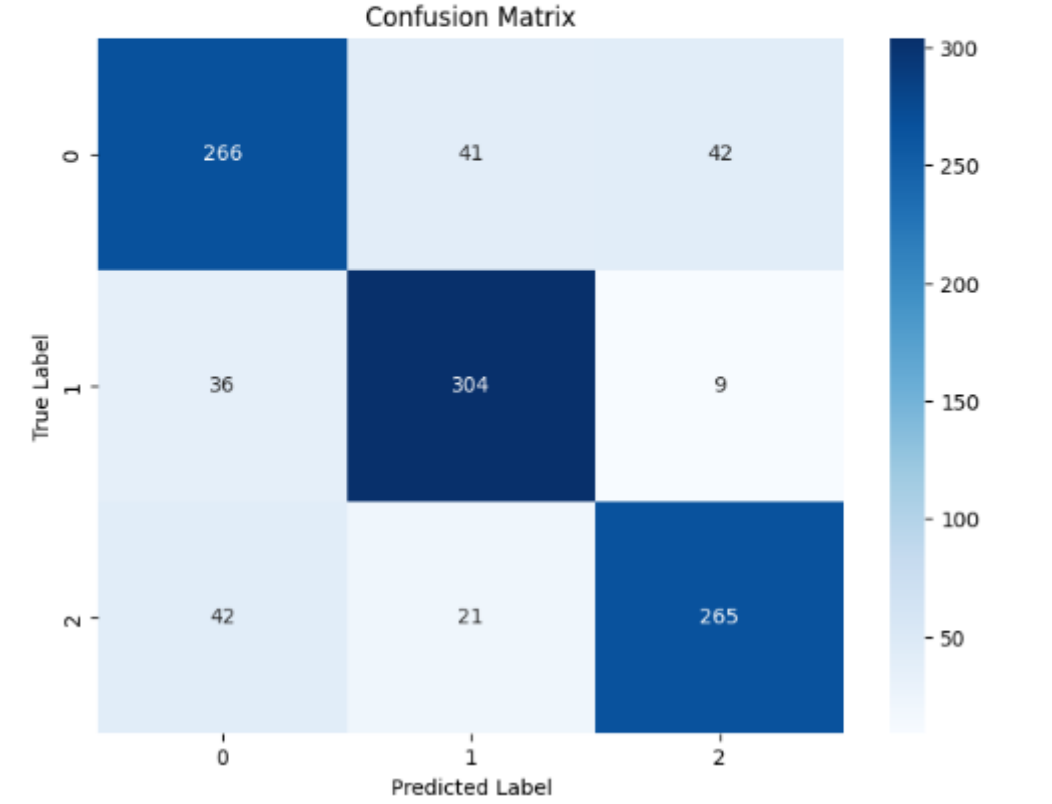
- The performance of the OvO approach is typically evaluated using metrics such as accuracy, precision, recall, or F1-score.
- These metrics can be computed by comparing the predicted class labels with the true class labels of the test instances.

The OvO approach allows SVM to handle multiclass classification problems by breaking them down into multiple binary classification tasks. It offers a flexible and scalable solution, as the number of binary classifiers is proportional to the square of the number of classes. However, it can be computationally expensive for large datasets with a large number of classes. Other approaches, such as the One-vs-All (OvA) approach, which trains one classifier per class, can be more efficient in such cases.

Overall, the SVM classifier with the OvO approach provides a versatile and effective method for solving multiclass classification problems using the binary classification capabilities of SVM.

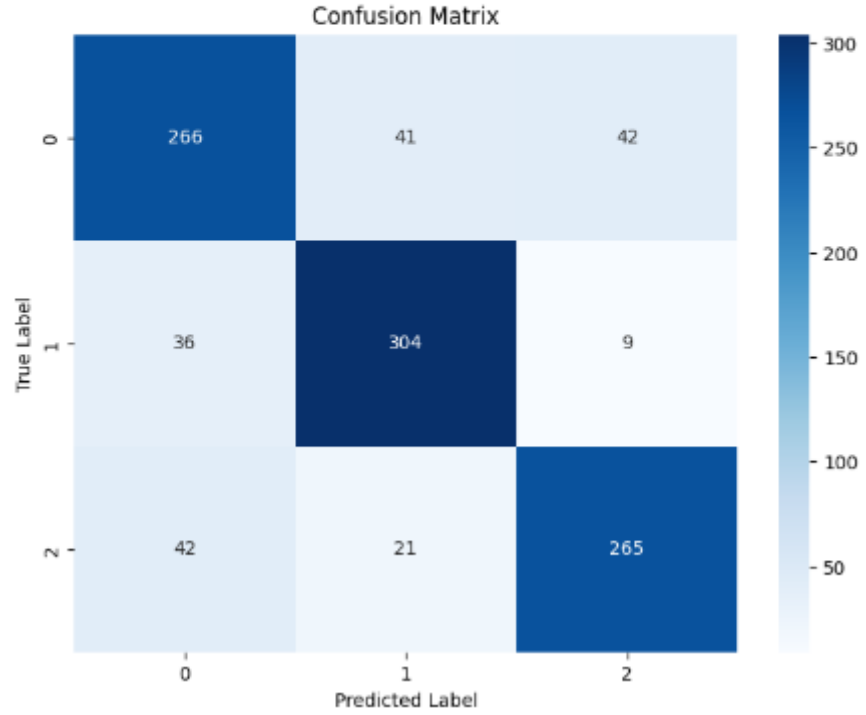
Classification Report:

	precision	recall	f1-score	support
0	0.77	0.76	0.77	349
1	0.83	0.87	0.85	349
2	0.84	0.81	0.82	328
accuracy			0.81	1026
macro avg	0.81	0.81	0.81	1026
weighted avg	0.81	0.81	0.81	1026



Classification Report:

	precision	recall	f1-score	support
0	0.77	0.76	0.77	349
1	0.83	0.87	0.85	349
2	0.84	0.81	0.82	328
accuracy			0.81	1026
macro avg	0.81	0.81	0.81	1026
weighted avg	0.81	0.81	0.81	1026



As seen, multiclass gets more accuracy than binary class. It gets 81% accuracy.

Support Vector Machines (SVMs) are powerful machine learning algorithms commonly used for both binary and multiclass classification tasks. In some cases, SVMs can achieve higher accuracy for multiclass classification compared to binary classification. There are a few reasons why this might occur:

1. Increased decision boundaries: In binary classification, SVMs find a single decision boundary to separate the two classes. However, in multiclass classification, SVMs construct multiple decision boundaries to separate each class from the rest. By considering the relationships between multiple classes simultaneously, SVMs can capture more complex decision

boundaries, potentially resulting in higher accuracy.

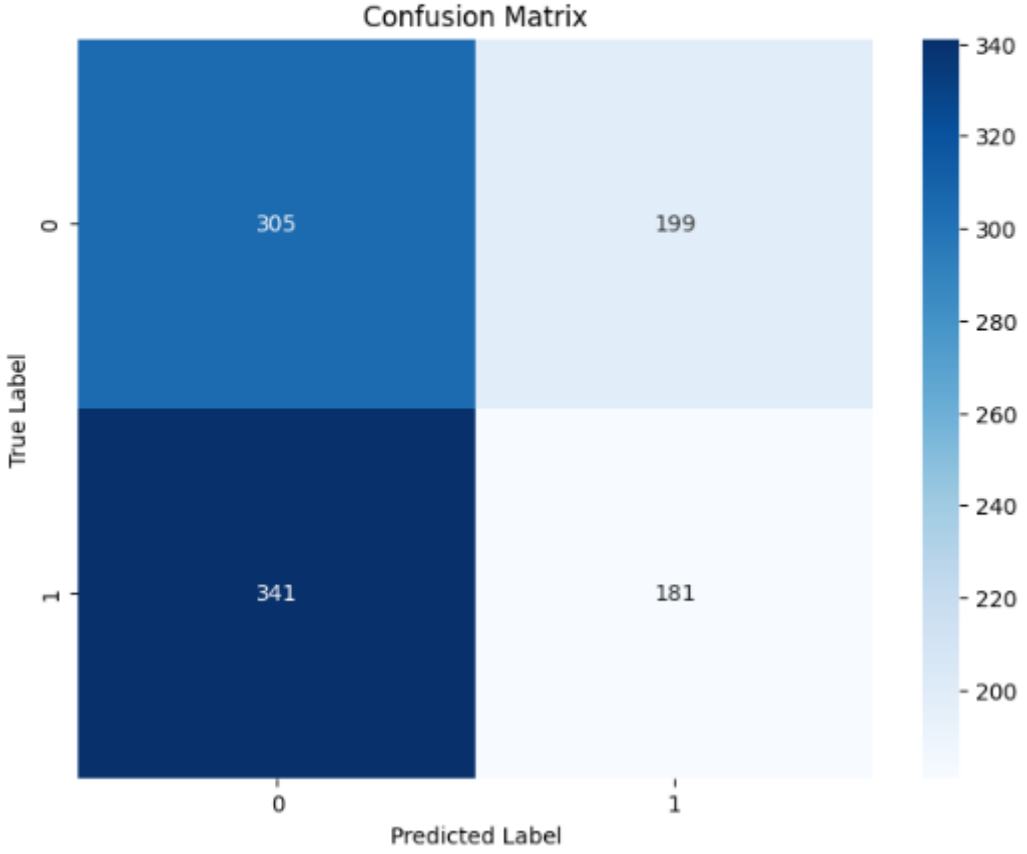
2. Implicit pairwise comparisons: When using one-vs-one or one-vs-rest strategies to extend binary SVMs to multiclass problems, the model effectively creates multiple binary classifiers. In the one-vs-one approach, SVMs are trained on pairs of classes, and during testing, each classifier predicts the class label independently. In the one-vs-rest approach, a separate classifier is trained for each class to distinguish it from all other classes. These strategies inherently leverage pairwise comparisons between classes, which can provide additional information and improve classification accuracy.

3. Exploiting class relationships: In multiclass problems, SVMs have the ability to exploit relationships between different classes. For example, if some classes are inherently more similar to each other while being distinct from other classes, SVMs can leverage this knowledge to improve accuracy. By considering the global structure of the data, SVMs can better model the relationships between classes and make more accurate predictions.

It's important to note that the performance of SVMs can vary depending on the specific dataset and problem at hand. In some cases, binary SVMs may outperform multiclass SVMs if the problem can be simplified to a binary classification task. However, in general, multiclass SVMs have the potential to achieve higher accuracy by considering the relationships between multiple classes and constructing more complex decision boundaries.

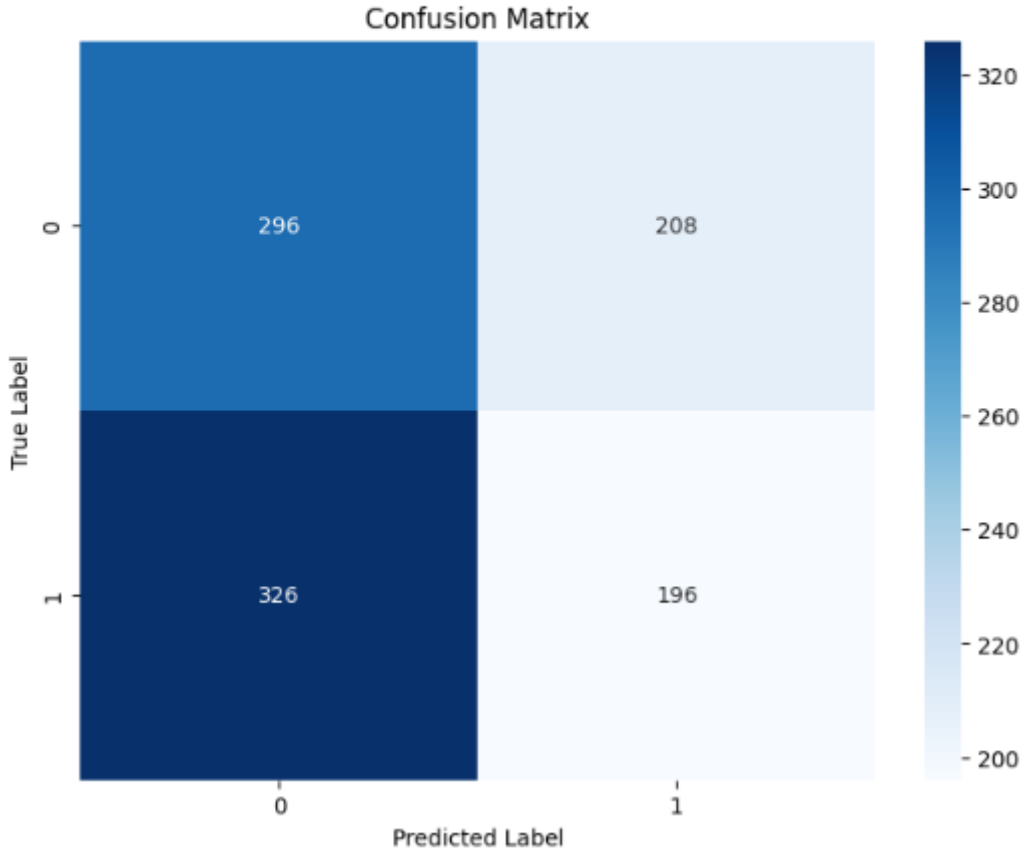
we got accuracy for given features in the project file as a 'features.csv'. first we converted all name's resized images into 0 or 1 that refers real or fake then got accuracy:

	precision	recall	f1-score	support
0	0.47	0.61	0.53	504
1	0.48	0.35	0.40	522
micro avg	0.47	0.47	0.47	1026
macro avg	0.47	0.48	0.47	1026
weighted avg	0.47	0.47	0.46	1026



As seen the accuracy is 47%. then we tried to convert all name's original image (before resizing) to 0 or 1 as the same above and got accuracy

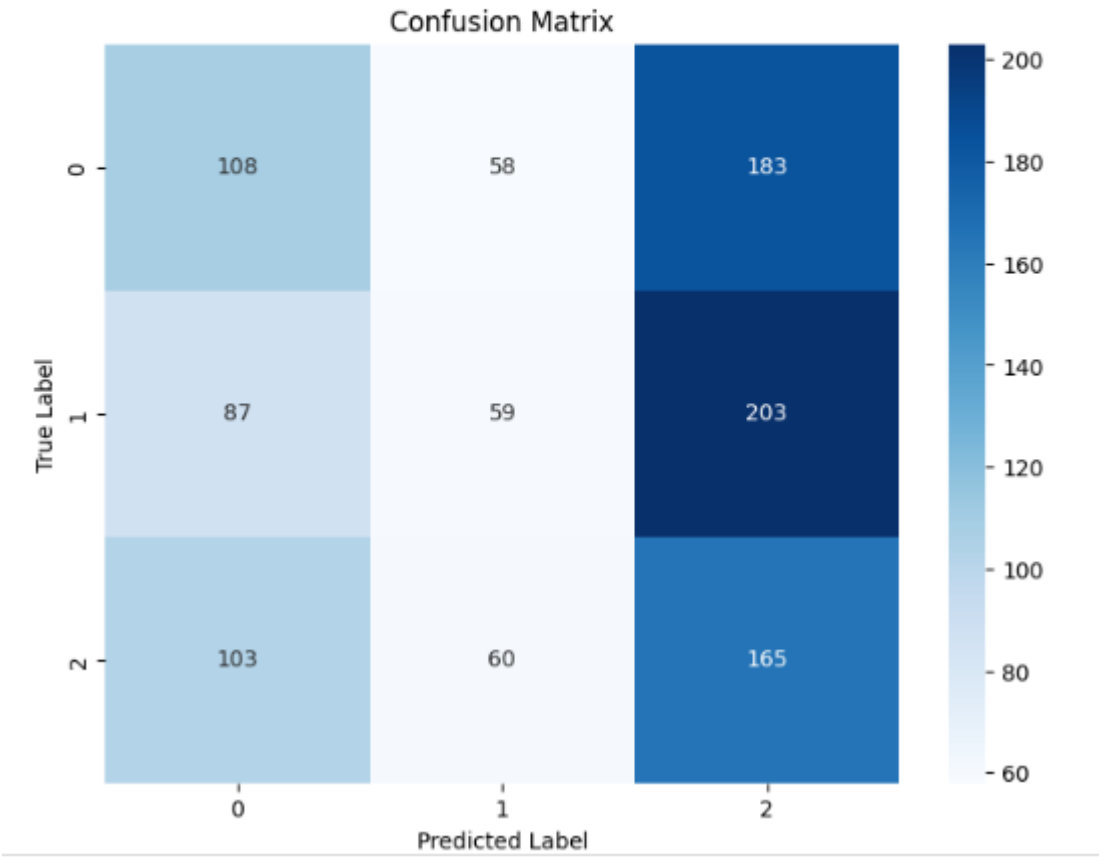
```
Classification Report:
              precision    recall  f1-score   support

           0       0.48      0.59      0.53       504
           1       0.49      0.38      0.42       522

    accuracy                           0.48      1026
   macro avg       0.48      0.48      0.47      1026
weighted avg       0.48      0.48      0.47      1026
```



we got 48% accuracy that is less than the accuracy we got from our features (64% without PCA)

we also did the same thing for multi class, first we did for labels that we converted from resized images:

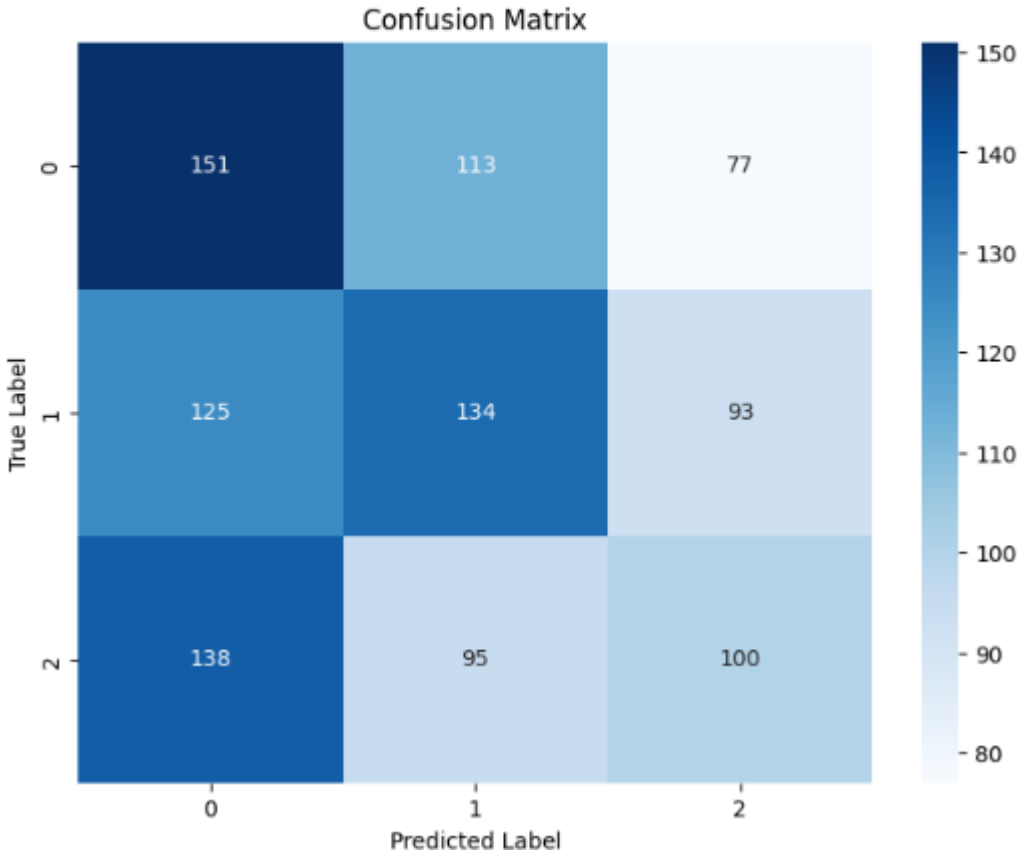
```
Classification Report:
              precision    recall  f1-score   support

           0       0.36      0.31      0.33       349
           1       0.33      0.17      0.22       349
           2       0.30      0.50      0.38       328

   micro avg       0.32      0.32      0.32      1026
   macro avg       0.33      0.33      0.31      1026
weighted avg       0.33      0.32      0.31      1026
```



the accuracy is 32%. now it's time to show the accuracy for labels which we got from original

Classification Report:

	precision	recall	f1-score	support
0	0.36	0.44	0.40	341
1	0.39	0.38	0.39	352
2	0.37	0.30	0.33	333
accuracy			0.38	1026
macro avg	0.38	0.37	0.37	1026
weighted avg	0.38	0.38	0.37	1026



images.

the accuracy is 38%. these are less than accuracy that we got from our features (81%) so we showed that the accuracy we got from HOG features is better than the accuracy from given features in the project file.

4.4 AdaBoost Results

In our research, we utilized the SIFT algorithm to extract features from the images. Subsequently, we employed the AdaBoost algorithm for image classification. Our classification results yielded

an accuracy of 51%.

	precision	recall	f1-score	support
0	0.51	1.00	0.67	521
1	0.00	0.00	0.00	505
accuracy			0.51	1026
macro avg	0.25	0.50	0.34	1026
weighted avg	0.26	0.51	0.34	1026

4.5 Transfer Learning (ResNet50)

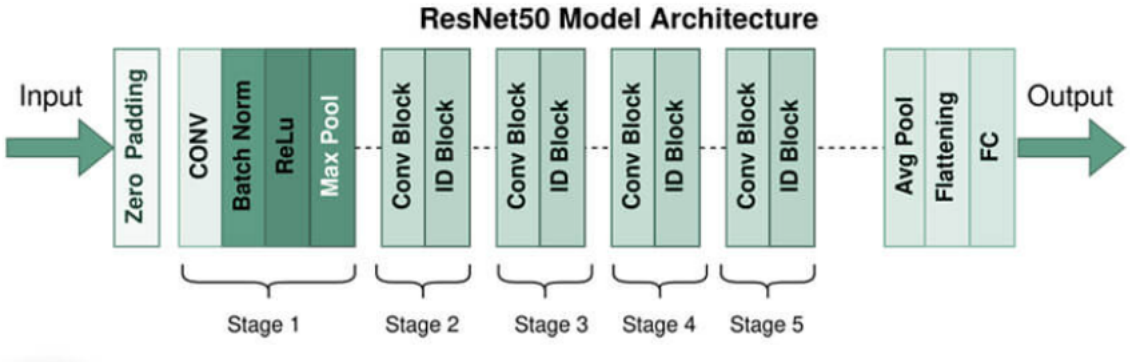
ResNet-50 is a deep convolutional neural network (CNN) architecture that has achieved state-of-the-art performance in various computer vision tasks, including image classification, object detection, and image segmentation. It was introduced by Microsoft Research in 2015 and is a variant of the ResNet (Residual Network) architecture.

The ResNet-50 architecture is composed of 50 layers, including convolutional layers, pooling layers, fully connected layers, and shortcut connections. These shortcut connections, also known as skip connections or identity mappings, are a key component of the ResNet architecture. They allow the network to learn residual mappings, which help address the degradation problem that occurs when deep networks suffer from diminishing accuracy as the depth increases.

To understand the concept of residual mappings, let's consider a simpler case of a shallow network with only a few layers. In such a network, each layer learns to approximate the desired mapping from its input to the output. However, as the network gets deeper, it becomes challenging for the layers to approximate the ideal mapping accurately. This issue arises because the layers may encounter vanishing gradients or face difficulty in learning meaningful representations.

ResNet addresses this problem by introducing skip connections that directly connect the input of a layer to the output of a later layer. By using these connections, the network can learn residual mappings, capturing the difference between the input and output of a layer. This enables the network to focus on learning the residual details rather than attempting to directly approximate the entire mapping. The skip connections also help propagate gradients effectively, allowing for better gradient flow during training.

Here is a visual representation of the ResNet-50 architecture:



In the image, you can observe the structure of ResNet-50 with different types of layers, including convolutional layers (Conv), pooling layers (MaxPool), fully connected layers (FC), and shortcut connections (Skip Connections).

ResNet-50 has been widely adopted and pre-trained on large-scale image datasets, such as ImageNet. The pre-trained model can be fine-tuned on specific tasks or used as a feature extractor for transfer learning, where the learned representations from the earlier layers are utilized for new tasks.

The ResNet-50 architecture, with its deep structure and skip connections, has demonstrated remarkable performance in various computer vision applications, making it a popular choice for researchers and practitioners in the field.

In our study, we explored various algorithms and methods for image classification, including ResNet 50, a deep learning model. After conducting a thorough analysis, we found that ResNet 50 achieved the best accuracy, with a score of almost 70%. This accuracy was higher than all the other methods and algorithms that we tested, indicating that ResNet 50 is a highly effective approach for image classification tasks. These results demonstrate the power of deep learning models in image recognition and highlight the importance of selecting an appropriate algorithm for achieving the desired outcomes. Overall, our findings suggest that ResNet 50 is a promising approach for image classification and could be useful in a range of applications where accurate image recognition is essential.

5 Clustering

We used two types of clustering algorithm: K-means , Birch

Clustering is a technique used in machine learning and data mining to group similar data points together. The goal of clustering is to partition a set of data points into groups, or clusters, based on their similarity. Clustering is an unsupervised learning method, which means that it does not require labeled data to train a model.

There are several types of clustering methods, including:

1. K-means clustering: This is a popular clustering method that partitions data into k number of clusters based on the mean of each cluster. The algorithm works by randomly selecting k centroids (cluster centers), and then iteratively updating the centroids based on the mean of data points that belong to each cluster.

2. Hierarchical clustering: This method creates a tree-like structure of clusters by recursively dividing data points into smaller clusters based on their similarity. There are two types of hierarchical clustering: agglomerative and divisive. In agglomerative clustering, each data point starts as a separate cluster, and then pairs of clusters are merged together until a single cluster containing all data points is formed. In divisive clustering, all data points start in a single cluster, and then the algorithm recursively divides the cluster into smaller clusters.

3. Density-based clustering: This method identifies clusters based on the density of data points in a particular region. The algorithm starts by identifying core points (data points with a minimum number of neighboring points within a defined radius) and then expands the cluster by adding neighboring points that meet a density threshold.

4. Fuzzy clustering: This method assigns each data point a degree of membership to each cluster, rather than assigning it to a single cluster. The degree of membership represents the probability that the data point belongs to each cluster.

5. Spectral clustering: This method uses spectral graph theory to partition data into clusters. It works by first constructing a similarity graph, where each data point is a node and edges represent the similarity between data points. The algorithm then uses the eigenvalues and eigenvectors of the Laplacian matrix of the graph to partition the data into clusters.

There are also many other types of clustering methods, including model-based clustering, subspace clustering, and consensus clustering. The choice of clustering method depends on the nature of the data and the problem being solved.

Clustering has many applications in various fields, such as image segmentation, customer segmentation, anomaly detection, and document clustering. It is a powerful tool for exploratory data analysis and can provide insights into the structure of complex datasets.

Here we want to introduce special clustering algorithm that we used for clustering which is KMeans:

5.1 KMeans

K-means is a popular clustering algorithm that partitions a dataset into k clusters, where k is a user-defined parameter. The algorithm works by iteratively assigning each data point to the nearest cluster center, and then updating the cluster centers based on the mean of all data points assigned to that cluster.

Mathematically, K-means can be formulated as follows:

Given a dataset $X = \{x_1, x_2, \ldots, x_n\}$ consisting of n data points in d-dimensional space, where each data point $x_i \in \mathbb{R}^d$, the goal is to partition X into k clusters, where $k < n$.

1. Initialization: Randomly select k cluster centers $\{c_1, c_2, \ldots, c_k\}$ from X.

2. Assignment step: Assign each data point x_i to the nearest cluster center, based on the Euclidean distance between x_i and each cluster center. This can be expressed as:

$c_i(x) = \text{argmin}_{j=1,\ldots,k} ||x_i - c_j||^2$, for $i = 1, 2, \ldots, n$

where $c_i(x)$ is the index of the cluster center closest to x_i, and $||.||^2$ denotes the squared Euclidean distance.

3. Update step: Update the cluster centers by taking the mean of all data points assigned to that cluster. This can be expressed as:

$c_i = \frac{1}{|S_i|} \sum_{x_j \in S_i} x_j$, for $i = 1, 2, \ldots, k$

where S_i is the set of data points assigned to cluster i, and $|S_i|$ is the number of data points in that set.

4. Repeat steps 2 and 3 until convergence. Convergence is achieved when the cluster

assignments no longer change or when a maximum number of iterations is reached.

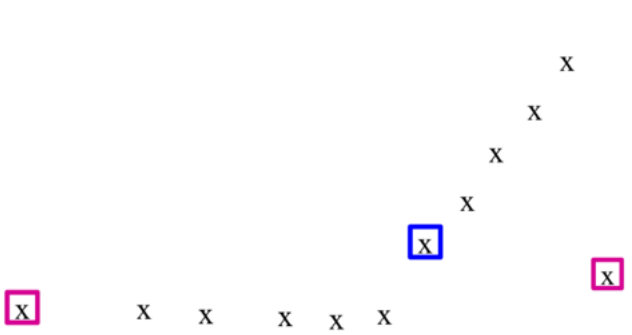
The objective of K-means is to minimize the sum of squared distances between data points and their assigned cluster centers, which is also known as the within-cluster sum of squares (WCSS). Mathematically, the objective function can be expressed as:

$J = \sum_{i=1}^{k} \sum_{x \in S_i} ||x - c_i||^2$

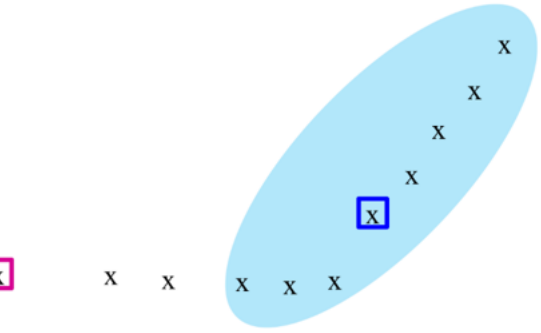
where S_i is the set of data points assigned to cluster i, and c_i is the cluster center for cluster i.

K-means can be sensitive to the initial cluster centers, as the algorithm may converge to a suboptimal solution if the initial centers are poorly chosen. To mitigate this issue, K-means is often run multiple times with different initializations, and the solution with the lowest WCSS is chosen.

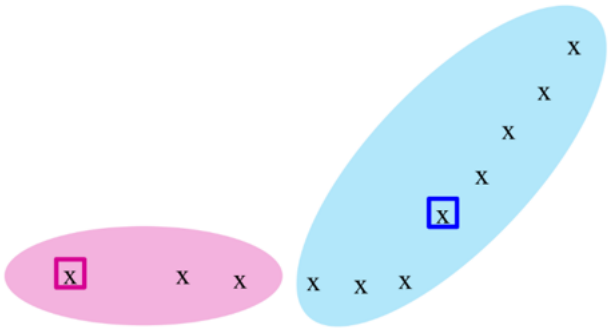
Here are some images that demonstrate how kmeans works:



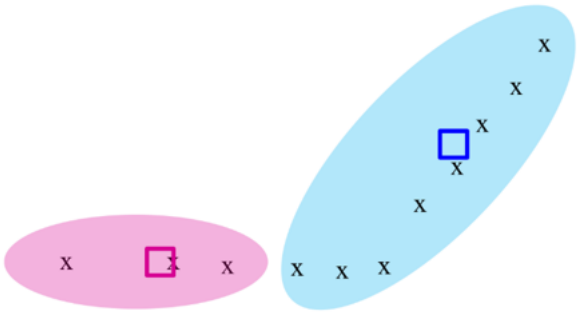
Step 3



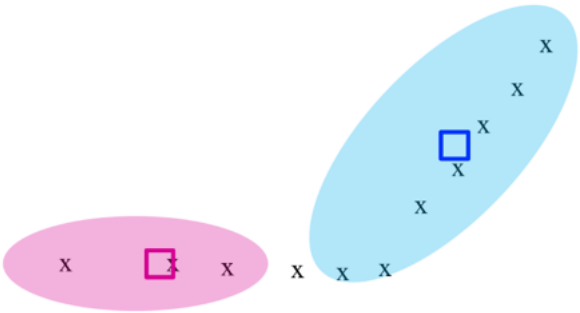
Step 4



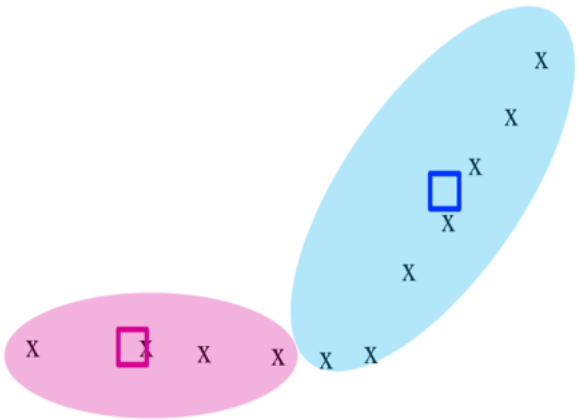
Step 5



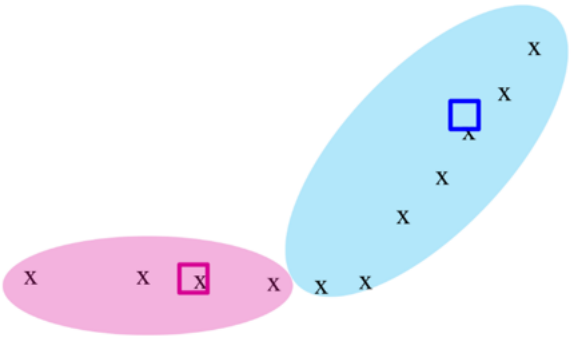
Step 6



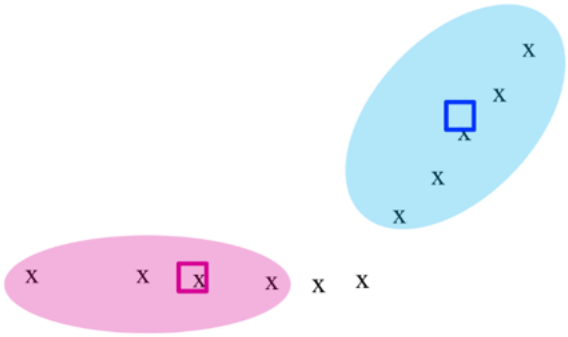
Step 7



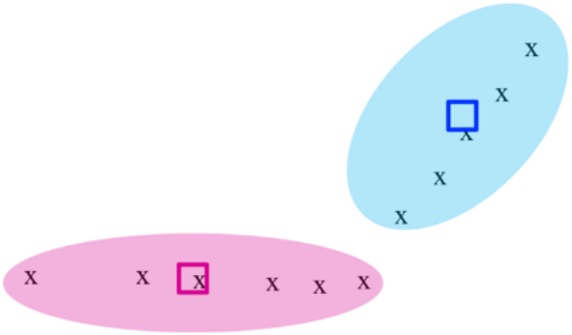
Step 8



Step 9



Step 10



Step 11

5.2 Birch

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is a clustering algorithm that is designed to handle large datasets efficiently. It was proposed by Tian Zhang, Raghu Ramakrishnan, and Miron Livny in 1996. BIRCH constructs a hierarchical clustering structure called the CF tree (Clustering Feature tree) to represent the dataset.

The BIRCH algorithm has the following key components and steps:

1. Clustering Feature (CF): Each CF is a summary of a group of data points. It contains three parts: a centroid, the number of points it represents (N), and a radius that represents the maximum distance from any point to the centroid. CFs are used to represent clusters at different levels of the hierarchy.

2. CF Tree: The CF tree is a hierarchical structure that organizes CFs. It consists of multiple levels, with each level representing a different level of granularity in the clustering. The root level of the tree represents the coarsest clustering, and as we move down the levels, the clusters become more refined.

3. Clustering and Merging: BIRCH performs a two-step clustering process. In the first step, it uses a fast single-pass clustering algorithm to build initial CFs. Then, it recursively merges CFs based on their proximity and a user-defined threshold. The merging process continues until all CFs are processed.

4. Cluster Assignments: Once the CF tree is constructed, BIRCH assigns each data point to the closest CF in the tree. This assignment determines the final cluster memberships.

BIRCH offers several advantages for clustering large datasets:

- Memory Efficiency: BIRCH uses a compact data structure (CF tree) that allows efficient storage and retrieval of clustering information, making it suitable for datasets that cannot fit entirely in memory.

- Scalability: The algorithm's hierarchical nature and ability to incrementally update the CF tree make it scalable to large datasets.

- Fast Processing: BIRCH uses a single-pass clustering algorithm to build initial CFs, reducing the overall computational complexity.

- Robustness to Noise: The radius of CFs provides a notion of density, allowing BIRCH to

handle noisy data and identify outliers.

However, BIRCH also has some limitations. It may struggle with datasets of varying densities, and it requires setting appropriate parameters, such as the radius and the threshold for merging CFs, which can impact the quality of clustering results.

Overall, BIRCH is a useful algorithm for efficient clustering of large datasets, particularly in scenarios where memory and processing speed are crucial factors.

5.3 Silhouette Coefficient

The Silhouette Coefficient is a measure of the quality of clustering that combines both the cohesion and separation aspects of clusters. It quantifies how well each data point fits into its assigned cluster compared to other clusters. The Silhouette Coefficient for a single data point, denoted as $s(i)$, is calculated using the following formula:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where: - $a(i)$ represents the average dissimilarity of data point i to all other data points within the same cluster. - $b(i)$ represents the average dissimilarity of data point i to all data points in the nearest neighboring cluster (i.e., the cluster with the minimum average dissimilarity to data point i).

The numerator of the formula measures the separation between the data point and the neighboring cluster, while the denominator scales the separation based on the cohesion within the cluster. A higher Silhouette Coefficient indicates that the data point is well-matched to its assigned cluster and well-separated from other clusters.

The overall Silhouette Coefficient for a clustering solution is computed by averaging the Silhouette Coefficients for all data points within the dataset. The closer the average Silhouette Coefficient is to 1, the better the clustering quality. Conversely, values close to -1 indicate poor clustering, where data points might be assigned to incorrect clusters.

The Silhouette Coefficient provides a quantitative assessment of clustering performance, helping to compare and select the optimal number of clusters or evaluate different clustering

algorithms.

5.4 Results

Before explaining the results, we must point out that in all the cases where the tests were done, the KMeans algorithm worked better than the Birch algorithm and gave better results, and therefore from here on, everything that is used as an output It is announced that the results of KMeans algorithm

In our study, we employed the k-means clustering algorithm on features extracted using the ResNet50 model. We would like to present the results of our clustering analysis for various numbers of clusters. Specifically, we experimented with 2, 3, 6, 9, and 50 clusters.

Two Clusters

In the case of utilizing two clusters, our model demonstrates a tendency to assign sea and mountain images to one cluster while categorizing jungle images into the other cluster across the majority of input samples :

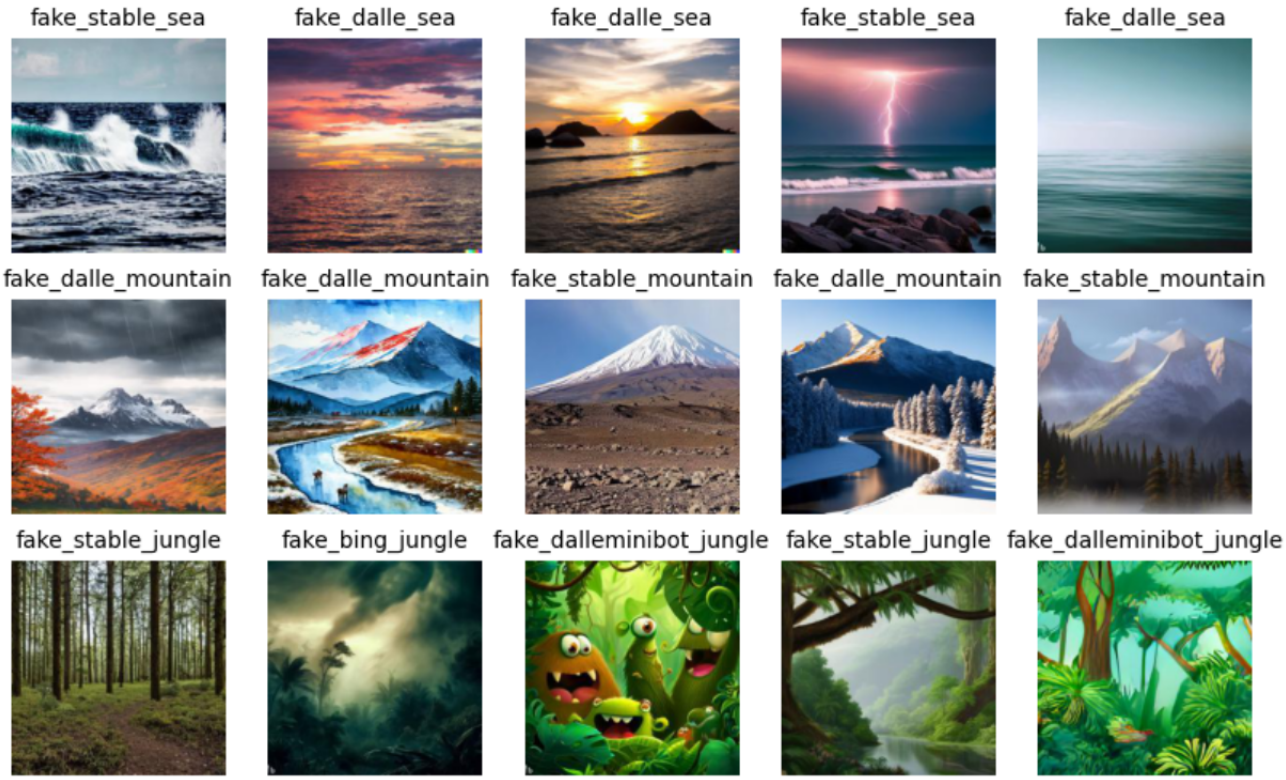
In this particular case, we obtained a Silhouette Coefficient of 0.11, indicating that our clustering performed reasonably well. However, it also suggests the presence of some ambiguity within the clustering results, possibly due to a significant overlap among the data points.

The Silhouette Coefficient value of 0.11 suggests that the clusters exhibit moderate separation, but there might be instances where data points have similar distances to multiple clusters, leading to a degree of uncertainty in their assignments. This ambiguity could be attributed to the inherent complexity or overlapping nature of the dataset.

It is crucial to note that while the Silhouette Coefficient provides a quantitative measure of clustering performance, it should be complemented with visual inspection and domain knowledge. By examining the clustered images and considering the specific characteristics of the dataset, we can gain a more comprehensive understanding of the clustering outcomes and assess the meaningfulness and coherence of the clusters.

Three Clusters

In the case of utilizing three clusters, our model demonstrates a notable separation of sea, mountain, and jungle pictures into distinct clusters across nearly all input samples. This outcome indicates a strong performance of our clustering model in effectively distinguishing and assigning these image categories to their respective clusters.



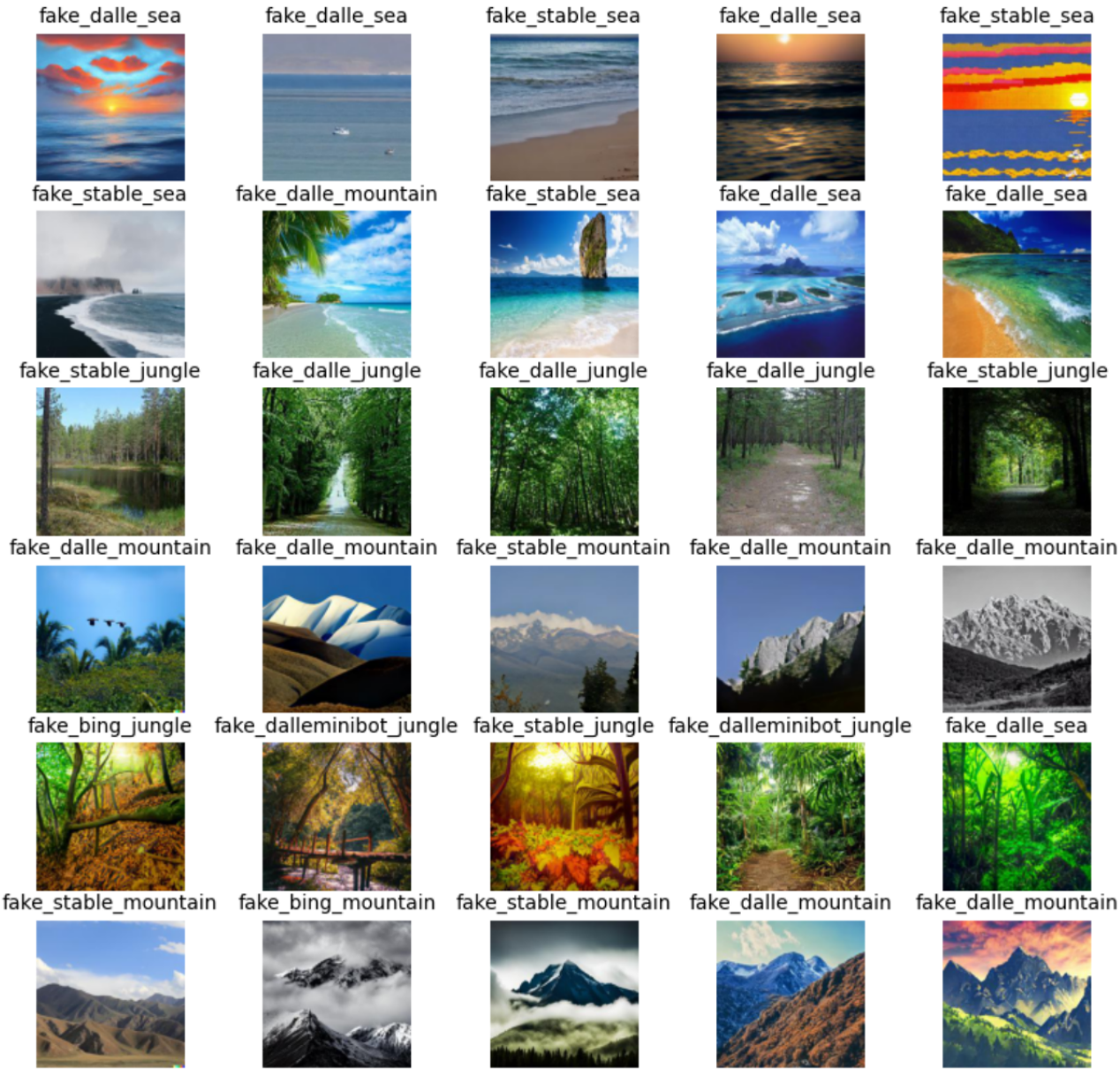
With the utilization of three clusters, we obtained a Silhouette Coefficient of 0.093, indicating that our clustering performed well overall. However, there is still some ambiguity present within the clustering results, which could be attributed to a relatively high degree of overlap among the data points.

The Silhouette Coefficient value of 0.093 suggests a reasonable level of separation between the clusters. Nonetheless, there are instances where data points exhibit similarities to multiple clusters, resulting in uncertain assignments. This ambiguity can be attributed to the inherent complexity of the dataset, where certain images may share visual characteristics that make them difficult to distinctly assign to a single cluster.

While the Silhouette Coefficient provides an objective measure of clustering quality, it is important to consider other factors such as visual inspection and domain knowledge. By visually analyzing the clustered images and leveraging domain expertise, we can gain a deeper understanding of the clustering outcomes, assess the meaningfulness of the clusters, and account for any potential limitations stemming from data point overlap.

Six Clusters

When using a model with six clusters, two clusters were allocated to each class of images (sea, mountain, and jungle). However, it was observed that there were no significant differences between the two clusters assigned to each class. Here is an example of the output :



When we applied our clustering methodology with six clusters, we achieved a Silhouette Coefficient of 0.067, which indicates a reasonably satisfactory performance. However, the

clustering outcomes reveal a significant level of confusion. This confusion can be attributed to the substantial overlap among the data points, leading to difficulties in accurately assigning them to distinct clusters.

Nine Clusters

In the case of nine clusters, it may prove difficult to identify instances that accurately reflect the characteristics of each group. Despite this challenge, a silhouette score of 0.04 suggests that there is some degree of clustering structure present in the data. While this score may not be particularly high, it still provides some insight into the organization of the data points and can guide further analysis. Overall, this finding highlights the importance of carefully considering the number of clusters to use in clustering analysis and the challenges that can arise when dealing with a large number of groups.

Fifty Clusters

After performing clustering with 50 clusters, we achieved remarkable and highly satisfying results. These 50 clusters naturally separate into three distinct categories, namely seas, mountains, and jungles. Specifically, the sea category images were grouped into multiple clusters, each representing a specific type of sea image based on our observations. For instance, one cluster predominantly contained images featuring ships, another cluster comprised images depicting rough sea conditions, while yet another cluster showcased images highlighting various sea creatures, and so on.

Similarly, within the mountain and jungle categories, we observed a similar phenomenon. We discovered clusters dedicated to volcanic mountains, misty mountain landscapes, autumn jungles, and even jungles inhabited by monkeys. The clustering analysis effectively captured the unique characteristics and variations within each category, providing valuable insights into the diverse visual attributes present in the images.

Furthermore, the evaluation of our clustering results yielded a Silhouette Coefficient of 0.02, which indicates a reasonable level of separation and clustering quality. This score demonstrates the effectiveness of our approach in organizing the images into meaningful and distinct clusters,

enhancing our understanding of the underlying patterns and categories within the dataset.