ARIZONA STATE UNIVERSITY
CSE 515 SLN 95433: **Multimedia and Web Databases** Fall 2023

Course Project: Phase #3 Report

Group Details: Group #15
Caleb Panikulam - cpanikul@asu.edu
Gokul Vasudeva - gvasude2@asu.edu
Joshua Martin Noronha - jnoronha@asu.edu
Kiran Sthanusubramonian - ksthanus@asu.edu
Sandipan De - sandipan@asu.edu
Shankar Harinarayanan - sharina1@asu.edu

Contents

1 Abstract **3**

2 Keywords **3**

3 Structure of Report **3**

4 Relevant Definitions, Formulae and Corresponding Implementations **4**
4.1 Distance Measures 4
4.2 Inherent Dimensionality & Dimensionality Reduction 4
4.3 Clustering Algorithms 5
4.4 Classification Algorithms 6
4.5 Approximate Multidimensional Indexing (LSH) 7
4.6 Relevance Feedback 8
4.7 Accuracy Measures 10

5 The Tasks **11**
5.1 Task 0 11
5.1.1 Task 0a - Goal Description: 11
5.1.2 Task 0a - Assumptions Made: 11
5.1.3 Task 0a - Interface Specifications: 11
5.1.4 Task 0a - Implementation Approach: 11
5.1.5 Task 0a - Results: 12
5.1.6 Task 0a - Analysis of results: 13
5.1.7 Task 0b - Goal Description: 14
5.1.8 Task 0b - Assumptions Made: 14
5.1.9 Task 0b - Interface Specifications: 14
5.1.10 Task 0b - Implementation Approach: 14
5.1.11 Task 0b - Results: 15
5.1.12 Task 0b - Analysis of results: 16
5.2 Task 1 17
5.2.1 Goal Description: 17
5.2.2 Assumptions Made: 17
5.2.3 Interface Specifications: 17
5.2.4 Implementation Approach: 17
5.2.5 Results: 18
5.2.6 Task 1 - Analysis of results: 19
5.3 Task 2 20
5.3.1 Goal Description: 20
5.3.2 Assumptions Made: 20
5.3.3 Interface Specifications: 20

5.3.4 Implementation Approach: . . . 21
5.3.5 Results: . . . 22
5.3.6 Task 2 - Analysis of results . . . 27
5.4 Task 3 . . . 29
5.4.1 Goal Description: . . . 29
5.4.2 Assumptions: . . . 29
5.4.3 Interface Specifications: . . . 29
5.4.4 Implementation Approach: . . . 30
5.4.5 Results: . . . 32
5.4.6 Task 3 - Analysis of Results: . . . 38
5.5 Task 4 . . . 40
5.5.1 Task 4a - Goal Description: . . . 40
5.5.2 Task 4a - Assumptions Made: . . . 40
5.5.3 Task 4a - Interface Specifications: . . . 40
5.5.4 Task 4a - Implementation Approach: . . . 40
5.5.5 Task 4a - Results: . . . 42
5.5.6 Task 4b - Goal Description: . . . 42
5.5.7 Task 4b - Assumptions Made: . . . 42
5.5.8 Task 4b - Interface Specifications: . . . 42
5.5.9 Task 4b - Implementation Approach: . . . 43
5.5.10 Task 4b - Results: . . . 43
5.5.11 Task 4 - Analysis of Results . . . 62
5.6 Task 5 . . . 63
5.6.1 Goal Description: . . . 63
5.6.2 Assumptions: . . . 63
5.6.3 Interface Specifications: . . . 64
5.6.4 Implementation Approach: . . . 64
5.6.5 Results: . . . 66
5.6.6 Task 5 - Analysis of Results . . . 70

6 System Requirements & Execution Instructions 71

7 Related Works 71

8 Conclusions 72

9 Acknowledgements 72

A Specific Roles and Details of the Group Members 74

1 Abstract

In Phase 3 of our Course Project, we build on the feature descriptors and their respective reduced-size latent semantics extracted from Phases 1 and 2, respectively. Using these feature descriptors, we build clustering, classification, and approximate multidimensional indices (for example, using Locality-Sensitive Hashing) for efficient information retrieval (IR) for the Caltech101 Dataset. With each task, we build (and present) the necessary intuitions and attempt to effectively contrast their efficacy in terms of accuracy (precision and recall), time and space complexity. Finally, we also explore popular techniques used to improve the relevance of the overall information retrieval for the users, such as Relevance Feedback using Support Vector Machines and Probabilistic Models.

2 Keywords

image processing, vector models, feature extraction, similarity measure, information retrieval, inherent dimensionality, clustering, image classification, approximate classification, multidimensional indexing, relevance feedback

3 Structure of Report

Following this section, we introduce the relevant definitions and mathematical formulae (and key implementation details surrounding the formulae) used for the specific tasks in this Phase. Suitable use cases of these mathematical formulae during the tasks will be cited back to this section to ensure the report remains easily readable. Next, we will systematically discuss the goal description, assumptions (if any), implementation approach, and results (if any) for each Task described in the Specification Outline. Following this, we will go through separate sections on System Requirements & Execution Instructions to run the code base provided (alternatively, you can refer to the README.md file described in the primary code base for these details). The report will continue with separate sections on Related Works, Overall Conclusions, Acknowledgements, and notable References used to complete the tasks. The report will conclude with an Appendix detailing the efforts of each group member to complete this Phase-wise development.

4 Relevant Definitions, Formulae and Corresponding Implementations

4.1 Distance Measures

We implement 4 different Distance Measures for our experiments in this project. Distance Measures are either pre-computed or implemented using the Scipy Spatial Distance module [1]:

$$\text{Cosine Distance} = 1 - \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|}$$

$$\text{Correlation Distance} = 1 - \frac{(\mathbf{u} - \mathbf{u_{mean}}) \cdot (\mathbf{v} - \mathbf{v_{mean}})}{\|(\mathbf{u} - \mathbf{u_{mean}})\|_2 \cdot \|(\mathbf{v} - \mathbf{v_{mean}})\|_2}$$

$$\text{Manhattan Distance} = \sum_{i=1}^{n} |A_i - B_i|$$

$$\text{Euclidean Distance} = \sum_{i=1}^{n} \sqrt{(A_i^2 - B_i^2)}$$

4.2 Inherent Dimensionality & Dimensionality Reduction

Inherent Dimensionality (also popularly known as Intrinsic dimensionality) refers to the essential or inherent number of features or dimensions needed to describe a dataset or a system. It is a concept commonly used in data analysis, machine learning, and signal processing. The inherent dimensionality reflects the effective complexity of the data or system and helps in understanding the fundamental structure of the overall dataset.

For this phase of the project, we calculate the inherent dimensionality of any of the feature vector matrices extracted from Phases 1 and 2 (of our choice) for the Caltech101 Dataset.

The Inherent Dimensionality of these matrices can be calculated using multiple Dimensionality Reduction techniques. However, for this phase of the Project. we mainly focus on using the following Dimensionality Reduction technique:

Singular Value Decomposition (SVD):

$$A = U\Sigma V^T$$

where, A = Original Matrix

U = left singular vectors matrix, an $m \times m$ orthogonal matrix

Σ = diagonal singular values matrix, an $m \times n$ diagonal matrix

V^T = transpose of right singular vectors matrix, an $n \times n$ orthogonal matrix

SVD is implemented from scratch in our codebase, with the help of several numpy functions (especially the linear algebra module within numpy to calculate the Eigen Value Matrix Σ) [2].

Scree Test to Calculate Inherent Dimensionality:
The Scree test is a common technique used to estimate the inherent dimensionality of a dataset. This method is often applied in the context of dimensionality reduction techniques, such as Principal Component Analysis (PCA). The idea is to plot the explained variance as a function of the number of components (dimensions) and look for the "elbow" point in the plot, which indicates a point of diminishing returns in terms of explaining the variance.
After retrieving the eigenvalues using SVD, we calculate and plot the explained variance (calculated using an optimizer library) vs the number of components. The "elbow" in the plot is where the explained variance stops increasing rapidly, and this corresponds to the Inherent Dimensionality of the dataset.

$$\text{Explained Variance} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$$

$$k = \text{number of components considered}$$

$$p = \text{total number of components}$$

$$\lambda_i = \text{eigenvalue of i-th component}$$

4.3 Clustering Algorithms

Throughout the course of this project up till point, we have primarily worked with the **K-Means Clustering Algorithm** [3] (implemented from scratch using relevant matrix manipulation & distance measure functionality using NumPy, SciPy, and Scikit-learn).
However, for task 2 of this Phase, we implement the **DBScan Clustering Algorithm** [4] from scratch. DBScan is particularly useful for identifying clusters of points in a dataset based on their density in the feature space. Unlike k-means, which assume that clusters are spherical and of similar size, DBSCAN can find clusters of arbitrary shapes and handle clusters of different sizes. Additionally, DBSCAN is capable of identifying noise or outliers in the data. The formulae for each are as follows:

1. **K-Means Clustering:**

$$J = \sum_{i=1}^{K} \sum_{j=1}^{n} ||x_j^{(i)} - \mu_i||^2$$

$$\begin{aligned}
\text{where, } J &= \text{Objective Function to be Minimized} \\
K &= \text{Number of Clusters} \\
n &= \text{Number of Data Points} \\
x_j^{(i)} &= \text{j-th data point in cluster i} \\
\mu_i &= \text{centroid of cluster i}
\end{aligned}$$

2. **DBScan (Density-Based Spatial Clustering):**

$$\begin{aligned}
\text{Core}(p, \varepsilon, \text{MinPts}) &= |\{q \in D : \text{dist}(p, q) \leq \varepsilon\}| \geq \text{MinPts} \\
\text{Border}(p, \varepsilon, \text{MinPts}) &= |\{q \in D : \text{dist}(p, q) \leq \varepsilon\}| < \text{MinPts and} \\
&\quad \exists o \in D : \text{Core}(o, \varepsilon, \text{MinPts}) \text{ where } \text{dist}(p, o) \leq \varepsilon \\
\text{Noise}(p) &= \neg(\text{Core}(p, \varepsilon, \text{MinPts}) \vee \text{Border}(p, \varepsilon, \text{MinPts})) \\
\text{Clusters, } C &= \{p : \text{Core}(p, \varepsilon, \text{MinPts})\} \\
\varepsilon &= \text{ the maximum distance to be considered a neighbor} \\
\text{MinPts} &= \text{minimum number of data points required within distance } \varepsilon
\end{aligned}$$

4.4 Classification Algorithms

Classification algorithms are a type of machine learning algorithm used to categorize or label data points into different classes or groups based on their features. The goal of classification is to learn a mapping from input features to a target output variable, often a discrete label or category. In task 3, we primarily implement the following classifiers:

1. ***k*-NN Classifier: (NOTE: The term *k*-NN Classifier is used interchangeably with *m*-NN Classifier throughout this project, including the codebase)**
   The k-NN (k-Nearest Neighbors) classifier [5] is a supervised machine learning algorithm used for classification tasks. Given a dataset with labelled instances, the k-NN classifier classifies a new, unlabeled instance by considering the class labels of its m nearest neighbors in the feature space.

$$y_q = \text{argmax}_y \left(\sum_{i=1}^{k} \delta(y_i = y) \right)$$

$$\delta = \text{Euclidean Distance Measure}$$

2. **Decision Tree Classifier:** A decision tree is a predictive model that maps features (or attributes) to conclusions about the target value. It's a tree-like model of decisions and their possible consequences. Each internal node represents a feature (or attribute), each branch represents a decision rule, and each leaf node represents the outcome. The general formula for decision trees can be represented as follows:

$$T(X) = \begin{cases} \text{Outcome}_1 & \text{if } f_1(X) \text{ is true} \\ \text{Outcome}_2 & \text{if } f_2(X) \text{ is true} \\ \vdots & \\ \text{Outcome}_k & \text{if } f_k(X) \text{ is true} \end{cases}$$

where,

$$\begin{aligned} T(X) &= \text{decision tree function for the} \\ &\quad \text{input feature vector X} \\ f_1(x),\ f_2(x),\ ....,\ f_k(x) &= \text{conditions based on the features} \\ Outcome_1,\ Outcome_2,\ ....\ ,\ Outcome_k &= \text{possible outcomes} \end{aligned}$$

3. **Personalized Page Rank (PPR)-based Classifier:** Personalized PageRank is an algorithm that measures the importance of nodes in a graph with respect to a given set of seed nodes. The basic idea is to compute the PageRank score for each node based on random walks through the graph, but with a bias towards the seed nodes.

$$\begin{aligned} p(t+1) &= (1-\alpha) * M * p(t) + \alpha * p_0 \\ p(t+1) &= \text{updated PageRank} \\ p(t) &= \text{vector of PageRank scores at time}t \\ \alpha &= \text{random jump probability} \\ M &= \text{transition matrix of the graph} \\ p_0 &= \text{personalized node or seed node's probability vector} \end{aligned}$$

4.5 Approximate Multidimensional Indexing (LSH)

Multidimensional indexing refers to the process of efficiently accessing and retrieving data in a multidimensional array or dataset. This is a common operation in various fields, such as databases, data science, and image processing. When dealing with large datasets, the efficiency of indexing becomes crucial for performance.
Approximate multidimensional indexing involves methods that provide fast and efficient access to data, even if the result is not exact but within an acceptable range or

approximation. This is particularly useful in scenarios where exact matches are not required or the data is inherently imprecise.
In this phase of the project, we specifically focus on implementing the following approximate multidimensional indexing for Task 4:

Locality-Sensitive Hashing (for Euclidean distance measure):
Locality-Sensitive Hashing (LSH) [6] is a technique used to approximate near neighbors in high-dimensional spaces. When dealing with Euclidean distances, LSH can be applied to find approximate nearest neighbors efficiently [7], which is valuable in scenarios where exact matches are computationally expensive or impractical. In contrast to conventional hashing functions, which attempt to minimize collisions, hash functions for LSH attempt to maximize collisions. The basic idea is to hash similar points to the same "buckets" with high probability.

Our implementation of Locality-Sensitive Hashing for Euclidean distances involves projecting high-dimensional vectors onto lower-dimensional subspaces and then quantizing the values to form hash codes. One common approach is to use random hyperplanes for the projection. The general formula used is:

$$h(x) = \left\lfloor \frac{a \cdot x + b}{w} \right\rfloor$$

where,

$x =$ input vector with dimensions m

$a =$ random vector with dimensions m (hash function hyperplane)

$b =$ random offset term

$w =$ bin width

More precisely, the total number of bins we create depends on the maximum and minimum projection values of our input vector set with the random hyperplane and the bin width.
To make the algorithm more robust, we create L layers of hash functions of count h, each having a bin width w. A hash code is generated for each input vector per layer. At query time, any image with a hash code similar to the input query image (from any of the L layers) is added to the candidate set. The images in the candidate are set in increasing order of their Euclidean distances with respect to the input image.

4.6 Relevance Feedback

Relevance feedback is a concept used in information retrieval systems [8], such as search engines, to improve the precision and relevance of search results based on user feedback. The primary goal is to enhance the user experience by refining the search

results to match the user's information needs better. Relevance feedback systems typically involve an iterative process incorporating user feedback to adjust and optimize the search algorithm. In this phase of the project, we specifically focus on implementing two distinct types of relevance feedback systems for the tag set {"Very Relevant (R+)", "Relevant (R)", "Irrelevant (I)", and "Very Irrelevant (I-)}:

1. **Support Vector Machine (SVM)-based Relevance Feedback Systems:** Support Vector Machines (SVMs) are supervised machine learning models used for classification and regression tasks. SVMs are particularly effective in high-dimensional spaces and well-suited for linear and non-linear problems. The fundamental idea behind SVMs is to find a hyperplane that best separates data points from different classes. SVMs can also efficiently handle non-linear decision boundaries by using a kernel function. The kernel trick allows SVMs to implicitly map input data into a higher-dimensional space (when required) where a hyperplane can separate the classes.
   Since we have 4 relevant tags for our relevance feedback system, we create multiple SVMs, classifying each particular tag in our tag set. For classification tasks, the following are the formulae used to implement SVMs:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \max(0, 1 - y_i(w \cdot x_i + b))$$
$$\text{subject to } y_i(w \cdot x_i + b) \geq 1$$
$$\text{where,}$$
$$w - \text{weight vector}$$
$$x = \text{input vector}$$
$$b = \text{bias}$$
$$y = \text{class label}$$
$$C = \text{regularization parameter}$$

   We primarily implement the Radial Basis (Gaussian) Kernel Function defined by:

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

2. **Probabilistic Relevance Feedback Systems:**
   We use the Probabilistic Feature Term Readjustment for incorporating feedback into the results generated. In this case, we calculate significance of each feature (f_k) in the manually labelled results. This is given by the equation:

$$sig(f_k) = log\left(\frac{\frac{p(f_k|R)}{1-p(f_k|R)} + \alpha\frac{p(f_k|VR)}{1-p(f_k|VR)}}{\frac{p(f_k|IR)}{1-p(f_k|IR)} + \alpha\frac{p(f_k|VIR)}{1-p(f_k|VIR)}}\right)$$

$$\begin{aligned} p(f_k|R) &= \text{Probability of feature given relevant label} \\ p(f_k|VR) &= \text{Probability of feature given very relevant label} \\ p(f_k|IR) &= \text{Probability of feature given irrelevant label} \\ p(f_k|VIR) &= \text{Probability of feature given very irrelevant label} \\ \alpha &= \text{Scalar Weight} \end{aligned}$$

The equation only works with binary feature spaces, so we convert the feature space into binary by using thresholding with $f_k = 1$ when $f_k > 0.5$ and $f_k = 0$ otherwise

To find the updated query vector we use the equation:
$q_{i+1} = q_i + \beta sig(f) q_i$

$$\begin{aligned} q_{i+1} &= \text{New query vector for (i+1)-th iteration} \\ q_i &= \text{Query vector for the i-th iteration} \\ sig(f) &= \text{Significance vector for the feedback} \\ \beta &= \text{Scalar weight} \end{aligned}$$

4.7 Accuracy Measures

We use the following accuracy measures [10] throughout most of the tasks for this phase of the project.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{True Negatives} + \text{False Positives} + \text{False Negatives}}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5 The Tasks

5.1 Task 0

5.1.1 Task 0a - Goal Description:

Implement a program which computes and prints the "inherent dimensionality" associated with the even-numbered Caltech101 images.

5.1.2 Task 0a - Assumptions Made:

1. We assume that we can use any feature vectors extracted from Phase 1 as the base matrix to demo inherent dimensionality calculation.

2. No prior knowledge of output requirements was provided, so we showcase results for the inherent dimensionality of Caltech101 Image feature vectors of our choice.

5.1.3 Task 0a - Interface Specifications:

1. We allow the user to choose any of the feature vectors extracted from Phase 1 of the Project, i.e. color_moments, hog, avgpool, layer3, fc, or resnet.

5.1.4 Task 0a - Implementation Approach:

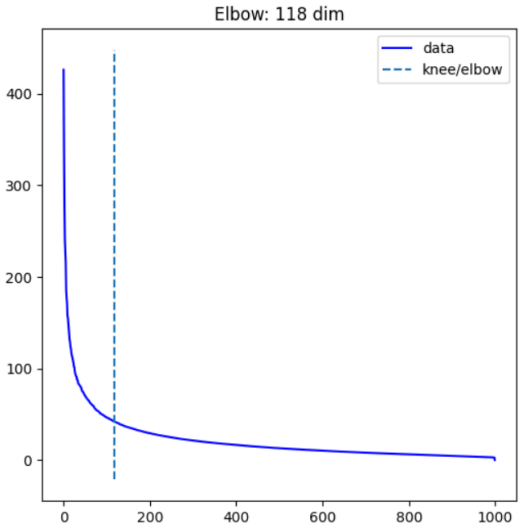
1. We use Singular Value Decomposition (SVD) to retrieve the Core Matrix of our Feature Space. We chose SVD as it was a direct extension of Phase 2 of the Project. We experimented with the possibility of using Multi-Dimensional Scaling Reduction to solve this problem but ran into implementation bugs, particularly for the Feature Vector Matrices. Hence, we reverted to SVD, which was giving good results.

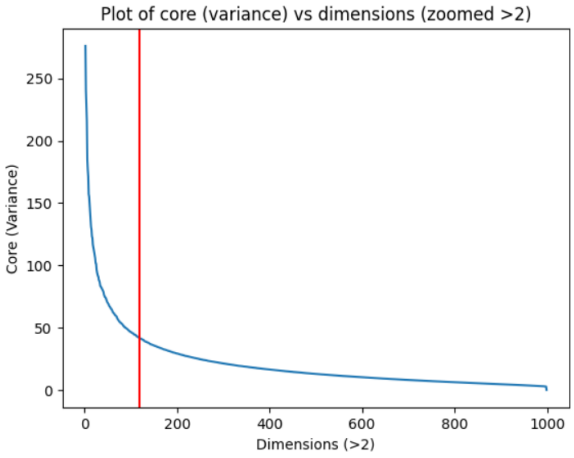
2. After calculating the core matrix, we extract the eigenvalues (diagonal of the core matrix) and pass it to the Scree-test which calculates the final inherent dimensionality of the original feature vector matrix (more details in 4).

5.1.5 Task 0a - Results:

Inherent Dimensionality of Caltech101 Even-Numbered Images - Feature Space: Resnet

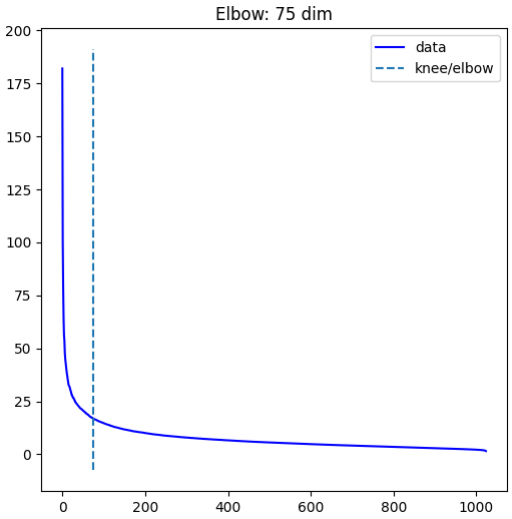
```
With 2 dimensions, we can explain 49.008692371182235% of the variance
With 118 dimensions, we can explain 90.74899586494222% of the variance
With 500 dimensions, we can explain 91.9179795686005% of the variance

Hence, the inherent dimensionality is approximately 118 dimensions for the chosen feature space: resnet
```

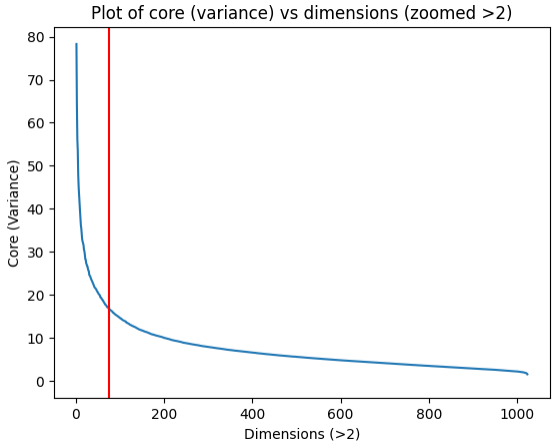




Inherent Dimensionality of Caltech101 Even-Numbered Images - Feature Space: Avgpool

```
With 2 dimensions, we can explain 36.303410165381344% of the variance
With 75 dimensions, we can explain 75.15973463317268% of the variance
With 500 dimensions, we can explain 81.36717921873489% of the variance

Hence, the inherent dimensionality is approximately 75 dimensions for the chosen feature space: avgpool
```





5.1.6 Task 0a - Analysis of results:

We noticed that different feature models provided different inherent dimensionality as follows:

1. ResNet, FC: ≈115

2. AvgPool: ≈75

3. Layer3, Color, HOG: ≈30

From this, we conclude that ResNet and FC had extracted features from the initial images, which were not present in models like Color and HOG, and that is the reason for the higher inherent dimensionality that they possessed. This could be due to the later layers of the Resnet based feature spaces viewing the entire image and extracting relevant features, unlike the local grid based Color and HOG feature spaces.

5.1.7 Task 0b - Goal Description:

Implement a program which computes and prints the "inherent dimensionality" associated with each unique label of the even-numbered Caltech101 images.

5.1.8 Task 0b - Assumptions Made:

1. We assume that we can use any feature vectors of our choice extracted from Phase 1 as the base matrix to demo inherent dimensionality calculation.

2. No prior knowledge of output requirements was provided, so we showcase results for the inherent dimensionality of unique label feature vectors of our choice.

5.1.9 Task 0b - Interface Specifications:

1. We allow the user to choose any of the feature vectors extracted from Phase 1 of the Project, i.e. color_moments, hog, avgpool, layer3, fc, or resnet.

5.1.10 Task 0b - Implementation Approach:

1. We filter out the feature space vectors for each label l in the Caltech101 Dataset.

2. We use Singular Value Decomposition (SVD) to retrieve the Core Matrix of our Label-specific Feature Vectors. We chose SVD as it was a direct extension of Phase 2 of the Project. We experimented with the possibility of using Multi-Dimensional Scaling Reduction to solve this problem but ran into implementation bugs, particularly for the Feature Vector Matrices. Hence, we reverted to SVD, which was giving good results (similar explanation to task 0a).

3. After calculating the core matrix, we extract the eigenvalues (diagonal of the core matrix) and pass it to the scree-test which calculates the final inherent dimensionality of the original feature vector matrix (similar explanation to task 0a).

5.1.11 Task 0b - Results:

Inherent Dimensionality per label - Feature Space: resnet (Truncated Output)

```
Label: Faces            Inherent Dimensionality: 20     % Variance explained: 93.78331751591523
Label: Faces_easy       Inherent Dimensionality: 14     % Variance explained: 93.93544402673689
Label: Leopards         Inherent Dimensionality: 11     % Variance explained: 96.16377857704835
Label: Motorbikes       Inherent Dimensionality: 21     % Variance explained: 94.62915955469623
Label: accordion        Inherent Dimensionality: 4      % Variance explained: 86.65468354127151
Label: airplanes        Inherent Dimensionality: 21     % Variance explained: 93.55846100716178
Label: anchor           Inherent Dimensionality: 4      % Variance explained: 80.61745070873836
Label: ant              Inherent Dimensionality: 5      % Variance explained: 85.77002661948676
Label: barrel           Inherent Dimensionality: 3      % Variance explained: 75.95182218282488
Label: bass             Inherent Dimensionality: 9      % Variance explained: 92.65954900703143
Label: beaver           Inherent Dimensionality: 4      % Variance explained: 82.1475444256416
Label: binocular        Inherent Dimensionality: 3      % Variance explained: 81.15679760458971
Label: bonsai           Inherent Dimensionality: 8      % Variance explained: 86.9665648038658
Label: brain            Inherent Dimensionality: 6      % Variance explained: 84.53058486837847
Label: brontosaurus     Inherent Dimensionality: 5      % Variance explained: 84.1025352383262
Label: buddha           Inherent Dimensionality: 5      % Variance explained: 86.26468607540832
Label: butterfly        Inherent Dimensionality: 4      % Variance explained: 79.18621850005533
Label: camera           Inherent Dimensionality: 6      % Variance explained: 91.16039850372576
Label: cannon           Inherent Dimensionality: 4      % Variance explained: 83.70395310843844
Label: car_side         Inherent Dimensionality: 2      % Variance explained: 93.4016672010545 6
Label: ceiling_fan      Inherent Dimensionality: 3      % Variance explained: 82.33760201127478
Label: cellphone        Inherent Dimensionality: 1      % Variance explained: 75.40692850060772
Label: chair            Inherent Dimensionality: 7      % Variance explained: 87.48581270461175
Label: chandelier       Inherent Dimensionality: 5      % Variance explained: 81.80430719499014
Label: cougar_body      Inherent Dimensionality: 4      % Variance explained: 85.18774393115629
...
Label: wild_cat         Inherent Dimensionality: 4      % Variance explained: 87.49668415094598
Label: windsor_chair    Inherent Dimensionality: 1      % Variance explained: 85.24553070744449
Label: wrench           Inherent Dimensionality: 3      % Variance explained: 84.21457382679606
Label: yin_yang         Inherent Dimensionality: 1      % Variance explained: 75.62758641769871
```

Inherent Dimensionality per label - Feature Space: avgpool (Truncated Output)

```
--------- Per-label Inherent Dimensionality Results for Feature Space: avgpool ----------

Label: Faces            Inherent Dimensionality: 19     % Variance explained: 88.12376020774337
Label: Faces_easy       Inherent Dimensionality: 11     % Variance explained: 86.05407785702234
Label: Leopards         Inherent Dimensionality: 8      % Variance explained: 91.67070094595496
Label: Motorbikes       Inherent Dimensionality: 23     % Variance explained: 88.52407956437666
Label: accordion        Inherent Dimensionality: 2      % Variance explained: 72.02324603639767
Label: airplanes        Inherent Dimensionality: 22     % Variance explained: 87.10176638422746
Label: anchor           Inherent Dimensionality: 2      % Variance explained: 61.536004680077774
Label: ant              Inherent Dimensionality: 3      % Variance explained: 67.26481457533535
Label: barrel           Inherent Dimensionality: 3      % Variance explained: 63.86743134338424
Label: bass             Inherent Dimensionality: 4      % Variance explained: 73.9520880516684
Label: beaver           Inherent Dimensionality: 3      % Variance explained: 68.35893655900303
Label: binocular        Inherent Dimensionality: 2      % Variance explained: 67.88095028985774
Label: bonsai           Inherent Dimensionality: 6      % Variance explained: 70.80962860220183
Label: brain            Inherent Dimensionality: 4      % Variance explained: 69.49451936422678
Label: brontosaurus     Inherent Dimensionality: 5      % Variance explained: 76.68617524188373
Label: buddha           Inherent Dimensionality: 3      % Variance explained: 68.28386273283355
Label: butterfly        Inherent Dimensionality: 5      % Variance explained: 70.02570432721132
Label: camera           Inherent Dimensionality: 3      % Variance explained: 70.082921289620 71
Label: cannon           Inherent Dimensionality: 4      % Variance explained: 73.50758488031775
Label: car_side         Inherent Dimensionality: 2      % Variance explained: 82.88534794537436
Label: ceiling_fan      Inherent Dimensionality: 2      % Variance explained: 72.5090109303024
Label: cellphone        Inherent Dimensionality: 1      % Variance explained: 60.46665378098483
Label: chair            Inherent Dimensionality: 5      % Variance explained: 72.06430813887306
...
Label: wild_cat         Inherent Dimensionality: 2      % Variance explained: 71.42910671963206
Label: windsor_chair    Inherent Dimensionality: 2      % Variance explained: 76.17219235171866
Label: wrench           Inherent Dimensionality: 3      % Variance explained: 21.50098743104918
Label: yin_yang         Inherent Dimensionality: 3      % Variance explained: 71.42307614656833
```

5.1.12 Task 0b - Analysis of results:

We noticed that a few labels like Faces and motorbikes had inherent dimensionality around 20, while most other labels had lesser than 10 inherent dimensionality. This could be because faces have more details and require more dimentions to be represented.

5.2 Task 1

5.2.1 Goal Description:

Implement a program which,

1. for each unique label l, computes the corresponding k latent semantics (of your choice) associated with the even-numbered Caltech101 images, and

2. for the odd-numbered images, predicts the most likely labels using distances/similarities computed under the label-specific latent semantics.

The system should also output per-label precision, recall, and F1-score values, as well as output an overall accuracy value.

5.2.2 Assumptions Made:

1. During experimentation, we assume that we can calculate the latent semantics using any feature vector space and dimensionality reduction technique of our choice.

2. During experimentation, we also decide the value k in our k latent semantics.

3. We assume that if our classifier predicts Faces_easy it also predicts Faces since the labels are very similar. We do this to the classes that contain the names of other classes at the beginning of that class.

5.2.3 Interface Specifications:

1. For better interactivity, we decided to allow the user to enter the feature space and k value for the number of latent semantics.

5.2.4 Implementation Approach:

1. We dimensionally reduce the feature vectors (ex: Resnet) for each specific label in the Caltech101 Dataset using Singular Value Decomposition (SVD). We experimented with different reducers, but leveraging our experience from Phase 2 implementations, we opted to stick with SVD due to best performance.

2. We decided to use k = 118 since that was the inherent dimensionality arrived at for the Resnet feature space in Task 0.

3. For the even images, we run K-Nearest Neighbor clustering with k=5 to generate 5 clusters, and compute the mean vector of each cluster, resulting in 5 representative vectors per label.

4. Using the label representative vectors, we predict the labels of each of the odd-numbered images based on the unique label ranker we developed for Phase 2 of this Course Project. The top 3 predicted labels are printed. Only the closest label is used for calculating performance scores.

5.2.5 Results:

k-Latent Semantics: Prediction Results per Label for: k = 5, Feature Space = Resnet, Dimensionality Reduction = SVD (Truncated Output)

```
IMG_ID: 1       True label: Faces       Pred labels: ('Faces', 'lobster', 'dalmatian')
IMG_ID: 3       True label: Faces       Pred labels: ('electric_guitar', 'Faces', 'saxophone')
IMG_ID: 5       True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 7       True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'euphonium')
IMG_ID: 9       True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'euphonium')
IMG_ID: 11      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 13      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'electric_guitar')
IMG_ID: 15      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 17      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'electric_guitar')
IMG_ID: 19      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 21      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'euphonium')
IMG_ID: 23      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'euphonium')
IMG_ID: 25      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'euphonium')
IMG_ID: 27      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 29      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'euphonium')
IMG_ID: 31      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'euphonium')
IMG_ID: 33      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'electric_guitar')
IMG_ID: 35      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'euphonium')
IMG_ID: 37      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'electric_guitar')
IMG_ID: 39      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'euphonium')
IMG_ID: 41      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'euphonium')
IMG_ID: 43      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'euphonium')
IMG_ID: 45      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'euphonium')
IMG_ID: 47      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'euphonium')
IMG_ID: 49      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'euphonium')
 ...
IMG_ID: 8669    True label: yin_yang    Pred labels: ('dolphin', 'mandolin', 'crocodile')
IMG_ID: 8671    True label: yin_yang    Pred labels: ('euphonium', 'panda', 'camera')
IMG_ID: 8673    True label: yin_yang    Pred labels: ('euphonium', 'metronome', 'headphone')
IMG_ID: 8675    True label: yin_yang    Pred labels: ('scissors', 'wrench', 'euphonium')
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

k-Latent Semantics: Accuracy Scores per Label for: k = 5, Feature Space = Resnet, Dimensionality Reduction = SVD (Truncated Output)

```
Per-label Precision, Recall, F1 Scores:

Label: Faces            Precision: 0.9908       Recall: 0.9908   F1 score:0.9908
Label: Faces_easy       Precision: 1.0          Recall: 0.9862   F1 score:0.9931
Label: Leopards         Precision: 0.5417       Recall: 0.39     F1 score:0.4535
Label: Motorbikes       Precision: 0.9763       Recall: 0.9298   F1 score:0.9525
Label: accordion        Precision: 0.08         Recall: 0.1481   F1 score:0.1039
Label: airplanes        Precision: 0.9636       Recall: 0.9275   F1 score:0.9452
Label: anchor           Precision: 0.0769       Recall: 0.0952   F1 score:0.0851
Label: ant              Precision: 0.037        Recall: 0.0476   F1 score:0.0417
Label: barrel           Precision: 0.0          Recall: 0.0      F1 score:0
Label: bass             Precision: 0.0645       Recall: 0.0741   F1 score:0.069
Label: beaver           Precision: 0.0488       Recall: 0.087    F1 score:0.0625
Label: binocular        Precision: 0.0          Recall: 0.0      F1 score:0
Label: bonsai           Precision: 0.3333       Recall: 0.1719   F1 score:0.2268
Label: brain            Precision: 0.2647       Recall: 0.1837   F1 score:0.2169
Label: brontosaurus     Precision: 0.0345       Recall: 0.0455   F1 score:0.0392
Label: buddha           Precision: 0.1714       Recall: 0.1429   F1 score:0.1558
Label: butterfly        Precision: 0.0769       Recall: 0.0435   F1 score:0.0556
Label: camera           Precision: 0.2143       Recall: 0.24     F1 score:0.2264
Label: cannon           Precision: 0.0909       Recall: 0.1429   F1 score:0.1111
Label: car_side         Precision: 0.9118       Recall: 1.0      F1 score:0.9538
Label: ceiling_fan      Precision: 0.0541       Recall: 0.087    F1 score:0.0667
Label: cellphone        Precision: 0.2121       Recall: 0.2333   F1 score:0.2222
Label: chair            Precision: 0.0          Recall: 0.0      F1 score:0
 ...
Label: yin_yang         Precision: 0.1667       Recall: 0.1      F1 score:0.125


Overall accuracy: 39.60350391885662%
```

k-Latent Semantics: Prediction Results per Label for: k = 118, Feature Space = Resnet, Dimensionality Reduction = SVD (Truncated Output)

```
IMG_ID: 7737    True label: stegosaurus Pred labels: ('stegosaurus', 'brontosaurus', 'rhino')
IMG_ID: 7739    True label: stegosaurus Pred labels: ('stegosaurus', 'brontosaurus', 'starfish')
IMG_ID: 7741    True label: stegosaurus Pred labels: ('stegosaurus', 'brontosaurus', 'crocodile_head')
IMG_ID: 7743    True label: stegosaurus Pred labels: ('stegosaurus', 'elephant', 'brontosaurus')
IMG_ID: 7745    True label: stegosaurus Pred labels: ('stegosaurus', 'beaver', 'elephant')
IMG_ID: 7747    True label: stegosaurus Pred labels: ('rhino', 'elephant', 'stegosaurus')
IMG_ID: 7749    True label: stegosaurus Pred labels: ('stegosaurus', 'rhino', 'beaver')
IMG_ID: 7751    True label: stegosaurus Pred labels: ('stegosaurus', 'rhino', 'elephant')
IMG_ID: 7753    True label: stegosaurus Pred labels: ('stegosaurus', 'crocodile_head', 'sea_horse')
IMG_ID: 7755    True label: stegosaurus Pred labels: ('stegosaurus', 'rooster', 'water_lilly')
IMG_ID: 7757    True label: stegosaurus Pred labels: ('stegosaurus', 'octopus', 'crocodile')
IMG_ID: 7759    True label: stop_sign   Pred labels: ('stop_sign', 'bass', 'umbrella')
IMG_ID: 7761    True label: stop_sign   Pred labels: ('yin_yang', 'stop_sign', 'snoopy')
IMG_ID: 7763    True label: stop_sign   Pred labels: ('stop_sign', 'umbrella', 'bass')
IMG_ID: 7765    True label: stop_sign   Pred labels: ('stop_sign', 'car_side', 'wrench')
IMG_ID: 7767    True label: stop_sign   Pred labels: ('stop_sign', 'bass', 'platypus')
IMG_ID: 7769    True label: stop_sign   Pred labels: ('stop_sign', 'bass', 'dollar_bill')
IMG_ID: 7771    True label: stop_sign   Pred labels: ('stop_sign', 'car_side', 'anchor')
IMG_ID: 7773    True label: stop_sign   Pred labels: ('stop_sign', 'umbrella', 'yin_yang')
IMG_ID: 7775    True label: stop_sign   Pred labels: ('stop_sign', 'car_side', 'anchor')
IMG_ID: 7777    True label: stop_sign   Pred labels: ('stop_sign', 'yin_yang', 'menorah')
```

k-Latent Semantics: Accuracy Scores per Label for: k = 118, Feature Space = Resnet, Dimensionality Reduction = SVD (Truncated Output)

```
Label: soccer_ball      Precision: 0.9655       Recall: 0.875    F1 score:0.918
Label: stapler          Precision: 0.9474       Recall: 0.7826   F1 score:0.8571
Label: starfish         Precision: 0.9333       Recall: 0.9767   F1 score:0.9545
Label: stegosaurus      Precision: 0.8667       Recall: 0.8966   F1 score:0.8814
Label: stop_sign        Precision: 0.9688       Recall: 0.9688   F1 score:0.9688
Label: strawberry       Precision: 1.0          Recall: 0.7778   F1 score:0.875
Label: sunflower        Precision: 0.95         Recall: 0.9048   F1 score:0.9268
Label: tick             Precision: 0.9615       Recall: 1.0      F1 score:0.9804
Label: trilobite        Precision: 0.9545       Recall: 0.9767   F1 score:0.9655
Label: umbrella         Precision: 1.0          Recall: 0.8378   F1 score:0.9118
Label: watch            Precision: 0.9831       Recall: 0.9667   F1 score:0.9748
Label: water_lilly      Precision: 0.4737       Recall: 0.5      F1 score:0.4865
Label: wheelchair       Precision: 0.8667       Recall: 0.8667   F1 score:0.8667
Label: wild_cat         Precision: 0.875        Recall: 0.8235   F1 score:0.8485
Label: windsor_chair    Precision: 0.7714       Recall: 0.9643   F1 score:0.8571
Label: wrench           Precision: 0.6875       Recall: 0.5789   F1 score:0.6286
Label: yin_yang         Precision: 0.7931       Recall: 0.7667   F1 score:0.7797


Overall accuracy: 90.73305670816045%
```

5.2.6 Task 1 - Analysis of results:

We consistently had bad results during implementing this task. However during experimentation, we noticed that using the core matrix of the even objects to reduce features produced better results than generating a new core matrix from all the odd objects. This resulted in a jump in accuracy of about 60 percent. This could be because when we reduce the odd images, we are utilizing the exact same latent space as the even images thereby reducing losses. Additionally through testing with multiple feature spaces, we noticed the best results with the Resnet feature space.

5.3 Task 2

5.3.1 Goal Description:

Implement a program which,

1. for each unique label l, computes the corresponding c most significant clusters associated with the even-numbered Caltech101 images (using DBScan algorithm); the resulting clusters should be visualized both
   (a) as differently colored point clouds in a 2-dimensional MDS space, and
   (b) as groups of image thumbnails.
2. for the odd-numbered images, predicts the most likely labels using the c label-specific clusters.

The system should also output per-label precision, recall, and F1-score values, as well as output an overall accuracy value.

5.3.2 Assumptions Made:

1. We assume that we can use any feature vectors or latent semantics extracted from Phase 1 / 2 as the base matrix to demo DBScan Clustering.
2. We assume that if our classifier predicts Faces_easy it also predicts Faces since the labels are very similar. We do this to the classes that contain the names of other classes at the beginning of that class.
3. We assume that the evaluation criteria is the classification precision, recall, f1 scores per label and overall accuracy, and not necessarily the DBSCAN clustering performance. The design considerations were in accordance to this.

5.3.3 Interface Specifications:

1. We take in the value of c (most significant cluster count) from the user.
2. After we calculate the significant clusters for each label l in the dataset, we provide the user to either display the per-label overall accuracy scores of our implementation OR visualize the point cloud and image thumbnails for a specific input label l from the user.

5.3.4 Implementation Approach:

1. We run DBSCAN on the subset of vectors for each unique label.
2. The EPS value, which is the maximum distance within which two vectors are said to be within the same dense cluster, and MIN PTS, which is the minumum number of vectors within EPS to call the vector under consideration a core vector, are computed using an optimization algorithm.
3. The way these parameters are found per label is by starting with an EPS of the minimum distance between any two vectors, and Initially choose MIN PTS to be square root of max possible points per cluster C.
4. We see how many clusters result using these parameters, and then increase EPS by a factor of 1.05 upto the the median distance between any two vectors.
5. We then incrementally decrease MIN PTS upto 2, and repeat the process.
6. We take the parameters that resulted in the number of clusterings closest to C.
7. We then transform the vectors to 2D MDS space by considering pairwise distances. Based on the cluster label predicted by DBSCAN, we visualize this in a 2D scatter plot. -1 indicates a noise vector, not considering to be a cluster.
8. We also group all vectors belonging to the same label, and visualize the thumbnails.
9. We then use a closest core vector policy to predict the labels of odd images in the chosen feature space.
10. Using these classifications, and the known label of the odd images, we compute per label precision, recall, f1 score and the overall accuracy.

5.3.5 Results:

DBScan Clustering - Number of Cluster per Label for: C = 5, Feature Space = Resnet (Truncated Output)

```
Per-label Best Clustering Results using DBSCAN for C = 5

1 of 101        Best clustering: 5 clusters     Label: Faces
2 of 101        Best clustering: 5 clusters     Label: Faces_easy
3 of 101        Best clustering: 5 clusters     Label: Leopards
4 of 101        Best clustering: 4 clusters     Label: Motorbikes
5 of 101        Best clustering: 2 clusters     Label: accordion
6 of 101        Best clustering: 5 clusters     Label: airplanes
7 of 101        Best clustering: 2 clusters     Label: anchor
8 of 101        Best clustering: 2 clusters     Label: ant
9 of 101        Best clustering: 2 clusters     Label: barrel
10 of 101       Best clustering: 1 clusters     Label: bass
11 of 101       Best clustering: 2 clusters     Label: beaver
12 of 101       Best clustering: 2 clusters     Label: binocular
13 of 101       Best clustering: 5 clusters     Label: bonsai
14 of 101       Best clustering: 5 clusters     Label: brain
15 of 101       Best clustering: 1 clusters     Label: brontosaurus
16 of 101       Best clustering: 3 clusters     Label: buddha
17 of 101       Best clustering: 2 clusters     Label: butterfly
18 of 101       Best clustering: 2 clusters     Label: camera
19 of 101       Best clustering: 2 clusters     Label: cannon
20 of 101       Best clustering: 4 clusters     Label: car_side
21 of 101       Best clustering: 3 clusters     Label: ceiling_fan
22 of 101       Best clustering: 2 clusters     Label: cellphone
23 of 101       Best clustering: 2 clusters     Label: chair
...
98 of 101       Best clustering: 1 clusters     Label: wild_cat
99 of 101       Best clustering: 4 clusters     Label: windsor_chair
100 of 101      Best clustering: 2 clusters     Label: wrench
101 of 101      Best clustering: 3 clusters     Label: yin_yang
```

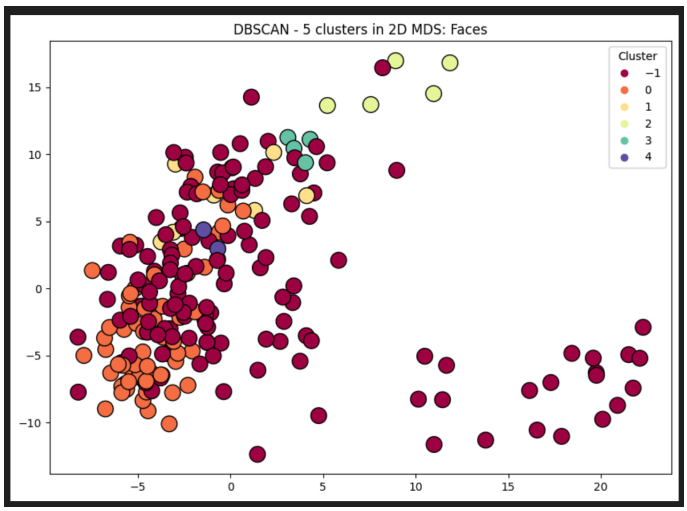
DBScan Clustering - Prediction Results per Label for: C = 5, Feature Space = Resnet (Truncated Output)

```
Running Predictions for DBSCAN for C = 5
IMG_ID: 1       True label: Faces       Pred labels: ('Faces', 'accordion', 'nautilus')
IMG_ID: 3       True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'crocodile_head')
IMG_ID: 5       True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'scorpion')
IMG_ID: 7       True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'llama')
IMG_ID: 9       True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'llama')
IMG_ID: 11      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'scorpion')
IMG_ID: 13      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'llama')
IMG_ID: 15      True label: Faces       Pred labels: ('Faces_easy', 'scorpion', 'crocodile')
IMG_ID: 17      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'llama')
IMG_ID: 19      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'accordion')
IMG_ID: 21      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 23      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 25      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 27      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 29      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 31      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 33      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'llama')
IMG_ID: 35      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 37      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 39      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 41      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'crocodile_head')
IMG_ID: 43      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'crocodile_head')
IMG_ID: 45      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'scorpion')
IMG_ID: 47      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
...
IMG_ID: 8669    True label: yin_yang    Pred labels: ('anchor', 'scissors', 'nautilus')
IMG_ID: 8671    True label: yin_yang    Pred labels: ('yin_yang', 'soccer_ball', 'brain')
IMG_ID: 8673    True label: yin_yang    Pred labels: ('yin_yang', 'brain', 'soccer_ball')
IMG_ID: 8675    True label: yin_yang    Pred labels: ('scissors', 'yin_yang', 'octopus')
```

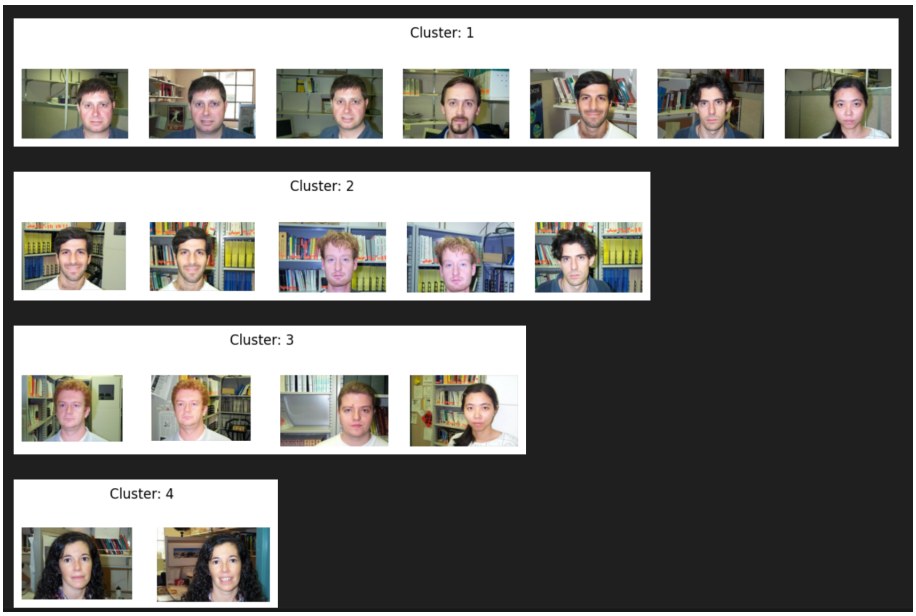
DBScan Clustering - Accuracy Scores per Label for C = 5, Feature Space = Resnet (Truncated Output)

```
Per-label Precision, Recall, F1 Scores for DBSCAN for C = 5:

Label: Faces            Precision: 1.0          Recall: 0.9816   F1 score:0.9907
Label: Faces_easy       Precision: 1.0          Recall: 0.9862   F1 score:0.9931
Label: Leopards         Precision: 0.9804       Recall: 1.0      F1 score:0.9901
Label: Motorbikes       Precision: 1.0          Recall: 0.9799   F1 score:0.9899
Label: accordion        Precision: 1.0          Recall: 0.963    F1 score:0.9811
Label: airplanes        Precision: 0.9888       Recall: 0.885    F1 score:0.934
Label: anchor           Precision: 0.6          Recall: 0.4286   F1 score:0.5
Label: ant              Precision: 0.7727       Recall: 0.8095   F1 score:0.7907
Label: barrel           Precision: 0.9583       Recall: 0.9583   F1 score:0.9583
Label: bass             Precision: 1.0          Recall: 0.7407   F1 score:0.8511
Label: beaver           Precision: 0.8261       Recall: 0.8261   F1 score:0.8261
Label: binocular        Precision: 0.8          Recall: 1.0      F1 score:0.8889
Label: bonsai           Precision: 0.9138       Recall: 0.8281   F1 score:0.8689
Label: brain            Precision: 0.7627       Recall: 0.9184   F1 score:0.8333
Label: brontosaurus     Precision: 1.0          Recall: 0.4091   F1 score:0.5806
Label: buddha           Precision: 0.9722       Recall: 0.8333   F1 score:0.8974
Label: butterfly        Precision: 0.9565       Recall: 0.4783   F1 score:0.6377
Label: camera           Precision: 1.0          Recall: 0.96     F1 score:0.9796
Label: cannon           Precision: 0.7826       Recall: 0.8571   F1 score:0.8182
Label: car_side         Precision: 0.9841       Recall: 1.0      F1 score:0.992
Label: ceiling_fan      Precision: 0.9375       Recall: 0.6522   F1 score:0.7692
Label: cellphone        Precision: 0.9333       Recall: 0.9333   F1 score:0.9333
Label: chair            Precision: 0.6          Recall: 0.3871   F1 score:0.4706
...
Label: yin_yang         Precision: 0.9167       Recall: 0.7333   F1 score:0.8148


Overall accuracy for DBSCAN for C = 5: 87.59797141539888%
```

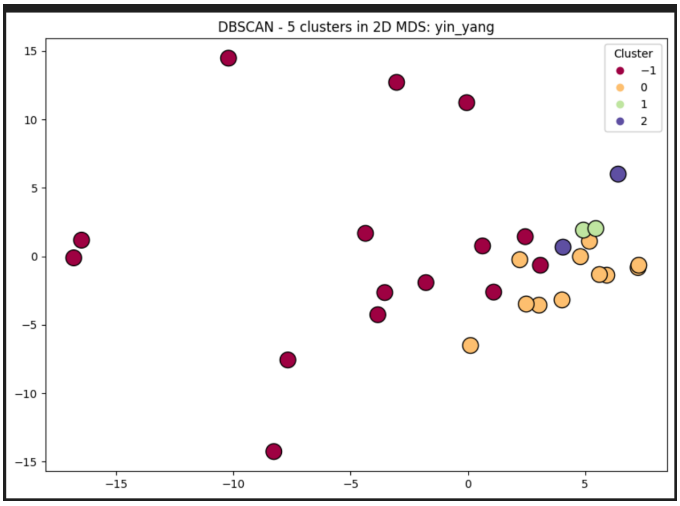
DBScan Clustering - Scatter Plot for Label = 0, C = 5, Feature Space = Resnet



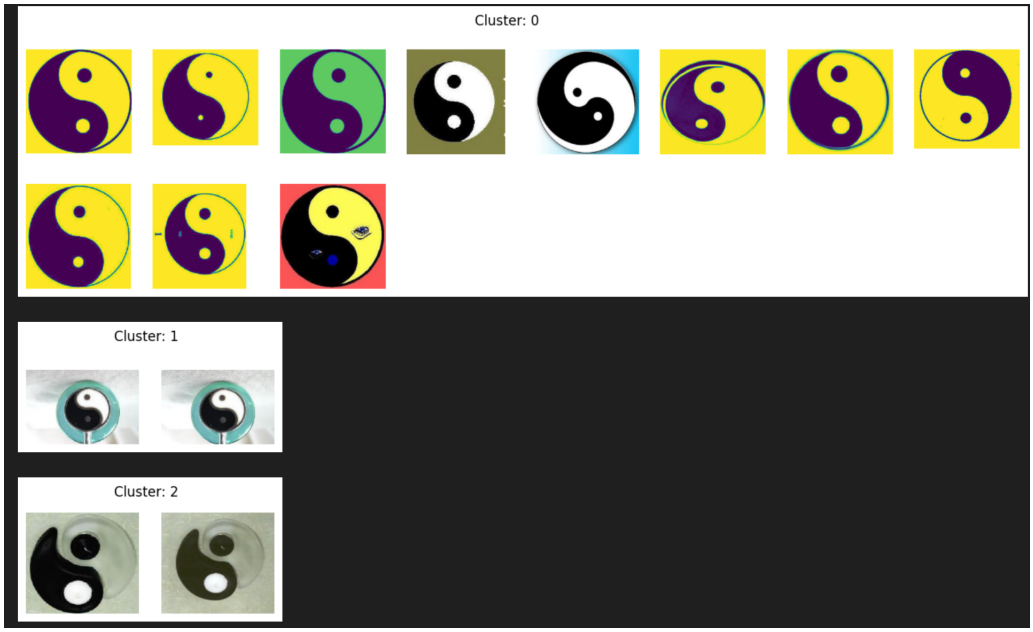
DBScan Clustering - Image Thumbnails for Label = 0, C = 5, Feature Space = Resnet



DBScan Clustering - Scatter Plot for Label = 100, C = 5, Feature Space = Resnet



DBScan Clustering - Image Thumbnails for Label = 100, C = 5, Feature Space = Resnet



DBScan Clustering - Number of Cluster per Label for: C = 10, Feature Space = Resnet (Truncated Output)

```
Per-label Best Clustering Results using DBSCAN for C = 10

1 of 101        Best clustering: 10 clusters    Label: Faces
2 of 101        Best clustering: 11 clusters    Label: Faces_easy
3 of 101        Best clustering: 8 clusters     Label: Leopards
4 of 101        Best clustering: 10 clusters    Label: Motorbikes
5 of 101        Best clustering: 2 clusters     Label: accordion
6 of 101        Best clustering: 9 clusters     Label: airplanes
7 of 101        Best clustering: 2 clusters     Label: anchor
8 of 101        Best clustering: 2 clusters     Label: ant
9 of 101        Best clustering: 2 clusters     Label: barrel
10 of 101       Best clustering: 1 clusters     Label: bass
11 of 101       Best clustering: 2 clusters     Label: beaver
12 of 101       Best clustering: 2 clusters     Label: binocular
13 of 101       Best clustering: 6 clusters     Label: bonsai
14 of 101       Best clustering: 5 clusters     Label: brain
15 of 101       Best clustering: 1 clusters     Label: brontosaurus
16 of 101       Best clustering: 3 clusters     Label: buddha
17 of 101       Best clustering: 2 clusters     Label: butterfly
18 of 101       Best clustering: 2 clusters     Label: camera
19 of 101       Best clustering: 2 clusters     Label: cannon
20 of 101       Best clustering: 6 clusters     Label: car_side
21 of 101       Best clustering: 3 clusters     Label: ceiling_fan
22 of 101       Best clustering: 2 clusters     Label: cellphone
23 of 101       Best clustering: 2 clusters     Label: chair
...
98 of 101       Best clustering: 1 clusters     Label: wild_cat
99 of 101       Best clustering: 4 clusters     Label: windsor_chair
100 of 101      Best clustering: 2 clusters     Label: wrench
101 of 101      Best clustering: 3 clusters     Label: yin_yang
```

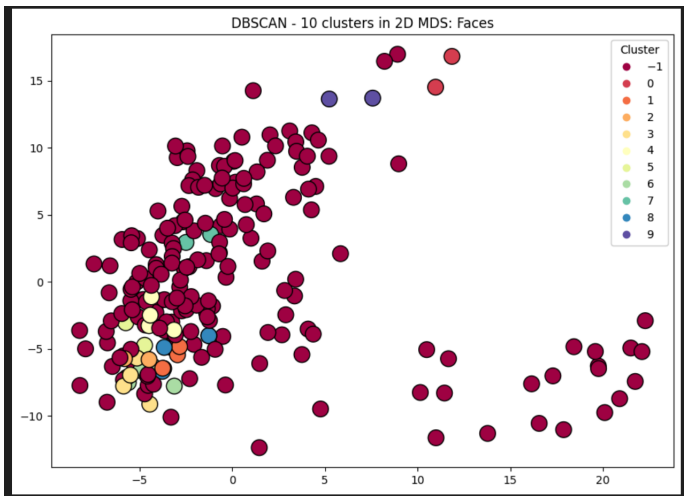
DBScan Clustering - Prediction Results per Label for: C = 10, Feature Space = Resnet (Truncated Output)

```
Running Predictions for DBSCAN for C = 10
IMG_ID: 1       True label: Faces       Pred labels: ('Faces', 'accordion', 'nautilus')
IMG_ID: 3       True label: Faces       Pred labels: ('Faces_easy', 'crocodile_head', 'Faces')
IMG_ID: 5       True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'scorpion')
IMG_ID: 7       True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'llama')
IMG_ID: 9       True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'llama')
IMG_ID: 11      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'scorpion')
IMG_ID: 13      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'llama')
IMG_ID: 15      True label: Faces       Pred labels: ('Faces_easy', 'scorpion', 'crocodile')
IMG_ID: 17      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'llama')
IMG_ID: 19      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'accordion')
IMG_ID: 21      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 23      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 25      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 27      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 29      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 31      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 33      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'llama')
IMG_ID: 35      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 37      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
IMG_ID: 39      True label: Faces       Pred labels: ('Faces_easy', 'Faces', 'lobster')
IMG_ID: 41      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'crocodile_head')
IMG_ID: 43      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'crocodile_head')
IMG_ID: 45      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'scorpion')
IMG_ID: 47      True label: Faces       Pred labels: ('Faces', 'Faces_easy', 'lobster')
...
IMG_ID: 8669    True label: yin_yang    Pred labels: ('anchor', 'scissors', 'nautilus')
IMG_ID: 8671    True label: yin_yang    Pred labels: ('yin_yang', 'soccer_ball', 'brain')
IMG_ID: 8673    True label: yin_yang    Pred labels: ('yin_yang', 'brain', 'soccer_ball')
IMG_ID: 8675    True label: yin_yang    Pred labels: ('scissors', 'yin_yang', 'octopus')
```

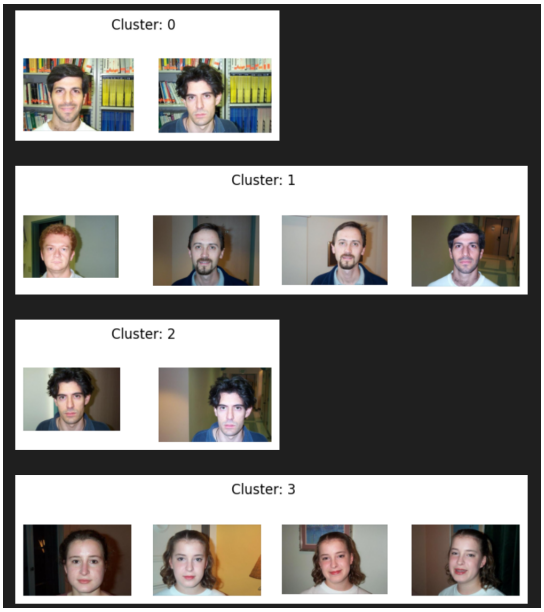
DBScan Clustering - Accuracy Scores per Label for C = 10, Feature Space = Resnet (Truncated Output)

```
Per-label Precision, Recall, F1 Scores for DBSCAN for C = 10:

Label: Faces            Precision: 1.0          Recall: 0.9677   F1 score:0.9836
Label: Faces_easy       Precision: 1.0          Recall: 0.9771   F1 score:0.9884
Label: Leopards         Precision: 0.9804       Recall: 1.0      F1 score:0.9901
Label: Motorbikes       Precision: 0.9876       Recall: 1.0      F1 score:0.9938
Label: accordion        Precision: 1.0          Recall: 0.963    F1 score:0.9811
Label: airplanes        Precision: 0.9863       Recall: 0.9      F1 score:0.9412
Label: anchor           Precision: 0.6          Recall: 0.4286   F1 score:0.5
Label: ant              Precision: 0.7727       Recall: 0.8095   F1 score:0.7907
Label: barrel           Precision: 0.9583       Recall: 0.9583   F1 score:0.9583
Label: bass             Precision: 1.0          Recall: 0.7407   F1 score:0.8511
Label: beaver           Precision: 0.8261       Recall: 0.8261   F1 score:0.8261
Label: binocular        Precision: 0.7619       Recall: 1.0      F1 score:0.8649
Label: bonsai           Precision: 0.9194       Recall: 0.8906   F1 score:0.9048
Label: brain            Precision: 0.7759       Recall: 0.9184   F1 score:0.8411
Label: brontosaurus     Precision: 1.0          Recall: 0.4091   F1 score:0.5806
Label: buddha           Precision: 0.9722       Recall: 0.8333   F1 score:0.8974
Label: butterfly        Precision: 0.9615       Recall: 0.5435   F1 score:0.6944
Label: camera           Precision: 1.0          Recall: 0.96     F1 score:0.9796
Label: cannon           Precision: 0.8571       Recall: 0.8571   F1 score:0.8571
Label: car_side         Precision: 0.9841       Recall: 1.0      F1 score:0.992
Label: ceiling_fan      Precision: 0.9375       Recall: 0.6522   F1 score:0.7692
Label: cellphone        Precision: 0.9333       Recall: 0.9333   F1 score:0.9333
Label: chair            Precision: 0.6          Recall: 0.3871   F1 score:0.4706
...
Label: yin_yang         Precision: 0.9167       Recall: 0.7333   F1 score:0.8148


Overall accuracy for DBSCAN for C = 10: 87.89764868603042%
```

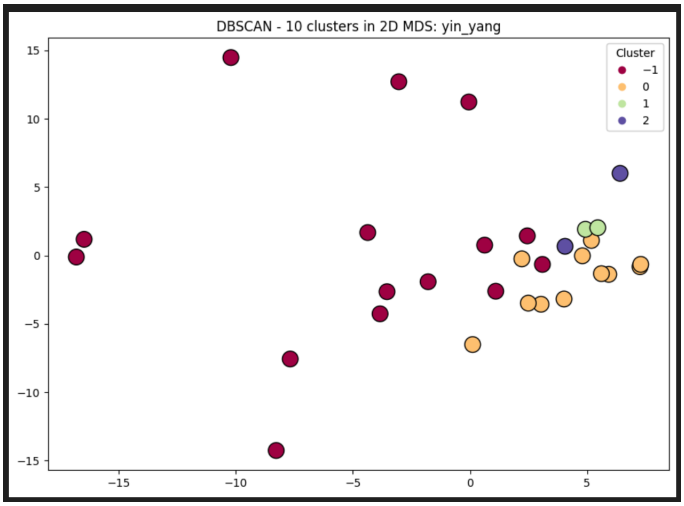
DBScan Clustering - Scatter Plot for Label = 0, C = 10, Feature Space = Resnet



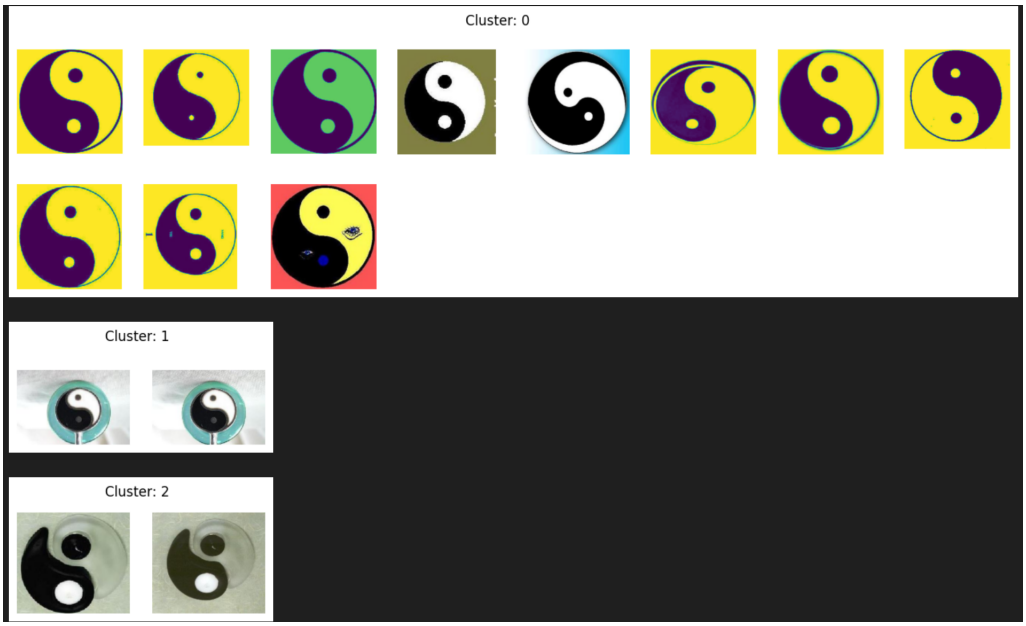
DBScan Clustering - Image Thumbnails for Label = 0, C = 10, Feature Space = Resnet



DBScan Clustering - Scatter Plot for Label = 100, C = 10, Feature Space = Resnet



DBScan Clustering - Image Thumbnails for Label = 100, C = 10, Feature Space = Resnet



5.3.6 Task 2 - Analysis of results

While running this task, we may not always achieve the required number of clusterings C for every label, but the optimization algorithm usually gets the result within 1 or 2 clusters. We noticed that using latent semantics resulted in better DBSCAN

clustering performance. However, this design decision hurt the follow up task of predicting the labels for odd images. The reason this could be happening is due to differences in dimensionality reduction between the training set (even images) and the testing set (odd images). For example, SVD's same core matrix might not be applicable between these two sets, resulting in reduced dimensions that do not accurately preserve density. This could result in the lower accuracy. Using a feature model directly, we were able to attain very good accuracies of greater than 87 percent. In particular, we observed good results with the Resnet feature space.

5.4 Task 3

5.4.1 Goal Description:

Implement a program which,

1. given even-numbered Caltech101 images,
   (a) creates an m-NN classifier (k-NN classifier) (for a user-specified m (k)),
   (b) creates a decision-tree classifier,
   (c) creates a PPR-based classifier.

   For this task, you can use a feature space of your choice.
2. for the odd-numbered images, predicts the most likely labels using the user-selected classifier.

The system should also output per-label precision, recall, and F1-score values, as well as output an overall accuracy value.

5.4.2 Assumptions:

1. We assume that we can use any feature vectors or latent semantics extracted from Phase 1 / 2 as the base matrix to demo each of the presented Classification algorithms.
2. The parameters for each Classification algorithm (not specified in the Task Description) can be modified to depict the best possible results received through experimentation.
3. We assume that if our classifier predicts Faces_easy it also predicts Faces since the labels are very similar. We do this to the classes that contain the names of other classes at the beginning of that class.
4. We run the Personalized Page Rank classifier for 20 iterations since we felt it provided a good balance between accuracy and speed.

5.4.3 Interface Specifications:

1. First, we prompt the user to select a Feature Space among: color_moments, hog, avgpool, layer3, fc, or resnet.
2. Second, we prompt the user to select the required classification algorithm: knn, personalized page rank, or decision tree.
3. Classifier Inputs:

(a) For the k-NN Classifier, we take in the value k from the user.

(b) For the personalized page rank algorithm, we prompt the user to input the random jump probability.

(c) For the decision tree, we prompt the user to select the variable selection mode to be used (between entropy-gain or gini index), the minimum sample leaf-size, and the maximum depth of the tree.

.

5.4.4 Implementation Approach:

1. We use the even-numbered & odd-numbered images of the Caltech101 dataset as our train & test datasets, respectively.

2. ***k*-NN Classifier:**

   (a) For this classifier, we simply train our model based on the **euclidean** distance measure with respect to the k-nearest neighbors.

   (b) During testing, we filter out the final result based on the **maximum** number of closest labels among the k-closest neighbors.

   (c) We try and select k as odd so that we don't face issues with the point lying exactly in between two points

   (d) We experimented with a number of distance functions and got the best accuracy with euclidean distance. This is probably because generally, a few features produce the distinction between two vectors.

3. **Personalized Page Rank-based Classifier (PPR):**

   (a) For implementing Personalized Page Rank, we first take distance between each vector pair and create a distance matrix. Since this process takes very long every time, we cache the distance matrix

   (b) We then add the query image to the end of the distance matrix and take the distances to each image from it

   (c) Next, we select the closest N nodes to each object and set a value of 1 for them in the similarity graph. The rest of the nodes get a value of 0 in the similarity graph

   (d) We set the teleportation start point as the last vector (query vector). This is because we want to find the closest, most relevant images to the query vector.

   (e) After this we take the M highest page rank objects and see their corresponding labels. We assign the maximum occurring label to the query point.

(f) We generally choose an odd value of M to avoid ties.

(g) We selected a low damping factor to emphasize the nodes closer to the query image.

4. **Decision Tree Classifier:**

   (a) For decision trees, the classification algorithm selects an optimal feature and its corresponding label to segment the dataset into two parts. For each possible splitting criterion, the process compares the information gained based on the entropy or Gini-impurity of the partitioned datasets and selects the optimal feature.

   (b) One of the issues we faced was the large training time for creating trees with no bounds on the depth; however, these gave the best results during our experimentation. Additionally, we notieced the best accuracy with a minimum leaf size of 25.

   (c) An extension to this implementation would be to develop early stopping criteria to prevent long training times and overfitting.

   (d) To calculate the information gain, we have used entropy and impurity-based measures such as the Gini index. This is configured as one of the parameters (mode) when defining the classifier. We have noticed that the Gini index provides higher test accuracy.

5.4.5 Results:

***k*-NN Classifier: Prediction Results per Label for: k = 1, Feature Space = Resnet (Truncated Output)**

```
Running Predictions for k-NN Classifier for k = 1
IMG_ID: 1       True label: Faces       Pred label: Faces_easy
IMG_ID: 3       True label: Faces       Pred label: Faces_easy
IMG_ID: 5       True label: Faces       Pred label: Faces_easy
IMG_ID: 7       True label: Faces       Pred label: Faces_easy
IMG_ID: 9       True label: Faces       Pred label: Faces_easy
IMG_ID: 11      True label: Faces       Pred label: Faces_easy
IMG_ID: 13      True label: Faces       Pred label: Faces
IMG_ID: 15      True label: Faces       Pred label: Faces_easy
IMG_ID: 17      True label: Faces       Pred label: Faces_easy
IMG_ID: 19      True label: Faces       Pred label: Faces
IMG_ID: 21      True label: Faces       Pred label: Faces
IMG_ID: 23      True label: Faces       Pred label: Faces_easy
IMG_ID: 25      True label: Faces       Pred label: Faces_easy
IMG_ID: 27      True label: Faces       Pred label: Faces
IMG_ID: 29      True label: Faces       Pred label: Faces_easy
IMG_ID: 31      True label: Faces       Pred label: Faces
IMG_ID: 33      True label: Faces       Pred label: Faces_easy
IMG_ID: 35      True label: Faces       Pred label: Faces
IMG_ID: 37      True label: Faces       Pred label: Faces_easy
IMG_ID: 39      True label: Faces       Pred label: Faces_easy
IMG_ID: 41      True label: Faces       Pred label: Faces_easy
IMG_ID: 43      True label: Faces       Pred label: Faces_easy
IMG_ID: 45      True label: Faces       Pred label: Faces_easy
IMG_ID: 47      True label: Faces       Pred label: Faces
...
IMG_ID: 8669    True label: yin_yang    Pred label: watch
IMG_ID: 8671    True label: yin_yang    Pred label: yin_yang
IMG_ID: 8673    True label: yin_yang    Pred label: yin_yang
IMG_ID: 8675    True label: yin_yang    Pred label: scissors
```

***k*-NN Classifier: Accuracy Scores per Label for: k = 1, Feature Space = Resnet (Truncated Output)**

```
Per-label Precision, Recall, F1 Scores for k-NN Classifier for k = 1:

Label: Faces            Precision: 1.0          Recall: 1.0      F1 score:1.0
Label: Faces_easy       Precision: 1.0          Recall: 1.0      F1 score:1.0
Label: Leopards         Precision: 0.9901       Recall: 1.0      F1 score:0.995
Label: Motorbikes       Precision: 0.9925       Recall: 1.0      F1 score:0.9963
Label: accordion        Precision: 0.871        Recall: 1.0      F1 score:0.931
Label: airplanes        Precision: 0.9754       Recall: 0.99     F1 score:0.9826
Label: anchor           Precision: 0.9091       Recall: 0.4762   F1 score:0.625
Label: ant              Precision: 0.7273       Recall: 0.7619   F1 score:0.7442
Label: barrel           Precision: 0.9583       Recall: 0.9583   F1 score:0.9583
Label: bass             Precision: 0.875        Recall: 0.7778   F1 score:0.8235
Label: beaver           Precision: 0.7826       Recall: 0.7826   F1 score:0.7826
Label: binocular        Precision: 0.8889       Recall: 1.0      F1 score:0.9412
Label: bonsai           Precision: 0.9492       Recall: 0.875    F1 score:0.9106
Label: brain            Precision: 0.9362       Recall: 0.898    F1 score:0.9167
Label: brontosaurus     Precision: 0.6471       Recall: 0.5      F1 score:0.5641
Label: buddha           Precision: 1.0          Recall: 0.9762   F1 score:0.988
Label: butterfly        Precision: 0.8056       Recall: 0.6304   F1 score:0.7073
Label: camera           Precision: 0.96         Recall: 0.96     F1 score:0.96
Label: cannon           Precision: 0.85         Recall: 0.8095   F1 score:0.8293
Label: car_side         Precision: 0.9841       Recall: 1.0      F1 score:0.992
Label: ceiling_fan      Precision: 0.9474       Recall: 0.7826   F1 score:0.8571
Label: cellphone        Precision: 0.9355       Recall: 0.9667   F1 score:0.9508
Label: chair            Precision: 0.8182       Recall: 0.5806   F1 score:0.6792
...
Label: yin_yang         Precision: 0.9565       Recall: 0.7333   F1 score:0.8302


Overall accuracy for k-NN Classifier for k = 1: 90.82526509912402%
```

***k*-NN Classifier: Prediction Results per Label for: k = 5, Feature Space = Resnet (Truncated Output)**

```
Running Predictions for k-NN Classifier for k = 5
IMG_ID: 1       True label: Faces       Pred label: Faces
IMG_ID: 3       True label: Faces       Pred label: Faces
IMG_ID: 5       True label: Faces       Pred label: Faces
IMG_ID: 7       True label: Faces       Pred label: Faces
IMG_ID: 9       True label: Faces       Pred label: Faces
IMG_ID: 11      True label: Faces       Pred label: Faces_easy
IMG_ID: 13      True label: Faces       Pred label: Faces
IMG_ID: 15      True label: Faces       Pred label: Faces_easy
IMG_ID: 17      True label: Faces       Pred label: Faces
IMG_ID: 19      True label: Faces       Pred label: Faces_easy
IMG_ID: 21      True label: Faces       Pred label: Faces
IMG_ID: 23      True label: Faces       Pred label: Faces_easy
IMG_ID: 25      True label: Faces       Pred label: Faces
IMG_ID: 27      True label: Faces       Pred label: Faces
IMG_ID: 29      True label: Faces       Pred label: Faces
IMG_ID: 31      True label: Faces       Pred label: Faces
IMG_ID: 33      True label: Faces       Pred label: Faces_easy
IMG_ID: 35      True label: Faces       Pred label: Faces
IMG_ID: 37      True label: Faces       Pred label: Faces_easy
IMG_ID: 39      True label: Faces       Pred label: Faces
IMG_ID: 41      True label: Faces       Pred label: Faces
IMG_ID: 43      True label: Faces       Pred label: Faces_easy
IMG_ID: 45      True label: Faces       Pred label: Faces
IMG_ID: 47      True label: Faces       Pred label: Faces
...
IMG_ID: 8669    True label: yin_yang    Pred label: soccer_ball
IMG_ID: 8671    True label: yin_yang    Pred label: yin_yang
IMG_ID: 8673    True label: yin_yang    Pred label: yin_yang
IMG_ID: 8675    True label: yin_yang    Pred label: scissors
```

***k*-NN Classifier: Accuracy Scores per Label for: k = 5, Feature Space = Resnet (Truncated Output)**

```
Label: Faces            Precision: 1.0          Recall: 0.9954   F1 score:0.9977
Label: Faces_easy       Precision: 1.0          Recall: 0.9908   F1 score:0.9954
Label: Leopards         Precision: 0.9615       Recall: 1.0      F1 score:0.9804
Label: Motorbikes       Precision: 0.9828       Recall: 1.0      F1 score:0.9913
Label: accordion        Precision: 0.9          Recall: 1.0      F1 score:0.9474
Label: airplanes        Precision: 0.9365       Recall: 0.995    F1 score:0.9648
Label: anchor           Precision: 1.0          Recall: 0.4762   F1 score:0.6452
Label: ant              Precision: 0.85         Recall: 0.8095   F1 score:0.8293
Label: barrel           Precision: 0.9565       Recall: 0.9167   F1 score:0.9362
Label: bass             Precision: 0.95         Recall: 0.7037   F1 score:0.8085
Label: beaver           Precision: 0.8696       Recall: 0.8696   F1 score:0.8696
Label: binocular        Precision: 0.9412       Recall: 1.0      F1 score:0.9697
Label: bonsai           Precision: 0.9683       Recall: 0.9531   F1 score:0.9606
Label: brain            Precision: 0.8462       Recall: 0.898    F1 score:0.8713
Label: brontosaurus     Precision: 0.8333       Recall: 0.4545   F1 score:0.5882
Label: buddha           Precision: 1.0          Recall: 0.9524   F1 score:0.9756
Label: butterfly        Precision: 0.8788       Recall: 0.6304   F1 score:0.7342
Label: camera           Precision: 0.9259       Recall: 1.0      F1 score:0.9615
Label: cannon           Precision: 0.9474       Recall: 0.8571   F1 score:0.9
Label: car_side         Precision: 0.9841       Recall: 1.0      F1 score:0.992
Label: ceiling_fan      Precision: 1.0          Recall: 0.7826   F1 score:0.878
Label: cellphone        Precision: 0.9355       Recall: 0.9667   F1 score:0.9508
Label: chair            Precision: 0.8333       Recall: 0.4839   F1 score:0.6122
...
Label: yin_yang         Precision: 0.92         Recall: 0.7667   F1 score:0.8364


Overall accuracy for k-NN Classifier for k = 5: 90.43337943752881%
```

***k*-NN Classifier: Prediction Results per Label for: k = 10, Feature Space = Resnet (Truncated Output)**

```
Running Predictions for k-NN Classifier for k = 10
IMG_ID: 1       True label: Faces       Pred label: Faces_easy
IMG_ID: 3       True label: Faces       Pred label: Faces
IMG_ID: 5       True label: Faces       Pred label: Faces
IMG_ID: 7       True label: Faces       Pred label: Faces
IMG_ID: 9       True label: Faces       Pred label: Faces
IMG_ID: 11      True label: Faces       Pred label: Faces_easy
IMG_ID: 13      True label: Faces       Pred label: Faces
IMG_ID: 15      True label: Faces       Pred label: Faces
IMG_ID: 17      True label: Faces       Pred label: Faces
IMG_ID: 19      True label: Faces       Pred label: Faces_easy
IMG_ID: 21      True label: Faces       Pred label: Faces
IMG_ID: 23      True label: Faces       Pred label: Faces_easy
IMG_ID: 25      True label: Faces       Pred label: Faces
IMG_ID: 27      True label: Faces       Pred label: Faces
IMG_ID: 29      True label: Faces       Pred label: Faces
IMG_ID: 31      True label: Faces       Pred label: Faces
IMG_ID: 33      True label: Faces       Pred label: Faces_easy
IMG_ID: 35      True label: Faces       Pred label: Faces
IMG_ID: 37      True label: Faces       Pred label: Faces_easy
IMG_ID: 39      True label: Faces       Pred label: Faces
IMG_ID: 41      True label: Faces       Pred label: Faces_easy
IMG_ID: 43      True label: Faces       Pred label: Faces
IMG_ID: 45      True label: Faces       Pred label: Faces
IMG_ID: 47      True label: Faces       Pred label: Faces
...
IMG_ID: 8669    True label: yin_yang    Pred label: watch
IMG_ID: 8671    True label: yin_yang    Pred label: yin_yang
IMG_ID: 8673    True label: yin_yang    Pred label: yin_yang
IMG_ID: 8675    True label: yin_yang    Pred label: scissors
```

***k*-NN Classifier: Accuracy Scores per Label for: k = 10, Feature Space = Resnet (Truncated Output)**

```
Per-label Precision, Recall, F1 Scores for k-NN Classifier for k = 10:

Label: Faces            Precision: 1.0          Recall: 0.9954   F1 score:0.9977
Label: Faces_easy       Precision: 1.0          Recall: 0.9908   F1 score:0.9954
Label: Leopards         Precision: 0.9434       Recall: 1.0      F1 score:0.9709
Label: Motorbikes       Precision: 0.9779       Recall: 1.0      F1 score:0.9888
Label: accordion        Precision: 0.871        Recall: 1.0      F1 score:0.931
Label: airplanes        Precision: 0.9215       Recall: 0.9975   F1 score:0.958
Label: anchor           Precision: 1.0          Recall: 0.1905   F1 score:0.32
Label: ant              Precision: 0.8947       Recall: 0.8095   F1 score:0.85
Label: barrel           Precision: 0.9565       Recall: 0.9167   F1 score:0.9362
Label: bass             Precision: 0.9048       Recall: 0.7037   F1 score:0.7917
Label: beaver           Precision: 0.8636       Recall: 0.8261   F1 score:0.8444
Label: binocular        Precision: 0.9412       Recall: 1.0      F1 score:0.9697
Label: bonsai           Precision: 0.9683       Recall: 0.9531   F1 score:0.9606
Label: brain            Precision: 0.902        Recall: 0.9388   F1 score:0.92
Label: brontosaurus     Precision: 0.8571       Recall: 0.2727   F1 score:0.4138
Label: buddha           Precision: 1.0          Recall: 0.9524   F1 score:0.9756
Label: butterfly        Precision: 0.9412       Recall: 0.6957   F1 score:0.8
Label: camera           Precision: 0.9615       Recall: 1.0      F1 score:0.9804
Label: cannon           Precision: 0.8889       Recall: 0.7619   F1 score:0.8205
Label: car_side         Precision: 0.9841       Recall: 1.0      F1 score:0.992
Label: ceiling_fan      Precision: 1.0          Recall: 0.7826   F1 score:0.878
Label: cellphone        Precision: 0.875        Recall: 0.9333   F1 score:0.9032
Label: chair            Precision: 0.9091       Recall: 0.3226   F1 score:0.4762
...
Label: yin_yang         Precision: 0.8889       Recall: 0.8      F1 score:0.8421


Overall accuracy for k-NN Classifier for k = 10: 89.7648686030428%
```

Personalized Page Rank-based Classifier: Prediction Results per Label for: M = 5, N = 4, p = 0.15, Feature Space = Resnet (Truncated Output)

```
Distance Matrix computed
Running Predictions for Personalized Page Rank Classifier with M = 4, N = 5, and Random Jump Probability = 0.15
IMG_ID: 1       True label: Faces       Pred labels: Faces_easy
IMG_ID: 3       True label: Faces       Pred labels: Faces
IMG_ID: 5       True label: Faces       Pred labels: Faces
IMG_ID: 7       True label: Faces       Pred labels: Faces
IMG_ID: 9       True label: Faces       Pred labels: Faces_easy
IMG_ID: 11      True label: Faces       Pred labels: Faces_easy
IMG_ID: 13      True label: Faces       Pred labels: Faces_easy
IMG_ID: 15      True label: Faces       Pred labels: Faces_easy
IMG_ID: 17      True label: Faces       Pred labels: Faces
IMG_ID: 19      True label: Faces       Pred labels: Faces_easy
IMG_ID: 21      True label: Faces       Pred labels: Faces
IMG_ID: 23      True label: Faces       Pred labels: Faces_easy
IMG_ID: 25      True label: Faces       Pred labels: Faces
IMG_ID: 27      True label: Faces       Pred labels: Faces
IMG_ID: 29      True label: Faces       Pred labels: Faces
IMG_ID: 31      True label: Faces       Pred labels: Faces
IMG_ID: 33      True label: Faces       Pred labels: Faces_easy
IMG_ID: 35      True label: Faces       Pred labels: Faces
IMG_ID: 37      True label: Faces       Pred labels: Faces_easy
...
IMG_ID: 8669    True label: yin_yang    Pred labels: anchor
IMG_ID: 8671    True label: yin_yang    Pred labels: yin_yang
IMG_ID: 8673    True label: yin_yang    Pred labels: yin_yang
IMG_ID: 8675    True label: yin_yang    Pred labels: scissors
```

Personalized Page Rank-based Classifier: Accuracy Scores per Label for: M = 5, N = 4, p = 0.15, Feature Space = Resnet (Truncated Output)

```
Per-label Precision, Recall, F1 Scores for Personalized Page Rank Classifier with M = 4, N = 5, and Random Jump Probability = 0.15:

Label: Faces            Precision: 1.0          Recall: 1.0     F1 score:1.0
Label: Faces_easy       Precision: 1.0          Recall: 0.9908  F1 score:0.9954
Label: Leopards         Precision: 0.9804       Recall: 1.0     F1 score:0.9901
Label: Motorbikes       Precision: 0.9828       Recall: 1.0     F1 score:0.9913
Label: accordion        Precision: 0.8966       Recall: 0.963   F1 score:0.9286
Label: airplanes        Precision: 0.95         Recall: 0.9975  F1 score:0.9732
Label: anchor           Precision: 0.75         Recall: 0.1429  F1 score:0.24
Label: ant              Precision: 0.85         Recall: 0.8095  F1 score:0.8293
Label: barrel           Precision: 0.9565       Recall: 0.9167  F1 score:0.9362
Label: bass             Precision: 0.95         Recall: 0.7037  F1 score:0.8085
Label: beaver           Precision: 0.8636       Recall: 0.8261  F1 score:0.8444
Label: binocular        Precision: 0.9412       Recall: 1.0     F1 score:0.9697
Label: bonsai           Precision: 0.9688       Recall: 0.9688  F1 score:0.9688
Label: brain            Precision: 0.8113       Recall: 0.8776  F1 score:0.8431
Label: brontosaurus     Precision: 0.8333       Recall: 0.2273  F1 score:0.3571
Label: buddha           Precision: 0.975        Recall: 0.9286  F1 score:0.9512
Label: butterfly        Precision: 0.9062       Recall: 0.6304  F1 score:0.7436
Label: camera           Precision: 0.9615       Recall: 1.0     F1 score:0.9804
Label: cannon           Precision: 0.8571       Recall: 0.8571  F1 score:0.8571
Label: car_side         Precision: 0.9841       Recall: 1.0     F1 score:0.992
Label: ceiling_fan      Precision: 1.0          Recall: 0.7826  F1 score:0.878
Label: cellphone        Precision: 0.9062       Recall: 0.9667  F1 score:0.9355
Label: chair            Precision: 0.7692       Recall: 0.3226  F1 score:0.4545
...
Label: yin_yang         Precision: 0.8846       Recall: 0.7667  F1 score:0.8214


Overall accuracy for Personalized Page Rank Classifier with M = 4, N = 5, and Random Jump Probability = 0.15: 89.48824343015215%
```

Personalized Page Rank-based Classifier: Prediction Results per Label for: M = 5, N = 4, p = 0.5, Feature Space = Resnet (Truncated Output)

```
Computing distance matrix
Distance Matrix computed
Running Predictions for Personalized Page Rank Classifier with M = 4, N = 5, and Random Jump Probability = 0.5
IMG_ID: 1       True label: Faces       Pred labels: Faces_easy
IMG_ID: 3       True label: Faces       Pred labels: Faces
IMG_ID: 5       True label: Faces       Pred labels: Faces
IMG_ID: 7       True label: Faces       Pred labels: Faces
IMG_ID: 9       True label: Faces       Pred labels: Faces_easy
IMG_ID: 11      True label: Faces       Pred labels: Faces_easy
IMG_ID: 13      True label: Faces       Pred labels: Faces_easy
IMG_ID: 15      True label: Faces       Pred labels: Faces_easy
IMG_ID: 17      True label: Faces       Pred labels: Faces
IMG_ID: 19      True label: Faces       Pred labels: Faces_easy
IMG_ID: 21      True label: Faces       Pred labels: Faces
IMG_ID: 23      True label: Faces       Pred labels: Faces_easy
IMG_ID: 25      True label: Faces       Pred labels: Faces
IMG_ID: 27      True label: Faces       Pred labels: Faces
IMG_ID: 29      True label: Faces       Pred labels: Faces
IMG_ID: 31      True label: Faces       Pred labels: Faces
IMG_ID: 33      True label: Faces       Pred labels: Faces_easy
IMG_ID: 35      True label: Faces       Pred labels: Faces
IMG_ID: 37      True label: Faces       Pred labels: Faces_easy
IMG_ID: 39      True label: Faces       Pred labels: Faces
IMG_ID: 41      True label: Faces       Pred labels: Faces_easy
IMG_ID: 43      True label: Faces       Pred labels: Faces
...
IMG_ID: 8669    True label: yin_yang    Pred labels: anchor
IMG_ID: 8671    True label: yin_yang    Pred labels: yin_yang
IMG_ID: 8673    True label: yin_yang    Pred labels: yin_yang
IMG_ID: 8675    True label: yin_yang    Pred labels: scissors
```

Personalized Page Rank-based Classifier: Accuracy Scores per Label for: M = 5, N = 4, p = 0.5, Feature Space = Resnet (Truncated Output)

```
Per-label Precision, Recall, F1 Scores for Personalized Page Rank Classifier with M = 4, N = 5, and Random Jump Probability = 0.5:

Label: Faces            Precision: 1.0          Recall: 1.0     F1 score:1.0
Label: Faces_easy       Precision: 1.0          Recall: 0.9908  F1 score:0.9954
Label: Leopards         Precision: 0.9804       Recall: 1.0     F1 score:0.9901
Label: Motorbikes       Precision: 0.9852       Recall: 1.0     F1 score:0.9925
Label: accordion        Precision: 0.8667       Recall: 0.963   F1 score:0.9123
Label: airplanes        Precision: 0.9477       Recall: 0.9975  F1 score:0.972
Label: anchor           Precision: 0.6667       Recall: 0.0952  F1 score:0.1667
Label: ant              Precision: 0.85         Recall: 0.8095  F1 score:0.8293
Label: barrel           Precision: 0.9565       Recall: 0.9167  F1 score:0.9362
Label: bass             Precision: 1.0          Recall: 0.7037  F1 score:0.8261
Label: beaver           Precision: 0.9048       Recall: 0.8261  F1 score:0.8636
Label: binocular        Precision: 0.9412       Recall: 1.0     F1 score:0.9697
Label: bonsai           Precision: 0.9688       Recall: 0.9688  F1 score:0.9688
Label: brain            Precision: 0.7963       Recall: 0.8776  F1 score:0.835
Label: brontosaurus     Precision: 0.8333       Recall: 0.2273  F1 score:0.3571
Label: buddha           Precision: 0.975        Recall: 0.9286  F1 score:0.9512
Label: butterfly        Precision: 0.8788       Recall: 0.6304  F1 score:0.7342
Label: camera           Precision: 0.9615       Recall: 1.0     F1 score:0.9804
Label: cannon           Precision: 0.8571       Recall: 0.8571  F1 score:0.8571
Label: car_side         Precision: 0.9841       Recall: 1.0     F1 score:0.992
Label: ceiling_fan      Precision: 1.0          Recall: 0.7826  F1 score:0.878
Label: cellphone        Precision: 0.9062       Recall: 0.9667  F1 score:0.9355
Label: chair            Precision: 0.8571       Recall: 0.3871  F1 score:0.5333
...
Label: yin_yang         Precision: 0.8846       Recall: 0.7667  F1 score:0.8214


Overall accuracy for Personalized Page Rank Classifier with M = 4, N = 5, and Random Jump Probability = 0.5: 89.44213923467036%
```

Decision Tree Classifier: Prediction Results per Label for: mode = gini, minimum leaf size = 25, maximum depth = infinity (no restriction on tree growth), Feature Space = Resnet (Truncated Output)

```
Running Predictions for Decision Tree Classifier with mode = gini, minimum sample size = 25, maximum tree-depth = inf:
IMG_ID: 1   True label: Faces    Pred label: Faces
IMG_ID: 3   True label: Faces    Pred label: Faces_easy
IMG_ID: 5   True label: Faces    Pred label: Faces
IMG_ID: 7   True label: Faces    Pred label: Faces
IMG_ID: 9   True label: Faces    Pred label: Faces
IMG_ID: 11  True label: Faces    Pred label: Faces
IMG_ID: 13  True label: Faces    Pred label: Faces_easy
IMG_ID: 15  True label: Faces    Pred label: Faces
IMG_ID: 17  True label: Faces    Pred label: Faces
IMG_ID: 19  True label: Faces    Pred label: Faces
IMG_ID: 21  True label: Faces    Pred label: Faces
IMG_ID: 23  True label: Faces    Pred label: Faces
IMG_ID: 25  True label: Faces    Pred label: Faces
IMG_ID: 27  True label: Faces    Pred label: Faces
IMG_ID: 29  True label: Faces    Pred label: Faces
IMG_ID: 31  True label: Faces    Pred label: Faces
IMG_ID: 33  True label: Faces    Pred label: Faces
IMG_ID: 35  True label: Faces    Pred label: Faces
IMG_ID: 37  True label: Faces    Pred label: Faces_easy
IMG_ID: 39  True label: Faces    Pred label: Faces
IMG_ID: 41  True label: Faces    Pred label: Faces
IMG_ID: 43  True label: Faces    Pred label: Faces_easy
...
IMG_ID: 8669    True label: yin_yang    Pred label: anchor
IMG_ID: 8671    True label: yin_yang    Pred label: yin_yang
IMG_ID: 8673    True label: yin_yang    Pred label: yin_yang
IMG_ID: 8675    True label: yin_yang    Pred label: chandelier
```

Decision Tree Classifier: Accuracy Scores per Label for: mode = gini, minimum leaf size = 25, maximum depth = infinity (no restriction on tree growth), Feature Space = Resnet (Truncated Output)

```
Per-label Precision, Recall, F1 Scores for Decision Tree Classifier with mode = gini, minimum sample size = 25, maximum tree-depth = inf:

Label:  Faces           Precision:  0.7802197802197802  Recall: 0.9815668202764977  F1 score:   0.8693877551020408
Label:  Faces_easy      Precision:  0.9953703703703703  Recall: 0.9862385321100917  F1 score:   0.9907834101382489
Label:  Leopards        Precision:  0.9056603773584906  Recall: 0.96                F1 score:   0.9320388349514563
Label:  Motorbikes      Precision:  0.9872448979591837  Recall: 0.9699248120300752  F1 score:   0.9785082174462705
Label:  accordion       Precision:  0.9259259259259259  Recall: 0.9259259259259259  F1 score:   0.9259259259259259
Label:  airplanes       Precision:  0.9632352941176471  Recall: 0.9825              F1 score:   0.9727722772277227
Label:  anchor          Precision:  0.05263157894736842 Recall: 0.7619047619047619  F1 score:   0.05
Label:  ant             Precision:  0.8333333333333334  Recall: 0.7142857142857143  F1 score:   0.7692307692307692
Label:  barrel          Precision:  0.9523809523809523  Recall: 0.8333333333333334  F1 score:   0.8888888888888889
Label:  bass            Precision:  0.4                 Recall: 0.5185185185185185  F1 score:   0.45161290322580644
Label:  beaver          Precision:  0.34146341463414637 Recall: 0.6086956521739131  F1 score:   0.4375
Label:  binocular       Precision:  0.9411764705882353  Recall: 1.0                 F1 score:   0.9696969696969697
Label:  bonsai          Precision:  0.7397260273972602  Recall: 0.84375             F1 score:   0.7883211678832116
Label:  brain           Precision:  0.6382978723404256  Recall: 0.6122448979591837  F1 score:   0.625
Label:  brontosaurus    Precision:  0.2857142857142857  Recall: 0.2727272727272727  F1 score:   0.2790697674418604
Label:  buddha          Precision:  0.7352941176470589  Recall: 0.5952380952380952  F1 score:   0.6578947368421053
Label:  butterfly       Precision:  0.4107142857142857  Recall: 0.5                 F1 score:   0.45098039215686275
Label:  camera          Precision:  0.92                Recall: 0.92                F1 score:   0.92
Label:  cannon          Precision:  0.6956521739130435  Recall: 0.7619047619047619  F1 score:   0.7272727272727272
Label:  car_side        Precision:  1.0                 Recall: 0.9193548387096774  F1 score:   0.9579831932773109
Label:  ceiling_fan     Precision:  0.6470588235294118  Recall: 0.4782608695652174  F1 score:   0.55
Label:  cellphone       Precision:  0.9545454545454546  Recall: 0.7                 F1 score:   0.8076923076923077
Label:  chair           Precision:  0.46808510638297873 Recall: 0.7096774193548387  F1 score:   0.5641025641025641
...
Label:  yin_yang    Precision:  0.8947368421052632  Recall: 0.5666666666666667  F1 score:   0.6938775510204083


Overall accuracy for Decision Tree Classifier with mode = gini, minimum sample size = 25, maximum tree-depth = inf: 79.25311203319502%
```

Decision Tree Classifier: Experiments Configuration with Corresponding Training Times

Mode	Min-Sample	Max-Depth	Accuracy (Test)	Train Time	Trained Tree-Depth	Comments
gini	25	Unbounded	79.25%	2hr 25mins	122	train size = complete data
gini	3	Unbounded	67.82%	25mins	92	train size = 868, [image_id %5==0]
gini	5	Unbounded	64.13%	1hr 40mins	104	train size = 1515, 15 samples from each class
gini	15	Unbounded	64.80%	55 mins	102	train size = 1515, 15 samples from each class
entropy	15	Unbounded	34.35%	7 mins	12	train size = 1515, 15 samples from each class
gini	25	Unbounded	68.05%	54 mins	97	train size = 1515, 15 samples from each class
entropy	25	Unbounded	33.86%	6 mins	11	train size = 1515, 15 samples from each class
gini	35	Unbounded	65.91%	59 mins	92	train size = 1515, 15 samples from each class
gini	25	45	51.38%	45 mins	46	train size = 1515, 15 samples from each class

5.4.6 Task 3 - Analysis of Results:

We found the best results in the case of the resnet feature space, this could be because the final layers of the resnet captures the larger features more effectively. ***k*-NN:** As seen in the results section, we get really good accuracy for k = 1. For the other values of k we get similar or worse accuracy. In our testing we found good accuracy for k = 1 to 3. This means that as we consider more values of k ¿ 4, we consider more points which could be farther away from the points and belong to other classes.

PPR: As seen in the results, we get quite a good accuracy for M = 5 and N = 4 and damping factor (p) = 0.5. The p = 0.5 emphasizes that we want to find the important cluster centers closer to the starting point (query vector). We experimented with a lot of other values for M, N and p and found these to be the best combination of accuracy and time taken. However, as is the case with the previous phase too, our page rank implementation takes quite long to completely run, this is why we proceeded to cache the results.

Decision Tree: In case of decision tree, we got worse results than the previous two methods. This could be because some of our features in the feature space are redundant. Hence, we tried experimenting with other reduced dimensions but didn't get much better results. Additionally, the run time of the program was also quite high due to the gini index calculation for the entire feature space, hence we cached the results.
Furthermore, in the case of decision tree, we got worse results than the previous two methods. We noticed that the run time of the program was also significantly high as the algorithm iterated through all feature-value combinations to find the optimal split. The large precision of feature vector elements made almost all the elements for any index unique, and as a result, the execution time of the program was significantly high.
To reduce the execution time, we have reduced the precision of the elements of the

feature vector to two decimal points - this reduced the number of unique values for any index and made the training time reasonable. However, we assume this may have impacted the accuracy of the model.
In order to reduce the training time, we have added stopping criteria as the minimum sample size for each node and a maximum depth of the tree. It has been observed that if we allow to grow the tree unbounded although the accuracy improves, the training time also increases. Minimum sample size also allows to limit the depth of the tree indirectly and decreases training time.
Another probable cause of low test accuracy could be that the model is overfitted with the training data. We have tried to downsize the training dataset by using only one image per five images – although it lowered training time, testing accuracy also decreased.
On further investigation, we observed that the number of samples per label is not evenly distributed. While some classes have as many as 400 samples, some classes have fewer than 30 samples. Thus, we tried to use a training set containing only 15 samples from each class. It improved the accuracy marginally; however, the training time increased drastically.
Although we tried different parameters, each execution took a significant amount of time and thus limited the number of iterations we could execute. With the resnet feature space, using the leaf sample size as 25, we observed the highest test accuracy of 9

5.5 Task 4

5.5.1 Task 4a - Goal Description:

Implement a Locality Sensitive Hashing (LSH) tool (for Euclidean distance), which takes as input (a) the number of layers, L, (b) the number of hashes per layer, h, and (c) a set of vectors as input and creates an in-memory index structure containing the given set of vectors. Use the following paper for reference: "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions" (by Alexandr Andoni and Piotr Indyk). Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122. [6]

5.5.2 Task 4a - Assumptions Made:

1. In addition the L and h, we accept an additional parameter W, which we usually provide as 5. W is the number of buckets per hash.

2. We assume that the input vectors can either be a JSON file that contains a list of vectors, or a feature model.

3. In addition to the vectors, we also store some auxiliary data per vector in the LSH, such as the image ID. This is useful in the subsequent tasks that use the LSH for retrieval.

4. We assume that the set of vectors provided for indexing can fit within the memory of the system on which the LSH index is being created and run.

5.5.3 Task 4a - Interface Specifications:

1. First, we take in the required number of layers L, the number of hashes per layer h, and also additionally prompt the user for bucket width w.

2. Second, for the set of vectors, we allow the user to enter the name of a json file containing the vectors to be indexed, or ask for a feature space for the Caltech101 Dataset, i.e. color_moments, hog, avgpool, layer3, fc, or resnet.

5.5.4 Task 4a - Implementation Approach:

1. The exact description of how the Locality-Sensitive Indexing is implemented is presented in Section 4.5.

2. Each hash function is a random hyperplane having the same dimensions as a vector in our input dataset. We index each image with respect to the bucket width w for the range between [lowest dot product, highest dot product] calculated between our input vectors and the respective hash function (hyperplane).

3. Based on the bucketed indices, we generate the hash code for an entire layer.

4. During query time, we take the union of all data points having at least one similar hash code across a layer in comparison to the original query image. This union is called our candidate data set.

5. If we found that we did not retrieve enough candidates to satisfy t (the limit of the number of objects to retrieve), then we use a strategy whereby we incrementally look at adjacent buckets until we have enough candidates to satisfy t.

6. The incremental adjacent buckets operates by generating all bit strings of the number of indices in each layer. We first start with delta=1, and incremental/decrement our original index value by delta, resulting 2 new indices for each bit string. We then fetch the vectors in these 2 new indices, and perform union with our current candidates.

7. We increment delta and repeat until we reach the required number of candidates.

8. We finally rank the images in our candidate data set in increasing order of **euclidean** distance in comparison to the original query image, since we observed the best performance with it.

5.5.5 Task 4a - Results:

Locality-Sensitive Hashing Results: t = 10, Image-ID = 2500, Feature Space = layer3

```
Image ID:  2500         Unique considered:  185         Total considered:  235
1.      Image ID: 2500  Distance: 0.0
2.      Image ID: 2308  Distance: 1.3429126739501953
3.      Image ID: 2254  Distance: 1.7147425413131714
4.      Image ID: 2398  Distance: 1.8075337409973145
5.      Image ID: 2226  Distance: 1.816244125366211
6.      Image ID: 2670  Distance: 1.8447668552398682
7.      Image ID: 2412  Distance: 1.8741718530654907
8.      Image ID: 2252  Distance: 1.8773486614227295
9.      Image ID: 2484  Distance: 1.8951730728149414
10.     Image ID: 2134  Distance: 1.9605820178985596
```

Locality-Sensitive Hashing Results: t = 10, Image-ID = 2500, Feature Space = resnet

```
Locality-Sensitive Hashing Results for L = 10, h = 10, W = 5
Image ID:  2500         Unique considered:  155         Total considered:  208
1.      Image ID: 2500  Distance: 0.0
2.      Image ID: 2308  Distance: 4.294017314910889
3.      Image ID: 2670  Distance: 4.866342067718506
4.      Image ID: 2254  Distance: 4.904757976531982
5.      Image ID: 2252  Distance: 4.94062614440918
6.      Image ID: 2484  Distance: 5.320182800292969
7.      Image ID: 2134  Distance: 5.43259334564209
8.      Image ID: 2570  Distance: 5.615748882293701
9.      Image ID: 2492  Distance: 5.646214962005615
10.     Image ID: 2398  Distance: 5.653390884399414
```

5.5.6 Task 4b - Goal Description:

Implement a similar image search algorithm using this index structure storing the even-numbered Caltech101 images and a visual model of your choice (the combined visual model must have at least 256 dimensions): for a given query image and integer t,

1. visualizes the t most similar images,
2. outputs the number of unique and overall number of images considered during the process.

5.5.7 Task 4b - Assumptions Made:

Similar to Task 4a.

5.5.8 Task 4b - Interface Specifications:

1. In general, we use an LSH index that we found to perform decently, that is, with L=10, h=3, W=5 with the ResNet feature model.

2. First, we take the input image as a filename or a valid image ID in the Caltech101 Dataset.

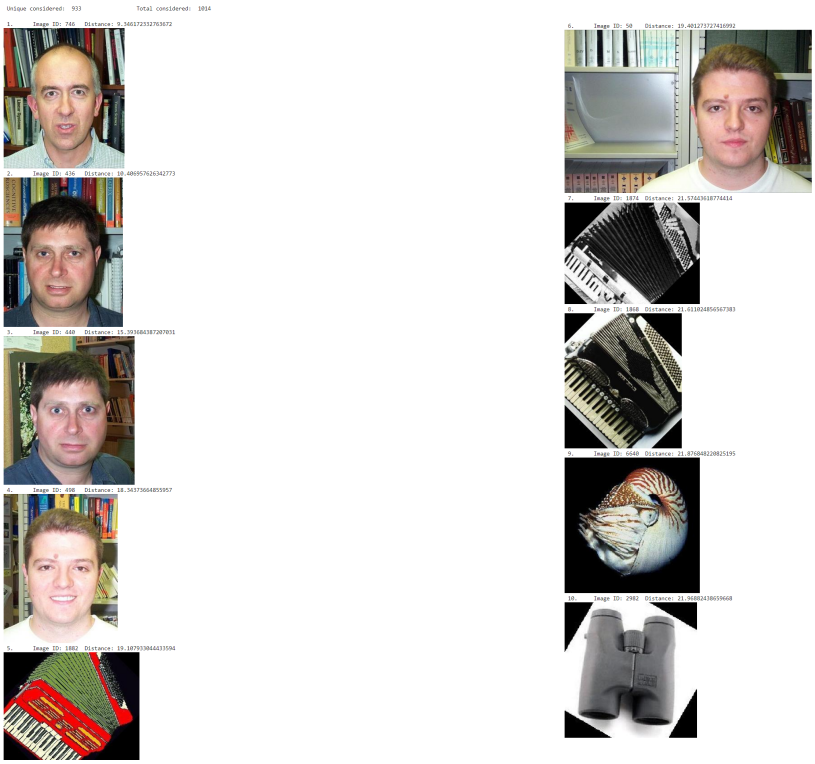
3. Second, we take the number of images, t, to be displayed.

5.5.9 Task 4b - Implementation Approach:

Similar to our implementation of Task 4a.

5.5.10 Task 4b - Results:

Locality-Sensitive Hashing Results: L = 3, h = 3, w = 5, t = 10, Image-ID = 1, Feature Space = resnet

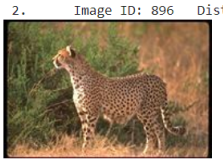


Locality-Sensitive Hashing Results: L = 3, h = 3, w = 5, t = 10, Image-ID = 881, Feature Space = resnet

Unique considered: 1002 Total considered: 1067

1. Image ID: 994 Distance: 6.573435306549072

2. Image ID: 896 Distance: 6.797601699829102



3. Image ID: 976 Distance: 6.931845188140869

4. Image ID: 894 Distance: 7.009654998779297

5. Image ID: 972 Distance: 7.044436454772949

6. Image ID: 872 Distance: 7.073945999145508

7. Image ID: 970 Distance: 7.164029121398926

8. Image ID: 870 Distance: 7.330625534057617

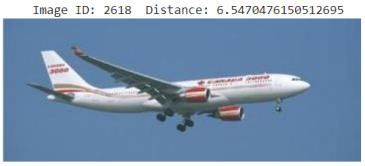
9. Image ID: 978 Distance: 7.55354642868042

10. Image ID: 878 Distance: 7.771692752838135

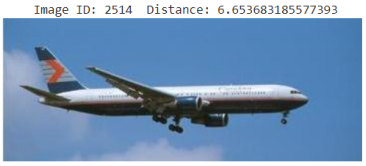
Locality-Sensitive Hashing Results: L = 3, h = 3, w = 5, t = 10, Image-ID = 2501, Feature Space = resnet

1. Image ID: 2630 Distance: 6.454123497009277

2. Image ID: 2618 Distance: 6.5470476150512695

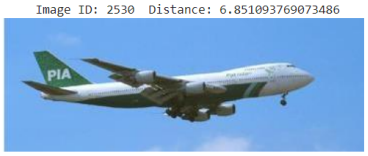


3. Image ID: 2514 Distance: 6.653683185577393



4. Image ID: 2412 Distance: 6.780489444732666

5. Image ID: 2530 Distance: 6.851093769073486



6. Image ID: 2550 Distance: 6.9408183097839355

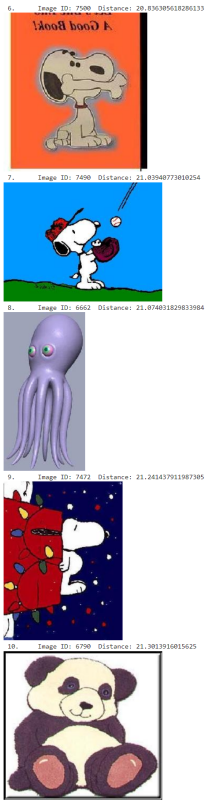
7. Image ID: 2590 Distance: 6.996563911437988

8. Image ID: 2612 Distance: 7.143371105194092

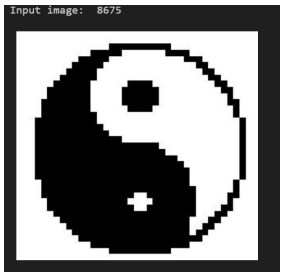
9. Image ID: 2684 Distance: 7.155673980712891

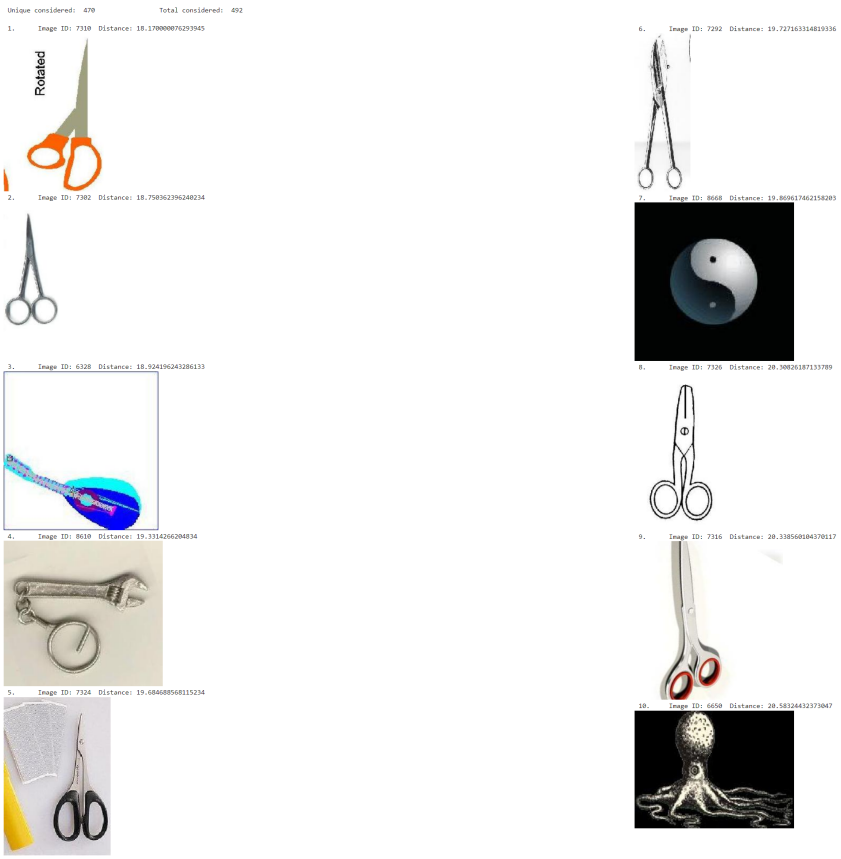
10. Image ID: 2472 Distance: 7.175907611846924

Locality-Sensitive Hashing Results: L = 3, h = 3, w = 5, t = 10, Image-ID = 5123, Feature Space = resnet

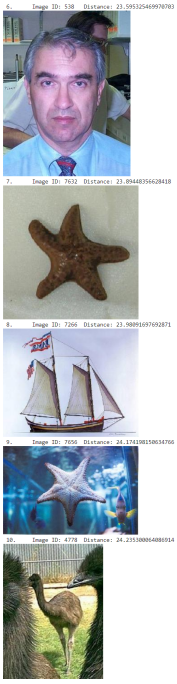


Locality-Sensitive Hashing Results: L = 3, h = 3, w = 5, t = 10, Image-ID = 8675, Feature Space = resnet





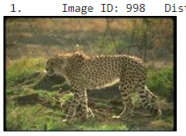
Locality-Sensitive Hashing Results: L = 3, h = 10, w = 5, t = 10, Image-ID = 1, Feature Space = resnet



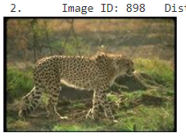
Locality-Sensitive Hashing Results: L = 3, h = 10, w = 5, t = 10, Image-ID = 881, Feature Space = resnet

Unique considered: 38 Total considered: 46

1. Image ID: 998 Distance: 6.0621562004089355

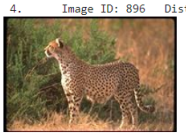


2. Image ID: 898 Distance: 6.399871826171875



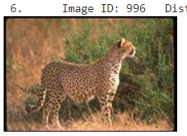
3. Image ID: 994 Distance: 6.573435306549072

4. Image ID: 896 Distance: 6.797601699829102



5. Image ID: 976 Distance: 6.931845188140869

6. Image ID: 996 Distance: 6.993564128875732



7. Image ID: 894 Distance: 7.009654998779297

8. Image ID: 972 Distance: 7.044436454772949

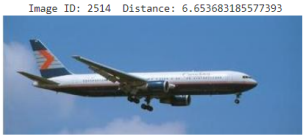
9. Image ID: 872 Distance: 7.073945999145508

10. Image ID: 970 Distance: 7.164029121398926

Locality-Sensitive Hashing Results: L = 3, h = 10, w = 5, t = 10, Image-ID = 2501, Feature Space = resnet

Unique considered: 79 Total considered: 80

1. Image ID: 2514 Distance: 6.653683185577393



2. Image ID: 2550 Distance: 6.9408183097839355

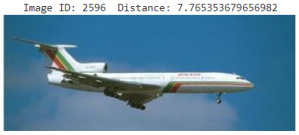
3. Image ID: 2590 Distance: 6.996563911437988

4. Image ID: 2472 Distance: 7.175907611846924

5. Image ID: 2436 Distance: 7.575500011444092

6. Image ID: 2216 Distance: 7.578601837158203

7. Image ID: 2596 Distance: 7.765353679656982

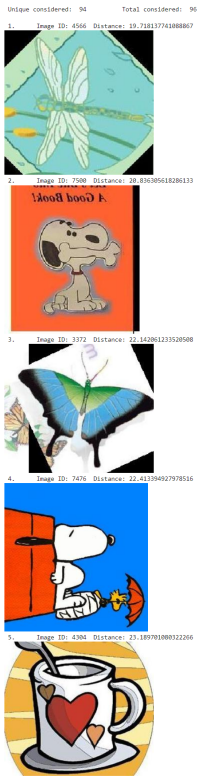


8. Image ID: 2548 Distance: 7.8660502433776855

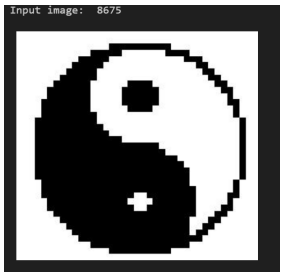
9. Image ID: 2708 Distance: 7.895928859710693

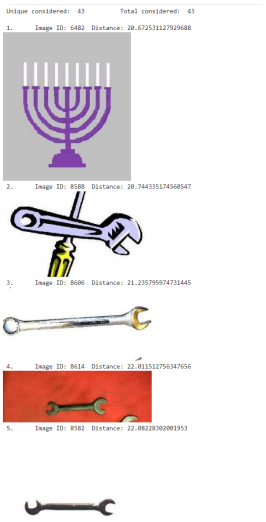
10. Image ID: 2592 Distance: 7.910541534423828

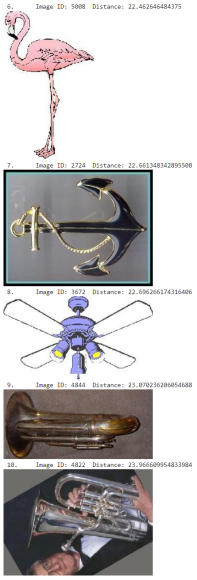
Locality-Sensitive Hashing Results: L = 3, h = 10, w = 5, t = 10, Image-ID = 5123, Feature Space = resnet



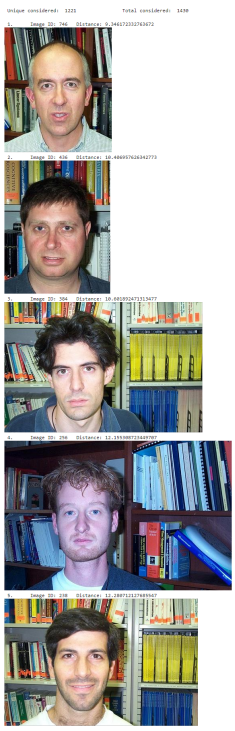
Locality-Sensitive Hashing Results: L = 3, h = 10, w = 5, t = 10, Image-ID = 8675, Feature Space = resnet







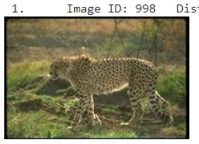
Locality-Sensitive Hashing Results: L = 10, h = 3, w = 5, t = 10, Image-ID = 1, Feature Space = resnet



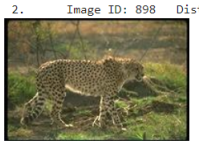
Locality-Sensitive Hashing Results: L = 10, h = 3, w = 5, t = 10, Image-ID = 881, Feature Space = resnet

Unique considered: 3005 Total considered: 4989

1. Image ID: 998 Distance: 6.0621562004089355

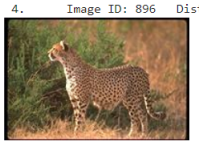


2. Image ID: 898 Distance: 6.399871826171875



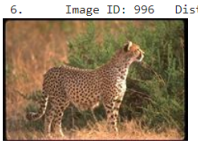
3. Image ID: 994 Distance: 6.573435306549072

4. Image ID: 896 Distance: 6.797601699829102



5. Image ID: 976 Distance: 6.931845188140869

6. Image ID: 996 Distance: 6.993564128875732



7. Image ID: 894 Distance: 7.009654998779297

8. Image ID: 972 Distance: 7.044436454772949

9. Image ID: 872 Distance: 7.073945999145508

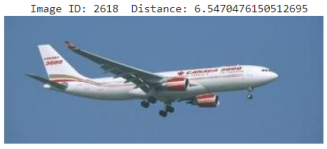
10. Image ID: 970 Distance: 7.164029121398926

Locality-Sensitive Hashing Results: L = 10, h = 3, w = 5, t = 10, Image-ID = 2501, Feature Space = resnet

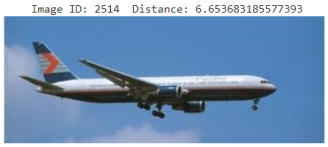
Unique considered: 2754 Total considered: 4

1. Image ID: 2630 Distance: 6.454123497009277

2. Image ID: 2618 Distance: 6.5470476150512695



3. Image ID: 2514 Distance: 6.653683185577393



4. Image ID: 2412 Distance: 6.780489444732666

5. Image ID: 2530 Distance: 6.851093769073486

6. Image ID: 2550 Distance: 6.940818309783935

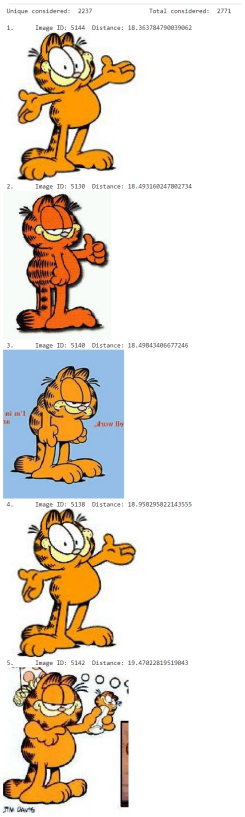
7. Image ID: 2590 Distance: 6.996563911437988

8. Image ID: 2612 Distance: 7.143371105194092

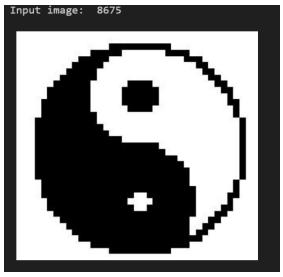
9. Image ID: 2684 Distance: 7.155673980712891

10. Image ID: 2472 Distance: 7.175907611846924

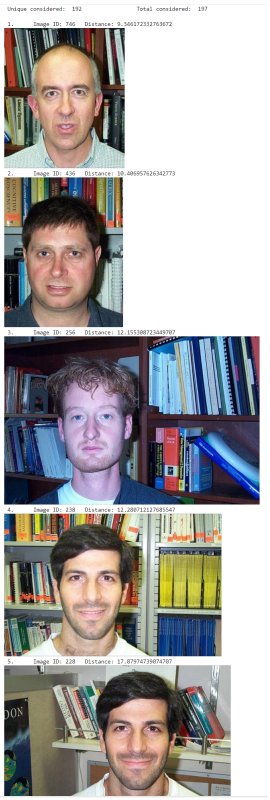
Locality-Sensitive Hashing Results: L = 10, h = 3, w = 5, t = 10, Image-ID = 5123, Feature Space = resnet



Locality-Sensitive Hashing Results: L = 10, h = 3, w = 5, t = 10, Image-ID = 8675, Feature Space = resnet



Locality-Sensitive Hashing Results: L = 10, h = 10, w = 5, t = 10, Image-ID = 1, Feature Space = resnet



Locality-Sensitive Hashing Results: L = 10, h = 10, w = 5, t = 10, Image-ID = 881, Feature Space = resnet

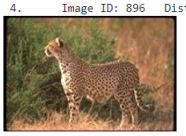
Unique considered: 38 Total considered: 50

1. Image ID: 998 Distance: 6.0621562004089355

2. Image ID: 898 Distance: 6.399871826171875

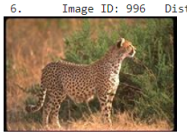
3. Image ID: 994 Distance: 6.573435306549072

4. Image ID: 896 Distance: 6.797601699829102



5. Image ID: 976 Distance: 6.931845188140869

6. Image ID: 996 Distance: 6.993564128875732



7. Image ID: 894 Distance: 7.009654998779297

8. Image ID: 972 Distance: 7.044436454772949

9. Image ID: 872 Distance: 7.073945999145508

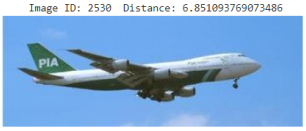
10. Image ID: 970 Distance: 7.164029121398926

Locality-Sensitive Hashing Results: L = 10, h = 10, w = 5, t = 10, Image-ID = 2501, Feature Space = resnet

Unique considered: 138 Total considered: 153

1. Image ID: 2412 Distance: 6.780489444732666

2. Image ID: 2530 Distance: 6.851093769073486



3. Image ID: 2550 Distance: 6.9408183097839355

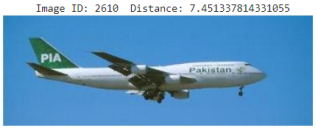
4. Image ID: 2590 Distance: 6.996563911437988

5. Image ID: 2612 Distance: 7.143371105194092

6. Image ID: 2684 Distance: 7.155673980712891

7. Image ID: 2472 Distance: 7.175907611846924

8. Image ID: 2610 Distance: 7.451337814331055

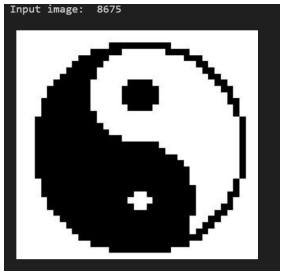


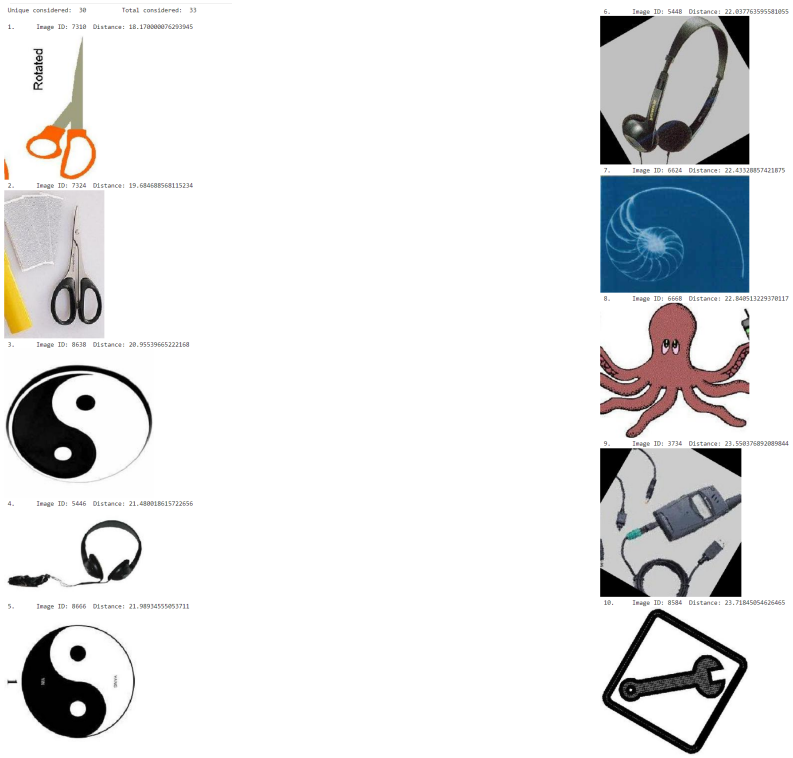
9. Image ID: 2594 Distance: 7.484836578369141

10. Image ID: 2500 Distance: 7.541537761688232

Locality-Sensitive Hashing Results: L = 10, h = 10, w = 5, t = 10, Image-ID = 5123, Feature Space = resnet

Locality-Sensitive Hashing Results: L = 10, h = 10, w = 5, t = 10, Image-ID = 8675, Feature Space = resnet





5.5.11 Task 4 - Analysis of Results

We observed decent performance in the LSH (that is, not too many unique and total vectors scanned, while still retrieving relevant vectors), using the input parameters as well as W=5. We observed that some images do not perform well. However, this was not due to the implementation of the LSH, but due to vectors within that image's label having substantial euclidean distance, resulting in other label vectors with around the same or lesser distance being ordered earlier. In addition to this, performance changes based on the random hash functions generated during the creation of the LSH. This also affects subsequent tasks.

5.6 Task 5

5.6.1 Goal Description:

Let us consider the tag set "Very Relevant (R+)", "Relevant (R)", "Irrelevant (I)", and "Very Irrelevant (I-)". Implement:

1. an SVM-based relevance feedback system,

2. a probabilistic relevance feedback system – use the following paper for reference: "Improving retrieval performance by relevance feedback." (by Gerard Salton and Chris Buckley). Journal of the American Society for Information Science. 41, pp. 288-297, 1990. [8]

which enables the user to tag some of the results returned by Task 4b and then return a new set of ranked results, relying on the feedback system selected by the user, either by revising the query or by re-ordering the existing results.

5.6.2 Assumptions:

1. We assume we can use any L, W, h combination for the Locality-Sensitive Hashing Index structure to demo the relevance feedback algorithms implemented. We have hardcoded an LSH index that we found to perform decently, that is, with L=10, h=3, W=5 with the resnet feature model.

2. Since there is no actual format of how the relevance feedback should be taken for each image, we define our own input mechanism.

3. Since we are using Locality-Sensitive Hashing (LSH) for evaluating the candidate set for the initial query, we assume we can directly re-use the same candidate set for re-ranking after relevance feedback is provided.

4. We fetch 30 times the given output size (t) of the images as our candidate images. This is because our implementation may return exactly t images which would make re-ranking images much harder.

5. We assume that we can run additional queries (query revision).

6. We assume that only negative feedback is not going to be provided in case of Probabilistic Relevance Feedback, our implementation is susceptible to deviation from the query in case of no positive feedback. If only negative feedback is provided we could get wrong candidate images.

7. We assume that only negative feedback is not going to be provided in case of SVM relevance feedback, our implementation only performs query revision on input of positive feedback. This is because purely negative feedback might result

in query revision in the wrong direction. If only negative feedback is provided, only re-ordering is performed.

8. We assume that the user might not label every result, but might label more than 1 at a time. However, the user must provide some new feedback to change the results.

5.6.3 Interface Specifications:

Stage 1:

1. First, we take the input image as a filename or a valid image ID in the Caltech101 Dataset.

2. Second, we take the number of images, t, to be displayed for each iteration of the feedback system.

3. Last, we take in the desired Feedback System between SVM-based or Probabilistic-based Relevance Feedback.

Stage 2 - Relevance Feedback System:

1. Now, for each of the presented t results, we require the user to provide the relevance tag (R+, R, I, I-) for any image of their choice in the format $\langle ImageID, Relevance \rangle$, separated by commas for multiple feedback inputs.

2. This feedback system is repeated until the user has completed the query process. At this moment, they can type in the letter Q to stop the execution.

5.6.4 Implementation Approach:

SVM-based Relevance Feedback System:

1. Initally, we query the LSH with the input image's query with a limit = t*100. The reason we do this is in-case the user gives negative feedback, we can at least re-order the results.

2. Using the same strategy as task 4, the LSH will rank the results, and provide an initial set of results, which we limit to t and display.

3. The user provides feedback for some subset of these t images.

4. Using this feedback, we train a binary SVM using an RBF (Guassian) hyperplane, which distinguish between relevant and irrelevant. We call this the "outer" SVM, since we have opted for a hierarchical approach. R or R+ feedback is considered relevant, I or I- feedback is considered irrelevant.

5. Once this is done, we then train two more SVMs, "inner" SVM for relevant, and "inner" SVM for irrelevant. These distinguish between R and R+, I and I- respectively.

6. An SVM is only trained when we have at least one example for both of its possible binary classifications. For example, if we have an R, R+ and I- feedback, we train the outer SVM, and only the inner relevant SVM, not the inner irrelevant SVM.

7. We globally keep track of all the feedback provided as a dictionary, and update this dictionary as we get new feedback. At each iteration, the SVMs are retrained.

8. Whenever we get a relevant feedback for an image, we update our set of candidates by querying the LSH with the new relevant image as a query.

9. At display time, all our candidates are classified hierarchically using the SVMs we have generated, and are ranked, first based on an ordinal numbering of the relevance, then based on the distance to the original query image.

Probabilistic Relevance Feedback System:

1. In case of the Probabilistic Relevance Feedback System, we first binarize the feedback vectors as well as the query vector. In our case we used the threshold as 0.5 since we are using the resnet layer which ranges from [0, 1].

2. We then calculate the conditional probabilities of a feature occurring given a particular label.

3. This is done by performing the division of the number of times a feature appears in a particular label (Relevant, Very Relevant, Irrelevant, Very Irrelevant) with the total number of times that label appears.

4. Once this is done, we can calculate the significance of each feature.

5. We then multiply the query vector with a constant and the significance vector and add it to the original query vector. (more details provided in 4)

6. We use this new query vector to find distances from the candidate list of the original images. This then gives us the top relevant images.

7. In this algorithm, we observed better results when we don't repeatedly fetch new candidates from the LSH index. We instead just update the query vector for re-ranking within candidate images.

8. We experimented with a lot of values and settled on an $\alpha = 1.2$ and $\beta = 0.3$ in the equations referred to in 4.

5.6.5 Results:

SVM-based Relevance Feedback System Results:

Initial Results for LSH: L = 10, h = 10, w = 5, t = 10, Image-ID = 4200, Feature Space = layer3

Relevance Feedback Provided to SVM-based System:

```
Updating based on new relevance feedback: Image ID: 4222 feedback: R
Updating based on new relevance feedback: Image ID: 4212 feedback: R+
Updating based on new relevance feedback: Image ID: 5356 feedback: I
Updating based on new relevance feedback: Image ID: 5424 feedback: I
Updating based on new relevance feedback: Image ID: 4054 feedback: I-
Updating based on new relevance feedback: Image ID: 4208 feedback: R
Updating based on new relevance feedback: Image ID: 3940 feedback: I-
Updating based on new relevance feedback: Image ID: 4264 feedback: R
Updating based on new relevance feedback: Image ID: 4272 feedback: R
```

Updated Results from SVM-based Relevance Feedback System:

1. Image ID: 4212 Relevance: R+ Distance: 6.146203517913818

2. Image ID: 4200 Relevance: Predicted as R Distance: 9.990588719220832e-06

3. Image ID: 4222 Relevance: R Distance: 5.7801666259765625

4. Image ID: 4206 Relevance: Predicted as R Distance: 6.3062920570373535

5. Image ID: 8498 Relevance: Predicted as R Distance: 6.402686595916748

6. Image ID: 7394 Relevance: Predicted as R Distance: 6.763057231903076

7. Image ID: 4266 Relevance: Predicted as R Distance: 6.777328968048096

8. Image ID: 4228 Relevance: Predicted as R Distance: 6.828713893890381

9. Image ID: 4208 Relevance: R Distance: 6.841832160949707

10. Image ID: 4262 Relevance: Predicted as R Distance: 6.889274120330810

Probabilistic-based Relevance Feedback System Results:
Initial Results for LSH: L = 10, h = 10, w = 5, t = 10, Image-ID = 4200, Feature Space = layer3

Relevance Feedback Provided to Probabilistic-based System:

```
Updating based on new relevance feedback: Image ID:  4222  feedback: R
Updating based on new relevance feedback: Image ID:  4212  feedback: R+
Updating based on new relevance feedback: Image ID:  8498  feedback: I-
Updating based on new relevance feedback: Image ID:  4266  feedback: R
Updating based on new relevance feedback: Image ID:  4064  feedback: I-
Updating based on new relevance feedback: Image ID:  4186  feedback: R
```

Updated Results from Probabilistic-based Relevance Feedback System:

5.6.6 Task 5 - Analysis of Results

SVM Based Feedback System: We tried using a one vs all classifier and didn't get significantly good results. Our results drastically improved when we used the nested SVM-based classification, especially using a Gaussian hyperplane. There are still some irrelevant images in the results due to the nature of SVM classification; however, we can overcome this by providing sufficient feedback to train the SVMs. Overall we feel that the results are quite good.

Probabilistic Relevance Feedback System: In this case, we see a significant difference between the performance of even and odd images. In the case of even images, just a few instances of feedback could significantly improve the results. In the case of odd images, it takes a few more inputs for relevant images to appear in the results. But in our subjective opinion the results appear to be really good.

6 System Requirements & Execution Instructions

The System & Dependency Requirements, and Execution Instructions are provided in the README.md file provided with this report. Furthermore, clear information on how to visualize the outputs for each respective task is also mentioned in the README.md file. Justifications for choosing a particular visualization standard for each major task were provided earlier in this Report.

Code quality, clear documentation and reports, & an easy-to-use user experience were prioritised during the development of this entire phase of the project. We also hope that our code is easily extensible for any future work based on this Course Project. During the development of the project, our goal was to ensure our code was easily readable and extensible as this final phase of this project incorporated aspects of Phases 1 & 2.

Moreover, for System Requirements (to be more specific), this code repository utilizes Juypter Notebooks developed in Visual Studio Code and Python. The code has been tested on Python versions 3.9 - 3.11; due to using the latest packages, we recommend using the later versions of Python such as 3.9 or higher. Additionally, we ran our code in venv Python package, regular Python installation, and MiniConda Python environment and both the latest builds of Mac OS and Windows 11.

Any reasonably modern hardware should be able to execute all the tasks, albeit slowly relative to the performance. We have found acceptable performance on Apple Silicon M2 Pro with 16GB unified ram. We have also verified on 10th generation Intel Core i7 10700k and 16GB ram. A GPU is not required. Sufficient storage of at least 30 GB is recommended to execute the project.

7 Related Works

The authors of [11] provide a comprehensive survey and review of the various image retrieval and re-ranking techniques proposed over the years. This helps give an organized overview of the work done in this domain for quick reference. The comparison table also allows evaluating different techniques for further research. Additionally, their research on multi-modal and multi-latent graph based re-ranking techniques are presented which integrate information from multiple visual features and modalities for more robust performance were particularly useful in giving us ideas about how to approach reranking of the results.

The key contributions by the authors in [12] is the Multi-bin search approach. Multi-bin search examines not just the target bin where the query descriptor falls, but also searches nearby neighboring bins within a distance threshold to account for

quantization errors in hashing. This significantly boosts the retrieval precision of binary hashing methods for large-scale image retrieval. We use a similar approach in our LSH implementation to expand the search bins.

8 Conclusions

1. We learnt a lot from Phase 3 of the project. In particular we have developed a deeper understanding in the topics of multi-dimentional indexing, query result reordering with feedback, classification and dimensionality analysis.

2. Finding the inherent dimensionality was tricky since there was no correct value, that was until we figured out how to use explained variance as a measure of dimensionality.

3. A few tasks took a lot of time to run as well as test and iterate upon, like PPR and Decision tree even after numerous optimizations.

4. We made different design decisions and carefully selected the parameters for the classifiers. We also analyzed their impact on the results.

5. Additionally, implementing LSH has helped us understand how multidimensional indexing would happen in real-world scenarios. The performance improvements and simplicity of LSH over traditional methods of searching were really surprising to us.

6. Query reordering was one of the hardest problems we tackled in this project, since it included incorporating user subjectivity into an otherwise objective implementation.

9 Acknowledgements

We want to thank Prof. Selcuk Candan for his guidance, support, and initiative in executing a comprehensive project covering key course topics and giving us insight into the nitty-gritty details of implementing such relevant algorithms. This Project taught us critical insights into the complexities of algorithm implementations for even such a well-defined dataset in the Caltech101 dataset. We would also like to thank the teaching staff for their support.

References

[1] SciPy Spatial Distance (Distance Measures) Formulae, "https://docs.scipy.org/doc/scipy/reference/spatial.distance.html"

[2] Web MIT Lectures, Singular Value Decomposition, "https://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm"

[3] Education Ecosystem (Towards Data Science), K-Means Clustering in Machine Learning, "https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1"

[4] Soner Yildrim (Towards Data Science), DBSCAN Clustering, "https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556"

[5] Onel Harrison (Towards Data Science), K-nearest neighbors https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[6] "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions" (by Alexandr Andoni and Piotr Indyk). Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122.

[7] Alexey Grigorev (Machine Learning Bootcamp), Euclidean LSH, http://mlwiki.org/index.php/Euclidean_LSH

[8] "Improving retrieval performance by relevance feedback." (by Gerard Salton and Chris Buckley). Journal of the American Society for Information Science. 41, pp. 288-297, 1990.

[9] Rohith Gandhi (Towards Data Science), Support Vector Machines from Scratch, https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[10] Sumeet Kumar Agrawal (Analytics Vidhya), Metrics to Evaluate your Classification Model, https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/

[11] Joshi, Mayuri & Deshmukh, Revati & N.Hemke, Kalashree & Bhake, Ashwini & Wajgi, Rakhi. (2014). Image Retrieval and Re-Ranking Techniques - A Survey. Signal & Image Processing: An International Journal. 5. 1-14. 10.5121/sipij.2014.5201.

[12] A. Kamel, Y. B. Mahdi and K. F. Hussain, "Multi-bin search: Improved large-scale content-based image retrieval," 2013 IEEE International Conference on Image Processing, Melbourne, VIC, Australia, 2013, pp. 2597-2601, doi: 10.1109/ICIP.2013.6738535.

A Specific Roles and Details of the Group Members

Caleb Panikulam - cpanikul@asu.edu

1. Made significant contributions to the implementation for Task 0a and 0b of this project.
2. Made relevant contributions around discussions for Task 5.
3. Proofread final report.

Gokul Vasudeva - gvasude2@asu.edu

1. Made significant contributions to Tasks 0 and Task 1.
2. Implemented tasks 2, 4 and, 5 (SVM) completely.
3. Made relevant contributions to the final project report.

Joshua Martin Noronha - jnoronha@asu.edu

1. Made significant contributions to Tasks 0.
2. Implemented tasks 3 (knn & ppr) and 5 (Probabilistic Feedback) completely.
3. Made relevant contributions to the final project report.

Kiran Sthanusubramonian - ksthanus@asu.edu

1. Primary handler for the final project report.
2. Made relevant contributions to tasks 1 and 5.
3. Modified code across the project to better represent results for the final project report.

Sandipan De - sandipan@asu.edu

1. Implemented tasks 3 (decision tree) completely.
2. Made relevant contributions to the final project report.

Shankar Harinarayanan - sharina1@asu.edu

1. Implemented task 1 completely.
2. Made contributions to the final project report.