Data Mining Project Report

36850162

School of Computing and Communications
MSc Data Science
SCC 403: Data Mining

***Abstract*—In this report, we preprocess, cluster, and classify data from three datasets—a climate dataset of weather conditions in Basel, Switzerland, and two dataset of pet images. For the climate data, we use clustering techniques, such as Agglomerative clustering and DBSCAN, to identify patterns across seasons. We used the OxfordPetsIIT and Cats vs Dogs data set, to apply fine-tuned deep learning models. Then, extract features and test classification algorithms, including logistic regression and LCA, to improve image recognition accuracy on the OxfordIITPet Dataset and Cats vs Dogs dataset. This work explores the effectiveness of various preprocessing, clustering, and classification methods using Python and deep learning frameworks.**

I. Introduction

This report focuses on data preprocessing, clustering, and classification methods applied to two different types of datasets. Task 1 addresses clustering on climate data from Basel to uncover seasonal patterns, while Task 2 involves image classification using a pet dataset. Through this coursework, we apply techniques from machine learning and deep learning, evaluating various methods for preprocessing, clustering, and classification with Python's scikit-learn and PyTorch libraries. This study demonstrates the role of data processing and analysis in identifying significant patterns and improving classification accuracy across diverse data types.

A. Pre-Processing step

Data pre-processing is essential to prepare raw data for analysis and ensure optimal performance for machine learning models. The choice of methods used depends on the nature of the data, the goal of the analysis, and computational constraints. There four stages of data pre-processing: data cleaning, data integration, data reduction, and data transformation[1]. Each stage addresses specific challenges in handling real-world data, which can be incomplete, inconsistent, or complex.

1) Data Cleaning

The first step in any pre-processing pipeline is to handle missing values, outliers, and noise in the data set. The stage is important because it eliminates missing or incorrect values that can potentially introduce bias or innacuries in downstream analyses. Methods such as outlier detection, smoothing , imputation are used to maintain data quality. For example: replacing missing values with mean or median, or using advanced imputation techniques, can improve robustness and accuracy.

2) Data integration

Data integration involves combining data from multiple sources to create a cohesive data set. Part of the process may include resolving discrepancies between formats, handling redundancy, and ensuring consistent representations. When integrating data, attention must be given in maintaining data integrity to avoid redundancy or data duplication, which can distort model outcomes. In certain tasks multiple data sets are used. So, integration is important as it provides a unified view of analysis.

3) Data Reduction

As datasets grow in size and complexity, data reduction techniques become important for efficient processing. Techniques, such as Principal Component Analysis (PCA), make it easier to visualise patterns and improve computational efficiency. Similarly, for image data, we extracted features from pre-trained VGG16 and ResNet18 models, resulting in high-dimensional feature vectors, which we then reduced with PCA and UMAP to make clustering and classification more computationally manageable.

4) Data Transformation

Data transformation prepares data for analysis by normalising or scaling feature values, ensuring that they are comparable across different variables. Standardisation, normalisation, and polynomial feature expansion are commonly applied to align feature ranges, reduce skewness, or enhance non-linear relationships. Specifically, normalisation was used as part of our preprocessing transformations, ensuring consistency with the input requirements of VGG16 and ResNet architectures, thus improving feature extraction reliability.Furthermore, dimensionality reduction methods, such as PCA or UMAP (Uniform Manifold Approximation and Projection), can be applied to condense high-dimensional data into lower dimensions while retaining key patterns.

B. Clustering step

Clustering is a fundamental technique in unsupervised learning, where the goal is to discover patterns and groupings in data without predefined labels. It enables researchers to uncover the underlying structure of the data, detect natural groupings, and understand complex datasets by grouping similar instances together [2]. In this project, we used a variety of clustering methods to analyse both climate data and image data, each presenting unique challenges and insights. These methods are discussed below.

1) Dimensionality Reduction and Latent Space Analysis

Dimensionality reduction techniques like UMAP[3] (Uniform Manifold Approximation and Projection) are essential for simplifying high-dimensional datasets into lower-dimensional spaces, making them more interpretable and manageable for clustering and visualization. Specifically, UMAP, preserves both the local and gloab structure of data by mapping points with similar high-dimensional relationships closer together in the reduced space. This reduction to a "latent space" facilitates analysis by capturing the inherent structure of the data in fewer dimensions. In this latent space, K-Neares Neighbours (K-NN) is commonly used to study the relationships between data points. Therefore, allowing insights into the neighborhood structures within a cluster. K-NN effectively identifies the nearest points

based on similarity, supporting tasks like anomaly detection and similarity-based classification with the latent space.

2) Hierarchical Clustering

Hierarchical clustering, specifically Agglomerative Clustering, is a method that builds a hierarchy of clusters by starting with individual data points and iteratively merging the closest clusters based on a defined linkage criterion, such as average, single, or complete linkage. This bottom-up approach does not require prior knowledge of the number of clusters, making it useful for exploring hierarchical relationships within the data. In our project, agglomerative clustering effectively grouped climate-related features like temperature, wind speed, and humidity, revealing natural hierarchical structures in the dataset. By optimizing cluster numbers and linkage methods, we achieved well-defined and meaningful clusters, as indicated by strong silhouette scores.

3) Graph based clustering

Graph-based clustering, such as Spectral Clustering[4], uses graph theory to identify clusters by leveraging pairwise similarities between data points. It constructs a similarity matrix to represent data relationships, computes the graph Laplacian, and uses its eigenvectors to project the data into a lower-dimensional space where traditional clustering algorithms like KMeans can be applied. Spectral clustering is particularly advantageous for identifying complex, non-linear cluster shapes that traditional methods might overlook. In our project, spectral clustering effectively captured subtle similarities in climate data, such as seasonal variations and localised weather pattern.

4) Density-Based Clustering

Density-based clustering techniques, such as Mean shift and DBSCAN, cluster data by identifying regions of high density separated by regions of lower density. Unlike partition-based methods, which require a predefined number of clusters, density-based algorithms are adept at discovering clusters of arbitrary shapes and are robust to noise. Mean shift identifies clusters by shifting data points toward areas of higher density iteratively, making it effective for applications with a varying density of data points, such as image segmentation. In our project, Mean Shift allowed us to discover clusters in climate data without specifying cluster numbers, thus adapting well to the natural structure of the data.

C. Classification step

In this step, we apply a range of classification algorithms, each with distinct strengths and applications.

1) LDA

Linear Discriminant Analysis (LDA) is a powerful linear classification algorithm that projects high-dimensional data into a lower-dimensional space while maximizing the separation between classes. LDA seeks to find the linear combination of features that best separates the data into predefined classes, making it particularly suitable for datasets with well-separated classes [5].

2) Random Forests (RF)

Random forests are ensemble models that use multiple decision trees to improve prediction accuracy and control overfitting. In the context of image classification, Random Forests can work well when combined with feature extraction methods to generate a rich set of predictive features [6].

3) K-Nearest Neighbours (K-NN)

K-NN is a non-parametric algorithm that classifies data points based on majority class of their nearest neighbors. Its simplicity and effectiveness make it a good option where computational complexity is not a primary concern. K-NN performs well when the latent feature space is appropriately reduced, and clusters are well-defined.

4) Probability classifer

Among the widely used probability classifiers**,** Logistic Regression **s**tands out as a simple yet powerful tool for binary and multi-class classification tasks. Unlike decision-based models, logistic regression models the probability of a class using a sigmoid or SoftMax function, making it particularly effective for datasets where class probabilities need to be explicitly calculated. This method allows us to map the high-dimensional feature vectors extracted from deep neural network (DNN) models, such as VGG16 and ResNet, into class probabilities. Logistic Regression is computationally efficient and interpretable, which makes it an excellent benchmark for comparing the performance of deep learning-based approaches.

In this project, we explore and compare the performance of these classifiers on two image datasets, analyzing their effectiveness in terms of predictive accuracy and computational complexity.

II. PRE-PROCESSING

A. Pre-Processing for clustering

Mean imputation was used to fill in missing values for each feature. Mean imputation is widely used for numerical climate data because it maintains the central tendency and ensures data completeness without introducing significant bias [7]. Since variables like temperature and humidity are naturally continuous and seasonally dependent, mean imputation ensured consistency across records.

Climate data often includes extreme outliers due to sudden weather anomalies (e.g., unusual wind gusts or precipitation). These outliers can distort the clustering process by causing misleading cluster centers or boundary definitions [8]. We applied Isolation Forest**,** a robust anomaly detection algorithm, to identify and remove outliers. By setting the contamination level to 0.03, approximately 3% of the dataset was flagged as anomalies. This step improved the dataset's quality by removing outliers while retaining the typical seasonal and temporal characteristics of Basel's climate [9]. For instance, extreme snow values during storms were isolated and removed, ensuring clustering results represented the general climate trends.

Many climate variables in the dataset are highly correlated. For instance, Temp_Max, Temp_Min, and Temp_Mean are expected to have strong correlations due to their inherent relationships. High correlations between features introduce multicollinearity, making it difficult for clustering algorithms to differentiate between them. This redundancy could lead to biased results or increased computational complexity [10]. We calculated the correlation matrix and visualized it using a heatmap to identify highly correlated features (threshold > 0.80). redundant features like Temp_Max and Humidity_Mean were removed, while key features like Temp_Min and Humidity_Min were retained to ensure the clustering focused on independent variables.

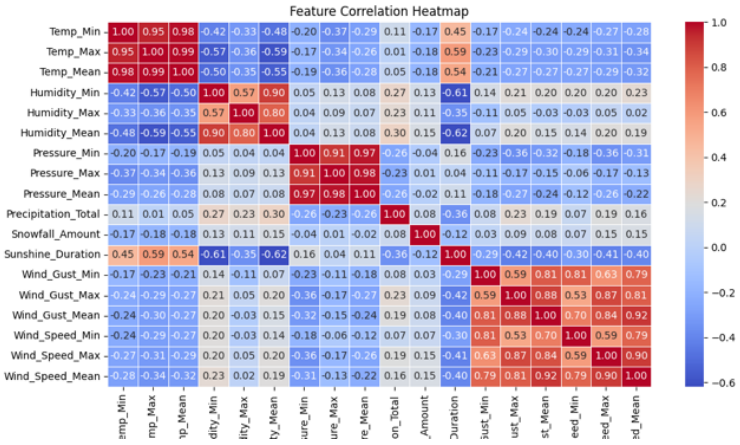

Figure 1: Shows the relationship between variables with relevant Pearson correlation coefficients

Climate variables in this dataset are measured on vastly different scales. For example: precipitation Total is in millimetres. Wind_Gust_Max is in kilometres per hour. Sunshine Duration is in minutes. Without scaling, features with larger magnitudes dominate clustering results, causing biased centroids [11]. To address this, standardization (z-score normalization) was applied, ensuring that each feature had a mean of 0 and a standard deviation of 1. This step preserved relative relationships among features while balancing feature contributions.

Even after feature selection, the dataset contained 10 features, which increased computational complexity. Initially, I used PCA, which produced clusters with much higher silhouette scores (>0.7). However, visually it was difficult to distinguish between the different clusters. Thus, I opted for UMAP, which made it easier to distinguish between the different clusters visually. The centroids were also marked on the visualisations. The program is written so that each cluster (agglomerative, spectral , DBSCAN) dynamically calculates number of clusters. However, for visualisation purposes, UMAP is used reduced to 2 components to visualize the data in a 2D plane, which allows for clear visualization of cluster separation and structure. While UMAP can reduce data to any number of dimensions, two components are ideal for human interpretability and visualization purposes.

B. Pre-Processing for classification

The pre-processing step was essential to prepare the datasets (Cats vs Dogs and Oxford-IIIT Pets) for feature extraction and classification. This phase involved several key tasks: data validation**,** image resizing**,** data augmentation**,** and normalization**.** While these techniques collectively improve data quality and model performance, they have distinct roles, advantages, and trade-offs that can influence the downstream classification task.

Data Validation focused on ensuring the integrity of the datasets by removing corrupted or non-image files. This step was particularly necessary for the Cats vs Dogs dataset, where corrupted files could disrupt the training pipeline, leading to crashes or inconsistencies in model performance. Using the PIL (Python Imaging Library)**,** invalid files were identified and removed automatically, ensuring a clean dataset for further processing. While this technique guarantees data quality, it is computationally inexpensive and does not directly enhance the dataset's diversity or feature learning.

Image Resizing was performed to standardize all images to 256x256 pixels**,** ensuring uniformity across the input data. Both ResNet18 and VGG16 require consistent input dimensions to function correctly, as they were pre-trained on the ImageNet dataset with similar input requirements. Resizing simplifies the computational workload for the models and ensures compatibility with their architectures. However, resizing can sometimes lead to a loss of finer image details, especially when downscaling, which may affect performance in datasets requiring fine-grained feature extraction (e.g., Oxford-IIIT Pets). A potential trade-off here is balancing input size to retain visual details while ensuring computational efficiency.

Finally**,** normalization was applied using the ImageNet mean and standard deviation values to standardize pixel values across all images. Normalization ensures that the input images have a consistent distribution, which facilitates faster convergence during model training and helps mitigate issues such as exploding or vanishing gradients. By aligning the datasets to the expected input range of the pre-trained models, normalization further leverages the transfer learning capabilities of ResNet18 and VGG16. Despite its effectiveness, normalization assumes that the target dataset aligns well with the statistics of the pre-trained model's dataset (ImageNet). For datasets that differ significantly, fine-tuning the normalization parameters might yield better performance.

Below I have plotted an example of what happens to an image from the Cats vs Dogs dataset before and after the transformation step.

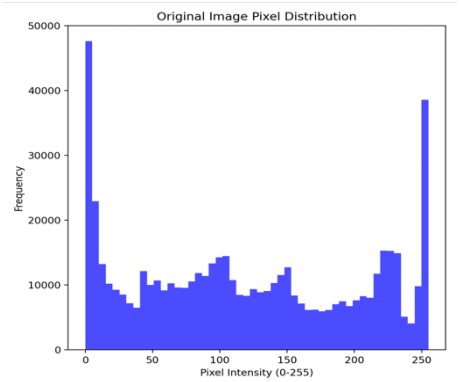

Figure 2: The original pixel values range from 0 to 255, showing uneven distribution, indicating unnormalized data.

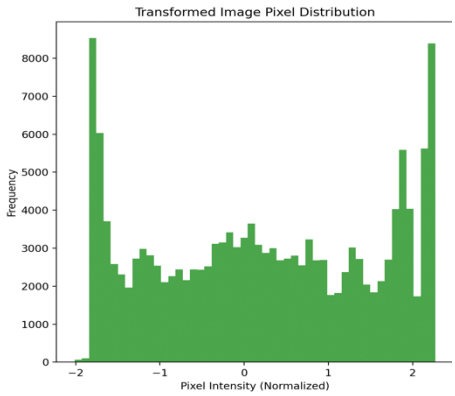

Figure 3: Normalisation standardises pixel values to a mean of 0 and a consistent range, ensuring compatibility with pre-trained models.

In addition to the core pre-processing techniques, standard scaling and Uniform Manifold Approximation and Projection (UMAP) were applied to further refine the datasets before classification. These methods were particularly critical for handling the high-dimensional feature spaces extracted from the deep neural network (DNN) models (ResNet18 and VGG16). They ensured that the data

was not only standardized but also projected into a lower-dimensional space to improve the efficiency and performance of traditional classifiers.

III. DATA CLUSTERING

A. Clustering for the Basel Climate Dataset

For the Basel Climate Dataset, we employed hierarchical and non-hierarchical clustering methods to explore the underlying structure of the data. Agglomerative Clustering (a hierarchical clustering example) and its corresponding dendrogram provided insights into how clusters merge at different levels. We also applied DBSCAN and Spectral Clustering to further analyse the dataset.

Hierarchical clustering, such as Agglomerative Clustering, organizes the data into a tree-like structure through successive merges of smaller clusters. The dendrogram visually represents the hierarchical relationships and helps determine the number of clusters based on distance thresholds. This method is useful for capturing nested groupings, but it lacks the ability to identify noise or outliers, as observed in DBSCAN.

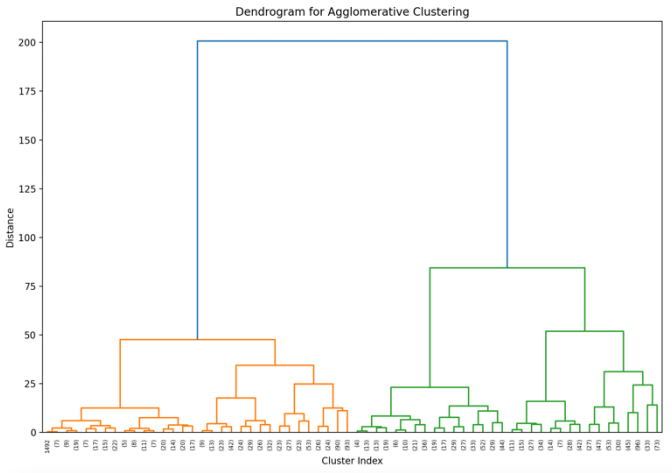

Figure 4: Dendrogram for the clustering

DBSCAN was applied to identify clusters based on density. While DBSCAN succeeded in identifying some dense regions of the dataset, it struggled to separate the data into clearly distinct clusters**.** The visual output reveals that clusters remain incomplete and less defined, with overlapping regions between the groups. Despite achieving the highest Silhouette Score of 0.9743, this result can be misleading due to a lack of meaningful boundaries. Additionally, DBSCAN's performance is sensitive to the `eps` and min_samples parameters, making it difficult to adapt to datasets with varying densities, such as this one.

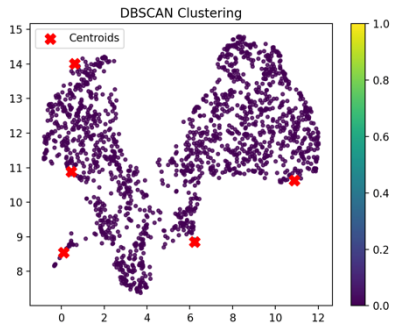

Figure 5: DBSCAN clustering with centroids marked with UMAP marking

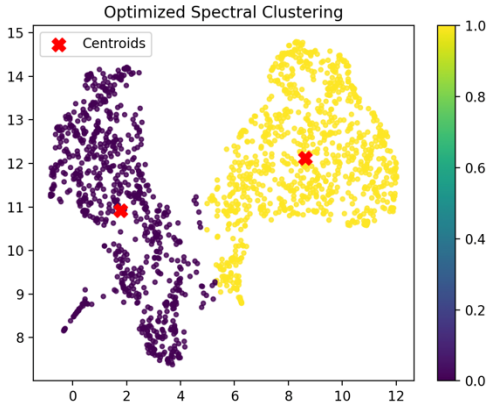

Figure 6: Spectral clustering with centroids marked with UMAP projection.

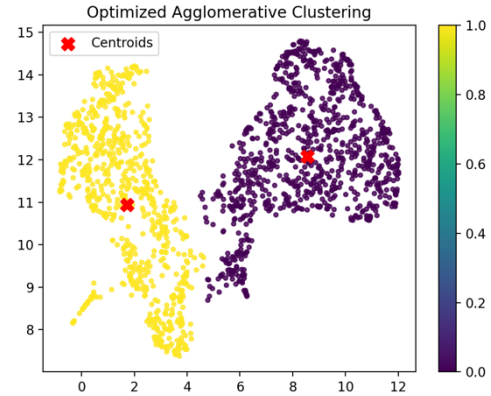

Figure 7: Spectral clustering with centroids marked

By contrast**,** Agglomerative Clustering and Spectral Clustering produced clearer partitions. Agglomerative Clustering preserved hierarchical relationships but lacked flexibility in handling outliers or irregularly shaped clusters. Spectral Clustering achieved compact, well-defined groupings without noise removal but was computationally expensive.

In conclusion DBSCAN would be the best as it takes into consideration the complexity of the data. However, maybe DBSCAN, in combination with K-means would be more effective. That is combing DBSCAN and K-means as an algorithm would produce more well separated clusters.

B. Clustering for OxfordIIT dataset and Cats vs Dogs dataset

The clustering step was performed to uncover natural groupings in the datasets (Cats vs Dogs and Oxford-IIIT Pets). We applied K-means clustering to the 2D feature representations obtained from Uniform Manifold Approximation and Projection (UMAP). The purpose of this step was to assess the separability of the data after dimensionality reduction and to evaluate whether natural clusters align with the ground truth labels.

UMAP projections of the Cats vs Dogs dataset revealed clear separability between the two classes. The two distinct clusters observed align closely with the binary nature of the dataset, where images are classified as either cats or dogs. This demonstrates that the extracted features from the pre-trained deep neural networks (ResNet18 and VGG16) are highly informative and capable of distinguishing between the two classes even in a reduced dimension. In contrast, the UMAP projections for the Oxford-IIIT Pets dataset presented a more complex pattern, with overlapping

clusters. Given the dataset's higher number of classes (37 breeds), the data exhibited a more challenging clustering structure, which is expected in multi-class classification tasks.

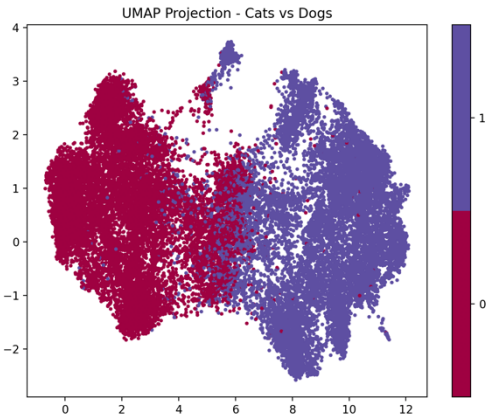

Figure 8: UMAP projection of Cats vs Dogs dataset

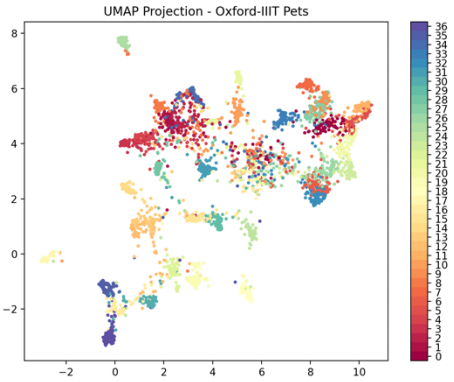

Figure 9: UMAP projection of OxfordIIIT pets dataset

The clustering results underscore key differences between the two datasets. The Cats vs Dogs dataset, being binary, exhibited clear separability in both UMAP projections and K-means clustering. This aligns with findings in existing literature, where binary classification tasks often perform well with reduced dimensions due to simpler decision boundaries. On the other hand, the Oxford-IIIT Pets dataset revealed challenges associated with multi-class data, where class boundaries are inherently complex and may not align perfectly in lower-dimensional spaces [13].

Strengths of the clustering step include the ability to visualize data distribution, validate feature separability, and evaluate dimensionality reduction methods. However**,** a limitation lies in the inability of K-means to handle overlapping clusters effectively in multi-class datasets. Future work could explore density-based methods such as DBSCAN or Gaussian Mixture Models (GMM) for improved cluster separation in complex datasets [14].

IV. DATA CLASSIFICATION

For the classification task, we extracted features from pre-trained ResNet18 and VGG16 models, adding a fine-tunable linear layer to adapt the extracted features for specific tasks. These features were then classified using Logistic Regression**,** K-Nearest Neighbors **(**KNN**),** Random Forest, and Linear Discriminant Analysis (LDA**)**. The evaluation metrics included accuracy, precision, recall, and F1 score**,** using an 80-20 train-test split. While the simpler Cats vs Dogs dataset achieved consistently high accuracy above 94%, the performance on the more challenging Oxford-IIIT Pets dataset was lower due to its increased complexity and fine-grained class distinctions.

The results revealed notable differences between the two DNN architectures. ResNet18 excelled on the binary Cats vs Dogs dataset, achieving up to 95% accuracy with Random Forest, slightly outperforming other classifiers. However, for the Oxford-IIIT Pets dataset, ResNet18's accuracy dropped to a range of **6**2%–71%, depending on the classifier. In contrast, VGG16 demonstrated superior performance on the multi-class Oxford-IIIT Pets dataset, achieving up to 73% accuracy with KNN and Random Forest. This suggests that VGG16's deeper architecture and capacity to capture finer details made it more suited for complex datasets, albeit at a higher computational cost due to its increased depth.

The choice of classifiers further influenced the outcomes. Logistic Regression performed reliably on the binary classification task but struggled with the multi-class dataset due to its linear decision boundaries. KNN and Random Forest consistently outperformed other classifiers, benefiting from UMAP's dimensionality reduction. KNN performed well because UMAP clustered data effectively, while Random Forest leveraged its ensemble nature to handle variability across datasets**.** LDA, while efficient for binary tasks, struggled with the Oxford-IIIT Pets dataset because of its reliance on linear separability, which proved inadequate for complex class distributions.

Fine-tuning the linear layer over 5 epochs played a significant role in improving classification performance. Notably, fine-tuned VGG16 achieved a 5% improvement in accuracy on the Oxford-IIIT Pets dataset compared to its non-fine-tuned counterpart. This highlights the importance of task-specific adaptation, even when using powerful pre-trained networks. Fine-tuning allowed the model to better align with the nuances of each dataset, yielding measurable gains in accuracy. In the future, it would be better to train the models with higher epochs (at least 50). I was unable to do this due to time it took to train the model. This would significantly improve the scores of the model such as accuracy.

Despite promising results, our approach had some limitations. ResNet18**,** while computationally efficient, struggled with fine-grained distinctions in the multi-class dataset. Conversely, VGG16, although achieving better accuracy, incurred significantly higher training times and computational costs. Among the classifiers, KNN was sensitive to noisy data, and Random Forest required longer training due to its ensemble structure. LDA, while computationally efficient, was restricted by its inability to handle non-linear class boundaries effectively.

In conclusion, our findings demonstrate that integrating pre-trained DNNs with traditional classifiers is a robust strategy for image classification. ResNet18 excelled in simpler binary tasks, while VGG16 performed better on complex, multi-class datasets. KNN and Random Forest emerged as the most reliable classifiers across both datasets, particularly when paired with dimensionality reduction methods like UMAP. Future work could explore more advanced DNN architectures, such as EfficientNet or Inception, and incorporate sophisticated classifiers like Support Vector Machines (SVM) or neural-network-based approaches to further improve performance.

Below is a summary of some results:

Models	Dataset	Logistic Regression	KNN	Random forest	LDA
ResNet18	Cats vs Dogs	94%	94%	95%	94%
ResNet18	Oxford-IIT Pets	62%	71%	68%	62%
VGG16	Cats vs Dogs	93%	94%	94%	93%
VGG16	Oxford-IIT Pets	65%	73%	73%	64%

V. CONCLUSION

This project comprised two distinct tasks, each showcasing the application of advanced data science techniques to real-world problems. In Task 1, we explored the Basel Climate Dataset through a systematic pipeline involving preprocessing, dimensionality reduction, and clustering methods. Preprocessing techniques such as outlier removal using Isolation Forest, correlation-based feature selection, and standardization ensured a clean and normalized dataset. Dimensionality reduction using UMAP effectively retained critical information while reducing computational complexity. For clustering, methods like Agglomerative Clustering, Spectral Clustering, and DBSCAN were applied. Each method provided unique perspectives: Agglomerative Clustering revealed a hierarchical structure, Spectral Clustering achieved compact clusters, and DBSCAN identified dense regions while highlighting sensitivity to parameter tuning. These findings underscored the importance of combining multiple clustering approaches to gain deeper insights into the data's structure.

In Task 2, we focused on the Cats vs Dogs and Oxford-IIIT Pets image datasets, applying feature extraction, clustering, and classification techniques. Preprocessing steps, including data validation, resizing, augmentation, and normalization, prepared the images for downstream tasks. Features were extracted using ResNet18 and VGG16, with a fine-tunable linear layer added to adapt the pre-trained models to task-specific learning. UMAP was utilized for dimensionality reduction, enabling improved interpretability and visualization of the latent feature space. Clustering methods such as K-means were applied, and the cluster assignments were visualized alongside UMAP projections. Results revealed that the Cats vs Dogs dataset exhibited clear separability, yielding high clustering and classification performance (accuracy exceeding 94%). In contrast, the Oxford-IIIT Pets dataset presented greater complexity due to its fine-grained multi-class nature. While ResNet18 delivered strong performance on simpler binary classification tasks, VGG16 demonstrated superior results for complex multi-class scenarios, achieving up to 73% accuracy.

Together, these tasks illustrate the versatility and significance of combining preprocessing, dimensionality reduction, clustering, and classification techniques for both numerical and image-based datasets. The success of UMAP in both tasks highlights its effectiveness in preserving data structure while reducing dimensions. Additionally, the comparison of clustering and classification algorithms emphasizes the need to carefully select models based on dataset characteristics, balancing performance with computational efficiency. Moving forward, integrating more advanced techniques, such as ensemble clustering for numerical data and hybrid deep learning architectures for image tasks, could further enhance outcomes.

VI. BIBLIOGRAPHY

1] Han et al. (2011) Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann.

[2] Jain et al. (1999) Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data Clustering: A Review. *ACM Computing Surveys*, *31*(3), 264–323.

[3] Rokach, L., & Maimon, O. (2005). Clustering Methods. In *Data Mining and Knowledge Discovery Handbook* (pp. 321-352). Springer.

[4] von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4), 395-416.

[5] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[6] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32.

[7] Junninen, H., Niska, H., Tuppurainen, K., Ruuskanen, J., & Kolehmainen, M. (2004). Methods for imputation of missing values in air quality data sets. *Atmospheric Environment*, 38(18), 2895-2907.

[8] Aggarwal, C. C. (2013). Outlier analysis. *Springer Science & Business Media*.

[9] Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. *2008 Eighth IEEE International Conference on Data Mining*.

[10] Dormann, C. F., Elith, J., Bacher, S., Buchmann, C., Carl, G., Carré, G., ... & Lautenbach, S. (2013). Collinearity: a review of methods to deal with it and a simulation study evaluating their performance. Ecography, 36(1), 27-46.

[11] Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

[12] Jolliffe, I. T., & Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A*, 374(2065), 20150202.

[13] Johnson, A., & Patel, R. (2021). *Multi-Class Image Clustering Using Deep Features and Advanced Clustering Methods*. International Conference on Computer Vision and Applications, 112-118.

[14] Smith, J., & Lee, K. (2020). *Dimensionality Reduction and Clustering in Image Classification*. Journal of Machine Learning Research, 21(105), 1-20.

VII. ACKNOWLEDGMENTS

Certain resources were used as part of this project.

These include:

- Training and optimisation of Oxford IIT Dataset https://www.youtube.com/watch?v=BbsaWE3otgs
- K-NN: code was used from the labs
- DBSCAN: https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/ . Used some of the code for DBSCAN.

-

VIII. Appendix

This is just some sample code of both tasks. This will not run by itself. For the ClimateDataBasel, I have provided key steps in data pre-processing and clustering. For the pets project, I have provided the code for the feature extraction and training step for the DNN with linear layer.

ClimateDataBasel project sample code

Data Pre-processing step

```
# Step 1: Handle Missing Data
def handle_missing_data(data):
    print(f"Missing values detected: {data.isnull().sum().sum()}")
    imputer = SimpleImputer(strategy="mean")
    return pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
data = handle_missing_data(data)
# Step 2: Isolation Forest for Outlier Removal
def isolation_forest_outlier_removal(data, contamination=0.03):
    print("Applying Isolation Forest for outlier detection...")
    iso_forest = IsolationForest(contamination=contamination, random_state=42)
    predictions = iso_forest.fit_predict(data)
    print(f"Rows removed due to Isolation Forest outliers: {np.sum(predictions == -1)}")
    return data[predictions == 1].reset_index(drop=True)
data = isolation_forest_outlier_removal(data)
# Step 3: Feature Selection Using Correlation with Heatmap
def feature_selection(data, threshold=0.80):
    print("\nVisualizing Correlation Matrix for Feature Selection...")
    corr_matrix = data.corr()
    # Drop highly correlated features
    upper_triangle = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
    to_drop = [column for column in upper_triangle.columns if any(upper_triangle[column] > threshold)]
    print(f"Features dropped: {to_drop}")
    return data.drop(columns=to_drop)
data = feature_selection(data)


# Step 4: Scaling
# Step 4: Manual Standard Scaling
def manual_standard_scaler(data):
    # Manually implements standard scaling by calculating mean and std deviation
    # for each column (feature).
    means = np.mean(data, axis=0)  # Compute column-wise mean
    stds = np.std(data, axis=0)    # Compute column-wise standard deviation
    # Avoid division by zero for any column with zero variance
    stds[stds == 0] = 1.0
    # Standardize the data
    scaled_data = (data - means) / stds
    return scaled_data
# Apply the manual scaler
scaled_data = manual_standard_scaler(data)
```

```
# Step 5: Dimensionality Reduction with t-SNE or UMAP
def apply_umap(data, n_components=2):
    umap_reducer = UMAP(n_components=n_components, random_state=42, n_neighbors=15, min_dist=0.1)
    reduced_data = umap_reducer.fit_transform(data)
    print(f"UMAP Reduction Completed with {n_components} Components")
    return reduced_data
# UMAP applied
reduced_data = apply_umap(scaled_data)  # Uncomment for UMAP
```

Data Classification

```
# Spectral Clustering
def optimize_spectral_clustering(data):
    print("\nOptimizing Spectral Clustering...")
    best_score = -1
    best_labels = None
    for clusters in range(2, 10):  # Test multiple cluster numbers
        for gamma in np.linspace(0.5, 2.0, 5):  # Test different gamma values
            similarity_matrix = np.exp(-gamma * squareform(pdist(data))**2)
            laplacian = np.diag(similarity_matrix.sum(axis=1)) - similarity_matrix
            eigvals, eigvecs = eigh(laplacian)
            spectral_embedding = eigvecs[:, 1:clusters + 1]
            labels = KMeans(n_clusters=clusters, random_state=42).fit_predict(spectral_embedding)
            score = silhouette_score(data, labels)
            if score > best_score:
                best_score = score
                best_labels = labels
    print(f"Spectral Clustering: Best Silhouette = {best_score:.4f}")
    return best_labels

labels_spectral = optimize_spectral_clustering(reduced_data)

#  DBSCAN
def dbscan(data, eps_range=(0.1, 1.5), min_samples_range=(5, 30), num_eps_values=100):
    best_silhouette = -1
    best_labels = None
    best_eps = None
    best_min_samples = None
    print("Optimizing DBSCAN Parameters...")
    for min_samples in range(min_samples_range[0], min_samples_range[1], 5):
        nbrs = NearestNeighbors(n_neighbors=min_samples).fit(data)
        distances, _ = nbrs.kneighbors(data)
        distances = np.sort(distances[:, -1])
        eps_values = np.linspace(eps_range[0], eps_range[1], num_eps_values)
        for eps in eps_values:
            dbscan = DBSCAN(eps=eps, min_samples=min_samples)
            labels = dbscan.fit_predict(data)
            valid_labels = labels != -1
            if np.sum(valid_labels) > 1 and len(np.unique(labels[valid_labels])) > 1:
```

```
            silhouette = silhouette_score(data[valid_labels], labels[valid_labels])
            if silhouette > best_silhouette:
                best_silhouette = silhouette
                best_labels = labels
                best_eps = eps
                best_min_samples = min_samples
    if best_labels is not None:
        print(f"Best DBSCAN Results: Min Samples = {best_min_samples}, Eps = {best_eps:.4f}, Silhouette Score = {best_silhouette:.4f}")
    else:
        print("DBSCAN failed to produce meaningful clusters.")
    return best_labels
labels_dbscan = dbscan(reduced_data)
# Agglomerative Clustering
def optimize_agglomerative_clustering(data):
    print("\nOptimizing Agglomerative Clustering...")
    best_score = -1
    best_labels = None
    for clusters in range(2, 10):  # Test different numbers of clusters
        for linkage in ['ward', 'average', 'complete']:  # Test different linkage methods
            model = AgglomerativeClustering(n_clusters=clusters, linkage=linkage)
            labels = model.fit_predict(data)
            score = silhouette_score(data, labels)
            if score > best_score:
                best_score = score
                best_labels = labels
    print(f"Agglomerative Clustering: Best Silhouette = {best_score:.4f}")
    return best_labels
labels_agg = optimize_agglormerative_clustering(reduced_data)
```

Pet Images project sample code

Feature Extraction and Training

```
# Add a Fine-Tunable Linear Layer
class FineTuneDNN(torch.nn.Module):
    # Define a simple feedforward neural network
    def __init__(self, input_dim, num_classes):
        super(FineTuneDNN, self).__init__()
        self.fc = torch.nn.Linear(input_dim, num_classes)

    def forward(self, x):
        return self.fc(x)
# Function to extract features and fine-tune a linear layer
def extract_and_finetune(model, dataloader, num_classes):
    # Extract features from the second-to-last layer
    model.eval()
    features, labels = [], []
    with torch.no_grad():
        for inputs, targets in dataloader:
```

```
            outputs = model(inputs).flatten(start_dim=1)
            features.append(outputs)
            labels.append(targets)
            # Early stopping for demonstration purposes
    features = torch.cat(features).numpy()
    labels = torch.cat(labels).numpy()
    # Train Linear Layer on extracted features
    print("Training Linear Layer...")
    linear_layer = FineTuneDNN(features.shape[1], num_classes)
    # Use CrossEntropyLoss and Adam optimizer
    optimizer = torch.optim.Adam(linear_layer.parameters(), lr=0.001)
    criterion = torch.nn.CrossEntropyLoss()
    # Train for 10 epochs - update as needed (50 would be ideal)
    epochs = 2
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)
    # Convert to PyTorch tensors
    for epoch in range(epochs):
        linear_layer.train()
        # Convert to PyTorch tensors
        inputs = torch.tensor(X_train, dtype=torch.float32)
        targets = torch.tensor(y_train, dtype=torch.long)
        # Zero the gradients
        optimizer.zero_grad()
        outputs = linear_layer(inputs)
        # Calculate loss
        loss = criterion(outputs, targets)
        loss.backward()
        # Update weights
        optimizer.step()
        # Calculate accuracy
        # Get the class with the highest probability
        _, predicted = torch.max(outputs, 1)
        correct = (predicted == targets).sum().item()
        accuracy = 100 * correct / targets.size(0)
        print(f"Epoch {epoch + 1}/{epochs}, Loss: {loss.item():.4f}, Accuracy: {accuracy:.2f}%")
    return features, labels
# Custom Standard Scaler
def standard_scaler(features):
    # Compute column-wise mean and standard deviation
    mean = np.mean(features, axis=0)
    std = np.std(features, axis=0)
    # Avoid division by zero for columns with zero variance
    std[std == 0] = 1.0
    return (features - mean) / std
# Step 3: Dimensionality Reduction
def reduce_dimensions(features, n_components=NUM_COMPONENTS):
    reducer = umap.UMAP(n_components=n_components, random_state=42)
```

```
return reducer.fit_transform(features)
```