



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Kalman-Enhanced Streaming Machine Learning for Real-Time Land Use Classification in Satellite Imagery

TESI DI LAUREA MAGISTRALE IN
MATHEMATICAL ENGINEERING - INGEGNERIA MATEMATICA

Author: **Nicola Francescon**

Student ID: 220697

Advisor: Prof. Emanuele Della Valle

Co-advisors: Giacomo Ziffer

Academic Year: 2023-24

Abstract

The rapid escalation of data generation in recent years has oriented Machine Learning toward applications designed to manage continuous, unbounded data streams, in a field called Streaming Machine Learning. This thesis investigates the application of Streaming Machine Learning in the classification of satellite imagery, a domain where timely analysis of incoming data is essential for effective decision-making in areas such as environmental monitoring, disaster response, and urban planning. Unlike traditional batch processing, Streaming Machine Learning enables real-time decision-making and adaptation to changes in data distribution, ensuring models remain relevant as new data arrives. To address the computational challenges associated with high-dimensional satellite data, this research proposes an optimized pipeline incorporating Streaming Linear Discriminant Analysis with Kalman Filtering, namely Kalman-SLDA, for robust classification performance. Advanced dimensionality reduction techniques, including Sparse Random Projection and a novel adaptation of the UMAP algorithm for streaming environments, are employed to mitigate the computational load without compromising classification accuracy. Experimental results demonstrate that Kalman-SLDA outperforms traditional classifiers in terms of accuracy and efficiency, even in the presence of class imbalance and concept drift. Furthermore, the proposed pipeline leverages dimensionality reduction to improve computational speed, presenting a viable solution for the real-time processing demands of satellite data. The findings underscore the importance of developing efficient algorithms for streaming data and contribute to support rapid, data-driven decision-making in dynamic environments. This work provides a foundation for future research into adaptive, scalable classification pipelines for satellite imagery, with the potential to improve a wide range of applications in geospatial analysis.

Keywords: Streaming Machine Learning, Dimensionality reduction, Satellite Image Classification, Real-Time Processing, Kalman filtering

Abstract in lingua italiana

Il rapido aumento della generazione di dati negli ultimi anni ha condotto il Machine Learning verso applicazioni sempre più orientate a gestire flussi di dati continui e illimitati, un campo denominato Streaming Machine Learning. Questa tesi esplora l'applicazione dello Streaming Machine Learning nella classificazione delle immagini satellitari, un ambito in cui l'analisi tempestiva dei dati è essenziale per guidare le decisioni in contesti quali il monitoraggio ambientale, la risposta alle catastrofi e la pianificazione urbana. A differenza dei tradizionali approcci batch, lo Streaming Machine Learning consente di prendere decisioni in tempo reale e di adattarsi ad eventuali cambiamenti nella distribuzione dei dati, garantendo che i modelli rimangano accurati con l'afflusso di nuove informazioni. Per affrontare i problemi computazionali associati ai dati satellitari ad alta dimensionalità, questa ricerca propone una pipeline ottimizzata che implementa il classificatore SLDA integrato con il filtro di Kalman, chiamato Kalman-SLDA, mirando a garantire una classificazione robusta delle immagini. Tecniche avanzate di riduzione della dimensionalità, tra cui Sparse Random Projection e un innovativo adattamento dell'algoritmo UMAP per contesti streaming, sono impiegate per ridurre il carico computazionale mantenendo un'elevata precisione nella classificazione. I risultati sperimentali evidenziano che Kalman-SLDA supera i classificatori tradizionali sia in termini di accuratezza che di efficienza, anche in presenza di classi sbilanciate e variazioni nella distribuzione dei dati. Inoltre, la pipeline proposta sfrutta la riduzione della dimensionalità per migliorare la velocità di elaborazione, presentando una soluzione valida per le esigenze di analisi in tempo reale dei dati satellitari. I risultati sottolineano l'importanza di sviluppare algoritmi efficienti per flussi di dati continui, contribuendo a supportare decisioni rapide ed informate in ambienti dinamici. Questo lavoro fornisce una base per future ricerche su pipeline di classificazione adattive per immagini satellitari, con il potenziale di migliorare una vasta gamma di applicazioni nell'analisi geospaziale.

Parole chiave: Streaming Machine Learning, Riduzione della dimensionalità, Classificazione di immagini satellitari, Elaborazione in tempo reale, Filtro di Kalman

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
1 Introduction	1
1.1 Contributions of the Thesis	3
1.2 Thesis Outline	3
2 Related Works	5
2.1 Introduction to Streaming Machine Learning	5
2.1.1 Concept Drift	7
2.2 Streaming Machine Learning Classifiers	9
2.2.1 Gaussian Naive Bayes	9
2.2.2 Softmax Regression	10
2.2.3 Streaming Linear Discriminant Analysis	11
2.3 Convolutional Neural Networks	13
2.3.1 MobileNet V3 Small	14
2.3.2 ResNet 18	15
2.4 Satellite Image Datasets	16
2.4.1 Satellite Image Classification	16
2.4.2 The Dataset: Functional Map of the World - Time	17
2.5 Dimensionality Reduction Techniques	18
2.5.1 Sparse Random Projection	19
2.5.2 Uniform Manifold Approximation and Projection	20
2.6 Evaluation	22
2.6.1 Evaluation Protocol	22
2.6.2 Evaluation Metrics	23

2.6.3	Metric Interpretation	25
3	Problem Statement	27
3.1	Research Questions	27
3.2	Research Challenges	27
4	Problem Solving	31
4.1	Proposed Approach	31
4.2	Analysis of Functional Map of the World - Time	33
4.3	Kalman-Streaming Linear Discriminant Analysis	36
4.3.1	SLDA Limitations	36
4.3.2	Kalman-SLDA	37
4.4	Streaming-UMAP and Batch-UMAP	41
4.4.1	UMAP: Offline technique	41
4.4.2	Streaming-UMAP	41
4.4.3	Batch-UMAP	42
5	Experiments and Results	45
5.1	Experimental Setup	45
5.1.1	Convolutional Neural Networks	45
5.1.2	Classification Models	46
5.1.3	Dimensionality Reduction Techniques	46
5.2	Analysis of Results	47
5.2.1	Comparison of SML Classifiers	47
5.2.2	Comparison of CNNs	53
5.2.3	Comparison of Dimensionality Reduction Methods	55
5.2.4	Detailed Analysis of the Complete Pipeline	65
5.2.5	Best Model Selection	75
6	Conclusions and Future Developments	79
6.1	Conclusions	79
6.2	Future Work	80
	Bibliography	83
	A Appendix A - Other results	87

A.1 Kalman-SLDA	87
A.2 SLDA	88
A.3 Gaussian Naive Bayes	89
A.4 Softmax Regression	90
List of Figures	93
List of Tables	95
Acknowledgements	97

1 | Introduction

In recent years, the exponential growth in data generation has driven a significant transformation in the field of Machine Learning, giving rise to what is now known as **Streaming Machine Learning (SML)** [4, 14]. This area studies innovative approaches that are specifically designed to handle potentially unbounded data streams, enabling continuous and adaptive data processing as new information becomes available. In contrast to traditional Machine Learning paradigms, which rely on static datasets processed in batch mode, Streaming Machine Learning excels in scenarios where data arrives in real time and cannot be feasibly stored or processed in large batches. This shift represents a critical advancement in leveraging data for real-time insights and responsive decision-making in the increasingly data-centric world of today.

The main advantage of SML algorithms is their capability to operate on dynamic data streams, enabling systems to make timely decisions that are essential in applications requiring immediate response. SML approaches are also inherently adaptive to **concept drift** [30], changes in the underlying data distribution over time, which is a common phenomenon in evolving data environments. By learning from new data and discarding outdated patterns, Streaming Machine Learning ensures that models remain relevant and accurate over time, effectively handling the evolving nature of real-world data. The SML significance spans across a wide range of fields, including finance, healthcare, transportation, and environmental monitoring. Its ability to analyze streaming data in real time has the potential to yield better insights, improve outcomes, and enhance operational efficiency. Furthermore, this paradigm shift challenges researchers to develop algorithms that are not only efficient and scalable but also resilient to the complexities of real-world data, including variability, noise, and shifts in data distribution.

In the context of satellite imagery, SML offers a powerful solution for managing the vast data volumes generated by **remote sensing technologies**. Traditional methods often struggle to keep pace with the continuous influx of satellite images, which can result in delayed analysis and missed opportunities for timely intervention. Streaming Machine Learning enables real-time processing of this incoming imagery, supporting immediate

classification and monitoring of diverse environmental and urban phenomena. This capability is particularly valuable in disaster response, where rapid assessment of satellite images can lead to life-saving interventions and resource allocation.

The advent of remote sensing has revolutionized planetary data collection and analysis, providing an innovative view of environmental changes, urban growth, agricultural dynamics, and natural disasters. Satellite imagery, therefore, stands as a vital resource for researchers and industries, offering insights that drive strategic decisions. Efficient and accurate classification of satellite images has thus become a critical field of study, with applications ranging from environmental monitoring to urban planning and precision agriculture. As data streams from satellites increase in both volume and frequency, integrating real-time processing capabilities becomes essential for achieving timely insights and enabling data-driven action.

Within this context, image classification plays a fundamental role, involving the categorization of images into predefined classes. This classification task is crucial for numerous applications, such as land-use mapping, resource management, and climate analysis, where accurate information is essential for informed decision-making. However, the complexity of satellite imagery, characterized by high dimensionality, diverse spectral bands, and varying spatial resolutions, poses considerable challenges to traditional classification techniques, necessitating more sophisticated methods.

This work investigates the development of an optimized pipeline for classifying satellite image streams, focusing on both classification accuracy and computational performance. The core methodologies explored include the adaptation of classification algorithms, such as **Streaming Linear Discriminant Analysis (SLDA)** [16], and the integration of advanced dimensionality reduction techniques, including **Sparse Random Projection** [5] and a novel adaptation of **UMAP** (Uniform Manifold Approximation and Projection)[26] designed for streaming contexts. By leveraging these tools, this research aims to achieve high classification performance while addressing the computational demands of processing high-dimensional satellite data.

The practical implications of this study provide solutions that can be applied to real-world scenarios where timely and reliable geospatial information is crucial. As the demand for rapid and accurate satellite-based data continues to grow, the methodologies developed in this thesis contribute to advancing the field of satellite image processing, enabling more informed, data-driven decisions across a wide range of sectors.

1.1. Contributions of the Thesis

This thesis addresses the classification of temporally correlated satellite image streams, which evolve over time and exhibit complex patterns due to changing environmental conditions and land use dynamics. The primary contributions provided by this work are as follows:

- **Kalman filter adaptation for Streaming Linear Discriminant Analysis (SLDA):** This thesis integrates the Kalman filter into the SLDA framework, enabling the model to handle concept drift by dynamically adjusting to evolving patterns in the data stream, which may include seasonal changes, landscape shifts, or human-driven land use alterations.
- **Development of a streaming version of Uniform Manifold Approximation and Projection (UMAP):** An adaptation of UMAP for streaming data is presented, designed to reduce the dimensionality of large-scale, evolving datasets like satellite imagery. By integrating data similarity, this method captures temporal correlations within the satellite stream, with the aim to improve downstream classification and computational efficiency.
- **Comprehensive performance comparison of dimensionality reduction techniques in streaming settings:** A detailed evaluation of various dimensionality reduction techniques is conducted, assessing their real-time applicability to satellite images. This analysis identifies strategies that maintain classification accuracy while optimizing computational speed for streaming contexts.
- **Experimental validation of proposed methods against state-of-the-art models:** The effectiveness of the proposed methods is demonstrated through extensive validation, comparing them to state-of-the-art algorithms for processing temporally correlated, evolving satellite image data.

1.2. Thesis Outline

This study is organized into six chapters.

- **Chapter 1: Introduction** provides an introduction to the problem and summarizes the major contributions of this thesis.
- **Chapter 2: Related Works** reviews relevant literature and existing research related to the topic of this study.

- **Chapter 3: Problem Statement** specifies the research questions guiding this work and highlights the main challenges encountered.
- **Chapter 4: Problem Solving** details the novel methods developed in this thesis, including the proposed approach to address the problem.
- **Chapter 5: Experiments and Results** discusses the results obtained, comparing the outcomes of different experiments.
- **Chapter 6: Conclusions and Future Developments** presents an overall summary of the work and suggests potential extensions to further enhance the study.

2 | Related Works

This chapter reviews and discusses existing works and materials related to the thesis topic. Section 2.1 introduces the field of Streaming Machine Learning, outlining its main challenges, which provide the basis for the methods explored in the subsequent sections. Section 2.2 reviews existing classification algorithms in Streaming Machine Learning scenarios, highlighting their strengths and weaknesses. Section 2.3 provides an overview of Convolutional Neural Networks (CNNs) and their relevance in image processing, particularly in streaming contexts. Section 2.4 provides an overview of common satellite image datasets and the most relevant techniques for processing such data. It also details the dataset utilized in this research, which is framed within a temporally correlated stream of satellite images. Section 2.5 addresses dimensionality reduction techniques and their applicability to streaming learning environments. Finally, Section 2.6 outlines the metrics employed to evaluate model performance and compare the different approaches in a streaming environment.

2.1. Introduction to Streaming Machine Learning

Streaming Machine Learning (SML) [4, 14] is a branch of Machine Learning focused on learning from continuous, potentially unbounded data streams. In such settings, data points are observed sequentially, used to update model parameters, and then discarded. Unlike traditional batch learning, where the entire dataset is available at once, SML incrementally refines its parameters with each new observation. This approach is particularly effective when storing or processing large datasets at once is impractical, allowing for real-time model updates without requiring significant memory resources. Incremental learning enables models to adapt over time without the computational cost of retraining on all historical data, making it essential for scenarios that demand low latency and efficient resource use. Figure 2.1 contrasts traditional offline learning with online learning. In *Offline Learning*, a finite batch of samples is processed at once to train the model, which is then tested on a separate dataset of unseen samples. The model parameters remain unchanged after training. In *Online Learning*, the model continuously processes

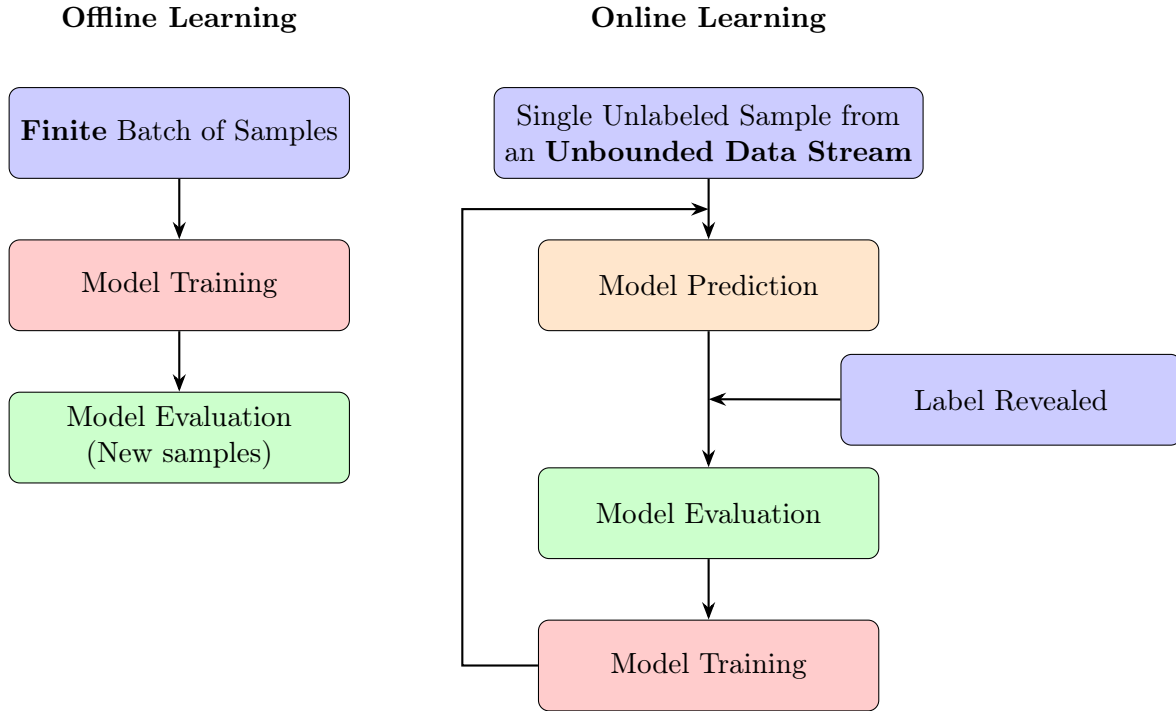


Figure 2.1: Comparison of Online and Offline Learning.

an unbounded stream of data, updating incrementally with each new observation. This approach allows the model to adapt to new patterns as they emerge in real time.

In the context of satellite image classification, SML can process vast quantities of image data collected from orbiting satellites in real time. This continuous flow of images enables rapid analysis of environmental conditions, urban expansion, or agricultural patterns. The unbounded nature of satellite data means that models need to adapt to shifting distributions—whether due to seasonal changes, weather patterns, or human activities—without revisiting previously seen data. For example, as new satellite images are captured over time, SML models can adapt to detect deforestation, monitor crop health, or track natural disasters as they evolve, without reprocessing earlier observations.

Currently, SML algorithms have demonstrated efficient learning with low-dimensional data, such as tabular datasets with a limited number of features. Processing very high-dimensional data, like raw images with hundreds of thousands of features, presents additional challenges. In these cases, a preliminary feature extraction step is necessary to reduce dimensionality [31]. Convolutional Neural Networks (CNNs), discussed in Section 2.3, are commonly used for this purpose, extracting meaningful features prior to classification and enabling SML to handle more complex data structures.

However, SML faces some critical challenges. First of all, an efficient processing is re-

quired. SML models must update parameters quickly as new data arrives at a high rate. This requires algorithms that are computationally lightweight and capable of updating in constant or near-constant time. Furthermore, as data volume grows, models must scale efficiently without proportional increases in computational or memory demands.

A major challenge in SML is **concept drift** [30], where the underlying data distribution changes over time. Traditional Machine Learning assumes a static data distribution, but streaming data often violate this assumption: data are not identically distributed over time, making previously learned patterns obsolete. Concept drift forces SML algorithms to continuously adapt without sacrificing model performance on previously learned concepts. Failure to account for concept drift can lead to significant degradation in prediction accuracy.

2.1.1. Concept Drift

Concept drift poses a significant challenge in Streaming Machine Learning because it violates the assumption of a static data distribution foundational to many traditional Machine Learning techniques. In real-world applications like satellite image classification, the relationship between input features and target variables may evolve over time due to seasonal effects, urbanization, or other environmental changes.

Concept drift can manifest in various ways, displayed in Figure 2.2 and detailed as follows:

- **Abrupt Drift:** A sudden, significant change in the data distribution often triggered by a major external event. For instance, in satellite imagery, a natural disaster such as a wildfire or flood may drastically alter the landscape, causing an immediate distribution shift.
- **Gradual Drift:** The data distribution changes slowly over time. Examples include urban area expansion or vegetation pattern shifts, where changes occur progressively, requiring the classifier to adapt over extended periods.
- **Reoccurring Drift:** Patterns in the data repeat over time. In satellite imagery, seasonal variations are a common cause of reoccurring drift, particularly for agricultural areas where crop growth cycles or snow cover influence the classification output at specific times of the year.
- **Incremental Drift:** Continuous, small changes in the data distribution accumulate gradually, as in the slow conversion of natural landscapes into farmland, which might not be immediately noticeable but eventually leads to significant changes in the land-use classification.

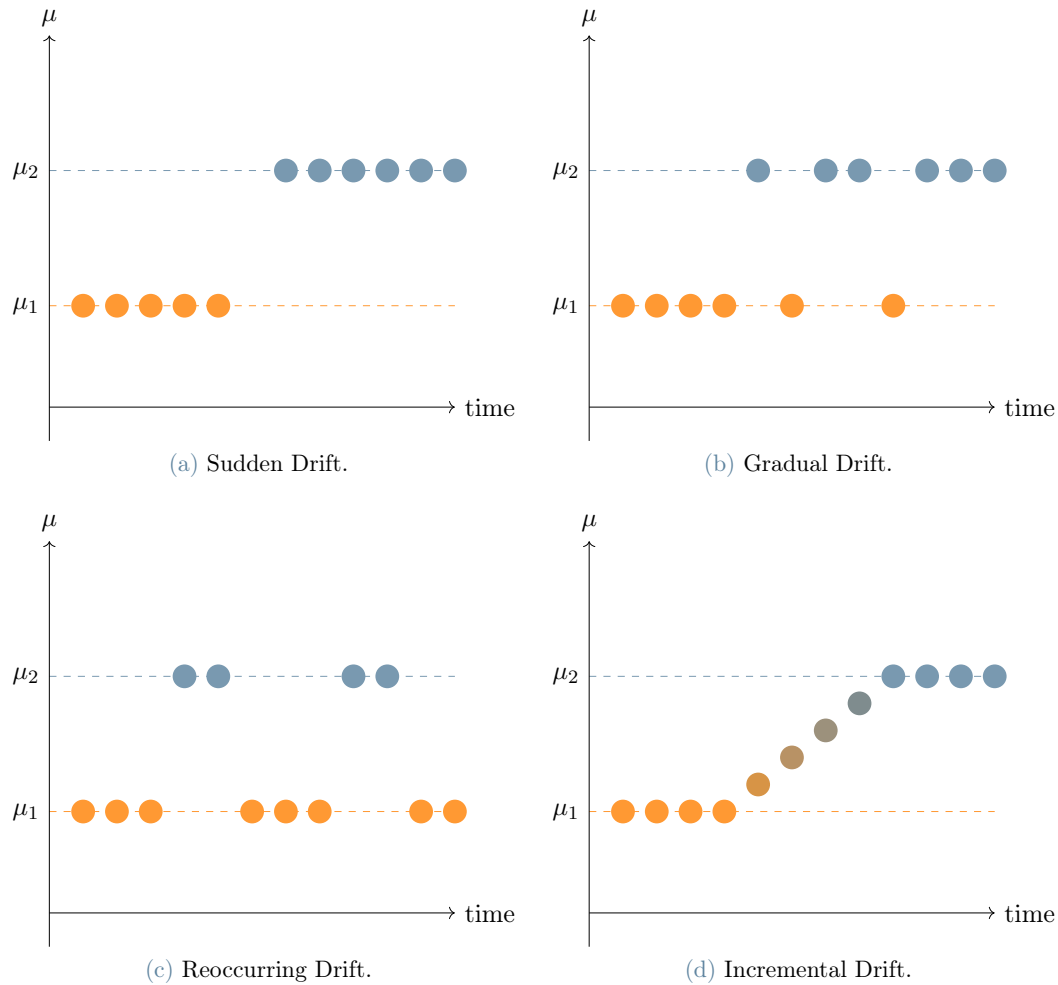


Figure 2.2: Different types of Concept Drift. The plots display the evolution of the mean μ for one-dimensional data.

To address concept drift, SML models incorporate adaptive mechanisms to adjust to shifts in the data distribution [13, 24], including:

- **Sliding Window Techniques:** These approaches consider only the most recent data, discarding outdated observations to help the model quickly adapt to current distributions.
- **Drift Detection Mechanisms:** These methods monitor model performance to detect concept drift. Upon detecting a drift, the model is either updated or retrained with recent data.

In summary, concept drift introduces significant challenges to accomplish any streaming learning task; however, various techniques exist to ensure SML models remain accurate and relevant over time.

2.2. Streaming Machine Learning Classifiers

In the context of learning from data streams, several families of algorithms have been proposed to handle the challenges associated with dynamic and evolving datasets, including instance-based classifiers, probability-based classifiers and decision tree-based classifiers. This thesis focuses on **probability-based classifiers**, a category known for their ability to produce probabilistic outputs that represent the likelihood of each class. These classifiers estimate the probability distribution over possible classes, allowing for more refined decision-making by incorporating uncertainty into predictions. The strength of probability-based classifiers lies in their ability to adapt to changing data distributions, making them suitable for real-time applications like satellite image classification. By leveraging probabilistic models, these classifiers can handle noisy, incomplete, or imbalanced data while offering interpretability through the probability scores they generate.

2.2.1. Gaussian Naive Bayes

One of the most used probability-based SML classifiers is Gaussian Naive Bayes [21], a simple yet effective model grounded in Bayes' theorem. Bayes' theorem describes the relationship between the conditional and marginal probabilities of random events, expressed as:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

where $P(C|X)$ is the posterior probability of class C given feature vector X , $P(X|C)$ is the likelihood of observing X given class C , $P(C)$ is the prior probability of class C , and $P(X)$ is the marginal likelihood of X .

Gaussian Naive Bayes assumes conditional independence between each pair of features given the class variable, simplifying the likelihood computation as follows:

$$P(X|C) = \prod_{i=1}^n P(X_i|C),$$

where X_i represents each feature in the feature vector X .

This model is suitable for tasks where the features are assumed to be normally distributed, characterizing each feature by its mean and variance for each class. The streaming adaptation of Gaussian Naive Bayes is straightforward: it maintains a count of observed features

and their corresponding statistics (mean and variance) for each class. As new data arrives, these statistics update to allow adaptation to concept drift, making Gaussian Naive Bayes efficient in streaming contexts with low computational requirements and rapid parameter updates.

2.2.2. Softmax Regression

For multi-class classification tasks, Softmax Regression [19] is a widely utilized probability-based classifier that extends Logistic Regression [23] to multiple classes, predicting probabilities across more than two classes.

The Softmax function is defined mathematically as follows:

$$\sigma(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}, \quad \text{for } k = 1, \dots, K$$

where $\mathbf{z} = [z_1, z_2, \dots, z_K]$ represents the class scores, K is the total number of classes, and $\sigma(\mathbf{z})_k$ is the predicted probability of class k . Softmax converts raw class scores (logits) into probabilities that sum to one, facilitating interpretation of the outputs as probabilities.

Training a Softmax model is generally performed through cross-entropy loss [25], quantifying the difference between predicted probabilities and true class labels. Cross-entropy loss function is expressed as:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^K y_i \log(\hat{y}_i)$$

where \mathbf{y} is the one-hot encoded true class vector and $\hat{\mathbf{y}}$ is the predicted probability vector.

Softmax Regression assumes linearly separable classes and aims to find the optimal linear boundary maximizing observed data likelihood. This linearity simplifies the model, making it computationally efficient for applications requiring interpretability and speed.

In a SML context, Softmax Regression adapts by incrementally updating parameters as data arrives, using online learning techniques. This approach allows the model to adjust weights for each new observation without retraining on the full dataset, an advantage in dynamic environments where data distributions may change over time.

2.2.3. Streaming Linear Discriminant Analysis

Streaming Linear Discriminant Analysis (SLDA) [16] builds on Linear Discriminant Analysis (LDA) [33], which assumes each class follows a multivariate Gaussian distribution. SLDA offers a novel approach outperforming many leading continual learning methods available at the time. Its key feature is an incremental learning process, where the model updates its parameters one sample at a time based on the true label of each new data point.

SLDA maintains a mean vector for each class, adjusting it as new samples arrive. Simultaneously, it updates a shared global covariance matrix across all classes. Before starting the incremental updates, the model requires a prior batch of samples $\{(x_1, y_1), \dots, (x_n, y_n)\}$ to establish initial parameters:

$$c_{k,0} = \sum_{i=1}^n \mathbb{1}_{\{y_i=k\}} \quad \forall k \text{ classes}$$

$$\boldsymbol{\mu}_{k,0} = \frac{\sum_{i=1}^n x_i \cdot \mathbb{1}_{\{y_i=k\}}}{c_{k,0}} \quad \forall k \text{ classes}$$

Here, $c_{k,0}$ is the count of observations for class k , and $\boldsymbol{\mu}_{k,0}$ represents the initial mean. The initial covariance matrix $\boldsymbol{\Sigma}_0$ is estimated using the Oracle Approximation Shrinkage (OAS) method [7].

As new data \mathbf{z}_t arrives at time t , SLDA computes a score for each class. The class with the highest score is selected as the predicted class C :

$$\mathbf{w}_k = \boldsymbol{\Sigma}_t^{-1} \boldsymbol{\mu}_{k,t}$$

$$s_k = \mathbf{w}_k^T \mathbf{z}_t - \frac{1}{2} (\boldsymbol{\mu}_{k,t}^T \boldsymbol{\Sigma}_t^{-1} \boldsymbol{\mu}_{k,t})$$

$$C = \underset{k}{\operatorname{argmax}} \{s_k\}$$

When the true label k_t for the point \mathbf{z}_t is revealed, the model updates its parameters as follows:

- **Class Mean Update:**

$$\begin{cases} \boldsymbol{\mu}_{k_t,t+1} = \frac{c_{k_t,t} \cdot \boldsymbol{\mu}_{k_t,t} + \mathbf{z}_t}{c_{k_t,t} + 1} \\ \boldsymbol{\mu}_{j,t+1} = \boldsymbol{\mu}_{j,t} & \forall j \neq k_t \\ c_{k_t,t+1} = c_{k_t,t} + 1 \\ c_{j,t+1} = c_{j,t} & \forall j \neq k_t \end{cases} \quad (2.1)$$

- **Covariance Matrix Update:**

$$\begin{cases} \boldsymbol{\Sigma}_{t+1} = \frac{t \cdot \boldsymbol{\Sigma}_t + \boldsymbol{\Delta}_t}{t + 1} \\ \boldsymbol{\Delta}_t = \frac{t \cdot (\mathbf{z}_t - \boldsymbol{\mu}_{k_t,t})(\mathbf{z}_t - \boldsymbol{\mu}_{k_t,t})^T}{t + 1} \end{cases} \quad (2.2)$$

SLDA is computationally demanding, especially in high-dimensional spaces, as it requires inverting a $d \times d$ matrix. Notably, SLDA converges to the result of offline LDA when presented with the entire data stream in batch form. However, the method does not account for concept drift, as early data points are weighted equally with later ones. Consequently, SLDA may struggle in dynamic environments where the underlying data distribution shifts over time. Another important aspect of Streaming Linear Discriminant Analysis is its implicit role in dimensionality reduction, particularly when dealing with high-dimensional feature vectors. Like its offline counterpart, Linear Discriminant Analysis, SLDA leverages class-discriminative information by maintaining class-specific mean vectors and a global covariance matrix.

SLDA can be seen as performing dimensionality reduction incrementally, as it continuously updates the model’s parameters with each new sample from the data stream. Specifically, SLDA retains class-separating information in the mean vectors and the shared covariance matrix, effectively reducing the impact of high-dimensionality on the classification process. Similar to LDA [29], SLDA seeks to project the input data into a lower-dimensional space where the between-class variance is maximized relative to the within-class variance. In the case of SLDA, however, this process happens incrementally as the data arrives, without needing to store the entire dataset. The projection is implicitly driven by the updates to the covariance matrix and the mean vectors, which capture the essential characteristics of each class as more samples are observed.

In high-dimensional settings, SLDA can help mitigate the *curse of dimensionality* [22] by reducing the effective number of features used in the classification process. While the

original input feature space might be very large, the model incrementally refines its understanding of the class boundaries through updates to a lower-dimensional representation that focuses on the most discriminative features. This leads to a computationally efficient and more robust classifier that performs well even with fewer observations relative to the number of features. Moreover, although SLDA processes data in a streaming fashion, it retains much of the information necessary for distinguishing between classes. In fact, due to its incremental learning nature, SLDA can perform effectively in high-dimensional spaces, often achieving better classification accuracy by avoiding overfitting and reducing noise in the data.

2.3. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are widely recognized as the state-of-the-art architecture for image classification [3, 11, 35], due to their ability to learn increasingly complex hierarchical features from visual data. CNNs start by detecting basic patterns, such as edges and curves, and progressively capture more complex structures, such as shapes, textures, and finally, high-level semantic features. Image classification is a supervised Computer Vision task that involves assigning a specific class label to each image based on its visual content. By recognizing patterns within the data, CNNs are able to categorize images into predefined classes, making them indispensable for a wide range of applications.

Despite the straightforward goal of image classification, two main challenges arise:

- **Variability:** Images can differ significantly from one another in terms of viewpoint, lighting, occlusions, backgrounds, and scale. Even the same subject may appear very different depending on these factors, complicating the development of a classifier that generalizes well across diverse conditions. For instance, in the context of satellite images, a building could be partially obscured by clouds or appear under various lighting conditions and weather patterns, affecting its appearance in each image.
- **Class Imbalance:** Real-world datasets often exhibit class imbalance, with certain classes having significantly fewer samples than others. This imbalance can lead to a bias in the classifier, making it favor more frequent classes and struggle to accurately recognize underrepresented ones. This disparity can negatively impact the predictive performance of the model, requiring strategies to ensure balanced learning across all classes.

Image classification finds application in numerous domains, including surveillance, security, industrial automation, and automotive systems. Reliable image classification is particularly crucial for tasks like land-use detection, facial recognition, and scene understanding. For satellite image analysis, CNNs play an essential role in transforming raw visual data into a format suitable for further processing. By removing the top layers, the networks act solely as feature extractors, producing compact representations of images that retain vital structural information. This approach allows the model to capture high-level image attributes while enabling more efficient handling of downstream classification tasks. To capture relevant features, this work uses two neural network architectures, both pre-trained on the ImageNet [10] dataset, a benchmark in the field of Computer Vision containing millions of labeled images across diverse categories.

2.3.1. MobileNet V3 Small

MobileNet V3 Small [20] is a state-of-the-art CNN architecture designed primarily for mobile and edge devices, balancing computational efficiency and accuracy. Its architecture, shown in Figure 2.3, makes it suitable for real-time applications where computational resources are limited.

Key innovations that enhance MobileNet V3 Small's performance include:

- **Squeeze-and-Excitation (SE) Blocks:** These blocks adaptively recalibrate channel-wise feature responses, allowing the network to emphasize important features while suppressing less informative ones. This mechanism enhances the model's representational power without significantly increasing computational costs.
- **Lightweight Attention Mechanisms:** MobileNet V3 Small incorporates lightweight attention modules to improve feature representation while maintaining efficiency. These modules enable the network to focus on essential parts of the input image, improving performance on complex tasks.
- **Efficient Architecture Search:** The architecture was optimized through a neural architecture search technique to identify most effective layers and configurations for mobile environments, ensuring high accuracy while remaining computationally efficient.

The feature embedding size from MobileNet V3 Small is 576 dimensions when used without the top classification layers. This compact representation captures essential features of the input image, facilitating downstream tasks such as classification.

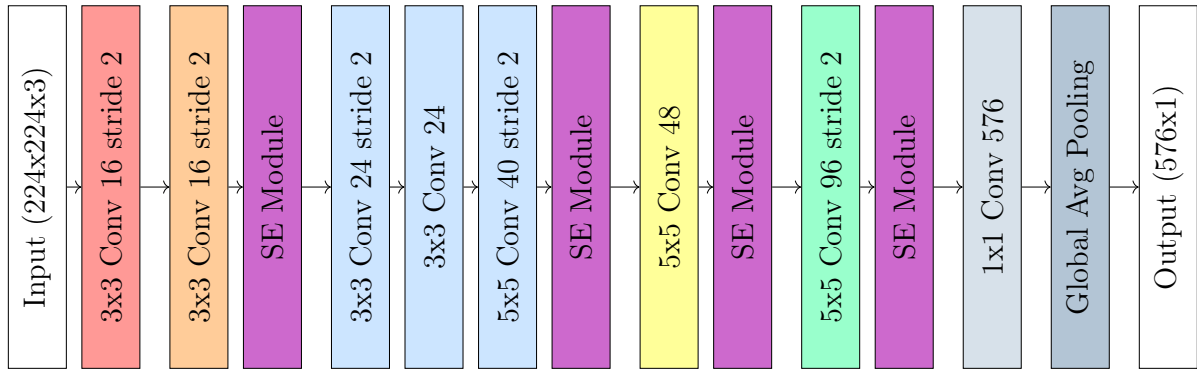


Figure 2.3: MobileNet V3 Small Architecture

2.3.2. ResNet 18

ResNet 18 [17] is a deeper CNN architecture that introduces residual connections to improve gradient flow during training. This innovation has established ResNet 18 as a robust benchmark for various image classification tasks, showing excellent real-world performance due to its ability to capture complex patterns in data.

The architecture's distinctive feature is the use of residual connections: the input to a layer is directly added to its output, forming a shortcut connection. This design enables the network to learn residual mappings rather than optimizing the original unreferenced mappings, which mitigates the vanishing gradient problem [6] often encountered in deep networks. As a result, ResNet 18 can be effectively trained even at greater depths without degradation in performance. The details about its layers are illustrated in Figure 2.4.

When used without its classification layers, ResNet 18 generates a 512-dimensional feature embedding. Like MobileNet V3 Small, this representation efficiently captures the critical features of the input image, supporting further analysis.

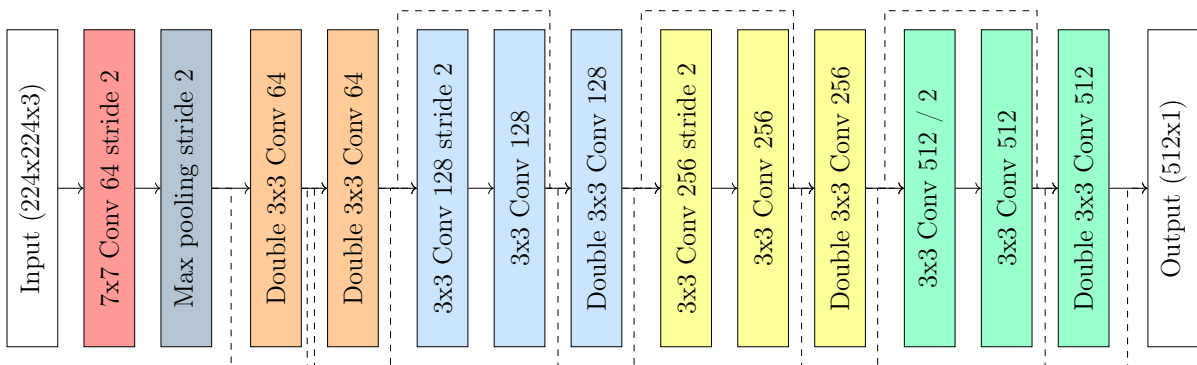


Figure 2.4: ResNet 18 Architecture. Dashed lines represent residual connections.

2.4. Satellite Image Datasets

Satellite image classification has become an essential task across various domains, such as environmental monitoring, urban planning, and agricultural assessment. Satellite imagery provides high-resolution visual data covering extensive geographical areas, enabling the detection and classification of land-use and land-cover features. However, analyzing satellite images for real-world applications poses several challenges due to data variability and temporal changes. Images taken at different times or under different conditions may exhibit autocorrelation with slight variations in structure or appearance, but these changes can also reflect genuine evolution in the imaged concepts (e.g., urban expansion, seasonal changes in vegetation). To build robust classifiers, it is necessary to handle both static patterns and dynamic evolutions in data.

2.4.1. Satellite Image Classification

Satellite image classification [1] aims to assign labels to regions of interest based on visual content, typically indicating land-use or land-cover categories. The main tasks include segmentation (pixel-wise classification), object detection (identifying specific objects), and image-level classification (assigning an entire image to a category). While segmentation and object detection offer detailed insights, image-level classification is a simpler, efficient solution, often adopted when processing large volumes of satellite data.

Popular datasets for satellite image classification include:

- **Functional Map of the World (FMoW)**: Contains 1 million images with 63 land-use categories. The FMoW dataset, used in this thesis, has been updated to include distribution shifts over time in the *Wilds* benchmark, which contains 126,165 samples in a more focused classification task.
- **EuroSAT** [18]: Based on Sentinel-2 satellite images, EuroSAT consists of 27,000 labeled images across 10 categories, focusing on European land types such as forest, river, and urban areas.
- Datasets from **DeepGlobe Land Cover Classification Challenge** [9]: The challenge proposes different datasets designed for land-cover classification, building detection and road extraction, providing annotated images from regions around the globe.

Classification techniques tailored for satellite imagery address unique challenges:

- **Multi-spectral and multi-temporal data**: Using multi-band data (e.g., infrared,

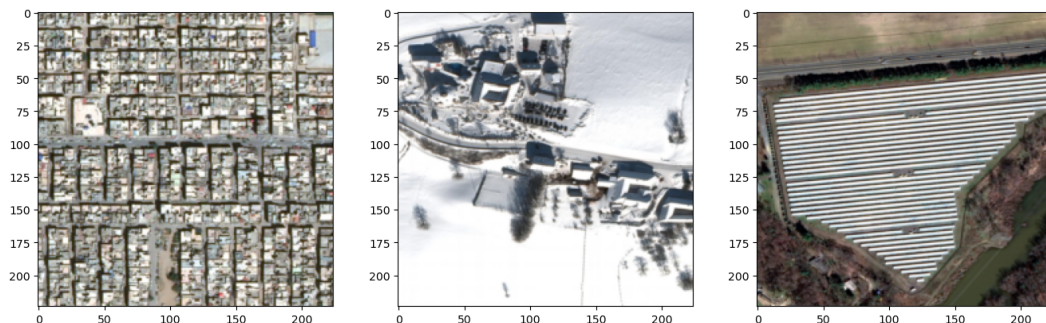
thermal) enhances classification accuracy, especially in distinguishing visually similar categories. Temporal information allows models to capture changes in land-use or land-cover over time.

- **Handling concept drift:** Temporal shifts in land-use require adaptive models capable of updating their representations. Approaches such as Streaming Machine Learning and adaptive models are employed to adjust to new data without retraining from scratch.

2.4.2. The Dataset: Functional Map of the World - Time

The dataset analyzed in this thesis originates from the *Functional Map of the World* (FMoW) [8], initially released in 2018. The original FMoW dataset includes approximately 1 million multispectral images across 63 target categories, covering different land-use types, such as residential, industrial, agricultural areas, and an additional class for false detections, where satellite imagery does not correspond to any designated land-use type. Data for this dataset are sourced from four Earth observation satellites, namely QuickBird 2, GeoEye 1, WorldView 2, and WorldView 3, capturing various spectral bands to enhance classification accuracy.

In 2021, FMoW was updated [34] and renamed as FMoW-Time. This revised dataset presents a simplified classification task with 126,165 images, eliminating the false detection category. The 2021 version, utilized for this thesis, focuses exclusively on three-band RGB images across 62 land-use categories. Each image sample, sized at 224×224 pixels, includes three RGB color channels and metadata specifying the land-use label and year of observation. However, these observations lack ordering within each year. A sample of



(a) An image representing **multi-unit residential** category. (b) An image representing **fire station** category. (c) An image representing **solar farm** category.

Figure 2.5: Samples of images taken from the FMoW-Time dataset.

images from the FMoW-Time dataset is shown in Figure 2.5.

The dataset includes various categories, such as airport, hospital, agricultural fields, and urban facilities, among others. The land use categories are detailed below:

airport, airport hangar, airport terminal, amusement park, aquaculture, archaeological site, barn, border checkpoint, burial site, car dealership, construction site, crop field, dam, debris/rubble, educational institution, electric substation, factory/powerplant, fire station, flooded road, fountain, gas station, golf course, ground transportation station, helipad, hospital, impoverished settlement, interchange, lake/pond, lighthouse, military facility, multi-unit residential, nuclear powerplant, office building, oil/gas facility, park, parking lot/garage, place of worship, police station, port, prison, race track, railway bridge, recreational facility, road bridge, runway, shipyard, shopping mall, single-unit residential, smokestack, solar farm, space facility, stadium, storage tank, surface mine, swimming pool, toll booth, tower, tunnel opening, waste disposal, water treatment facility, wind farm, zoo.

2.5. Dimensionality Reduction Techniques

An intermediate step to improve the computational efficiency of streaming classifiers is reducing the dimensionality of each feature vector using appropriate dimensionality reduction methods. The goal is to retain only the most relevant information from each sample while providing the classifiers with a reduced-dimensionality vector that can be processed more quickly. The underlying principle for achieving this goal is based on **data similarity** [28]: in a classification task, similar samples should ideally share the same label. Therefore, preserving the similarity among samples is crucial for retaining effective information for downstream tasks.

To achieve optimal performance, further reducing the number of features can also lead to performance improvements, provided the dimensionality reduction algorithm effectively condenses the information contained in the original d features. For clarity, let d denote the dimensionality of each sample after feature extraction through a CNN. Once dimensionality reduction is performed, the resulting features are fed to the classifier, which now processes feature vectors of size d' instead of d . The conclusions drawn for each classifier in Section 2.2 still apply when using these reduced feature vectors. In this thesis, two dimensionality reduction techniques are examined: Sparse Random Projection and Uniform Manifold Approximation and Projection (UMAP).

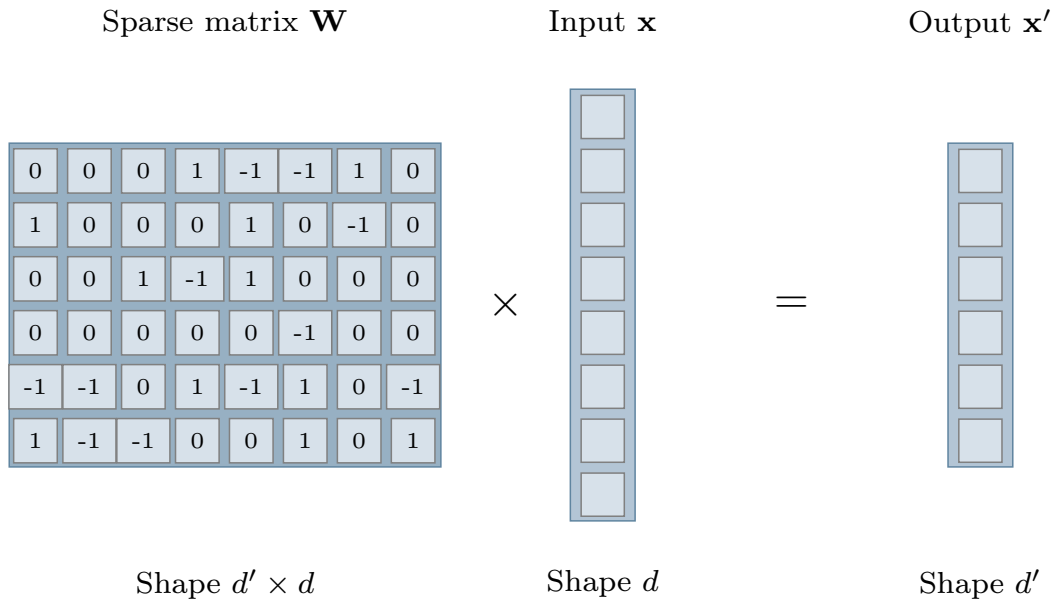


Figure 2.6: Sparse Random Projection Algorithm. Procedure is iterated for each data point.

2.5.1. Sparse Random Projection

Sparse Random Projection [5] is a computationally efficient method for dimensionality reduction. It projects high-dimensional data into a lower-dimensional space using a sparse random matrix of shape $d' \times d$. This matrix is filled with elements from $\{-1, 0, 1\}$, maintaining a density (ratio of non-zero elements in the matrix) of approximately 33% to achieve optimal performance, as it helps to preserve the integrity of the data. The reduced feature vector is obtained through a simple matrix-vector multiplication, resulting in a vector of shape d' . A brief overview of the method is provided in Figure 2.6.

The advantages of Sparse Random Projection include its low computational cost, as the use of sparse matrices significantly reduces memory usage and computational effort. This makes it particularly suitable for large datasets and streaming applications. Importantly, while this method is data agnostic and does not require assumptions about data distribution, it should preserve data similarity by projecting similar points closer together in the reduced space. However, it is essential to note that Sparse Random Projection may suffer from approximation errors and may not adequately capture complex structures or non-linear relationships in the data. Consequently, similar points in the original space are not guaranteed to maintain their proximity in the reduced dimensionality.

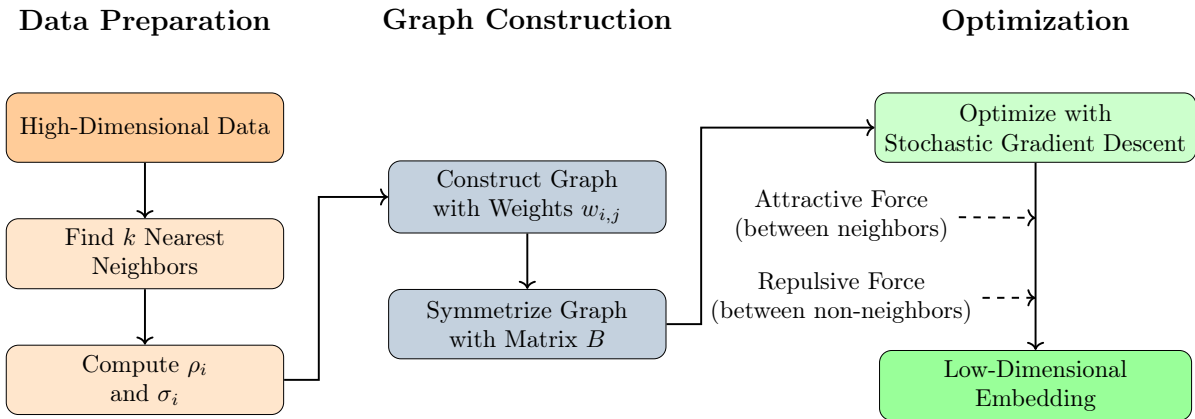


Figure 2.7: Uniform Manifold Approximation and Projection (UMAP) Algorithm.

2.5.2. Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection (UMAP) [26] is a non-linear dimensionality reduction technique designed to preserve both local and global structures in data, with a particular emphasis on maintaining data similarity. UMAP is particularly effective in providing clearer visualizations in lower-dimensional spaces, making it well-suited for exploratory data analysis. Due to its non-linear nature, UMAP is able to preserve complex relationships better than traditional linear methods. Furthermore, compared to other non-linear techniques like t-SNE, UMAP requires fewer computational resources and can handle arbitrarily high-dimensional data, such as images or text, reshaping their dimensionality to any target size.

UMAP is designed to preserve local similarities by ensuring that close points in the original high-dimensional space remain close in the reduced-dimensional space. This focus on data similarity is achieved through the construction of a weighted graph that represents the data structure, allowing UMAP to create an effective embedding that retains both local and global information.

Despite its advantages, UMAP currently has limitations in streaming contexts. It has primarily been developed for offline use and has not yet been adapted for real-time streaming environments. Additionally, UMAP can require substantial memory resources for complex datasets, which poses challenges for real-time applications. The algorithm is also sensitive to several parameters, such as the number of neighbors and the minimum distance of separation between similar points, which must be carefully tuned to achieve optimal performance.

The UMAP algorithm involves several steps, summarized in Figure 2.7. First, it identifies

the k nearest neighbors for each observation using a selected distance metric. Next, it computes ρ_i , the minimal positive distance from observation i to any of its neighbors:

$$\rho_i = \min\{d(x_i, x_{ij}) \mid 1 \leq j \leq k, d(x_i, x_{ij}) > 0\}$$

where $d(x_i, x_j)$ denotes the distance between points x_i and x_j .

Next, it calculates σ_i , a parameter that helps preserve the global structure in the embedding, by solving the equation:

$$\log_2(k) = \sum_{j=1}^k \exp\left(\frac{-\max(0, \text{dist}(x_i, x_j) - \rho_i)}{\sigma_i}\right)$$

The algorithm defines the weights of the graph based on the distances between points and the previously computed quantities:

$$w_{i,j} = \exp\left(\frac{-\max(0, \text{dist}(x_i, x_j) - \rho_i)}{\sigma_i}\right)$$

The weight $w_{i,j}$ represents the relative strength between the connected points x_i and x_j . After further normalization, it can be interpreted as the probability that x_i is connected to x_j , while $w_{j,i}$ represents the probability that x_j is connected to x_i , which may differ in principle. Considering the asymmetric nature of the weight matrix W , a symmetric matrix B can be constructed using the Hadamard operator (\circ):

$$B = W + W^T - W \circ W^T$$

Here, B represents the weights for an undirected graph, and the element $b_{i,j}$ represents the probability that x_i and x_j are connected. Given the structure represented by B , the low-dimensional representation Y of the initial dataset X is computed by building a weight matrix G . The optimization is performed through stochastic gradient descent, minimizing the cross-entropy loss between B and G . To maintain the local structure in the low-dimensional embedding, UMAP introduces two forces:

- **Attractive force** between x_i , with coordinates y_i in the reduced space, and one of its neighbors x_j , with coordinates y_j in the reduced embedding:

$$\frac{-2ab\|y_i - y_j\|_2^{2(b-1)}}{1 + a\|y_i - y_j\|_2^2} w_{i,j}(y_i - y_j)$$

- **Repulsive force** between x_i , with coordinates y_i in the reduced space, and a non-neighboring point x_q , with low-dimensional coordinates y_q :

$$\frac{2b}{(\epsilon + \|y_i - y_q\|_2^2)(1 + a\|y_i - y_q\|_2^{2b})}(1 - w_{i,q})(y_i - y_q)$$

These forces help preserve both local distances between nearby points and prevent overlap between distant points, ensuring a meaningful low-dimensional representation.

2.6. Evaluation

To evaluate the proposed models' performance, two metrics and a specific evaluation protocol are defined. Unlike the traditional Offline approach, where model performance is typically evaluated on a separate test set containing unseen samples at training phase, Streaming Machine Learning introduces a continuous evaluation process. In SML, the model's performance is assessed incrementally as new data points are observed. This approach requires evaluating the model's performance at every step to ensure real-time adaptability to changing data streams. By continuously updating the model with each new sample and evaluating its performance in real-time, prequential evaluation provides a more accurate reflection of how the models would perform in evolving environments, such as satellite image classification tasks.

2.6.1. Evaluation Protocol

The evaluation of the models in this study is conducted using **prequential evaluation** [12], a method particularly well-suited for scenarios where data is processed incrementally, such as in Streaming Machine Learning. In prequential evaluation, the model makes predictions for each incoming data point based on its current parameters and subsequently receives feedback by observing the true class label of the sample. Following this, the model updates its internal parameters to adapt to the new information, as described in Algorithm 2.1. This process of prediction followed by model update ensures that the model is continuously refining its understanding of the data stream as it evolves.

One of the key advantages of prequential evaluation is its ability to simulate real-world conditions where data arrives in a sequential, non-static manner, and models must adapt in real-time to handle new, previously unseen data. This continuous feedback loop allows for an ongoing assessment of model performance and its ability to generalize to new instances as they emerge. The evaluation metrics are updated incrementally with each

Algorithm 2.1 Prequential Evaluation Algorithm

Input: Unbounded data stream $D = \{(x_1, y_1), (x_2, y_2), \dots\}$, Model M

For each time step $t = 1, 2, \dots$:

- Predict $\hat{y}_t = M(x_t)$
(*Make prediction on the current sample*)
- Compare \hat{y}_t with true label y_t
- Update model M based on (x_t, y_t)
(*Update model parameters using the current sample*)
- Compute the evaluation metrics

Return: Desired metrics computed over the stream

new data point, making prequential evaluation an effective method for understanding how well a model performs in a dynamic environment. Unlike traditional batch learning, where the model is fixed after training and performance is evaluated on a test set, prequential evaluation provides a more realistic measure of how a model would operate in an online setting, where it is essential to deal with evolving data without retraining from scratch.

2.6.2. Evaluation Metrics

Two performance metrics are employed to evaluate the effectiveness of the models in a multiclass setting [15]:

- **Accuracy:** This is a general measure of classification performance that reflects the overall correctness of the model. In a multiclass setup, accuracy is calculated as the ratio of correctly predicted instances to the total instances:

$$\text{Accuracy} = \frac{\sum_{i=1}^K TP_i}{N}$$

where K is the number of classes, TP_i represents the true positives for class i , and N is the total number of instances.

- **Balanced Accuracy:** This metric is particularly suitable for imbalanced multiclass datasets, as it gives equal importance to each class. Balanced accuracy is computed as the average of recall obtained on each class:

$$\text{Balanced Accuracy} = \frac{1}{K} \sum_{i=1}^K \frac{TP_i}{TP_i + FN_i}$$

where FN_i denotes the false negatives for class i . This approach ensures that the

model's performance is not disproportionately influenced by the majority class.

To illustrate the computation of these metrics, consider the confusion matrix shown in Table 2.1. The matrix reflects a situation with class imbalance, particularly for Class 3, which has fewer samples than the other classes. This imbalance poses challenges for the model, as it correctly classifies only 3 out of 10 samples in Class 3.

The Accuracy is computed as the ratio of correctly predicted instances (diagonal elements) to the total instances:

$$\text{Accuracy} = \frac{50 + 45 + 3 + 47}{160} = \frac{145}{160} = 0.90625$$

So, the overall Accuracy is approximately 90.6%.

Balanced Accuracy considers each class's recall to give an equal weight to all classes, which is especially important in imbalanced datasets. Recall for each class is computed individually and then their average is the final result:

- **Class 1:**

$$\text{Recall}_{\text{Class 1}} = \frac{TP_1}{TP_1 + FN_1} = \frac{50}{50 + 2 + 1 + 0} = \frac{50}{53} \approx 0.9434$$

- **Class 2:**

$$\text{Recall}_{\text{Class 2}} = \frac{TP_2}{TP_2 + FN_2} = \frac{45}{1 + 45 + 3 + 1} = \frac{45}{50} = 0.9$$

- **Class 3:**

$$\text{Recall}_{\text{Class 3}} = \frac{TP_3}{TP_3 + FN_3} = \frac{3}{0 + 5 + 3 + 2} = \frac{3}{10} = 0.3$$

- **Class 4:**

$$\text{Recall}_{\text{Class 4}} = \frac{TP_4}{TP_4 + FN_4} = \frac{47}{0 + 1 + 2 + 47} = \frac{47}{50} = 0.94$$

Actual \ Predicted	Class 1	Class 2	Class 3	Class 4
Class 1	50	2	1	0
Class 2	1	45	3	1
Class 3	0	5	3	2
Class 4	0	1	2	47

Table 2.1: Example of confusion matrix.

The Balanced Accuracy is the average of the individual recalls:

$$\text{Balanced Accuracy} = \frac{0.9434 + 0.9 + 0.3 + 0.94}{4} = 0.77085$$

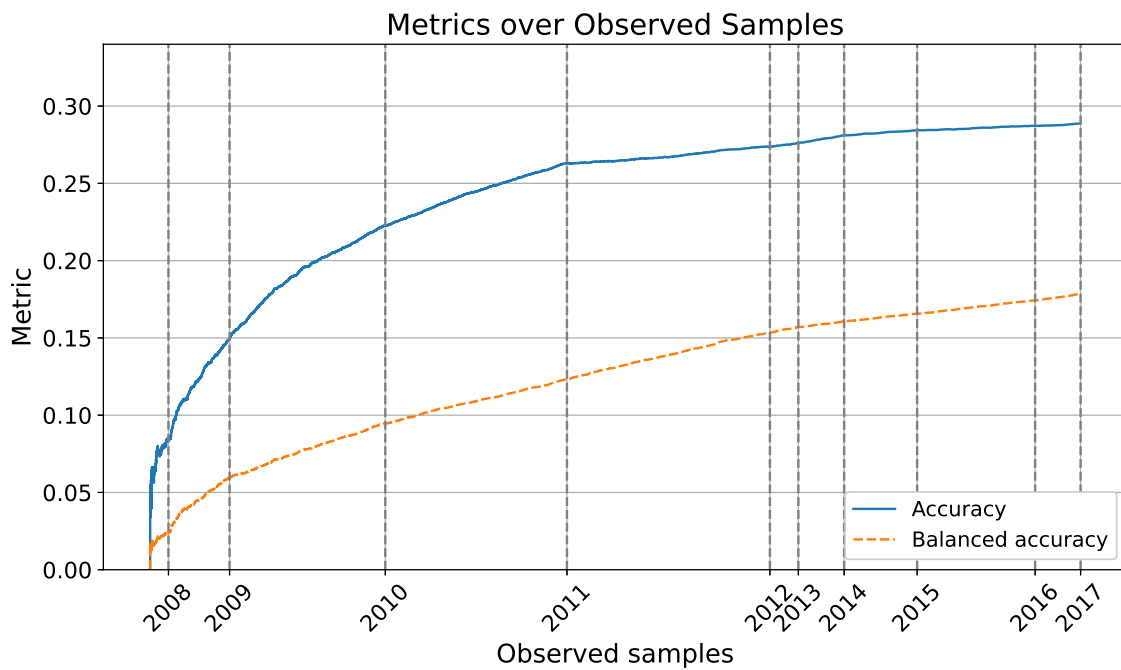
Thus, the Balanced Accuracy is approximately 77.1%, significantly lower with respect to the Accuracy.

2.6.3. Metric Interpretation

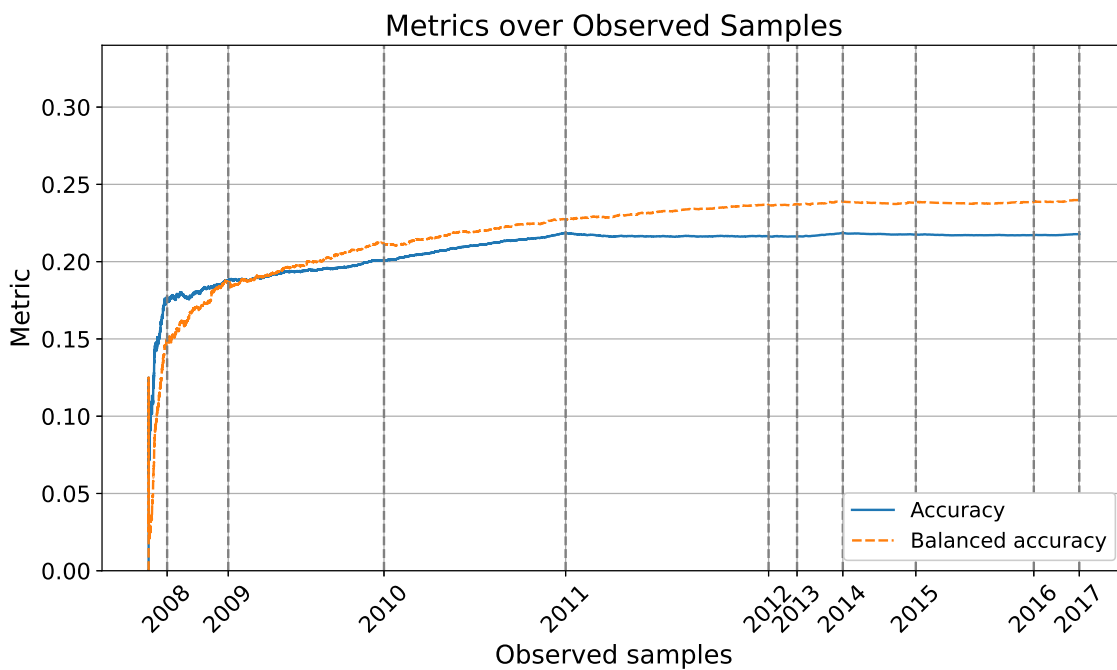
The interpretation of the metrics is critical for understanding model performance in a multiclass setting:

- A high accuracy matched with a relatively low balanced accuracy suggests that the model may be biased towards majority classes, potentially neglecting the minority classes. This scenario indicates that the model is not adequately capturing the distribution of the dataset, like in the case of the example in Table 2.1.
- Conversely, a higher balanced accuracy indicates that the model is effectively learning from all classes, demonstrating a more comprehensive understanding of the data. This situation reflects the model's capability to generalize across all classes, which is especially important in applications involving imbalanced multiclass datasets, like the one provided in this thesis.

To visualize these concepts, Figure 2.8a illustrates an example of behavior biased toward majority classes, while Figure 2.8b displays a more balanced performance. This challenge is particularly pronounced in the multiclass setting studied in this thesis, which involves 62 distinct classes. The large number of categories amplifies the risk of class imbalance, making it difficult for models to adequately represent minority classes. Consequently, this often results in significant discrepancies between overall accuracy and balanced accuracy. These differences highlight the importance of addressing class imbalance, as it ensures that the model's performance is not disproportionately influenced by the most frequent classes.



(a) Bias towards Majority Classes.



(b) Balanced Performance.

Figure 2.8: An example of possible performance plots illustrating majority class bias and balanced performance.

3 | Problem Statement

This Chapter outlines the research questions that form the foundation of this thesis. Section 3.1 provides an overview of the primary drivers that guided this work, while Section 3.2 highlights the major challenges encountered during the research process.

3.1. Research Questions

The goal of this thesis is to develop an efficient and scalable solution for satellite image classification using Streaming Machine Learning (SML) methods. The research is driven by the following research questions:

- **RQ1:** Can the current limitations of the state-of-the-art Streaming Linear Discriminant Analysis (SLDA) classifier be improved to efficiently **handle concept drift** in streaming data and maintain performance on continuously updating satellite image data?
- **RQ2:** Is it possible to develop a novel streaming dimensionality reduction approach that incorporates **data similarity**, inspired by existing offline methods?
- **RQ3:** Which SML method provides the **optimal trade-off** between classification performance and computational efficiency for evolving satellite images, eventually introducing dimensionality reduction?
- **RQ4:** How do the proposed methods perform compared to **state-of-the-art techniques** in terms of performance and scalability when applied to the context of temporally correlated satellite images?

3.2. Research Challenges

Satellite Image Classification in a streaming context presents several challenges that must be addressed to ensure accuracy, efficiency, and adaptability.

A significant challenge in learning from data streams is that they are typically temporally

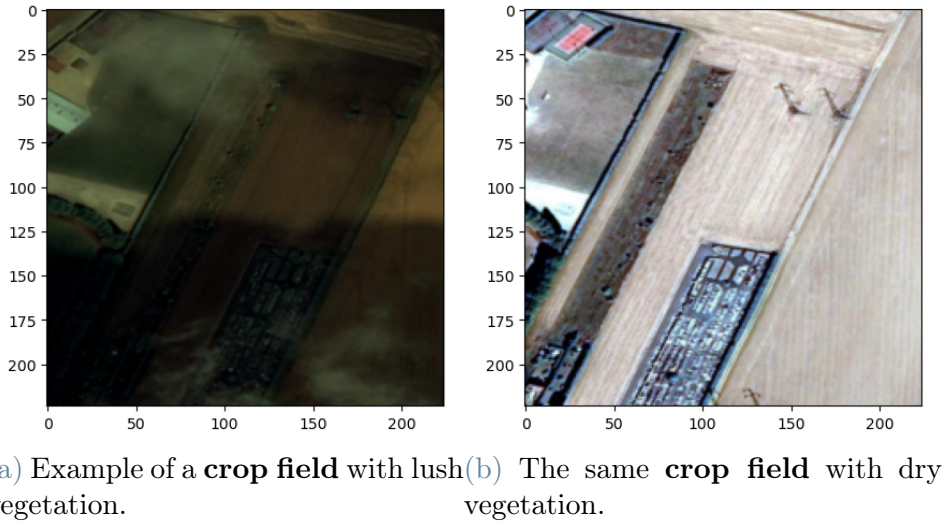
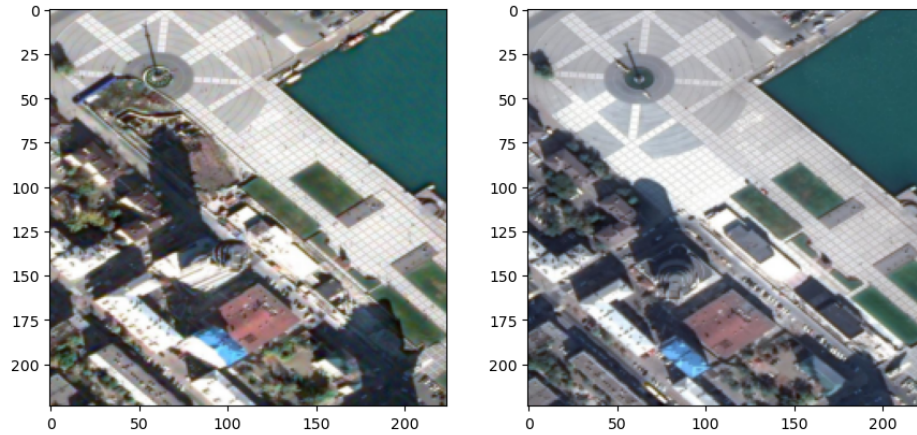


Figure 3.1: Illustration of seasonal variations in satellite images.

arranged [36], with changes in the distribution resulting from the passage of time. In the context of time-ordered image streams, the temporal variation in the data distribution is formally defined as temporal distribution shift [34]. Land-use categories, such as agricultural fields or urban areas, may appear differently over time due to seasonal changes or urbanization, while still representing the same class. These temporal variations can complicate the classification process, requiring models that are capable of adapting to evolving patterns in the data, even when the labels remain static. Figure 3.1 illustrates an example of seasonal effects, where two images of the same crop field exhibit contrasting vegetation states.

Concept drift introduces a second major challenge by relaxing the assumption of an identical data distribution over time, an assumption that many traditional Machine Learning models rely on. When the underlying data distribution shifts, static models struggle to maintain accuracy, necessitating the introduction of adaptive solutions that can detect and respond to these changes. Figure 3.2 illustrates this concept drift: two images taken in the same location at different times represent different classes due to completed construction between the capture dates.

Additionally, the high dimensionality of satellite images, which often contain vast amounts of information spread across multiple channels, must be taken into account. Processing and analyzing such large-scale data in real time is computationally expensive, requiring the use of dimensionality reduction techniques to reduce processing costs without losing critical information. Although dimensionality reduction algorithms are commonly employed to simplify data before classification, selecting the most effective approach is crucial. It



(a) An image representing a **construction site** from 2013. (b) An image from the same location, now classified as a **fountain** in 2016.

Figure 3.2: Example of concept drift in satellite images: The construction site of a fountain in the upper left portion of the image is completed over time.

requires determining the extent to which dimensionality can be reduced while still preserving the essential features necessary for accurate classification. Developing methods that incorporate data similarity while reducing dimensions is particularly important in maintaining classification quality.

Finally, achieving a balance between computational efficiency and classification accuracy is critical in streaming environments. High-dimensional data and the need for frequent model updates can increase processing times, which may not be feasible in real-time applications. Optimizing this trade-off between responsiveness and accuracy remains a central challenge.

4 | Problem Solving

This Chapter presents the rationale behind the proposed methodology and the development of novel methods introduced in this thesis. The discussion addresses the motivations that guided the design process, providing a comprehensive understanding of the employed approaches and the dataset considered.

4.1. Proposed Approach

As highlighted in Section 2.1, an effective Streaming Machine Learning (SML) classifier requires low-dimensional tabular input data to ensure both efficiency and scalability. However, the raw satellite images in the dataset [8], with dimensions $224 \times 224 \times 3$, contain approximately 150,000 features. This high-dimensional input is unsuitable for direct use in a streaming classifier due to the significant computational burden it imposes. Consequently, a preliminary feature extraction step is essential. Pre-trained Convolutional Neural Networks (CNNs) are employed for this purpose, enabling the transformation of raw image data into a more manageable feature set.

After feature extraction, the resulting set has an approximate dimensionality of $d = 500 - 600$, depending on the CNN used, making it feasible to feed streaming classifiers without further reduction. At this stage, the application of different classifiers is explored:

- **Gaussian Naive Bayes** and **Softmax Regression** effectively manage medium-dimensional data, with computational complexities of $O(d)$ and $O(d \times K)$ for each sample, respectively, where K is the number of classes. These classifiers offer a favorable trade-off between classification performance and computational efficiency.
- **Streaming Linear Discriminant Analysis (SLDA)** excels in performance when handling high-dimensional data due to its capabilities to leverage the curse of dimensionality. However, SLDA's computational complexity of $O(d^3)$ poses a significant challenge for large-scale applications, as it requires inverting a $d \times d$ matrix—a process that is computationally intensive. To mitigate this, the SLDA implementation leverages parallel computation, aiming to improve performance and make it

competitive with other classifiers.

A primary objective is to assess these classifiers, balancing their classification performance with their computational efficiency. To explore multiple configurations, dimensionality reduction techniques are applied to evaluate their impact on both execution time and accuracy, allowing for a more in-depth analysis of classifier performance within a reduced feature space.

It must be pointed out that dimensionality reduction is an optional step in the classification pipeline. The proposed classifiers are capable of learning effectively from the extracted feature set without any additional reduction, as shown in preliminary experiments in Section 5.2.1. However, this approach often leads to high computational costs, particularly in simulation scenarios where processing speed is critical. Thus, dimensionality reduction is introduced as an optimization strategy to explore potential reductions in computation time and resource usage, without compromising classification accuracy.

In this regard, Sparse Random Projection serves as a computationally efficient bench-

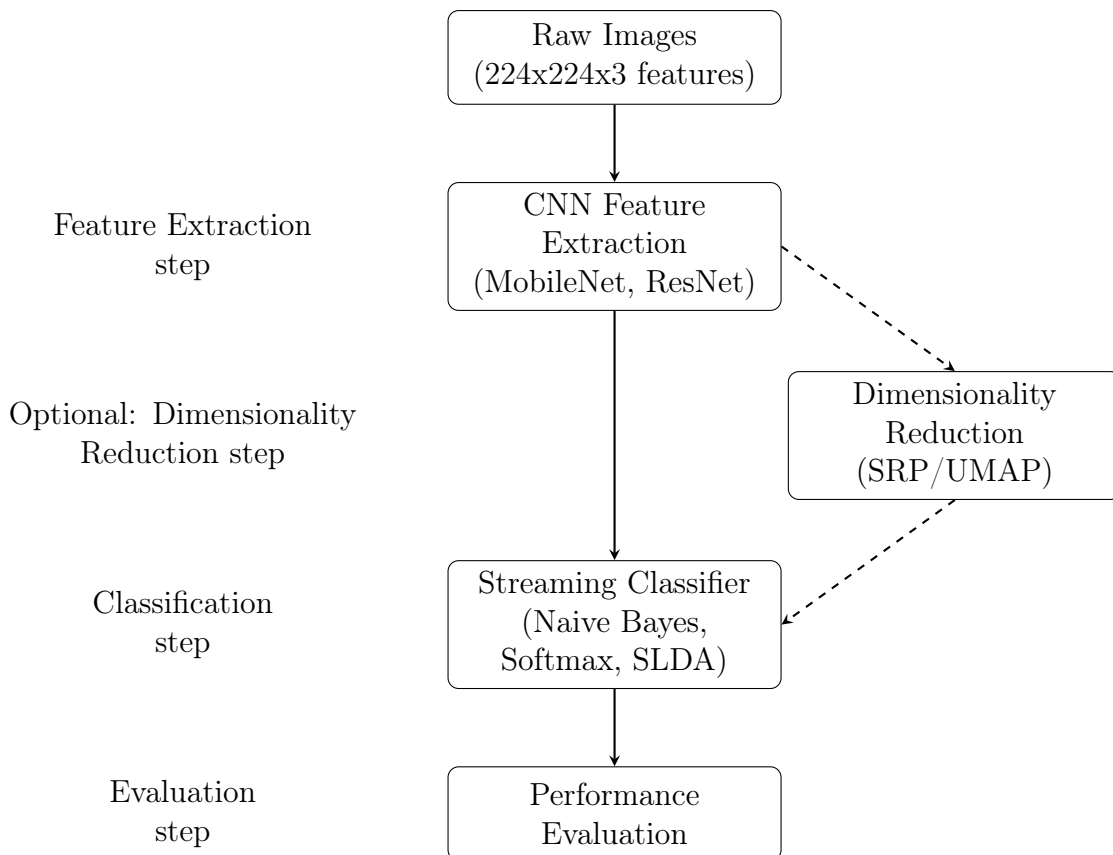


Figure 4.1: Pipeline for Streaming Machine Learning to deal with images, highlighting that dimensionality reduction is an optional step.

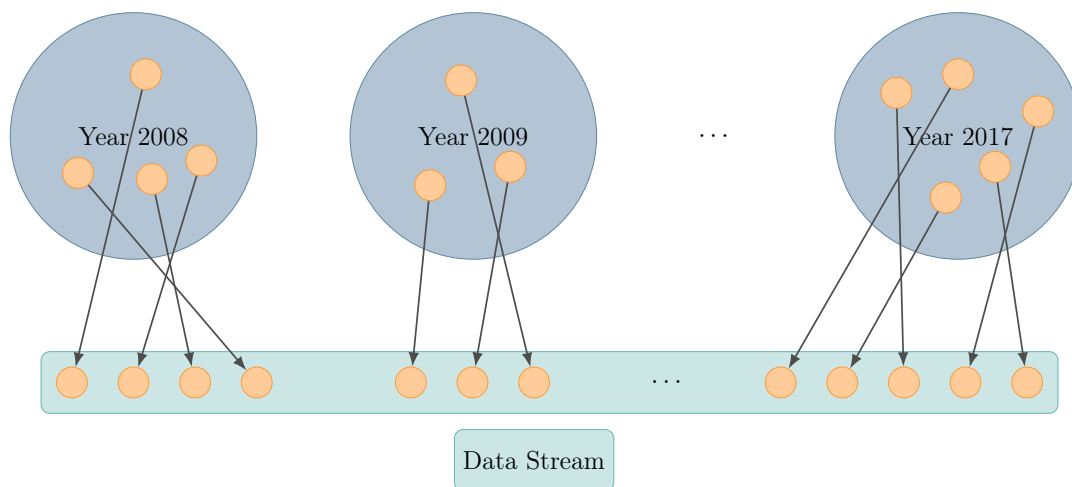


Figure 4.2: Setup of data stream starting from batches observed in different years.

mark. Additionally, a novel streaming adaptation of Uniform Manifold Approximation and Projection (UMAP) has been developed, based on the offline UMAP method.

Overall, incorporating dimensionality reduction helps to understand the trade-offs between execution time and classification performance, while reaffirming that it is not a mandatory step but rather a potential enhancement to the proposed approach in the context of learning from dynamically evolving satellite images. Figure 4.1 illustrates the general pipeline proposed, allowing flexibility in selecting specific methods for each stage.

Notably, the proposed dataset lacks a codified order among samples from the same year. To implement a streaming approach, an ordering is introduced to assess all methods in a streaming manner, as shown in Figure 4.2. In the proposed setup, each batch corresponds to a specific year of data acquisition. The visualization clarifies how the data from various years flows into the streaming classifier, emphasizing the need for an organized sequence in processing the satellite images.

4.2. Analysis of Functional Map of the World - Time

A preliminary analysis of the dataset is essential to identify the appropriate approaches for addressing the classification problem within the specified context. By examining the distribution of classes and their occurrences over time, insights can be gained into the characteristics of the data that will influence subsequent stages of the methodology.

The dataset exhibits some imbalance among the 62 categories, as illustrated in Table 4.1 and in Figure 4.3. Specifically, categories such as recreational facility contain more than 10,000 samples across the 16 years, while categories like nuclear powerplant and

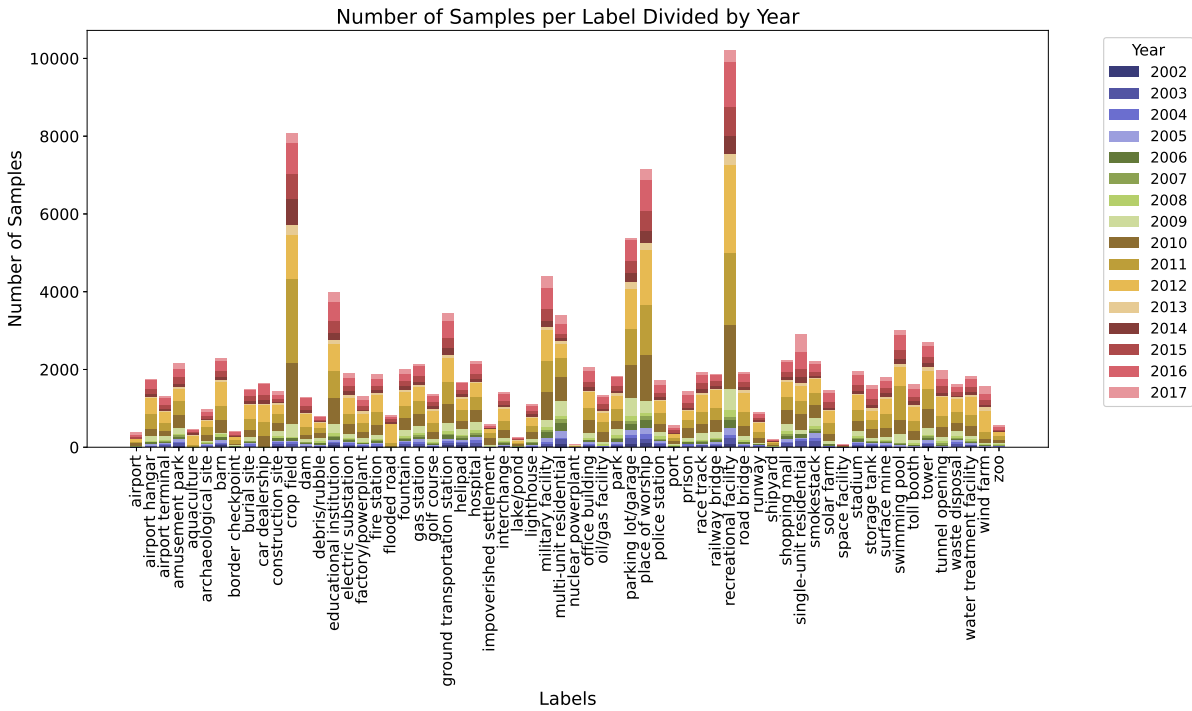


Figure 4.3: Distribution of the labels in the various years.

space facility have fewer than 100 occurrences in the entire dataset. Furthermore, the relative frequency of the images changes over time, as noted in Figure 4.4. For example, categories such as multi-unit residential are more frequent in the central years, indicating that this class was among the majority during those years and was less relevant in others. A similar trend is noted for crop field, which gained relevance in the later years, while smokestack was primarily observed in the initial years. Conversely, the category recreational facility maintains a relatively constant proportion of observation across all years.

The proposed methods will require an initial phase for parameter setting, performed offline on some initial training data. For consistency in the results, the samples observed in the first six years, comprising 12,874 data points, will be used solely for the initialization of the models requiring it, while the remaining data (113,291 samples) will be involved in the performance evaluation of the different models proposed in the thesis.

The separation illustrated in Figure 4.5 rigorously tests all models on previously unseen data, thereby ensuring a consistent performance evaluation across different methods.

Land-use label	Occurrences	Land-use label	Occurrences	Land-use label	Occurrences
airport	378	golf course	1365	recreational facility	10210
airport hangar	1761	ground transportation station	3448	road bridge	1926
airport terminal	1314	helipad	1683	runway	913
amusement park	2169	hospital	2201	shipyard	219
aquaculture	479	impoverished settlement	592	shopping mall	2239
archaeological site	983	interchange	1403	single-unit residential	2908
barn	2290	lake/pond	245	smokestack	2220
border checkpoint	415	lighthouse	1118	solar farm	1462
burial site	1485	military facility	4400	space facility	77
car dealership	1633	multi-unit residential	3388	stadium	1952
construction site	1442	nuclear powerplant	79	storage tank	1581
crop field	8075	office building	2066	surface mine	1803
dam	1288	oil/gas facility	1354	swimming pool	2998
debris/rubble	792	park	1836	toll booth	1629
educational institution	3979	parking lot/garage	5383	tower	2703
electric substation	1904	place of worship	7157	tunnel opening	1976
factory/powerplant	1318	police station	1713	waste disposal	1624
fire station	1870	port	555	water treatment facility	1819
flooded road	813	prison	1431	wind farm	1571
fountain	2014	race track	1933	zoo	554
gas station	2146	railway bridge	1883		

Table 4.1: Occurrences of the 62 labels, marking the majority class in red and the minority class in blue

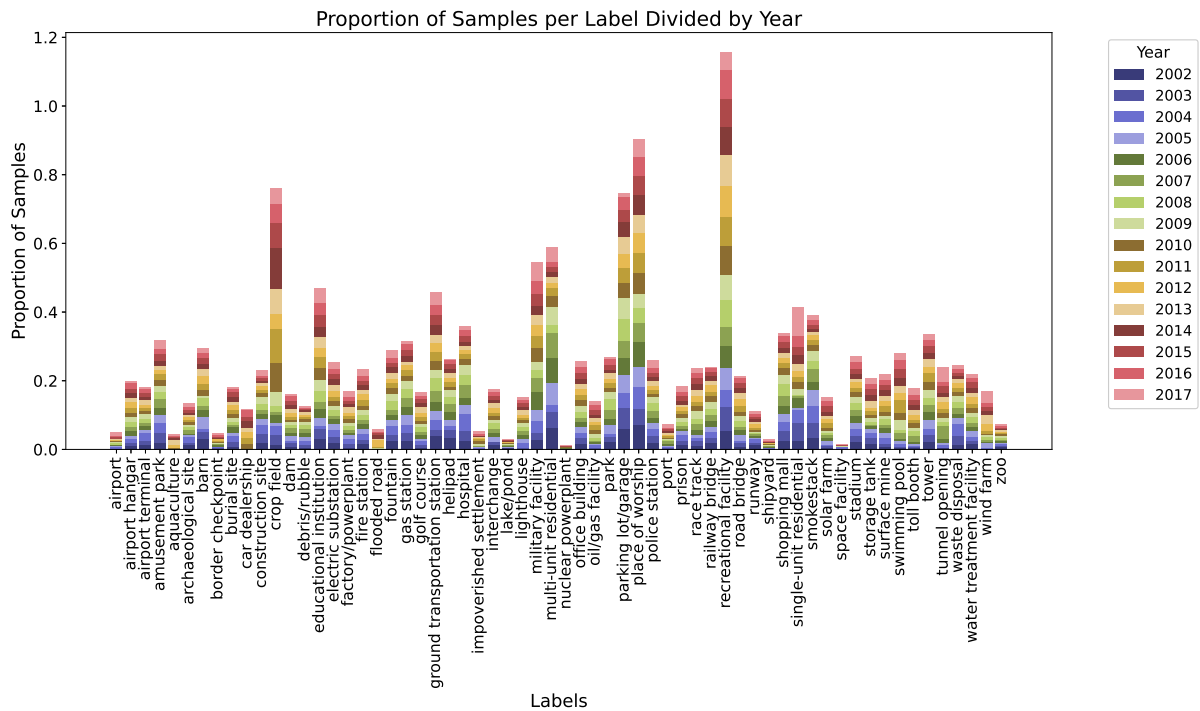


Figure 4.4: Proportion of the labels in the various years.

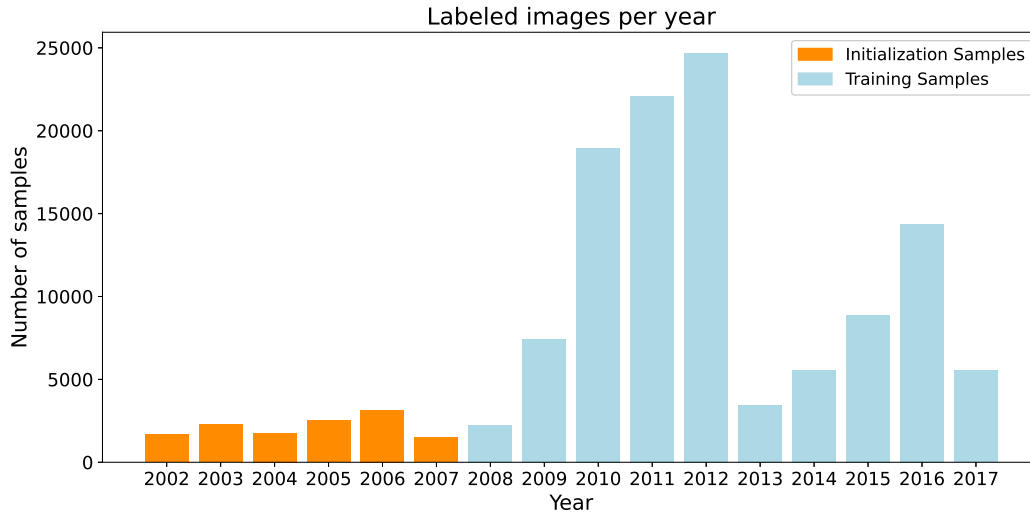


Figure 4.5: Distribution of samples across the years.

4.3. Kalman-Streaming Linear Discriminant Analysis

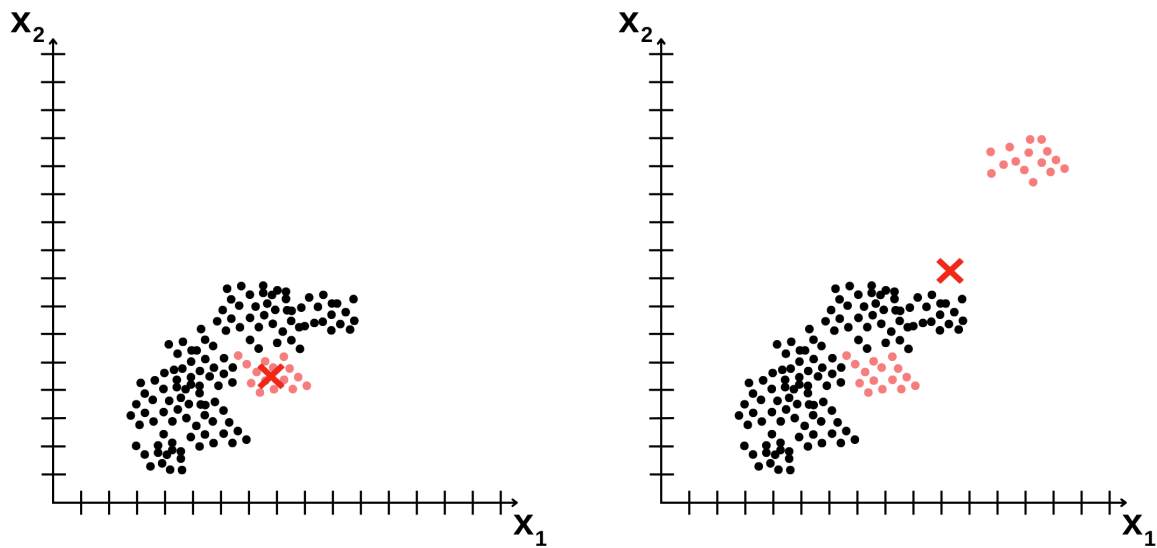
This section presents the development of an innovative Streaming Machine Learning classifier, building upon the foundational work of Hayes and Kanan on Streaming Linear Discriminant Analysis (SLDA) [16].

4.3.1. SLDA Limitations

As discussed in Section 2.2.3, the original SLDA algorithm struggles to handle concept drift effectively. In a stable scenario where all classes are linearly separable, observing a new data point in a different region assigns a weight of $\frac{1}{N+1}$ to this sample, where N represents the number of previously observed samples in class k . This weighting helps keep the sample mean closer to the original region, providing robustness against outliers.

On the other hand, when concept drift occurs and subsequent points are observed in the new region, the sample mean struggles to shift. For the mean to accurately reflect the new region, significantly more data points must be observed there than in the original region. If N points are observed in both the initial and new regions, the sample mean will lie between the two, needing more than N points in the new region to fully shift the mean. A visual benchmark is provided in Figure 4.6.

This limitation results in SLDA struggling to track concept drift, particularly when multiple drifts occur in rapid succession. Consequently, SLDA may underperform in terms



(a) The initial stable distribution for a bidimensional dataset. Class mean is marked with a red cross. (b) After observing N samples belonging to the red class both in the original region and in the new region, the class mean is still far from converging to the new distribution.

Figure 4.6: Impact of concept drift on SLDA for classification tasks.

of classification accuracy despite demanding higher computational resources compared to other classifiers like Gaussian Naive Bayes and Softmax Regression.

4.3.2. Kalman-SLDA

Given the structure of SLDA, which allows for the separate management of class means, the algorithm proves to be robust in scenarios with unbalanced classes. Even with a limited number of observations in a minority class, the classifier effectively tracks the distribution of that class. Since this native behavior is particularly effective, Kalman filtering is integrated in SLDA original model to update the mean vector and covariance matrix, maintaining the integrity of the original structure while addressing the limitations discussed previously.

Kalman Filtering

The Kalman filter operates on the principles of Bayesian estimation, assuming that the system state can be modeled using linear equations and is affected by Gaussian noise. The filter maintains a prediction of the system's state based on prior observations and continuously refines this prediction as new data becomes available. Kalman filters are

powerful recursive algorithms designed for estimating the state of a dynamic system from a series of noisy measurements [32]. Their significance lies in their ability to provide optimal state estimates in the presence of uncertainty, making them invaluable in real-time applications.

The Kalman filter consists of two main processes: prediction and update.

During the **Prediction** step, the filter predicts the current state based on the previous state and the system dynamics. It also predicts the uncertainty associated with this estimate.

$$\begin{cases} \hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k \\ P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \end{cases} \quad (4.1)$$

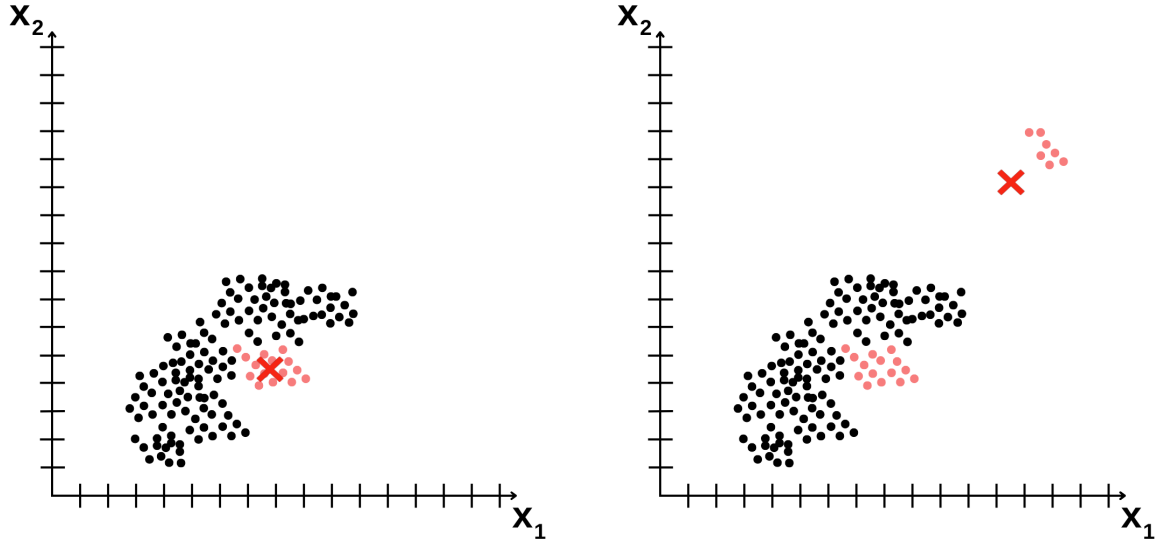
Here, $\hat{x}_{k|k-1}$ is the predicted state, F_k is the state transition model, B_k is the control input model, u_k is the control input, $P_{k|k-1}$ is the predicted uncertainty, and Q_k is the process noise covariance.

In the **Updating** step, a new measurement is received, and the filter updates the state estimate and its uncertainty. This involves calculating the Kalman gain, which determines how much weight to give to the new measurement relative to the current state estimate.

$$\begin{cases} K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \\ \hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k (z_k - H_k \hat{x}_{k|k-1}) \\ P_{k|k} = (I - K_k H_k) P_{k|k-1} \end{cases} \quad (4.2)$$

To clarify the notation, K_k is the Kalman gain, H_k is the measurement model, i.e. a transformation of the predicted state, z_k is the new measurement, R_k is the measurement noise covariance, and I is the identity matrix.

The introduction of the Kalman filter allows for dynamic adjustment of the weight assigned to each new sample, influencing how the mean and covariance are computed over time. The primary function of the Kalman filter is to adaptively adjust the weights of samples based on their predictions and observed outcomes, as expressed in the weight update equations where the Kalman gain determines the influence of the current sample on the model's estimates. This approach, illustrated in Figure 4.7, builds on the principles established in prior work [37].



(a) The initial stable distribution for a bidimensional dataset. Class mean is marked with a red cross. (b) Differently from SLDA, the model updates the mean quickly to follow concept drift, needing less observations to converge to the new distribution.

Figure 4.7: Impact of concept drift on Kalman-SLDA for classification tasks.

Model details

The model initializes with the following values:

$$r = 1000$$

$$q = 1$$

$$m = 6$$

$$x_{k,0} = 0 \quad \forall k \text{ classes}$$

$$p_{k,0} = 0 \quad \forall k \text{ classes}$$

The state predictions and updates are governed by the following relationships, derived from Equations (4.1) and (4.2):

$$V = m \times \mathbf{1}_{\{y_i \neq k\}} + 1$$

$$K_t = \frac{p_{k,t}}{p_{k,t} + r}$$

$$x_{k,t+1} = x_{k,t} + K_t(V - x_{k,t})$$

$$p_{k,t+1} = p_{k,t}(1 - K_t) + q$$

Here, K_t represents the Kalman gain, r denotes the measurement noise covariance, q

indicates the process noise covariance, and m is the error importance parameter, a value that controls how much a misclassification error is weighted. The measurement model H_t and the state transition model F_t are set to 1, while the control input model B_t is set to 0, given that the intention is to track a simple process: the weight assigned to each sample that should adapt over time.

The updated weights $x_{k,t+1}$ for class k are then used to compute updates for the covariance and class mean, balancing the influence of new observations against prior data. The updates for the mean and covariance follow the original model's formulations in Equations (2.1) and (2.2), incorporating the updated weights:

$$\begin{aligned}\boldsymbol{\mu}_{k,t+1} &= \frac{c_{k,t} \cdot \boldsymbol{\mu}_{k,t} + x_{k,t+1} \cdot \mathbf{z}_t}{c_{k,t} + x_{k,t+1}} \\ \boldsymbol{\mu}_{j,t+1} &= \boldsymbol{\mu}_{j,t} && \forall j \neq k_t \\ c_{k_t,t+1} &= c_{k_t,t} + 1 \\ c_{j,t+1} &= c_{j,t} && \forall j \neq k_t \\ \boldsymbol{\Sigma}_{t+1} &= \frac{t \cdot \boldsymbol{\Sigma}_t + x_{k,t+1} \cdot \boldsymbol{\Delta}_t}{t + x_{k,t+1}} \\ \boldsymbol{\Delta}_t &= \frac{t \cdot (\mathbf{z}_t - \boldsymbol{\mu}_{k,t})(\mathbf{z}_t - \boldsymbol{\mu}_{k,t})^T}{t + 1}\end{aligned}$$

Considerations on the proposed method

In the context of SLDA, the Kalman filter's adaptability allows for real-time updates to the model's estimates, making it particularly effective for streaming data and improving performance over traditional SLDA. As new samples are processed, the filter assigns appropriate weights based on the accuracy of previous predictions, refining the classification performance. This adaptability ensures that the model remains sensitive to changes in the data distribution, improving its robustness in dynamic environments.

A key aspect of this adaptation is that it remains robust to the initial parameters r and q , which regulate the convergence speed of the weight adaptation process. However, the error importance parameter m is the most critical to tune. It manages the trade-off between recency and historical information: a higher value of m emphasizes the importance of recently misclassified samples, while a lower value gives more weight to the history of the process. In the extreme case where $m = 0$, there is no distinction between recent and past samples, comparable to the behavior of the original SLDA.

Despite the advantages introduced by integrating Kalman filtering into SLDA, certain lim-

itations should be acknowledged. Firstly, the Kalman filter assumes linearity in the state dynamics and measurement processes. When these assumptions do not hold, performance may be suboptimal. Additionally, the effectiveness of the model is highly dependent on the accurate tuning of the error importance parameter m . Improper tuning of this parameter can lead to poor weight adaptation, negatively affecting classification performance. Therefore, careful selection of m is crucial to have an improvement of the performance compared to the baseline SLDA.

4.4. Streaming-UMAP and Batch-UMAP

This section presents a preliminary work to adapt the existing offline UMAP algorithm into a streaming setting. The rationale guiding the extension is rather simple, but can serve as a starting point for future work with a more detailed and extended implementation.

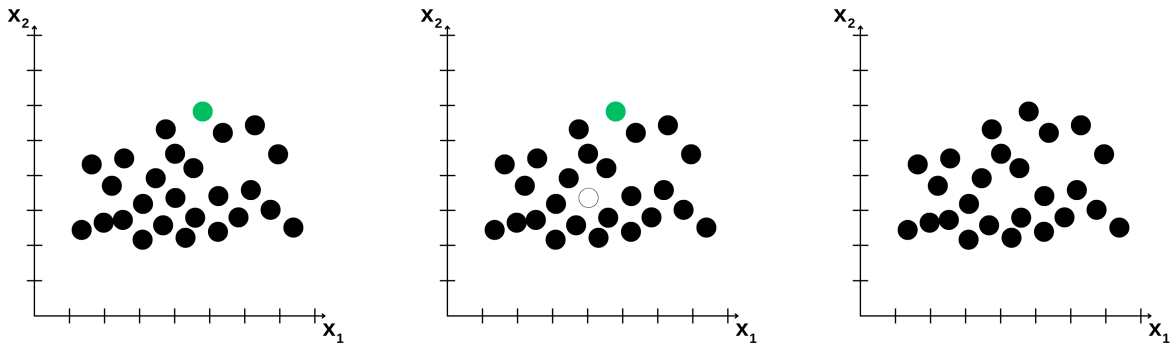
4.4.1. UMAP: Offline technique

Uniform Manifold Approximation and Projection (UMAP) is a dimensionality reduction technique that preserves both local and global structures in high-dimensional data. It is particularly useful for clustering and visualization, as it maintains the neighborhood relationships from the original high-dimensional space in its reduced representation. UMAP accomplishes this by using a force-directed layout algorithm, ensuring that data points close together in the original space remain close in the low-dimensional embedding, as better specified in Section 2.5.2.

However, UMAP was initially designed to operate in an offline setting, requiring access to the entire dataset at once. This design is not effective for real-time applications where data arrives continuously, such as in satellite image classification scenarios. Adapting UMAP to handle streaming data needs several key modifications to its underlying framework.

4.4.2. Streaming-UMAP

Streaming-UMAP is designed to handle data in a continuous manner, making it more suitable for real-time processing. The main innovation of Streaming-UMAP lies in the **sliding window approach**, which allows the embedding to be updated incrementally as new data arrives. Instead of embedding the entire dataset at once, only the most recent M observations are considered. As each new data point is added, the oldest one is removed from the window, shifting the window forward over time. Figure 4.8 displays a



(a) A single point (marked in green) is fit based on the embedding of the last $M = 25$ points observed.
 (b) The oldest point (marked in white) is removed from the sliding window.
 (c) The updated batch is used to fit the following point.

Figure 4.8: Iterative fitting of Streaming-UMAP on a bidimensional dataset.

visual example of the method.

Key modifications in Streaming-UMAP include:

- **Sliding Window:** The algorithm maintains a window of M observations, continuously updating the embedding by adding new data points and removing old ones.
- **Fitting New Data:** When a new point arrives, it is fitted into the existing low-dimensional space based on its relation to the points already in the window.
- **Dynamic Neighbor Updates:** The nearest neighbors and model parameters are updated in real-time, ensuring the embedding remains consistent as new data arrives.

This approach maintains the benefits of UMAP's original method while adapting to streaming data. However, challenges arise when no strong ordering in the samples is provided. This is not a problem in real time applications, but it is for the data at our disposal. A batch adaptation is therefore required in this situations, also suitable for problems of resource constrained machines where a continuous fitting can lead to out-of-memory issues.

4.4.3. Batch-UMAP

Batch-UMAP offers an alternative for handling data that arrives in periodic intervals or batches, such as when data are processed on a yearly basis. Instead of updating the embedding with each new data point, Batch-UMAP processes data in groups and refines

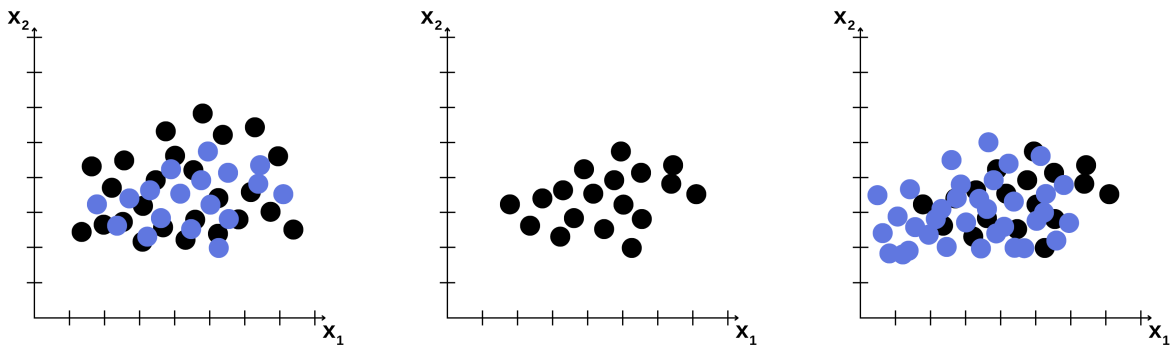
the embedding after each batch. In this method, data from year k are embedded using the existing embedding of data from year $k - 1$ as a reference. Once the batch from year k is processed, the embedding is updated, and the points from year $k - 1$ are discarded. This approach ensures that the embedding remains consistent across time while reducing computational overhead. A visualization of the procedure is provided in Figure 4.9.

The primary differences between Batch-UMAP and Streaming-UMAP are:

- **Batch-Wise Fitting:** Instead of continuous updates, data is processed in batches, with each batch's embedding starting from the previous year's embedding.
- **Efficiency for Large Datasets:** By updating only once per batch, this method can handle larger datasets more efficiently than continuous updates.
- **Hyperparameter Tuning:** Similar to Streaming-UMAP, Batch-UMAP requires careful tuning of key hyperparameters, such as the number of neighbors (`n_neighbors`) and the distance metric used.

Despite its advantages, Batch-UMAP can produce suboptimal results when handling batches of vastly different sizes, as smaller batches may not fully capture the global structure that larger ones establish. This limitation can lead to inconsistencies in feature representation across batches, especially in scenarios with significant data variation. Nevertheless, Batch-UMAP remains a suitable compromise when real-time updates are unnecessary, as it allows for effective dimensionality reduction in contexts where periodic data influxes are expected and computational efficiency is prioritized.

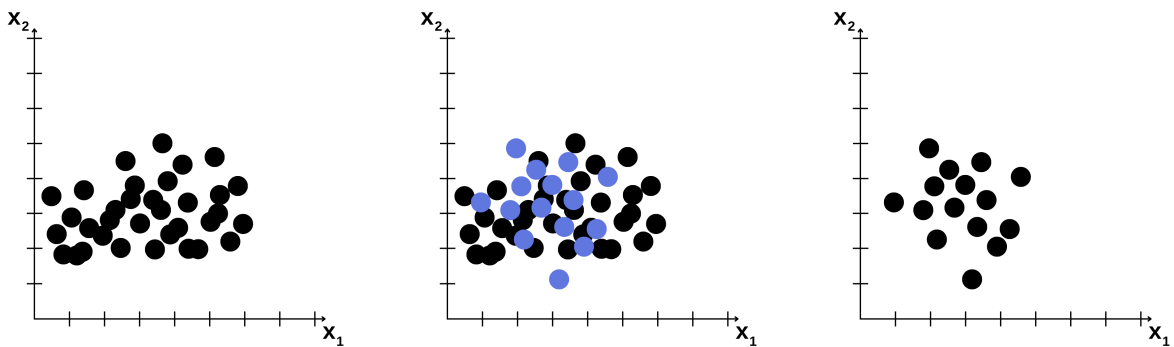
In summary, Streaming-UMAP is ideal for real-time data, enabling continuous updates that reflect ongoing changes in the data distribution. In contrast, Batch-UMAP is suitable for scenarios where data arrives in structured intervals, offering an efficient solution when real-time adjustments are less critical. Together, these methods extend the utility of UMAP in dynamic environments, providing flexible dimensionality reduction options that can be applied to streaming and batch processing contexts, each balancing adaptation speed and computational efficiency.



(a) Batch from year k (black points) serves as a basis to fit points from year $k + 1$ (marked in blue) in the reduced space.

(b) Batch from year k is removed, only points from year $k + 1$ are kept.

(c) Batch from year $k + 1$ (in black) serves as a basis to fit points from year $k + 2$ (marked in blue) in the reduced space.



(d) Batch from year $k + 1$ is removed, only points from year $k + 2$ are kept.

(e) Batch from year $k + 2$ (in black) serves as a basis to fit points from year $k + 3$ (marked in blue) in the reduced space.

(f) Batch from year $k + 2$ is removed, only points from year $k + 3$ are kept.

Figure 4.9: Iterative fitting of Batch-UMAP for a bidimensional dataset.

5 | Experiments and Results

This Chapter presents the experimental framework and the results of the proposed methodology for satellite image classification in a streaming environment. The experiments are designed to evaluate the performance of different Streaming Machine Learning (SML) classifiers and dimensionality reduction techniques, with a focus on accuracy and computational efficiency. Section 5.1 describes the experimental environment, outlining the models used and the configuration of the computational setup. Section 5.2 provides an analysis of the results, with particular attention to the trade-offs between execution time and classification performance across the different methods.

5.1. Experimental Setup

All simulations are conducted on a virtual machine with the following specifications:

- 100 GB of storage memory,
- 16 GB of RAM,
- 8-core processor to enable parallelization.

The experimental pipeline is fully replicable, with random seeds specified for each step to ensure consistency.¹

5.1.1. Convolutional Neural Networks

The first stage of the experiment involves Feature Extraction using two Convolutional Neural Networks (CNNs): **MobileNet v3 Small** and **ResNet18**. Pre-trained on the ImageNet dataset, both networks are used to extract relevant features from satellite images, later provided as input to the SML classifiers.

¹The thesis code is available at: <https://github.com/NicolaFrancescon/Thesis>

5.1.2. Classification Models

The following classifiers are employed for the task of satellite image classification:

- **Gaussian Naive Bayes,**
- **Softmax Regression,**
- **Streaming Linear Discriminant Analysis (SLDA),**
- **Kalman-SLDA.**

Gaussian Naive Bayes and Softmax Regression are implemented in Python using River library [27] and does not require any parameter tuning. SLDA also requires no specific hyperparameters, while Kalman-SLDA is initialized with the parameters outlined in Section 4.3.2 and here reported:

$$\begin{aligned}
 r &= 1000 \\
 q &= 1 \\
 m &= 6 \\
 x_{k,0} &= 0 \quad \forall k \text{ classes} \\
 p_{k,0} &= 0 \quad \forall k \text{ classes}
 \end{aligned}$$

For each classifier, the features extracted from the CNNs are scaled using a Streaming MinMaxScaler to standardize the input data and improve classification performance. The effect of scaling versus not scaling the input is discussed in Section 5.2.1.

5.1.3. Dimensionality Reduction Techniques

To better understand the trade-off between computational cost and classification performance, dimensionality reduction techniques are incorporated after feature extraction as an intermediate step. The two primary methods employed are **Sparse Random Projection** and **Batch-UMAP**. Although attempts to use **Streaming-UMAP** are made, computational constraints limited its extensive application. Sparse Random Projection is configured with a sparse matrix density of 0.33, based on recommendations from the literature [2]. Batch-UMAP and Streaming-UMAP are employed with 20 neighbors, cosine distance, the parameters $a = 65$ and $b = 5$ to balance the attractive and repulsive forces in the low-dimensional embedding, with 200 optimization epochs and 10 samples for the negative sample rate parameter, to improve the accuracy of the optimization at a higher computational cost. Due to the high computational cost of these methods, hyperparameters are tuned through trial and error rather than grid search. Further discussions are presented in Section 6.2. To evaluate the results, the metrics presented in Section 2.6 are

used. To understand the interpretation of the metrics, a careful review of the mentioned section is suggested.

5.2. Analysis of Results

The analysis of results centers on the trade-off between execution time and classification performance across the various classifiers and dimensionality reduction techniques. This trade-off is critical in selecting suitable models, particularly in resource-constrained environments where both speed and accuracy impact applicability. Faster classifiers may enable real-time or near-real-time decision-making, but this often comes at the cost of reduced accuracy, especially in high-dimensional datasets with complex structures. Conversely, more accurate classifiers may require extended processing times, which can limit their practicality in dynamic data settings. By examining this balance, the study aims to identify configurations that optimize performance without sacrificing efficiency.

5.2.1. Comparison of SML Classifiers

Classifiers are initially evaluated without dimensionality reduction, establishing a baseline for performance comparison. Table 5.1 presents the total execution time for classifying the entire dataset after feature extraction using MobileNet v3 Small, excluding feature extraction time and focusing only on classification time. This baseline provides a clear

Classifier	Total Time (minutes)	Time per Sample (seconds)
Kalman-SLDA	134	0.07
SLDA	132	0.07
Gaussian Naive Bayes	85	0.05
Softmax Regression	68	0.04

Table 5.1: Time required to classify the dataset using different SML classifiers.

Classifier	Average Accuracy	Average Balanced Accuracy
Kalman-SLDA	0.30	0.28
SLDA	0.25	0.27
GNB	0.21	0.22
SMR	0.25	0.13

Table 5.2: Average classification performance for different SML classifiers.

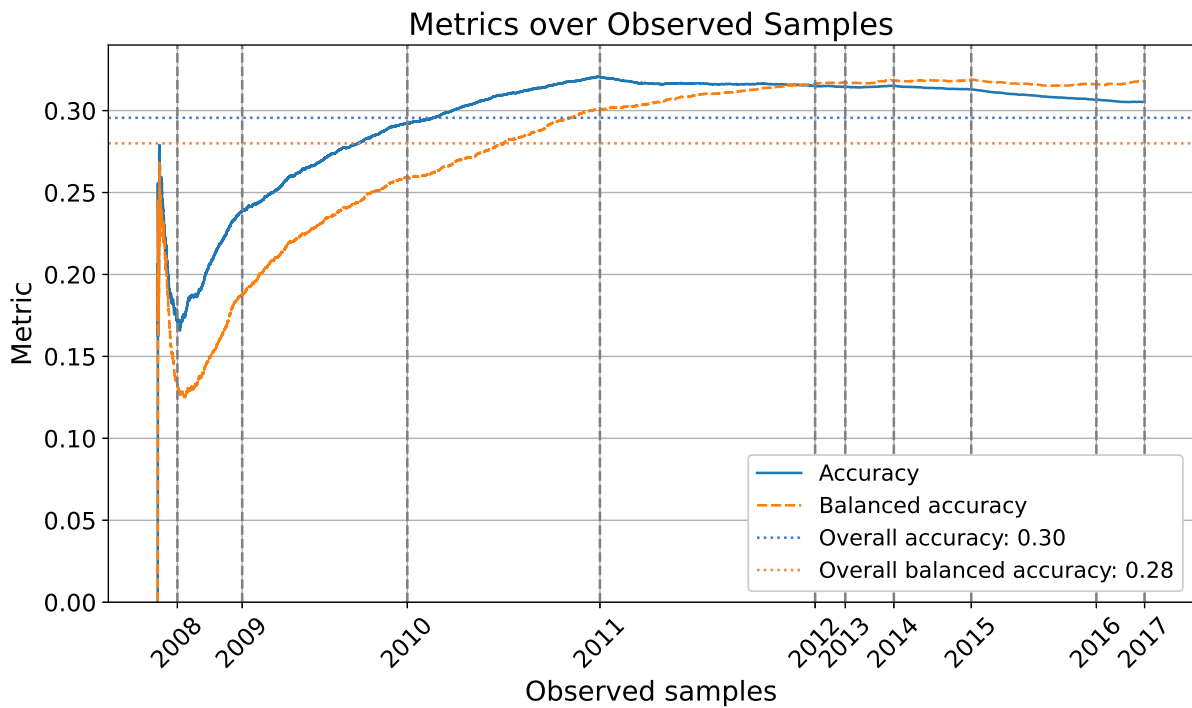


Figure 5.1: Classification performance for Kalman-SLDA.

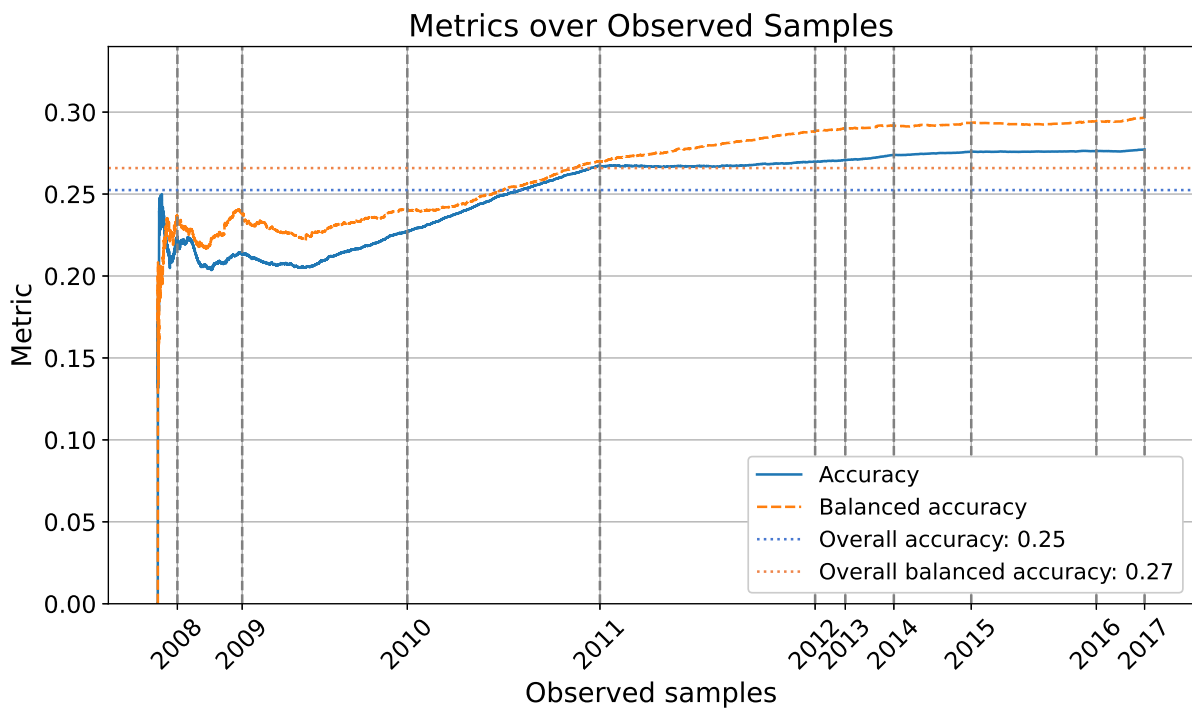


Figure 5.2: Classification performance for SLDA.

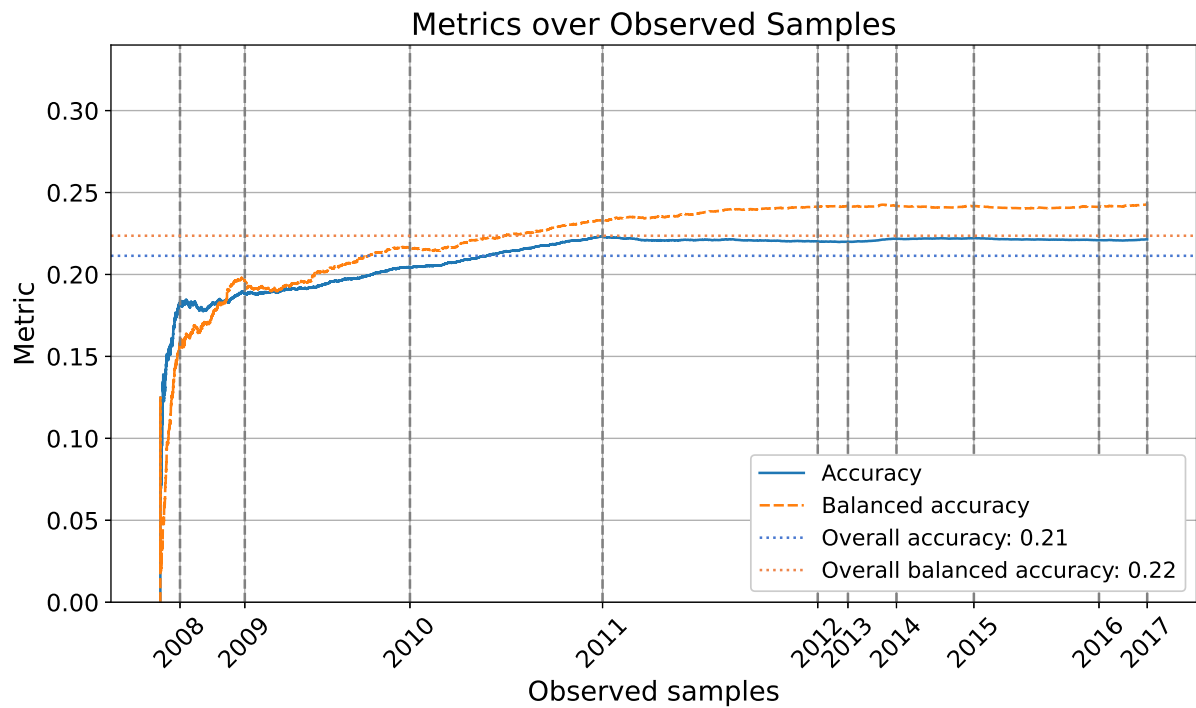


Figure 5.3: Classification performance for Gaussian Naive Bayes.

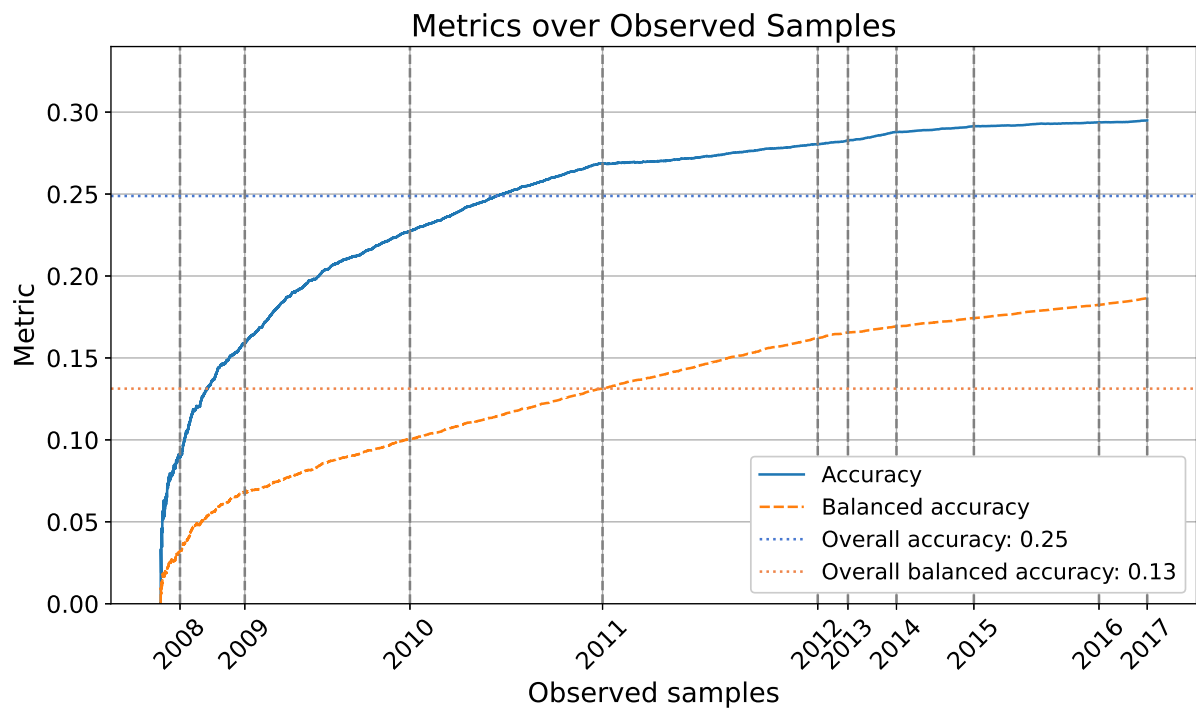


Figure 5.4: Classification performance for Softmax Regression.

reference for assessing the impact of dimensionality reduction techniques introduced later in the study. Additionally, by isolating classification time, this analysis highlights the computational demands of each classifier.

As shown in Table 5.1, Gaussian Naive Bayes and Softmax Regression demonstrate faster execution, while Kalman-SLDA and SLDA exhibit longer times, with Kalman-SLDA incurring an additional small computational overhead due to the presence of the Kalman filter. The feature extraction step, excluded from the Table 5.1, requires approximately **18 minutes** for the entire dataset, translating to about **0.01 seconds per sample**. This step has a limited impact on the total computation time for the entire simulation and is further discussed in Section 5.2.2.

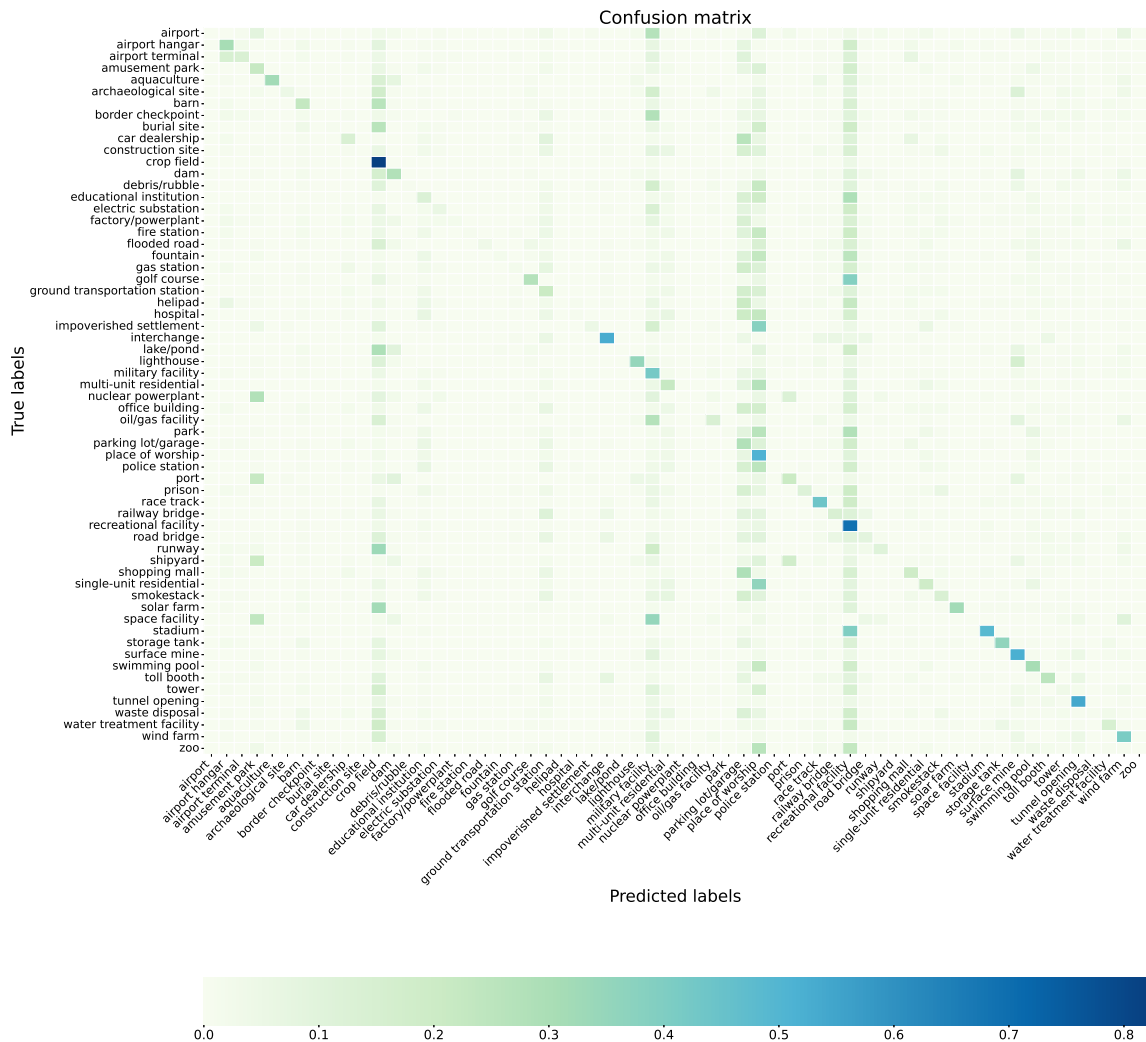


Figure 5.5: Confusion matrix for Softmax Regression.

The classification performance for each model is presented in Table 5.2. Despite being slower, SLDA and Kalman-SLDA achieve superior performance compared to the other methods. In particular, Kalman-SLDA experiences an initial performance drop due to the initialization of Kalman filter parameters. Once the reference values are reached, the model performance keeps improving towards the end of the dataset showing the highest performances between the different methods at the end of the evaluation period. Figures 5.1, 5.2, 5.3, 5.4 present the specific trends observed for the evaluated models across the years. Each figure highlights the performance of its specified model, allowing for a comprehensive comparison over the entire evaluation period.

Softmax Regression notably struggles to deal with class imbalance: the model focuses on the majority classes, in particular "crop field", "recreational facility", "military facility" and "place of worship", as can be noted from Figure 5.5. As shown in Table 4.1, the presented classes appear as the most frequent in the dataset, displaying an undesired result for Softmax Regression: the model puts attention on majority classes, and in a scenario with such label fragmentation and class imbalance, the problem is particularly evident.

Scaler Requirement

The application of a *Streaming MinMaxScaler* significantly improved classifier performance, as illustrated in Figures 5.6 and 5.7 and in Table 5.3. Scaling features in Streaming Machine Learning is essential to mitigate precision-related computational issues that arise when handling floating-point values. This process ensures that the classifier effectively learns from data that maintains consistent numerical ranges across features. This is particularly critical in streaming contexts, where data distributions may evolve over time, potentially impacting the performance and stability of the model.

Type of features	Average Accuracy	Average Balanced Accuracy
Scaled features	0.30	0.28
Non-scaled features	0.21	0.22

Table 5.3: Average classification performance for Kalman-SLDA with and without a preliminary step using a Streaming Scaler.

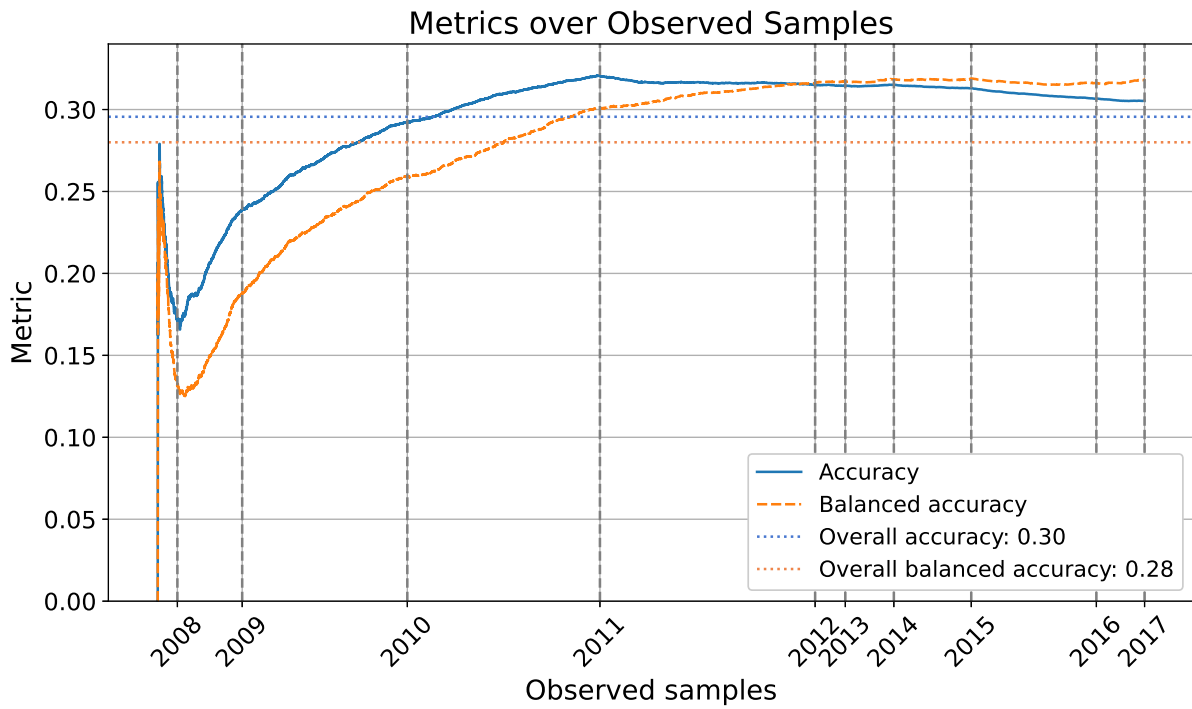


Figure 5.6: Classification performance for Kalman-SLDA using MinMaxScaler.

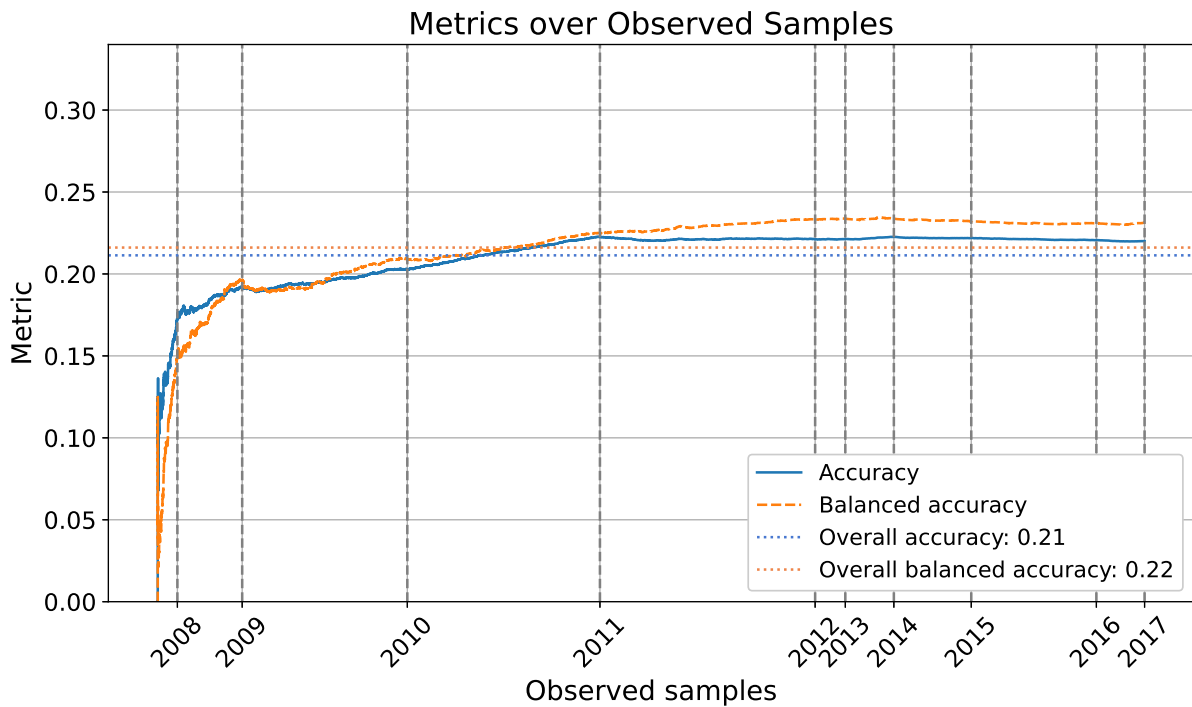


Figure 5.7: Classification performance for Kalman-SLDA without scaling features.

5.2.2. Comparison of CNNs

To validate the consistency of the results obtained in the previous section, the same pipeline is executed using features extracted by ResNet18. This procedure ensures that the conclusions drawn in the prior section are not dependent on the specific features extracted by MobileNet v3 Small. The neural networks present differing parameter counts and output dimensionalities. MobileNet v3 Small contains 927,008 parameters, producing feature vectors of dimension 576, while ResNet18 has 11,176,512 parameters and a feature vector dimension of 512. This difference in parameter count affects the execution time for feature extraction, as shown in Table 5.4. The parameter counts exclude those in the final dense layers, which are used exclusively for classification within the neural network and are not utilized for feature extraction.

ResNet18, with approximately eleven times the parameters of MobileNet v3 Small, requires roughly triple the time to extract features from a given image. Despite this, classification speed and accuracy show comparable trends when images are classified using features extracted from either network. Although MobileNet v3 Small’s higher-dimensional feature extraction results in a slightly longer classification time per image, the overall elapsed time, factoring in both feature extraction and classification, remains similar across

CNN	Total Time (minutes)	Total Time (seconds)	Time per Sample (seconds)
MobileNet v3 Small	18	1098	0.01
ResNet18	55	3290	0.03

Table 5.4: Elapsed time to extract features using different Convolutional Neural Networks.

CNN	Kalman-SLDA	SLDA	Gaussian Naive Bayes	Softmax Regression
MobileNet v3 Small	0.07	0.07	0.05	0.04
ResNet18	0.05	0.06	0.04	0.03

Table 5.5: Time in seconds per sample required to classify an image after feature extraction using each specified Convolutional Neural Network.

CNN	Kalman-SLDA	SLDA	Gaussian Naive Bayes	Softmax Regression
MobileNet v3 Small	0.08	0.08	0.06	0.05
ResNet18	0.08	0.09	0.07	0.06

Table 5.6: Time in seconds per sample required to execute the full classification pipeline using different Convolutional Neural Networks and Streaming Machine Learning classifiers.

different pipelines, as shown in Table 5.6. In terms of overall processing time, MobileNet v3 Small is preferred.

Moving to classification accuracy, the key results are presented in Figures 5.8, 5.9 and Table 5.7. Table 5.7 demonstrates that the choice of neural network has minimal impact on model performance. Since Kalman-SLDA and SLDA leverages the curse of dimensionality as described in Section 2.2.3, larger feature vectors provide a slight advantage in accu-

CNN	Classifier	Avg Accuracy	Avg Balanced Accuracy
MobileNet v3 Small	Kalman-SLDA	0.30	0.28
	SLDA	0.25	0.27
	GNB	0.21	0.22
	SMR	0.25	0.13
ResNet18	Kalman-SLDA	0.28	0.28
	SLDA	0.24	0.25
	GNB	0.21	0.22
	SMR	0.24	0.12

Table 5.7: Average accuracy and balanced accuracy for different SML classifiers with features extracted from different CNNs.

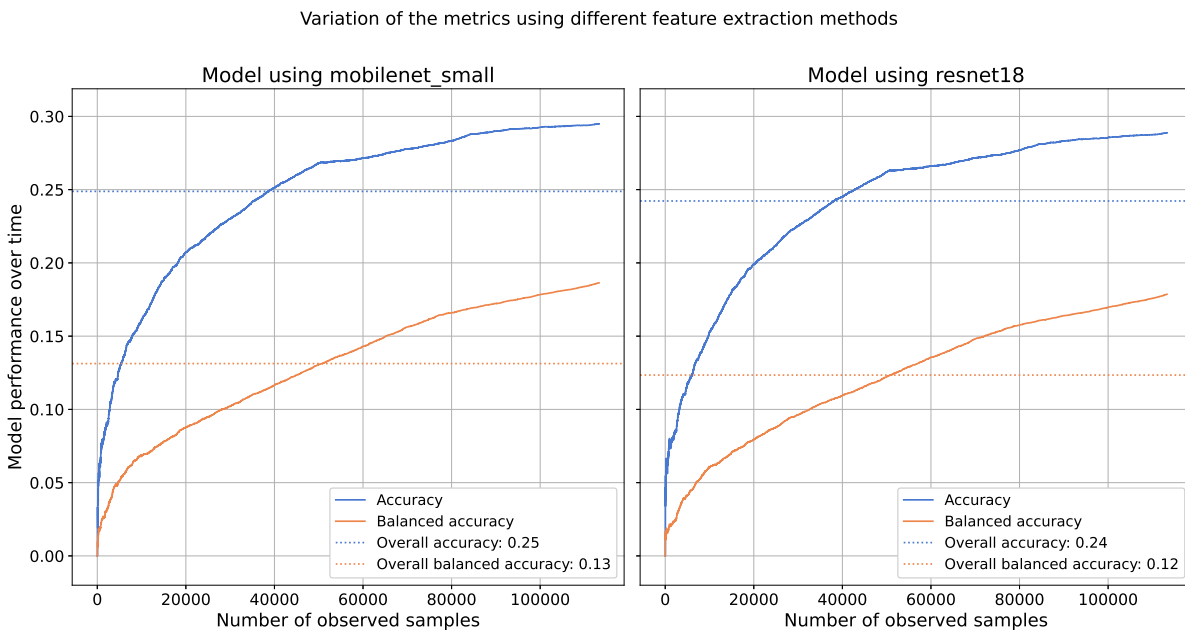


Figure 5.8: Classification performance for Softmax Regression with features from different CNNs.

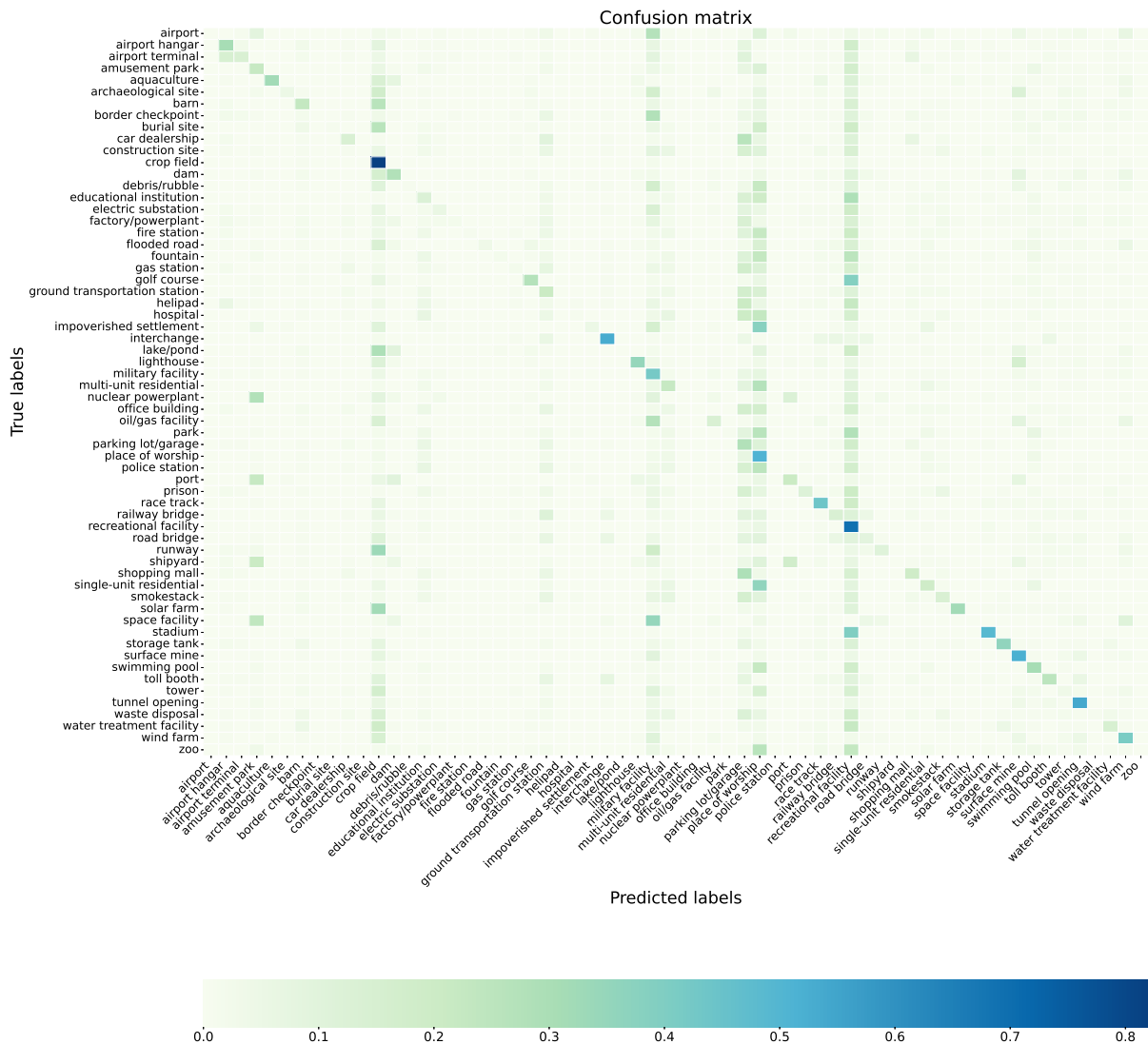


Figure 5.9: Confusion matrix for Softmax Regression with features from ResNet18.

racy; however, the difference is minimal. In addition, Figure 5.8 indicates that Softmax Regression’s performance issues with class imbalance persist across different CNNs. The model’s focus on certain classes remains consistent by comparing the confusion matrices in Figures 5.5 and 5.9.

5.2.3. Comparison of Dimensionality Reduction Methods

Dimensionality reduction techniques were applied to manage the high-dimensional feature space, aiming to improve computational efficiency and increase the number of tests to

determine the optimal balance between execution time and classification performance. This section presents the results for Sparse Random Projection and Batch-UMAP, with a brief discussion of Streaming-UMAP due to computational constraints.

Sparse Random Projection

Sparse Random Projection is a linear dimensionality reduction technique that transforms high-dimensional data via matrix-vector multiplication. Due to its structure, this method scales linearly in time with the number of dimensions in the reduced embedding. Execution times for Sparse Random Projection are illustrated in Figure 5.10. The Convolutional Neural Networks used for feature extraction, yielding projection matrices of differing sizes for dimensionality reduction, show similar reduction times despite matrix size variations.

Execution times for classification vary according to the dimensionality reduction level, as shown in Figure 5.11. SLDA and Kalman-SLDA show similar scaling patterns, while Gaussian Naive Bayes and Softmax Regression exhibit distinct but comparable behavior

Number of components	Kalman-SLDA	SLDA	Gaussian Naive Bayes	Softmax Regression
30 components	0.02	0.02	0.01	0.01
60 components	0.02	0.02	0.02	0.02
90 components	0.02	0.02	0.02	0.02
120 components	0.02	0.02	0.02	0.02
150 components	0.03	0.03	0.03	0.03
180 components	0.03	0.03	0.03	0.03
210 components	0.03	0.03	0.03	0.03
240 components	0.03	0.03	0.04	0.04
270 components	0.04	0.04	0.04	0.04
300 components	0.04	0.04	0.05	0.04
330 components	0.05	0.05	0.05	0.05
360 components	0.05	0.05	0.05	0.05
390 components	0.05	0.06	0.05	0.05
420 components	0.06	0.06	0.06	0.05
450 components	0.07	0.07	0.06	0.06
480 components	0.07	0.07	0.06	0.06
510 components	0.08	0.08	0.07	0.06
Baseline (576 components)	0.08	0.08	0.06	0.05

Table 5.8: Time in seconds per sample required to execute the entire classification pipeline with different Streaming Machine Learning classifiers after extracting the features with MobileNet v3 Small and reducing them through Sparse Random Projection. Blue lines mark the last feature size for which the execution time of the specific model is convenient compared to the baseline.

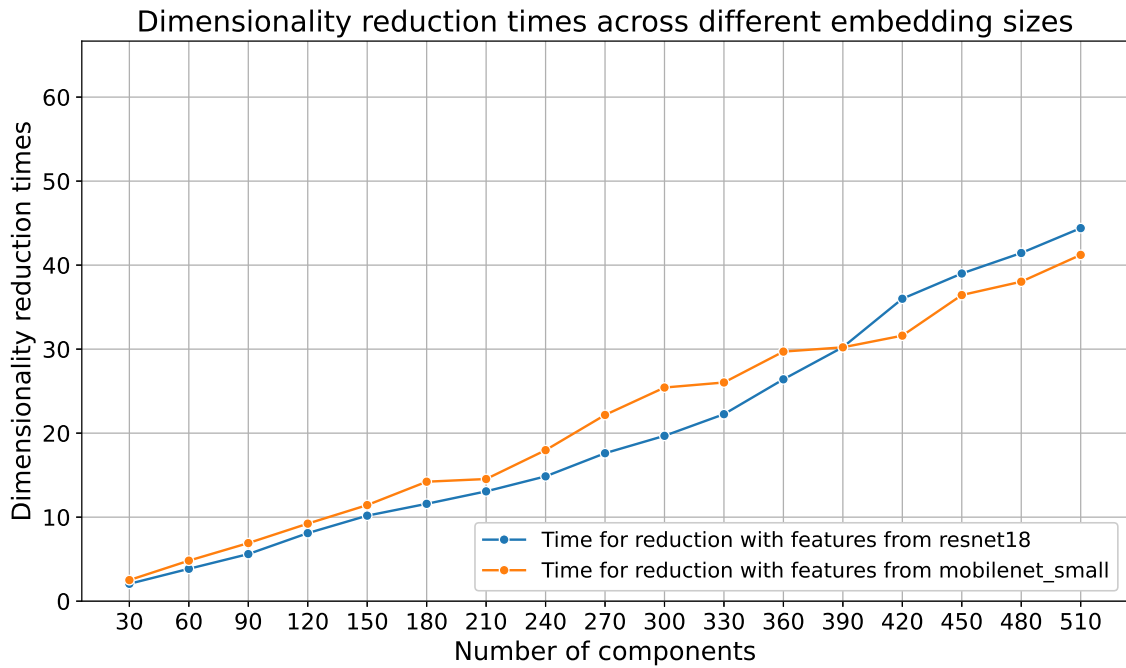


Figure 5.10: Execution time to reduce the feature set to the desired number of components for varying initial feature space shapes.

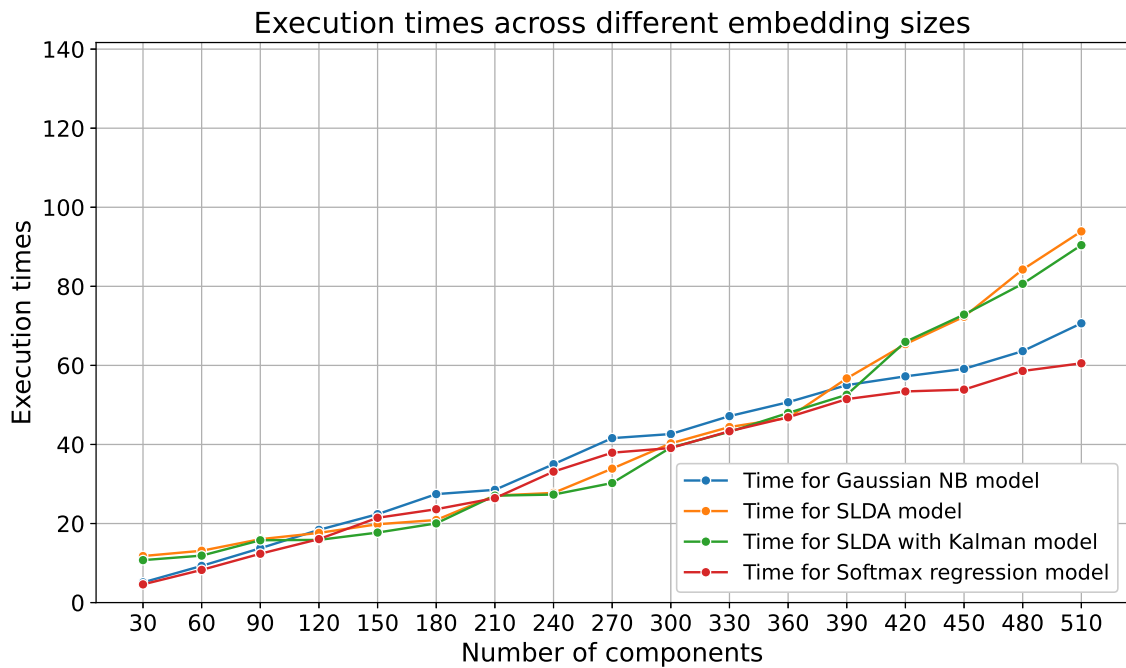


Figure 5.11: Execution time to classify the stream after reducing the feature set through Sparse Random Projection using features extracted from MobileNet v3 Small.

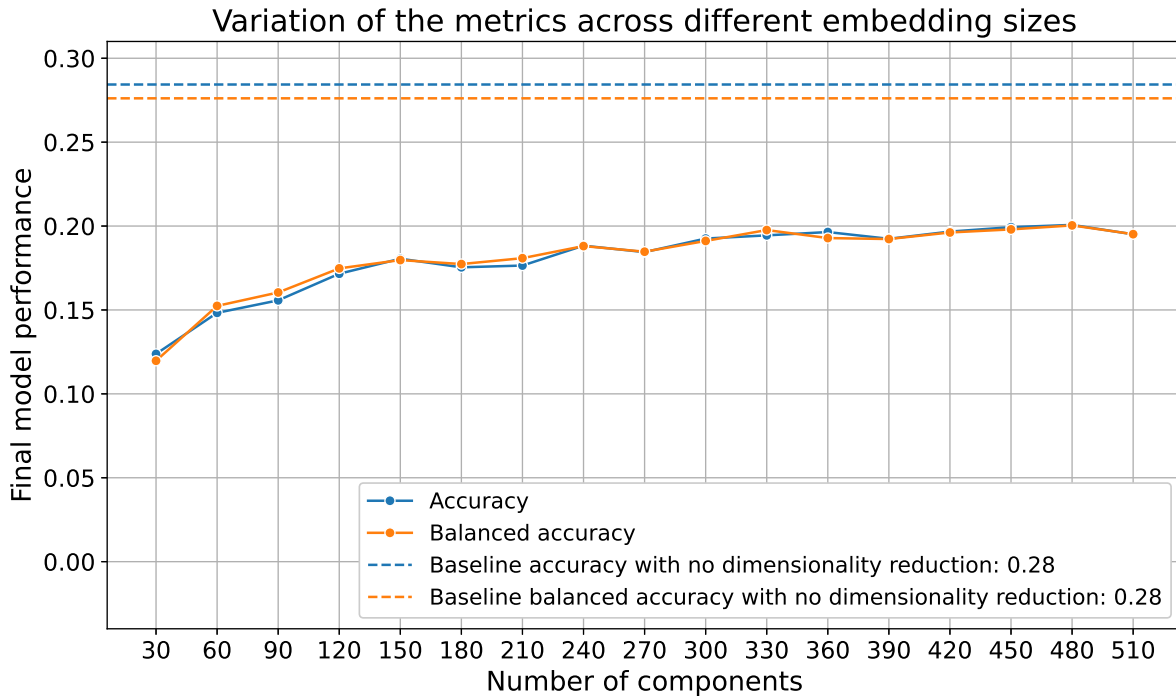


Figure 5.12: Classification performance of Kalman-SLDA classifier after extracting features through ResNet18 and reducing them through Sparse Random Projection to different sizes.

to one another. Although SLDA and Kalman-SLDA show increased processing times beyond 400 features, execution time alone should not be the only metric considered for result evaluation. Table 5.8 provides a more comprehensive view, displaying time per sample across different embedding sizes and classifiers. The same trends are observed for features extracted from ResNet18.

The main conclusion is that applying no dimensionality reduction is preferable in terms of execution times above 480 components for SLDA and Kalman-SLDA, while Gaussian Naive Bayes becomes inefficient beyond 400 components. For Softmax Regression, it is better to use the original features if the dimensionality exceeds 300.

Moving to the performance analysis, an example of empirical results obtained through various simulations is depicted in Figure 5.12. It shows the classification performance of Kalman-SLDA when starting from embeddings of varying sizes obtained through Sparse Random Projection. Performance gradually increases up to approximately 330 components, then settles, without reaching the baseline performance with no dimensionality reduction. This suggests that Sparse Random Projection cannot retain all the relevant

Number of components	Kalman-SLDA	SLDA	Gaussian Naive Bayes	Softmax Regression
30 components	0.13	0.10	0.14	0.03
60 components	0.16	0.13	0.19	0.04
90 components	0.17	0.14	0.19	0.05
120 components	0.17	0.15	0.21	0.06
150 components	0.18	0.16	0.21	0.07
180 components	0.19	0.17	0.22	0.07
210 components	0.19	0.17	0.22	0.08
240 components	0.19	0.18	0.23	0.09
270 components	0.19	0.18	0.23	0.09
300 components	0.19	0.18	0.23	0.09
330 components	0.19	0.17	0.23	0.09
360 components	0.19	0.18	0.23	0.10
390 components	0.20	0.18	0.23	0.10
420 components	0.20	0.18	0.23	0.10
450 components	0.19	0.17	0.23	0.11
480 components	0.20	0.18	0.23	0.11
510 components	0.20	0.18	0.23	0.11
Baseline (576 components)	0.28	0.27	0.22	0.13

Table 5.9: Average balanced accuracy of different classifiers after extracting features through MobileNet v3 Small and reducing them via Sparse Random Projection. Blue lines mark the feature size for which the classification performance does no longer improve.

information even with minimal reduction. For instance, reducing dimensionality from the original 512 features to 510 causes a drop in performance from 0.28 to nearly 0.20. Further results will be discussed in Section 5.2.4.

In conclusion, Sparse Random Projection proves to be a simple yet effective dimensionality reduction method to generate reduced-size feature spaces. It displays some classification performance losses, particularly above the critical threshold of 330 features. Nonetheless, the time saved in processing is substantial, making it optimal to reduce feature sizes to around 250 components in this scenario, as this achieves a favorable time-performance trade-off. The conclusions related to the performance are supported by Table 5.9.

Batch-UMAP

Batch-UMAP is a non-linear dimensionality reduction method developed within this thesis as an extension of the Uniform Manifold Approximation and Projection (UMAP) [26], which currently stands as a state-of-the-art offline dimensionality reduction technique. The objective of this implementation is to introduce a non-linear methodology capable

of potentially outperforming existing models by leveraging data similarity. Therefore, a comparison between Batch-UMAP and Sparse Random Projection is essential to evaluate the validity of this approach.

The core concept of Batch-UMAP is to adapt UMAP for incremental processing, applying it in a batch-wise manner. Specifically, each yearly batch of samples is reduced to a low-dimensional space, relying on the embedding of samples from the previous year to maintain temporal coherence. This iterative approach, while theoretically continuous, necessitates a deviation from a completely streaming environment: the dimensionality reduction for each batch can only proceed after the entire batch of samples is observed, a requirement central to the method’s functionality.

The first notable outcome, as illustrated in Figure 5.13, is that Batch-UMAP’s execution time remains approximately constant across various target dimensionalities and is not highly dependent on the dimensionality of the initial dataset. However, as shown in Figure 5.14, its execution time is consistently higher than Sparse Random Projection at all tested dimensionalities, implying an unsatisfactory result in terms of computational complexity. Similarly to Sparse Random Projection, Batch-UMAP influences the time required for classification, as depicted in Figure 5.15. Differently from models shown in

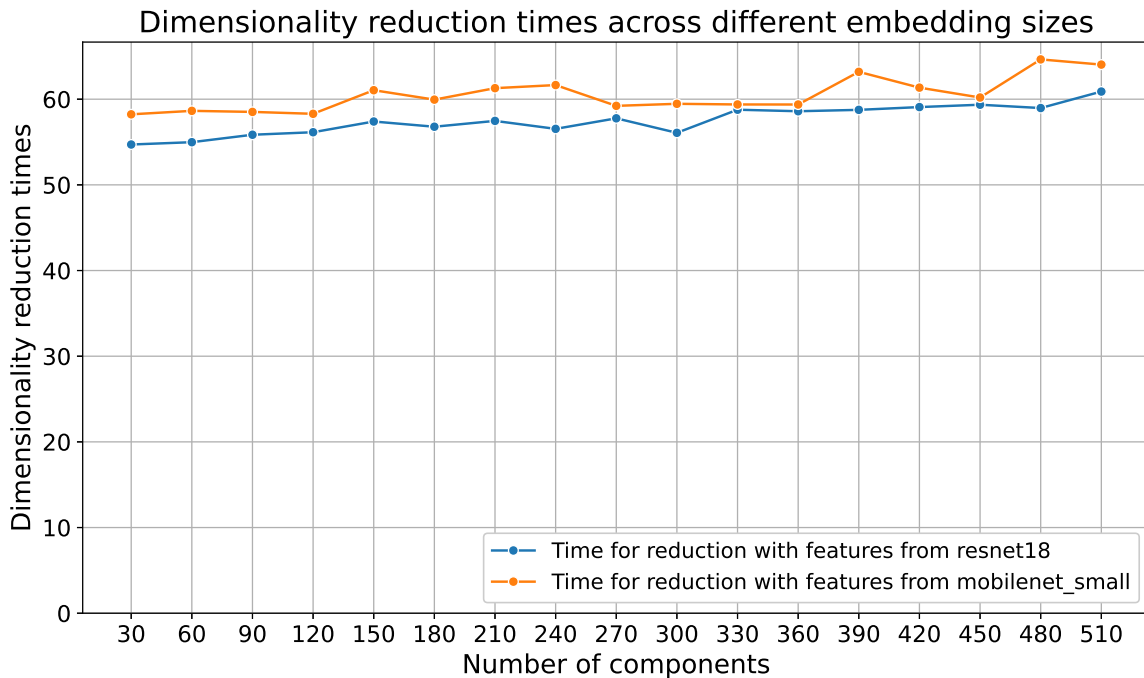


Figure 5.13: Execution time required to reduce the feature set to the specified number of components across varying initial feature space dimensions.

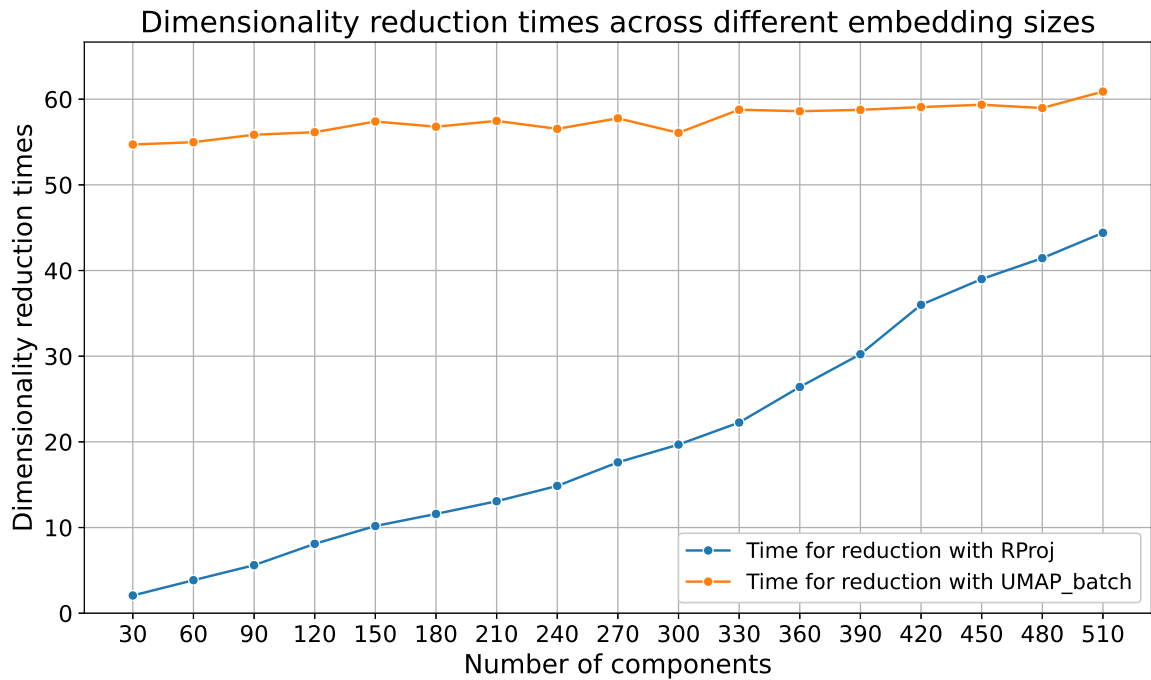


Figure 5.14: Execution time required to reduce the feature set to the desired number of components across different dimensionality reduction techniques.

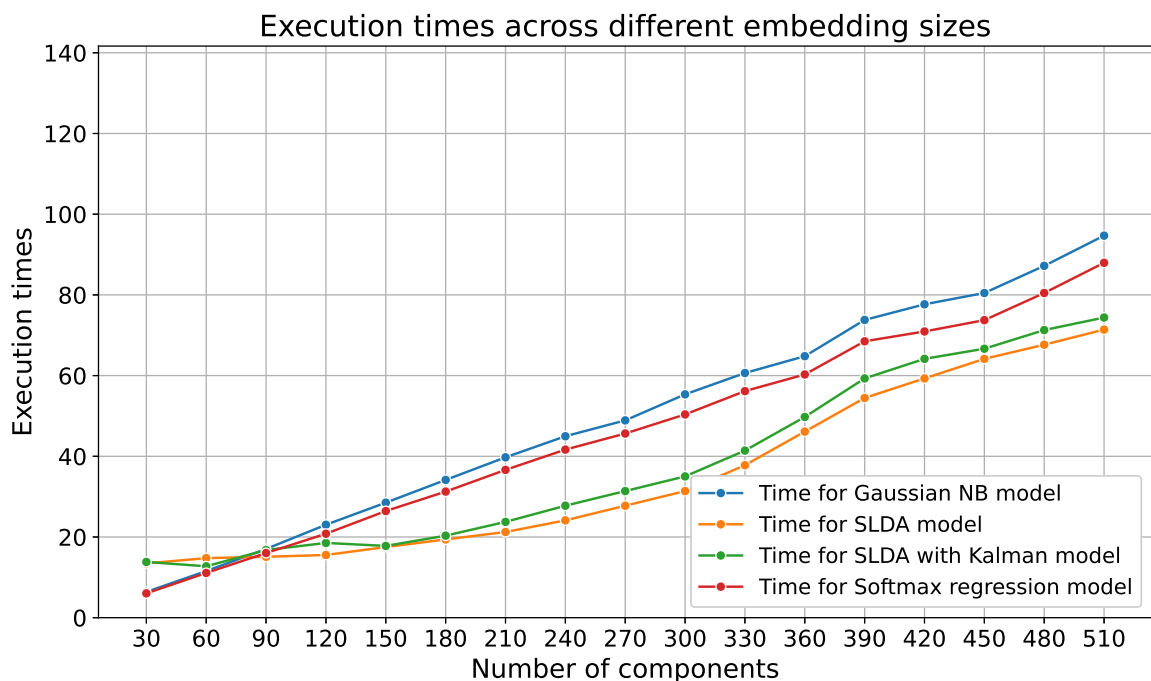


Figure 5.15: Execution time to classify the stream after reducing the feature set through Batch-UMAP using features extracted through MobileNet v3 Small.

Number of components	Kalman-SLDA	SLDA	Gaussian Naive Bayes	Softmax Regression
30 components	0.05	0.05	0.04	0.04
60 components	0.05	0.05	0.05	0.05
90 components	0.05	0.05	0.05	0.05
120 components	0.05	0.05	0.05	0.05
150 components	0.05	0.05	0.06	0.06
180 components	0.05	0.05	0.06	0.06
210 components	0.05	0.05	0.06	0.06
240 components	0.06	0.06	0.07	0.06
270 components	0.06	0.06	0.07	0.07
300 components	0.06	0.06	0.07	0.07
330 components	0.06	0.06	0.07	0.07
360 components	0.07	0.07	0.08	0.07
390 components	0.07	0.07	0.08	0.08
420 components	0.08	0.07	0.08	0.08
450 components	0.08	0.08	0.08	0.08
480 components	0.08	0.08	0.09	0.09
510 components	0.08	0.08	0.09	0.09
Baseline (576 components)	0.08	0.08	0.06	0.05

Table 5.10: Time per sample in seconds to execute the classification pipeline with different Streaming Machine Learning classifiers after feature extraction using MobileNet v3 Small and reduction through Batch-UMAP across different component sizes.

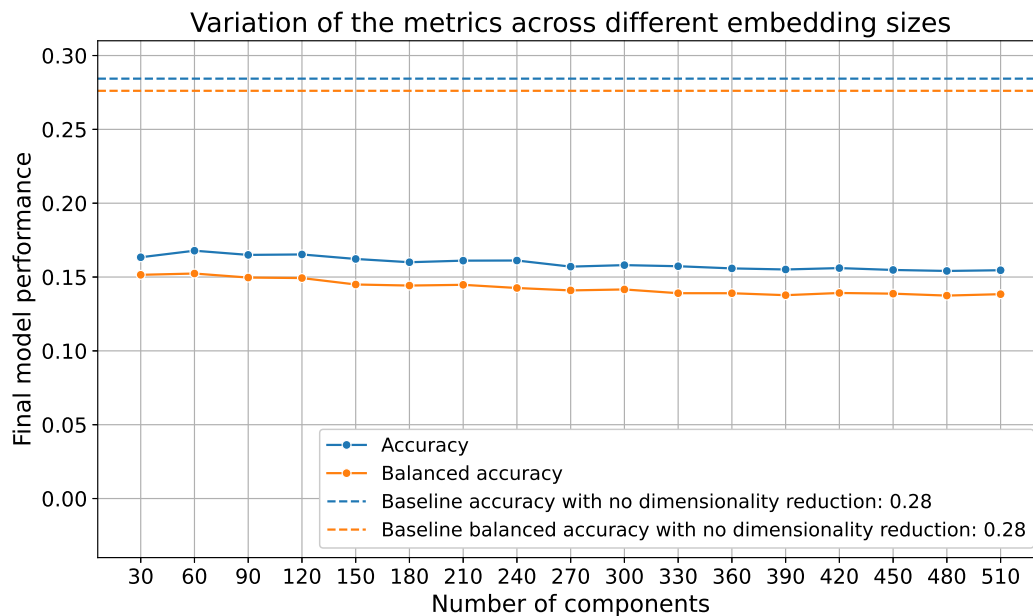


Figure 5.16: Classification performance of the Kalman-SLDA method after feature extraction through ResNet18 and reduction with Batch-UMAP across different component sizes.

Number of components	Kalman-SLDA	SLDA	Gaussian Naive Bayes	Softmax Regression
30 components	0.15	0.15	0.15	0.05
60 components	0.15	0.11	0.16	0.06
90 components	0.15	0.07	0.16	0.07
120 components	0.15	0.07	0.17	0.07
150 components	0.14	0.07	0.16	0.08
180 components	0.14	0.07	0.16	0.08
210 components	0.14	0.06	0.17	0.08
240 components	0.14	0.06	0.16	0.09
270 components	0.14	0.06	0.16	0.09
300 components	0.14	0.05	0.17	0.09
330 components	0.14	0.05	0.16	0.09
360 components	0.14	0.06	0.17	0.09
390 components	0.14	0.05	0.17	0.09
420 components	0.14	0.05	0.16	0.09
450 components	0.14	0.05	0.16	0.09
480 components	0.14	0.05	0.17	0.09
510 components	0.14	0.06	0.16	0.09
Baseline (576 components)	0.28	0.27	0.22	0.13

Table 5.11: Average balanced accuracy of different classifiers after extracting features through MobileNet v3 Small and reducing them via Batch-UMAP. Blue lines mark the feature size for which the classification performance does no longer improve.

Figure 5.11, SLDA and Kalman-SLDA exhibit the fastest performance, while Gaussian Naive Bayes is now the slowest. Table 5.10 extends these results to reflect the entire pipeline with MobileNet v3 Small feature extraction.

In conclusion, applying no dimensionality reduction is preferable when the component count exceeds approximately 400 with SLDA and Kalman-SLDA, while Softmax Regression require more processing time than without dimensionality reduction as early as 60 components. Gaussian Naive Bayes requires less time than the baseline with at most 120 features.

In terms of classification performance, a comparison across embedding sizes is illustrated in Figure 5.16 for Kalman-SLDA and in Table 5.11 for all the proposed classifiers. The figure shows a gradual decline in performance as the dimensionality increases, with Accuracy and Balanced Accuracy stabilizing around 0.14 for Kalman-SLDA, a trend that differs from Sparse Random Projection. Consequently, Batch-UMAP does not currently outperform existing methods and may require further refinement. This method effectively captures data similarity and is a promising alternative compared to the state-of-the-art methods,

Year	Number of Samples
2002	1676
2003	2279
2004	1755
2005	2512
2006	3155
2007	1497
2008	2261
2009	7439
2010	18957
2011	22111
2012	24704
2013	3465
2014	5572
2015	8885
2016	14363
2017	5534

Table 5.12: Distribution of samples per year of observation. Years highlighted in orange are used for model initialization, while those in blue are incrementally fit through Batch-UMAP.

but challenges arise when batches differ significantly in size. For instance, as detailed in Table 5.12, smaller sample sizes in years such as 2009 and 2015 complicate the embedding process for the subsequent year.

In conclusion, the current implementation of Batch-UMAP encounters challenges in surpassing the state-of-the-art Sparse Random Projection for dimensionality reduction. While the design effectively incorporates data similarity, the computational overhead limits its suitability for real-time settings, and performance outcomes remain suboptimal. As it stands, Batch-UMAP achieves its best trade-off by reducing the dataset to the minimum component count considered (30 features), maintaining comparable performance to higher dimensions while reducing processing time.

Streaming-UMAP

Attempts to apply Streaming-UMAP encountered significant computational limitations within the current experimental setup, primarily due to the memory constraints and the

processing power required for real-time updates. The graph-based nature of UMAP, particularly during the construction and neighbor-finding steps, demands substantial memory resources, which posed challenges as the dataset's dimensionality and volume increased. Given the hardware limitations the streaming approach could not be tested under realistic conditions. Additionally, the dynamic adjustment of embeddings in real-time required processing power that exceeded the available computational resources. These limitations prevented the effective application of Streaming-UMAP in the experimental setup, highlighting the need for specialized hardware or optimizations to handle such tasks efficiently.

5.2.4. Detailed Analysis of the Complete Pipeline

The application of dimensionality reduction led to a significant decrease in execution time across all classifiers, though its impact on classification accuracy varied by model. This section provides an in-depth comparison of classifier performance, focusing on both accuracy and execution time across different dimensionalities. Visualizations of the trade-offs between time efficiency and accuracy further illustrate these findings.

Softmax Regression

The initial analysis focuses on Softmax Regression classifier. As discussed in Section 5.2.1, this classifier faces challenges when handling class imbalance and high label fragmentation. These limitations persist with reduced features, as performance gradually improves towards baseline levels (with no dimensionality reduction) yet keeping an evident difference between Accuracy and Balanced Accuracy, as shown in Figure 5.17. Results are consistent across different feature extractors and dimensionality reduction techniques, with further details provided in Table 5.13 and Appendix A.4. Execution times per sample for the entire pipeline are summarized and compared in Table 5.14.

Features extracted using MobileNet V3 Small and reduced through Sparse Random Projection offer the best performance in terms of execution time at any dimensionality. Execution time and the performance metrics increase monotonically with the number of features, leading to the following recommended pipeline to balance the efficiency-accuracy trade-off:

- Extract features using MobileNet V3 Small.
- Reduce features via Sparse Random Projection to a size of approximately 250 or avoid dimensionality reduction to prioritize accuracy.
- Apply Softmax Regression.

Number of components	MobileNet + SRP	ResNet + SRP	MobileNet + Batch-UMAP	ResNet + Batch-UMAP
30 components	0.03	0.03	0.05	0.04
60 components	0.04	0.04	0.06	0.06
90 components	0.05	0.05	0.07	0.06
120 components	0.06	0.05	0.07	0.07
150 components	0.07	0.06	0.08	0.08
180 components	0.07	0.07	0.08	0.08
210 components	0.08	0.07	0.08	0.08
240 components	0.09	0.08	0.09	0.08
270 components	0.09	0.08	0.09	0.08
300 components	0.09	0.08	0.09	0.09
330 components	0.09	0.09	0.09	0.09
360 components	0.10	0.09	0.09	0.09
390 components	0.10	0.09	0.09	0.09
420 components	0.10	0.09	0.09	0.09
450 components	0.11	0.10	0.09	0.09
480 components	0.11	0.10	0.09	0.09
510 components	0.11	0.10	0.09	0.09
Baseline (no dim. reduction)	0.13	0.12	0.13	0.12

Table 5.13: Average balanced accuracy of Softmax Regression after feature extraction and reduction using different methods across different component sizes. Blue lines mark the feature size for which the classification performance does no longer improve.

Number of components	MobileNet + SRP	ResNet + SRP	MobileNet + Batch-UMAP	ResNet + Batch-UMAP
30 components	0.01	0.03	0.04	0.06
60 components	0.02	0.04	0.05	0.06
90 components	0.02	0.04	0.05	0.07
120 components	0.02	0.04	0.05	0.07
150 components	0.03	0.04	0.05	0.07
180 components	0.03	0.05	0.06	0.08
210 components	0.03	0.05	0.06	0.08
240 components	0.04	0.05	0.06	0.08
270 components	0.04	0.06	0.07	0.08
300 components	0.04	0.06	0.07	0.09
330 components	0.05	0.06	0.07	0.09
360 components	0.05	0.07	0.07	0.09
390 components	0.05	0.08	0.08	0.1
420 components	0.05	0.08	0.08	0.1
450 components	0.06	0.08	0.08	0.1
480 components	0.06	0.09	0.09	0.1
510 components	0.06	0.09	0.09	0.11
Baseline (no dim. reduction)	0.05	0.06	0.05	0.06

Table 5.14: Time per sample in seconds to execute the classification pipeline for Softmax Regression classifier after feature extraction and reduction using different methods across different component sizes.

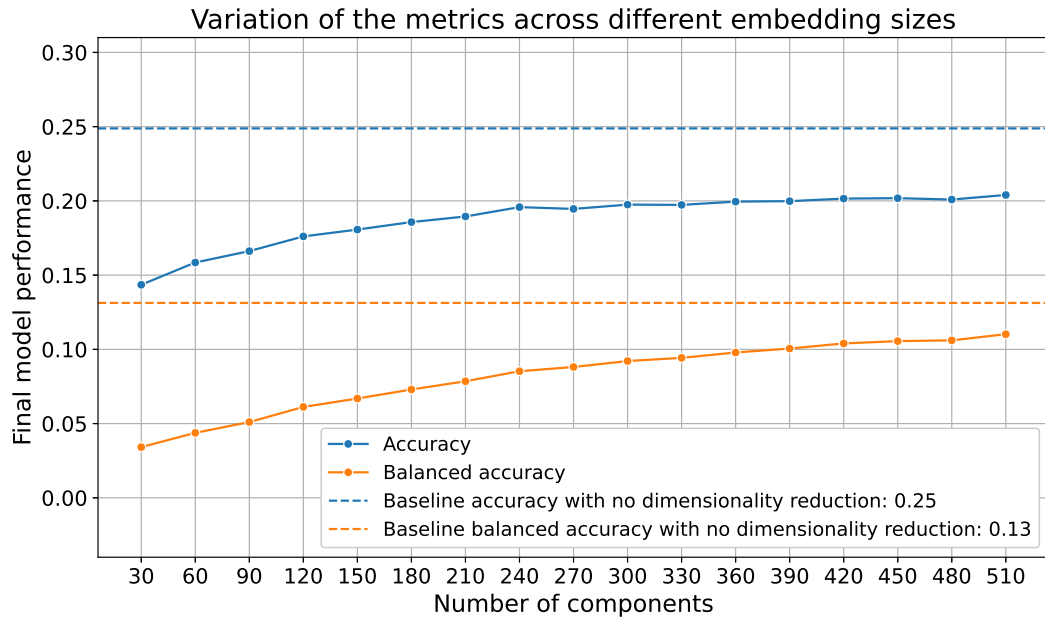


Figure 5.17: Classification performance of Softmax Regression method after feature extraction via MobileNet V3 Small and reduction using Sparse Random Projection across various component sizes.

SLDA

Turning to SLDA, the results at various dimensionalities using Sparse Random Projection are presented in Figure 5.18. Similar to Softmax Regression, accuracy increases with the number of dimensions, yet the model does not reach the baseline performance obtained without dimensionality reduction. This suggests that Sparse Random Projection, while computationally efficient, fails to preserve all information within the reduced feature space, resulting in some degree of information loss. The same trend is observed with features extracted by ResNet18, as shown in Appendix A.2.

For features reduced using Batch-UMAP, a different trend emerges, as shown in Figure 5.19. Accuracy declines to approximately 0.06 after initially reaching 0.15 at 30 dimensions. This suggests that the proposed algorithm may not effectively enhance SLDA performance, despite showing a slight improvement at 30 dimensions compared to Sparse Random Projection. The comparison of different pipelines in terms of performance is presented in Table 5.15, confirming the degradation due to Batch-UMAP.

Dimensionality reduction has a more substantial impact on execution time compared to Softmax Regression classifier, as displayed in Table 5.16. Reducing features to fewer than 300 components with Sparse Random Projection allows the model to execute in less than

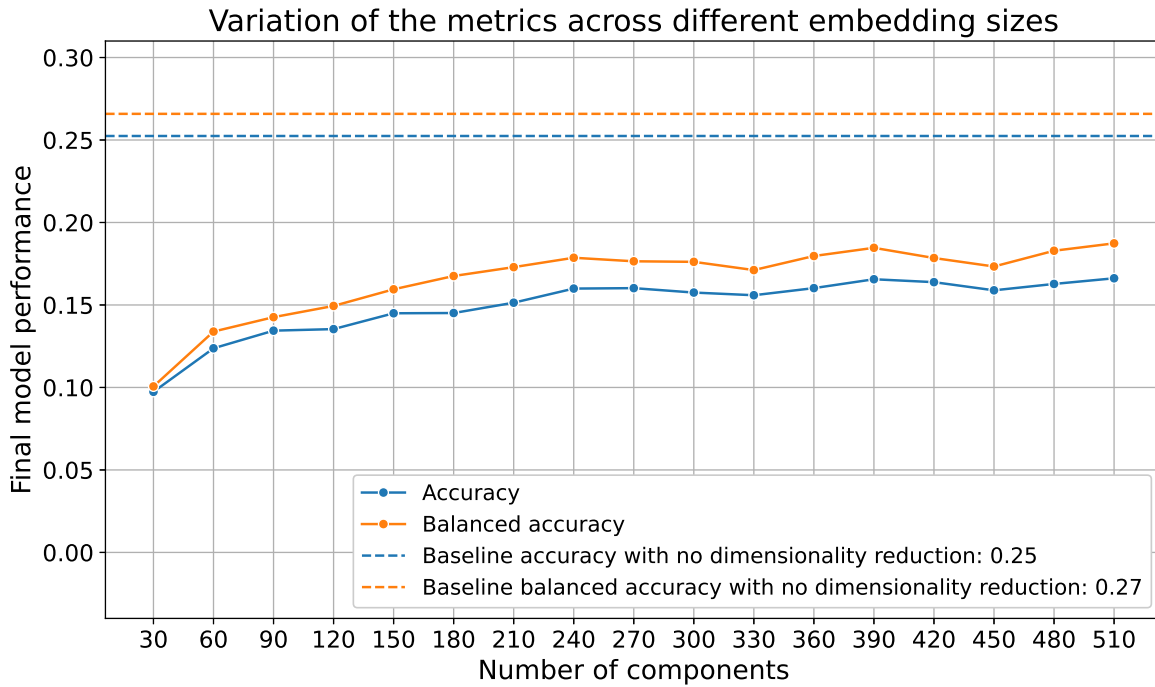


Figure 5.18: Classification performance of SLDA method after feature extraction via MobileNet V3 Small and reduction through Sparse Random Projection across various component sizes.

Number of components	MobileNet + SRP	ResNet + SRP	MobileNet + Batch-UMAP	ResNet + Batch-UMAP
30 components	0.10	0.10	0.15	0.14
60 components	0.13	0.13	0.11	0.12
90 components	0.14	0.14	0.07	0.09
120 components	0.15	0.16	0.07	0.07
150 components	0.16	0.17	0.07	0.06
180 components	0.17	0.16	0.07	0.06
210 components	0.17	0.17	0.06	0.06
240 components	0.18	0.17	0.06	0.05
270 components	0.18	0.17	0.06	0.05
300 components	0.18	0.18	0.05	0.05
330 components	0.17	0.19	0.05	0.05
360 components	0.18	0.18	0.06	0.05
390 components	0.18	0.19	0.05	0.04
420 components	0.18	0.19	0.05	0.05
450 components	0.17	0.19	0.05	0.04
480 components	0.18	0.19	0.05	0.05
510 components	0.18	0.19	0.06	0.04
Baseline (no dim. reduction)	0.27	0.25	0.27	0.25

Table 5.15: Average balanced accuracy of SLDA after feature extraction and reduction using different methods across different component sizes. Blue lines mark the feature size for which the classification performance does no longer improve.

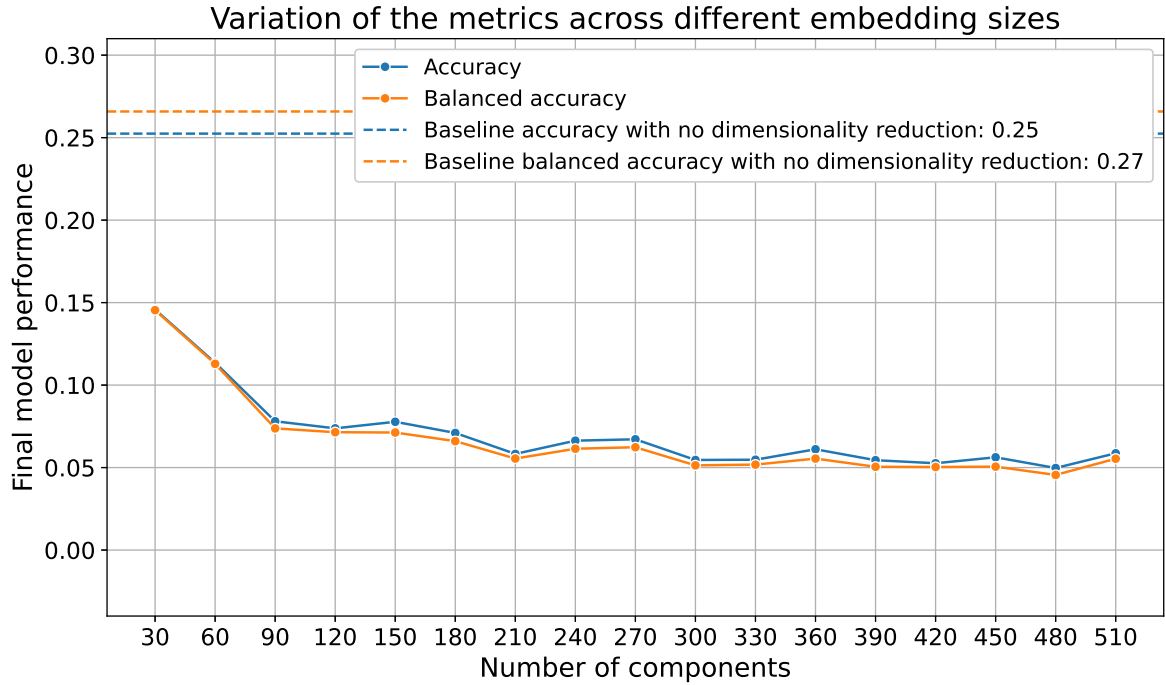


Figure 5.19: Classification performance of SLDA method after feature extraction via MobileNet V3 Small and reduction with Batch-UMAP across various component sizes.

Number of components	MobileNet + SRP	ResNet + SRP	MobileNet + Batch-UMAP	ResNet + Batch-UMAP
30 components	0.02	0.04	0.05	0.06
60 components	0.02	0.04	0.05	0.06
90 components	0.02	0.04	0.05	0.06
120 components	0.02	0.04	0.05	0.06
150 components	0.03	0.04	0.05	0.07
180 components	0.03	0.05	0.05	0.07
210 components	0.03	0.05	0.05	0.07
240 components	0.03	0.05	0.06	0.07
270 components	0.04	0.06	0.06	0.08
300 components	0.04	0.06	0.06	0.08
330 components	0.05	0.07	0.07	0.08
360 components	0.05	0.07	0.07	0.08
390 components	0.06	0.08	0.07	0.09
420 components	0.06	0.08	0.07	0.09
450 components	0.07	0.09	0.08	0.09
480 components	0.07	0.09	0.08	0.1
510 components	0.08	0.1	0.08	0.1
Baseline (no dim. reduction)	0.08	0.09	0.08	0.09

Table 5.16: Time per sample in seconds to execute the classification pipeline for SLDA classifier after feature extraction and reduction using different methods across different component sizes.

half the time required for the baseline MobileNet V3 Small performance.

The recommended pipeline, balancing accuracy and computational efficiency, is as follows:

- Extract features using MobileNet V3 Small, this CNN offers the best trade-off between accuracy and efficiency.
- Apply Sparse Random Projection to reduce dimensionality to approximately 250 features, or omit dimensionality reduction to prioritize accuracy.
- Use SLDA for classification.

Kalman-SLDA

Kalman-SLDA exhibits a similar behavior to SLDA when features are reduced using Sparse Random Projection. The addition of Kalman filtering improves performance up to approximately 240 features but does not reach the baseline version without dimensionality reduction. The Kalman-SLDA adaptation surpasses the performance of SLDA at all dimensionalities, demonstrating an overall improvement, as shown in Figure 5.20. Results for features extracted with ResNet18 confirm this pattern, as depicted in Appendix A.1.

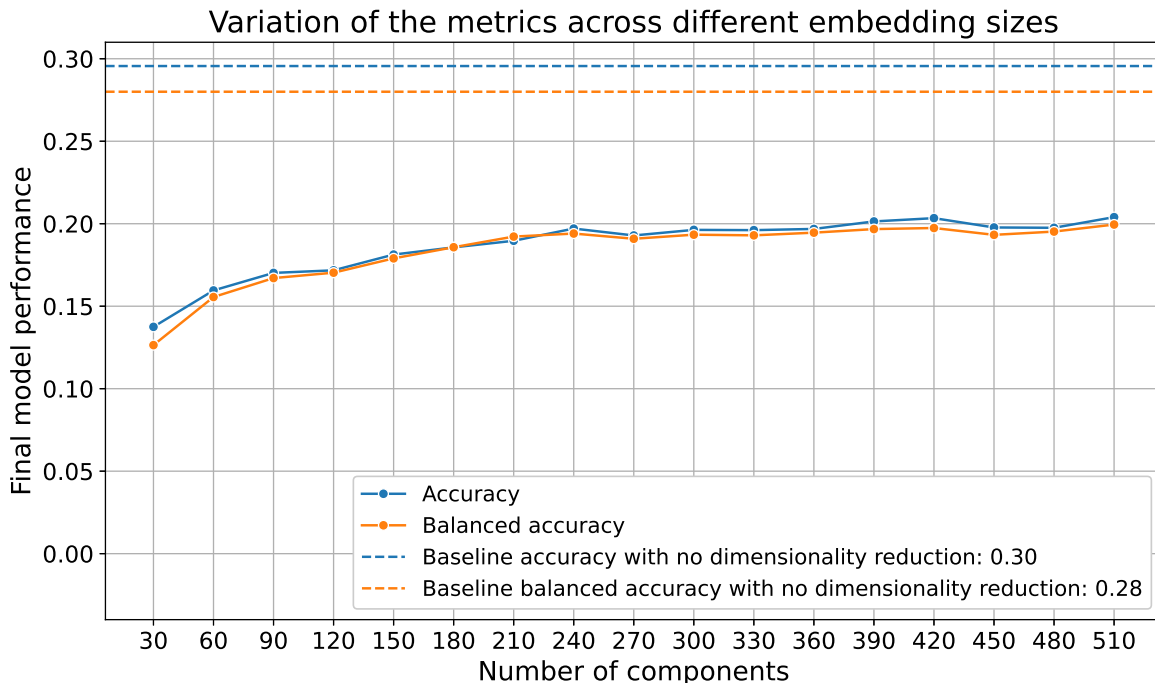


Figure 5.20: Classification performance of Kalman-SLDA method after feature extraction via MobileNet V3 Small and reduction with Sparse Random Projection across various component sizes.

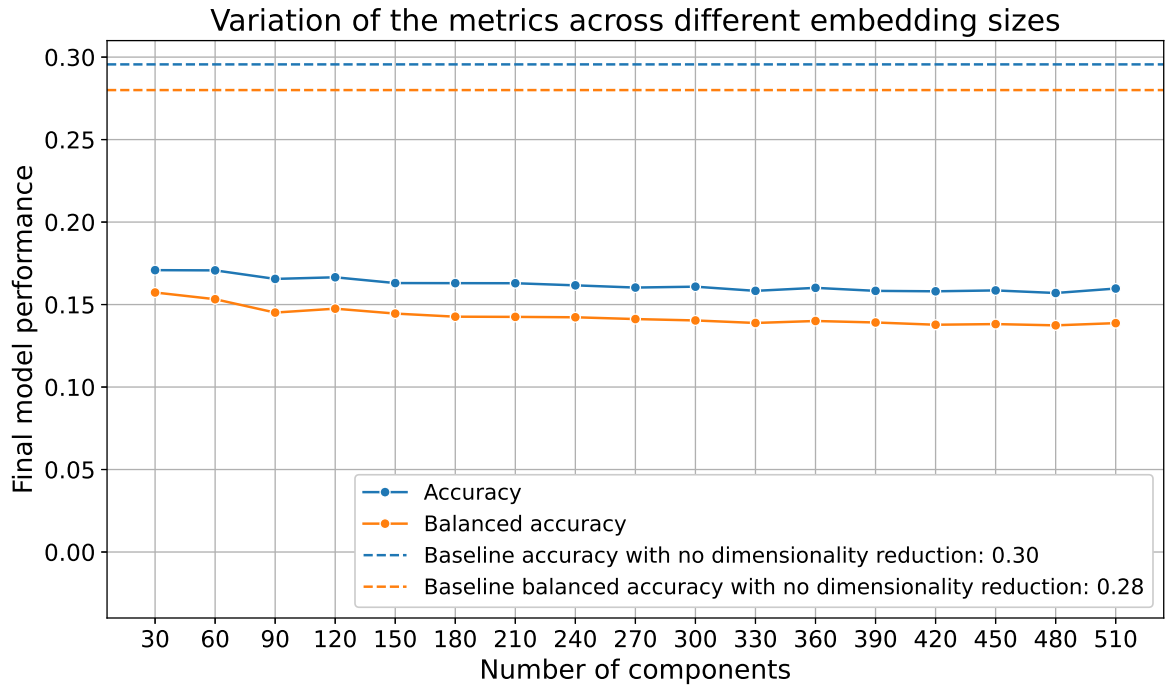


Figure 5.21: Classification performance of Kalman-SLDA method after feature extraction via MobileNet V3 Small and reduction with Batch-UMAP across various component sizes.

Number of components	MobileNet + SRP	ResNet + SRP	MobileNet + Batch-UMAP	ResNet + Batch-UMAP
30 components	0.13	0.12	0.15	0.15
60 components	0.16	0.15	0.15	0.15
90 components	0.17	0.16	0.15	0.15
120 components	0.17	0.17	0.15	0.15
150 components	0.18	0.18	0.14	0.14
180 components	0.19	0.18	0.14	0.14
210 components	0.19	0.18	0.14	0.14
240 components	0.19	0.19	0.14	0.14
270 components	0.19	0.18	0.14	0.14
300 components	0.19	0.19	0.14	0.14
330 components	0.19	0.20	0.14	0.14
360 components	0.19	0.19	0.14	0.14
390 components	0.20	0.19	0.14	0.14
420 components	0.20	0.20	0.14	0.14
450 components	0.19	0.20	0.14	0.14
480 components	0.20	0.20	0.14	0.14
510 components	0.20	0.20	0.14	0.14
Baseline (no dim. reduction)	0.28	0.28	0.28	0.28

Table 5.17: Average balanced accuracy of Kalman-SLDA after feature extraction and reduction using different methods across different component sizes. Blue lines mark the feature size for which the classification performance does no longer improve.

Number of components	MobileNet + SRP	ResNet + SRP	MobileNet + Batch-UMAP	ResNet + Batch-UMAP
30 components	0.02	0.04	0.05	0.06
60 components	0.02	0.04	0.05	0.06
90 components	0.02	0.04	0.05	0.06
120 components	0.02	0.04	0.05	0.06
150 components	0.03	0.04	0.05	0.07
180 components	0.03	0.05	0.05	0.07
210 components	0.03	0.05	0.05	0.07
240 components	0.03	0.05	0.06	0.07
270 components	0.04	0.06	0.06	0.08
300 components	0.04	0.06	0.06	0.08
330 components	0.05	0.07	0.06	0.08
360 components	0.05	0.07	0.07	0.09
390 components	0.05	0.08	0.07	0.09
420 components	0.06	0.08	0.08	0.1
450 components	0.07	0.09	0.08	0.1
480 components	0.07	0.09	0.08	0.1
510 components	0.08	0.1	0.08	0.11
Baseline (no dim. reduction)	0.08	0.09	0.08	0.09

Table 5.18: Time per sample in seconds to execute the classification pipeline for Kalman-SLDA classifier after feature extraction and reduction using different methods across different component sizes.

When using Batch-UMAP for dimensionality reduction, the behavior diverges from SLDA: although performance declines with increased component numbers, the effect is more controlled. At 30 dimensions, the model with Batch-UMAP outperforms the Sparse Random Projection variant, as shown in Figure 5.21 and Table 5.17. However, as dimensionality increases, the performance stabilizes rather than improves, indicating that Batch-UMAP may introduce concept drifts not inherent to the original dataset. While these drifts reduce SLDA’s performance significantly, Kalman-SLDA is robust enough to handle them, maintaining a reasonable performance level even in less favorable conditions.

Table 5.18 supports the findings observed with SLDA concerning computational efficiency. Both SLDA and Kalman-SLDA exhibit comparable performance across different dimensionalities and pipeline configurations, as the modifications in Kalman-SLDA are minimal in terms of computational demand but significant in terms of accuracy. The performance metrics using the different classifiers can be compared by examining Tables 5.15 and 5.17. The proposed pipeline can leverage Sparse Random Projection to reduce execution time or retain the original feature set to enhance classification accuracy.

- Extract features using MobileNet V3 Small.
- Apply Sparse Random Projection to reduce the feature dimensionality to approximately 200 components, or avoid dimensionality reduction to prioritize accuracy.

- Use Kalman-SLDA for classification.

Gaussian Naive Bayes

Figure 5.22 presents the performance of the Gaussian Naive Bayes classifier. In contrast to other methods, Gaussian Naive Bayes achieves classification performance comparable to the baseline (no dimensionality reduction) when features are reduced using Sparse Random Projection. Although Kalman-SLDA outperforms Gaussian Naive Bayes in accuracy, Gaussian Naive Bayes is highly time-efficient with dimensionality reduction. Reducing the feature dimensionality to approximately 250 components yields the best performance across methods applying dimensionality reduction and optimizes Gaussian Naive Bayes' computational efficiency. Results are consistent for features extracted with ResNet18, as detailed in Appendix A.3 and Table 5.19.

Table 5.20 reflects a trend consistent with the other classifiers: execution time per sample increases monotonically with the number of components.

Gaussian Naive Bayes model confirm results discussed in Section 5.2.3, as Batch-UMAP fails to achieve the performance levels reached with features reduced through Sparse Random Projection, as depicted in Figure 5.23. Furthermore, performance declines slightly

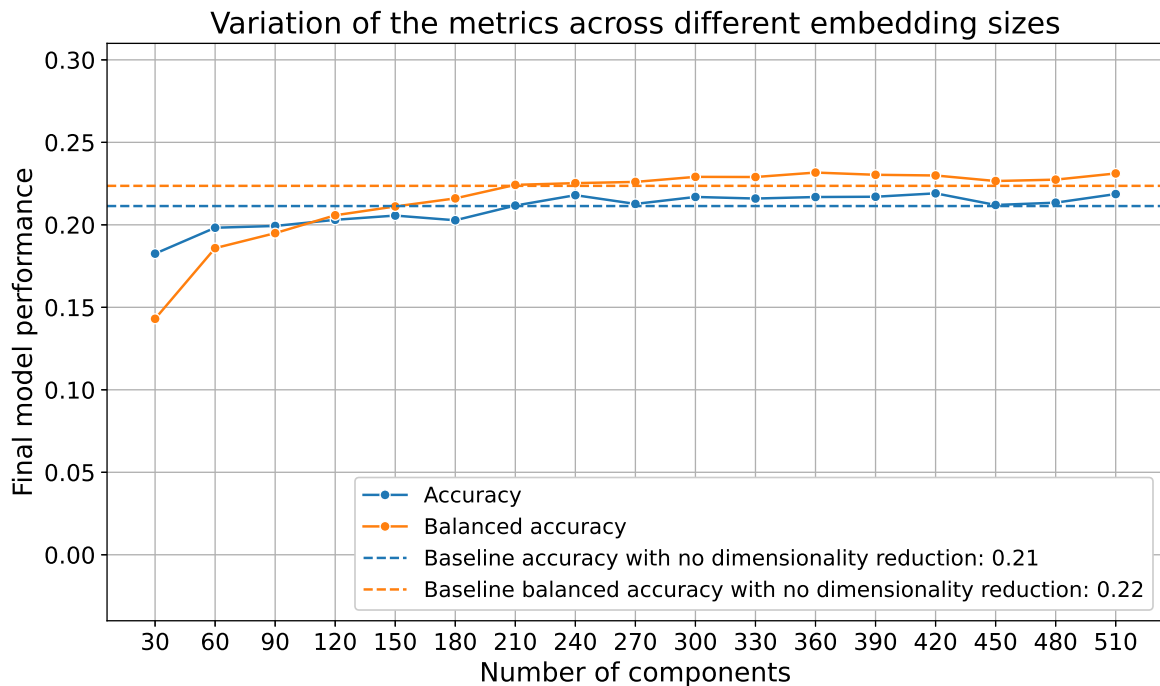


Figure 5.22: Classification performance of Gaussian Naive Bayes after feature extraction via MobileNet V3 Small and reduction with Sparse Random Projection.

Number of components	MobileNet + SRP	ResNet + SRP	MobileNet + Batch-UMAP	ResNet + Batch-UMAP
30 components	0.14	0.13	0.15	0.15
60 components	0.19	0.17	0.16	0.15
90 components	0.19	0.18	0.16	0.15
120 components	0.21	0.20	0.17	0.15
150 components	0.21	0.21	0.16	0.16
180 components	0.22	0.20	0.16	0.15
210 components	0.22	0.21	0.17	0.16
240 components	0.23	0.21	0.16	0.15
270 components	0.23	0.22	0.16	0.16
300 components	0.23	0.22	0.17	0.16
330 components	0.23	0.22	0.16	0.16
360 components	0.23	0.22	0.17	0.16
390 components	0.23	0.22	0.17	0.16
420 components	0.23	0.22	0.16	0.16
450 components	0.23	0.22	0.16	0.15
480 components	0.23	0.22	0.17	0.15
510 components	0.23	0.23	0.16	0.16
Baseline (no dim. reduction)	0.22	0.22	0.22	0.22

Table 5.19: Average balanced accuracy of Gaussian Naive Bayes after feature extraction and reduction using different methods across different component sizes. Blue lines mark the feature size for which the classification performance does no longer improve.

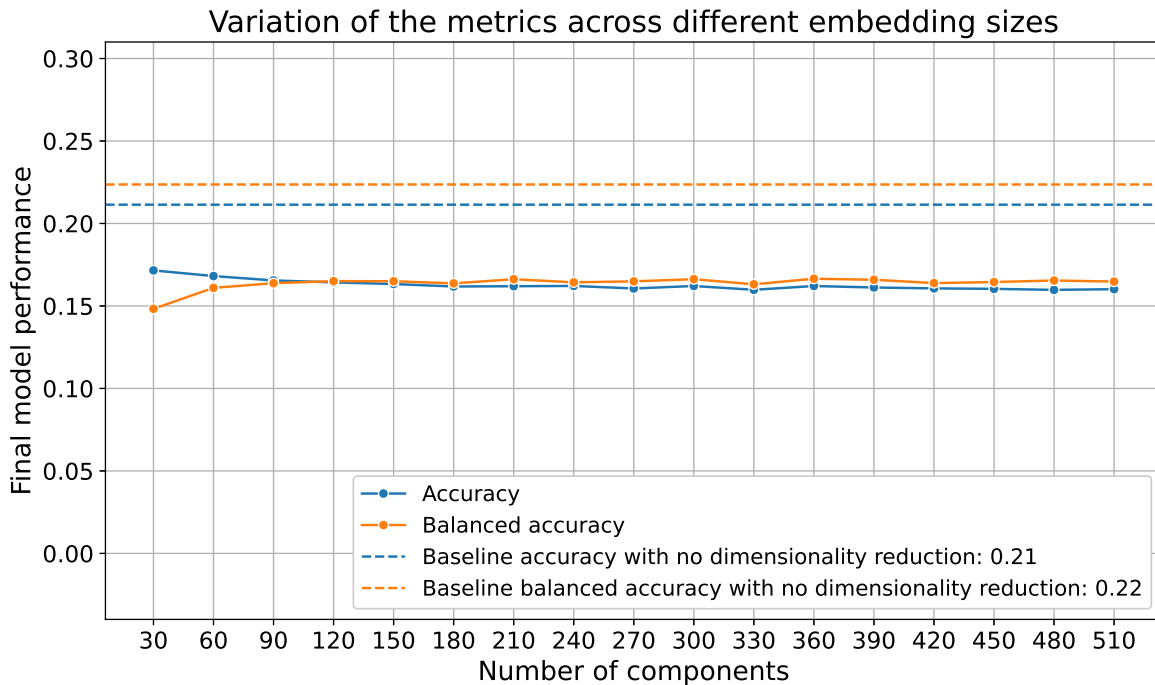


Figure 5.23: Classification performance of Gaussian Naive Bayes after feature extraction via MobileNet V3 Small and reduction with Batch-UMAP across various component sizes.

Number of components	MobileNet + SRP	ResNet + SRP	MobileNet + Batch-UMAP	ResNet + Batch-UMAP
30 components	0.01	0.03	0.04	0.06
60 components	0.02	0.04	0.05	0.06
90 components	0.02	0.04	0.05	0.07
120 components	0.02	0.04	0.05	0.07
150 components	0.03	0.05	0.06	0.07
180 components	0.03	0.05	0.06	0.08
210 components	0.03	0.05	0.06	0.08
240 components	0.04	0.05	0.07	0.08
270 components	0.04	0.06	0.07	0.09
300 components	0.05	0.06	0.07	0.09
330 components	0.05	0.06	0.07	0.09
360 components	0.05	0.07	0.08	0.1
390 components	0.05	0.08	0.08	0.1
420 components	0.06	0.09	0.08	0.1
450 components	0.06	0.09	0.08	0.11
480 components	0.06	0.09	0.09	0.11
510 components	0.07	0.1	0.09	0.11
Baseline (no dim. reduction)	0.06	0.07	0.06	0.07

Table 5.20: Time per sample in seconds to execute the classification pipeline for Gaussian Naive Bayes classifier after feature extraction and reduction using different methods across different component sizes.

as feature dimensionality increases with Batch-UMAP.

The pipeline offering the optimal balance between computational efficiency and classification accuracy is outlined as follows:

- Extract features using MobileNet V3 Small.
- Apply Sparse Random Projection to reduce dimensionality to approximately 250 components.
- Use Gaussian Naive Bayes for classification.

5.2.5. Best Model Selection

To conclude this chapter, a summary of the most accurate model is presented. The Kalman-SLDA classifier, using features extracted by MobileNet V3 Small, achieved the highest classification performance, as shown in Table 5.21. Notably, results without dimensionality reduction are excluded from this comparison, as previous analyses have demonstrated that no classifier matches the performance reached with the original high-dimensional features.

The model's performance is illustrated in Figure 5.24, with the corresponding confusion

matrix shown in Figure 5.25. In this scenario, dimensionality reduction is unnecessary, as Kalman-SLDA effectively leverages curse of dimensionality to extract essential information from the high-dimensional features. The processing time for this pipeline is **0.08 seconds per sample**, a practical runtime that could be further reduced by parallelizing operations across more cores in the experimental environment.

CNN	Classifier	Avg Accuracy	Avg Balanced Accuracy
MobileNet v3 Small	Kalman-SLDA	0.30	0.28
	SLDA	0.25	0.27
	GNB	0.21	0.22
	SMR	0.25	0.13
ResNet18	Kalman-SLDA	0.28	0.28
	SLDA	0.24	0.25
	GNB	0.21	0.22
	SMR	0.24	0.12

Table 5.21: Average accuracy and balanced accuracy for different SML classifiers with non-reduced features.

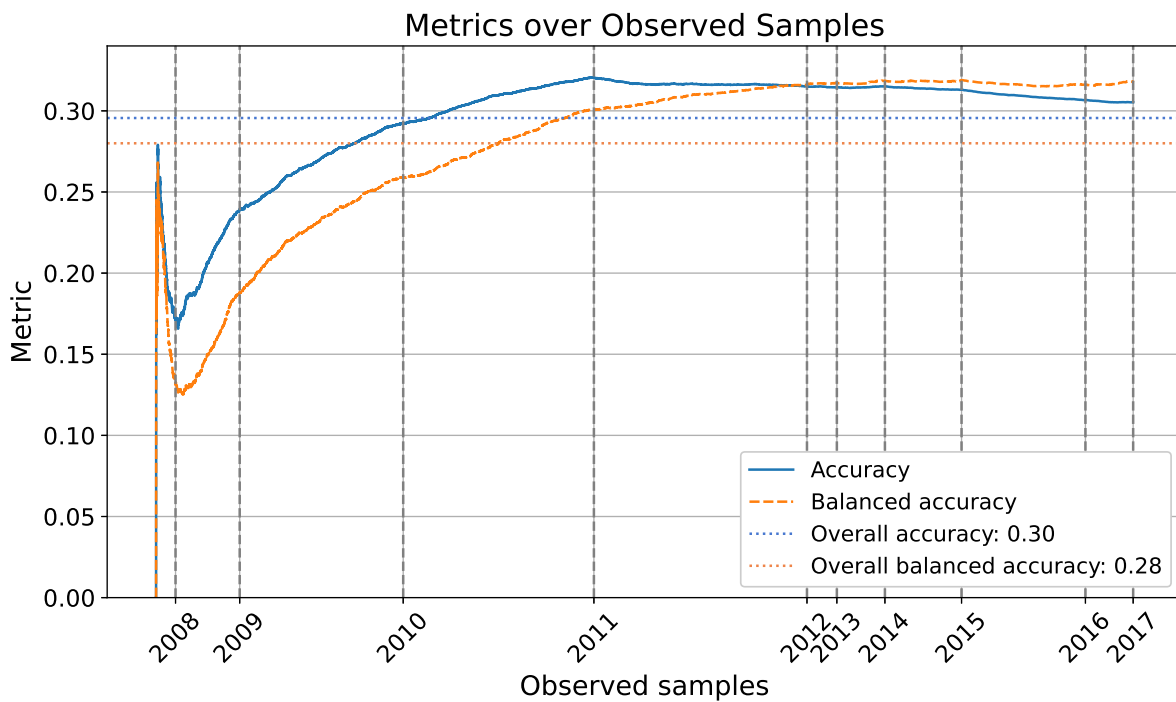


Figure 5.24: Performance of Kalman-SLDA classifier with features extracted using MobileNet v3 Small.

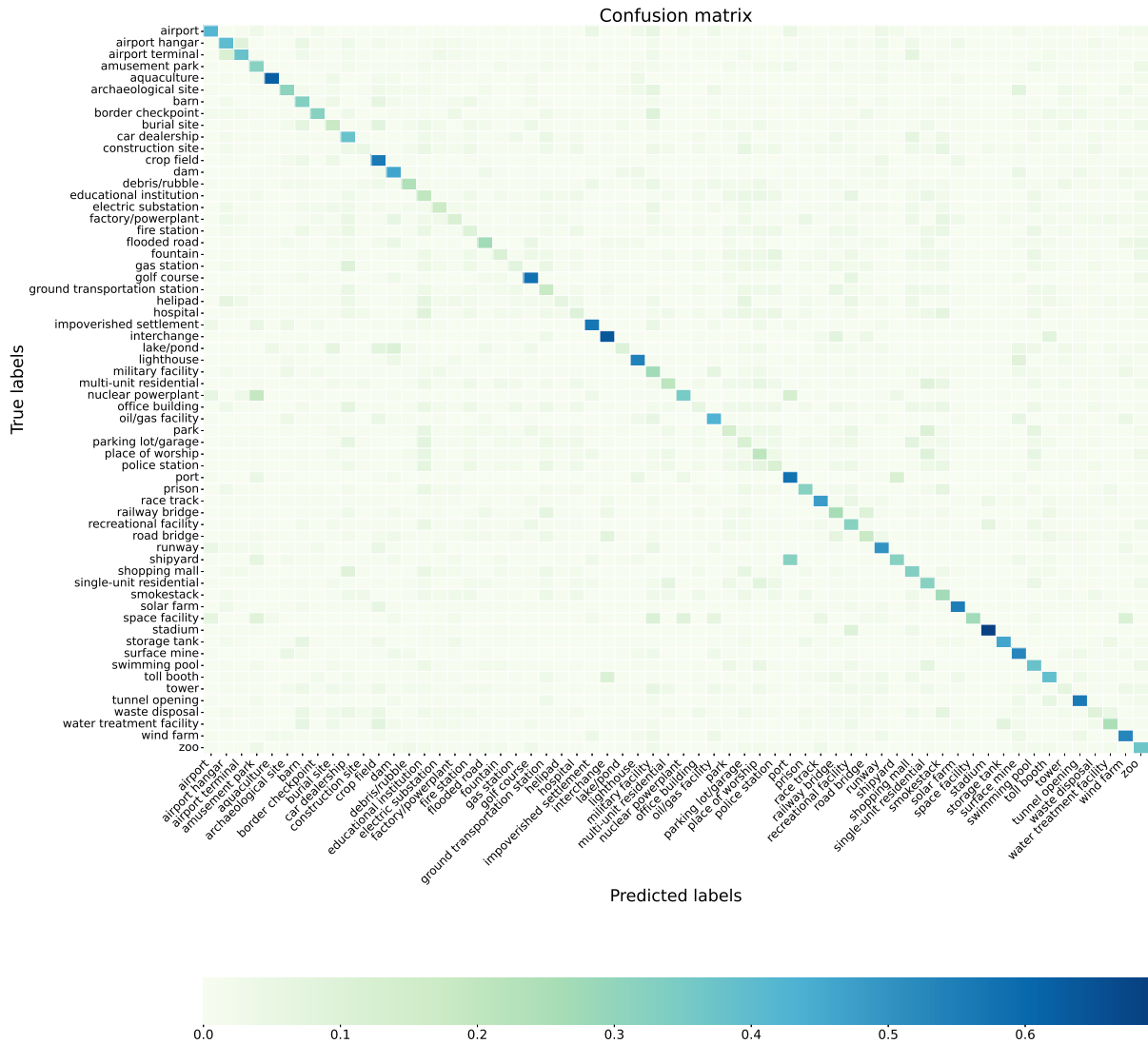


Figure 5.25: Confusion matrix of Kalman-SLDA classifier with features extracted using MobileNet v3 Small.

6 | Conclusions and Future Developments

The final conclusions of this thesis are presented in Section 6.1, while potential improvements to the work discussed are addressed in Section 6.2.

6.1. Conclusions

The primary objective of this study was to develop an optimal pipeline for classifying satellite image streams, focusing on both accuracy and computational performance. A foundational understanding of the interactions between dimensionality reduction techniques and classification algorithms in the context of satellite image stream analysis is provided. The insights gained contribute to optimizing performance while balancing accuracy and computational efficiency, laying the groundwork for future advancements in real-time data processing frameworks. Key findings from the experimental campaign are summarized as follows:

- **RQ1:** Kalman-SLDA effectively overcomes the limitations of traditional SLDA in handling concept drift without increasing computational requirements. This enhancement establishes Kalman-SLDA as a more efficient model for high-dimensional classification tasks with high label fragmentation, particularly suitable for evolving data streams where concept drift needs to be managed.
- **RQ2:** Batch-UMAP is presented as an innovative dimensionality reduction approach for Streaming Machine Learning; however, it underperformed relative to the state-of-the-art technique of Sparse Random Projection. While Batch-UMAP did not demonstrate sufficient competitive advantage over existing methods, its positive aspect is that it integrates data similarity to enhance downstream tasks.
- **RQ3:** Kalman-SLDA emerges as the optimal choice among the evaluated SML classifiers, achieving a strong balance between accuracy and efficiency. Parallelized computations makes it well-suited for satellite image classification, achieving high

classification performance while effectively managing class imbalance and complex data distributions. **Dimensionality reduction** improves the computational efficiency of classifiers in terms of speed, yet it also leads to a decline in overall classification performance. Classifiers such as Kalman-SLDA benefit from high-dimensional features that capture critical information in class mean vectors, indicating that reduced feature vectors may adversely affect classification accuracy. In contrast, Gaussian Naive Bayes displayed a more robust adaptability to reduced dimensionality, highlighting the need to consider classifier-specific responses to dimensionality reduction.

- **RQ4:** Kalman-SLDA demonstrates strong relevance as an SML classifier, balancing high accuracy with computational efficiency and showing improvements over SLDA without increasing computational demands. However, Batch-UMAP does not meet expectations in comparison to other state-of-the-art dimensionality reduction methods, suggesting that it needs refinement to be competitive. Furthermore, **Softmax Regression** reveals limitations in handling class imbalance with a large number of classes, typically favoring majority classes while neglecting minority ones. This issue, not anticipated at the study’s outset, surfaced during comparative analyses with state-of-the-art models. It underscores the importance of developing techniques that effectively address class imbalance, such as implementing sampling strategies to improve minority class representation without compromising overall performance.

6.2. Future Work

This research presents several directions for future exploration that could significantly improve the scalability, efficiency, and accuracy of dimensionality reduction and classification in dynamic and high-dimensional datasets.

In the current experimental setup, attempts to apply Streaming-UMAP faced computational constraints, primarily due to memory limitations and the lack of hardware optimized for high-throughput, incremental dimensionality reduction. UMAP’s graph-based computation is notably memory-intensive, with its complexity scaling exponentially as data volume and dimensionality increase. This imposes significant challenges, especially when handling large batches or high-dimensional data, which often lead to memory saturation and extended processing times. Additionally, real-time updates in Streaming-UMAP necessitate substantial processing power to balance between embedding accuracy and the rapid influx of new data. Overcoming these constraints may require specialized hardware configurations capable of parallelizing critical stages in UMAP’s algorithm, such as

graph construction and neighbor-finding, to facilitate a more efficient execution. While these limitations prevent the effective application of Streaming-UMAP under conventional experimental setups, future studies could explore this technique in environments with advanced computational resources. This approach holds considerable promise as an alternative for real-time dimensionality reduction in streaming data scenarios, especially with the development of hardware and algorithmic enhancements that accommodate UMAP's computational demands.

Another promising avenue for future research involves optimizing UMAP for evolving data distributions, specifically through Batch-UMAP hyperparameter tuning. As observed, selecting optimal parameters for Batch-UMAP remains challenging, particularly in high-dimensional datasets where class distributions and data relationships shift over time. Effective tuning could enable Batch-UMAP to more precisely capture underlying patterns in these evolving environments, ultimately yielding more robust and adaptive embeddings. Exploration of adaptive hyperparameter strategies that dynamically adjust to the data distribution could help mitigate the limitations of static settings in UMAP. Furthermore, expanding the scope of UMAP's adaptability to evolving environments could open up new applications in fields requiring continuous data integration, such as real-time monitoring systems or online learning models.

In light of the high fragmentation observed in the dataset's label structure, future work could investigate the impact of merging labels with similar land-use characteristics to enhance model performance. This approach could benefit classifiers like Softmax Regression, which has shown suboptimal performance when handling a large number of fine-grained categories. By grouping labels based on similarities in land-use types, it may be possible to reduce label sparsity and create a more cohesive training set that improves the model's predictive power. Additionally, clustering-based techniques for label merging could be explored to automate this process, potentially leading to improved model generalization in land-use classification tasks.

Bibliography

- [1] S. Abburu and S. B. Golla. Satellite image classification methods and techniques: A review. *International journal of computer applications*, 119(8), 2015.
- [2] D. Achlioptas. Database-friendly random projections. In *PODS*. ACM, 2001.
- [3] D. Bhatt, C. Patel, H. Talsania, J. Patel, R. Vaghela, S. Pandya, K. Modi, and H. Ghayvat. Cnn variants for computer vision: History, architecture, application, challenges and future scope. *Electronics*, 10(20), 2021.
- [4] A. Bifet, R. Gavaldà, G. Holmes, and B. Pfahringer. *Machine learning for data streams: with practical examples in MOA*. MIT press, 2018.
- [5] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *KDD*, pages 245–250. ACM, 2001.
- [6] L. Borawar and R. Kaur. Resnet: Solving vanishing gradient in deep networks. In *Proceedings of International Conference on Recent Trends in Computing: ICRTC 2022*, pages 235–247. Springer, 2023.
- [7] Y. Chen, A. Wiesel, Y. C. Eldar, and A. O. H. III. Shrinkage algorithms for MMSE covariance estimation. *IEEE Trans. Signal Process.*, 58(10):5016–5029, 2010.
- [8] G. A. Christie, N. Fendley, J. Wilson, and R. Mukherjee. Functional map of the world. In *CVPR*, pages 6172–6180. Computer Vision Foundation / IEEE Computer Society, 2018.
- [9] I. Demir, K. Koperski, D. Lindenbaum, G. Pang, J. Huang, S. Basu, F. Hughes, D. Tuia, and R. Raskar. Deepglobe 2018: A challenge to parse the earth through satellite images. In *CVPR Workshops*, pages 172–181. Computer Vision Foundation / IEEE Computer Society, 2018.
- [10] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. IEEE Computer Society, 2009.

- [11] A. F. Gad. *Practical Computer Vision Applications Using Deep Learning with CNNs*. Apress Berkeley, 2018.
- [12] J. Gama, R. Sebastião, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In *KDD*, pages 329–338. ACM, 2009.
- [13] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, 2014.
- [14] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. Gama. Machine learning for streaming data: state of the art, challenges, and opportunities. *SIGKDD Explor.*, 21(2):6–22, 2019.
- [15] M. Grandini, E. Bagli, and G. Visani. Metrics for multi-class classification: an overview. *CoRR*, abs/2008.05756, 2020.
- [16] T. L. Hayes and C. Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. In *CVPR Workshops*, pages 887–896. Computer Vision Foundation / IEEE, 2020.
- [17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.
- [18] P. Helber, B. Bischke, A. Dengel, and D. Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens.*, 12(7):2217–2226, 2019.
- [19] D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression, Second Edition*, chapter 2, pages 31–46. John Wiley & Sons, Ltd, 2000.
- [20] A. Howard, R. Pang, H. Adam, Q. V. Le, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, and Y. Zhu. Searching for mobilenetv3. In *ICCV*, pages 1314–1324. IEEE, 2019.
- [21] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *UAI*, pages 338–345. Morgan Kaufmann, 1995.
- [22] M. Köppen. The curse of dimensionality. In *5th online world conference on soft computing in industrial applications (WSC5)*, volume 1, pages 4–8, 2000.
- [23] M. P. LaValley. Logistic regression. *Circulation*, 117(18):2395–2399, 2008.
- [24] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under concept drift: A review. *IEEE Trans. Knowl. Data Eng.*, 31(12):2346–2363, 2019.

- [25] A. Mao, M. Mohri, and Y. Zhong. Cross-entropy loss functions: Theoretical analysis and applications. In *International conference on Machine learning*, pages 23803–23828. PMLR, 2023.
- [26] L. McInnes and J. Healy. UMAP: uniform manifold approximation and projection for dimension reduction. *CoRR*, abs/1802.03426, 2018.
- [27] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem, and A. Bifet. River: machine learning for streaming data in python. *J. Mach. Learn. Res.*, 22:110:1–110:8, 2021.
- [28] N. Passalis and A. Tefas. Dimensionality reduction using similarity-induced embeddings. *IEEE Trans. Neural Networks Learn. Syst.*, 29(8):3429–3441, 2018.
- [29] D. Sachin and T. Archana. Dimensionality reduction and classification through pca and lda. *International journal of computer Applications*, 122(17), 2015.
- [30] A. Tsymbal. The problem of concept drift: Definitions and related work. *Computer Science Department, Trinity College Dublin*, 106, 05 2004.
- [31] V. Vaquet, F. Hinder, J. Vaquet, J. Brinkrolf, and B. Hammer. Online learning on non-stationary data streams for image recognition using deep embeddings. In *SSCI*, pages 1–7. IEEE, 2021.
- [32] G. Welch. An introduction to the kalman filter. In *International Conference on Computer Graphics and Interactive Techniques*, 1995.
- [33] P. Xanthopoulos, P. M. Pardalos, T. B. Trafalis, P. Xanthopoulos, P. M. Pardalos, and T. B. Trafalis. Linear discriminant analysis. *Robust data mining*, pages 27–33, 2013.
- [34] H. Yao, C. Choi, B. Cao, Y. Lee, P. W. Koh, and C. Finn. Wild-time: A benchmark of in-the-wild distribution shift over time. In *NeurIPS*, 2022.
- [35] H.-J. Yoo. Deep convolution neural networks in computer vision: A review. *IEIE Transactions on Smart Processing and Computing*, 4(1):35–43, 2015.
- [36] G. Ziffer, A. Bernardo, E. Della Valle, V. Cerqueira, and A. Bifet. Towards time-evolving analytics: Online learning for time-dependent evolving data streams. *Data Science*, 6(1-2):1–16, 2023.
- [37] G. Ziffer, A. Bernardo, E. D. Valle, and A. Bifet. Kalman filtering for learning with evolving data streams. In *IEEE BigData*, pages 5337–5346. IEEE, 2021.

A | Appendix A - Other results

The Appendix contains plots that support and illustrate the results discussed in previous sections. These visualizations provide additional insights into the data patterns and performance metrics that underlie the claims made throughout the analysis.

A.1. Kalman-SLDA

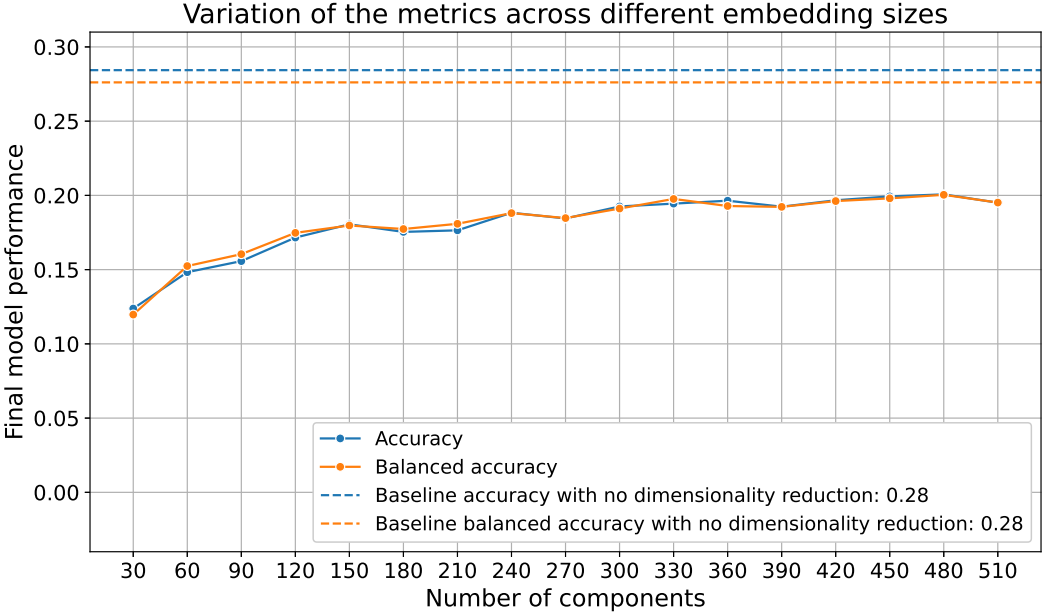


Figure A.1: Classification performance of Kalman-SLDA method after feature extraction through ResNet18 and reduction with Sparse Random Projection across different component size.

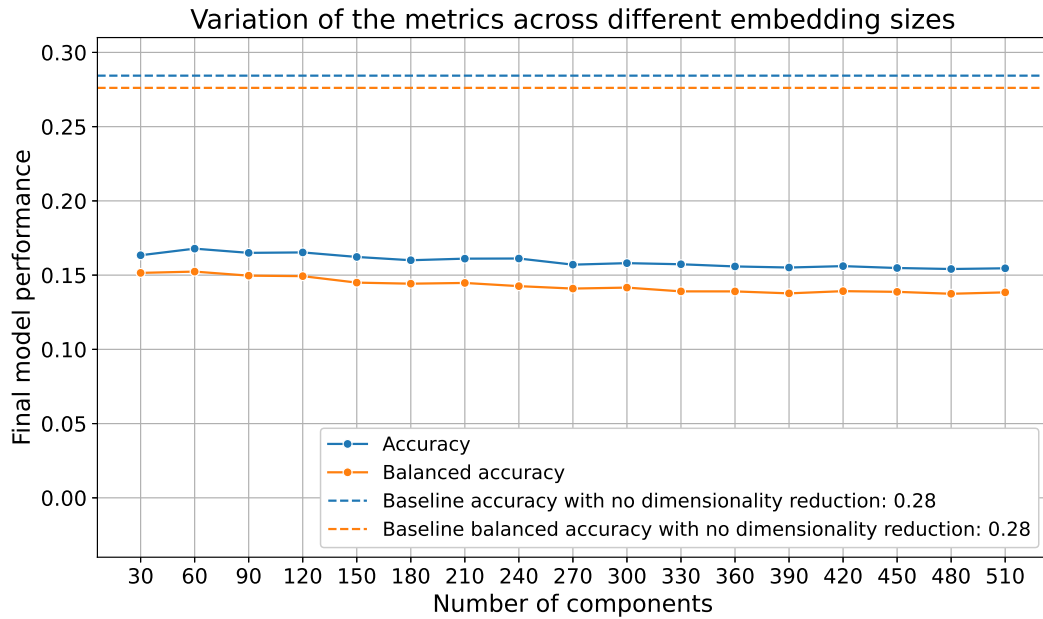


Figure A.2: Classification performance of Kalman-SLDA method after feature extraction through ResNet18 and reduction with UMAP-batch across different component size.

A.2. SLDA

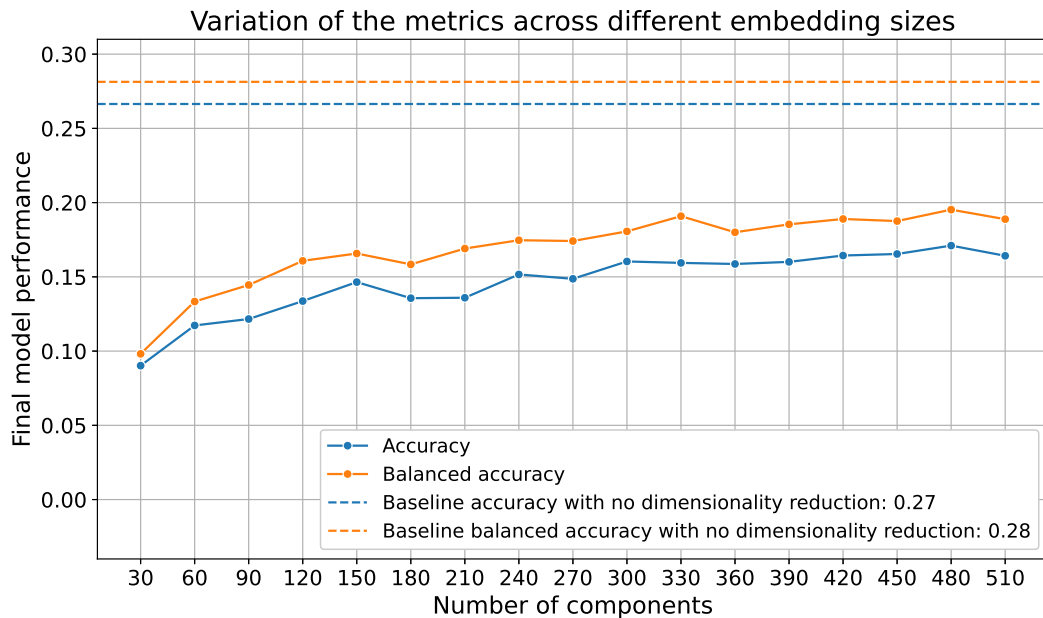


Figure A.3: Classification performance of SLDA method after feature extraction through ResNet18 and reduction with Sparse Random Projection across different component size.

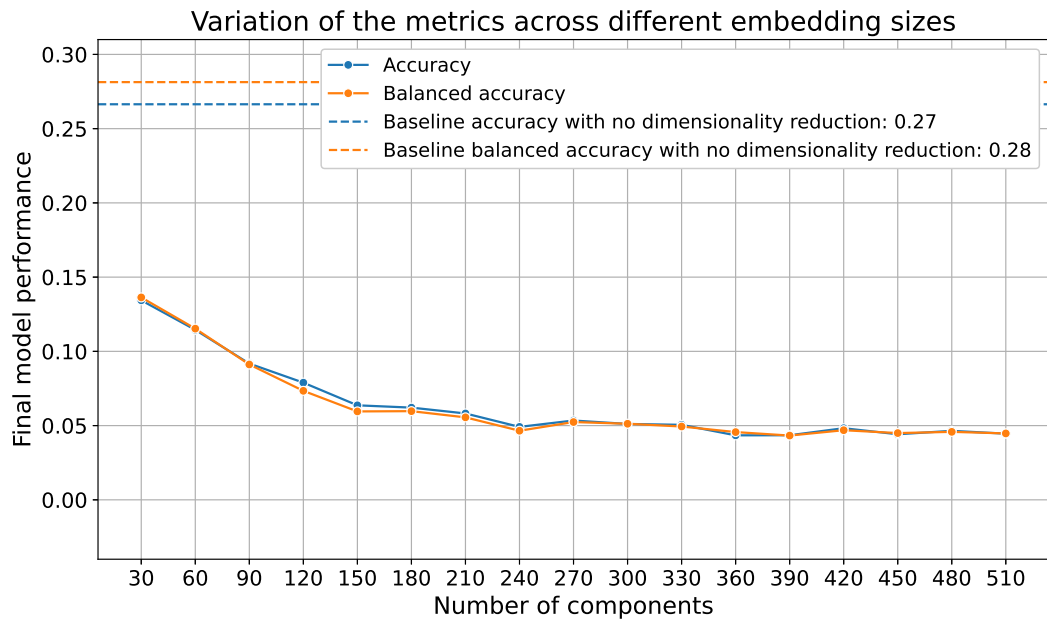


Figure A.4: Classification performance of SLDA method after feature extraction through ResNet18 and reduction with UMAP-batch across different component size.

A.3. Gaussian Naive Bayes

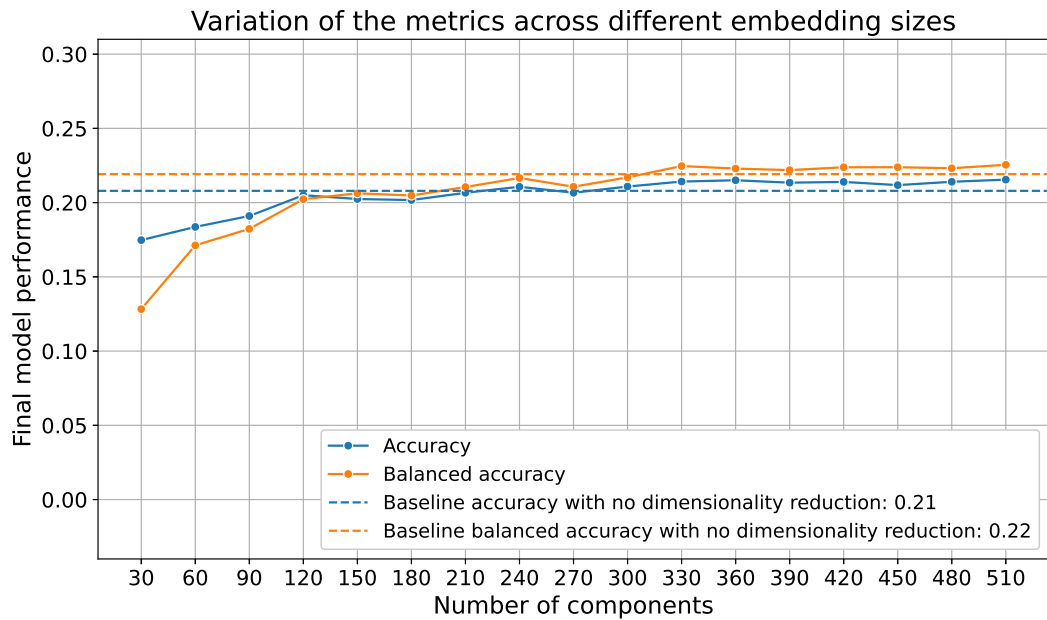


Figure A.5: Classification performance of Gaussian Naive Bayes method after feature extraction through ResNet18 and reduction with Sparse Random Projection across different component size.

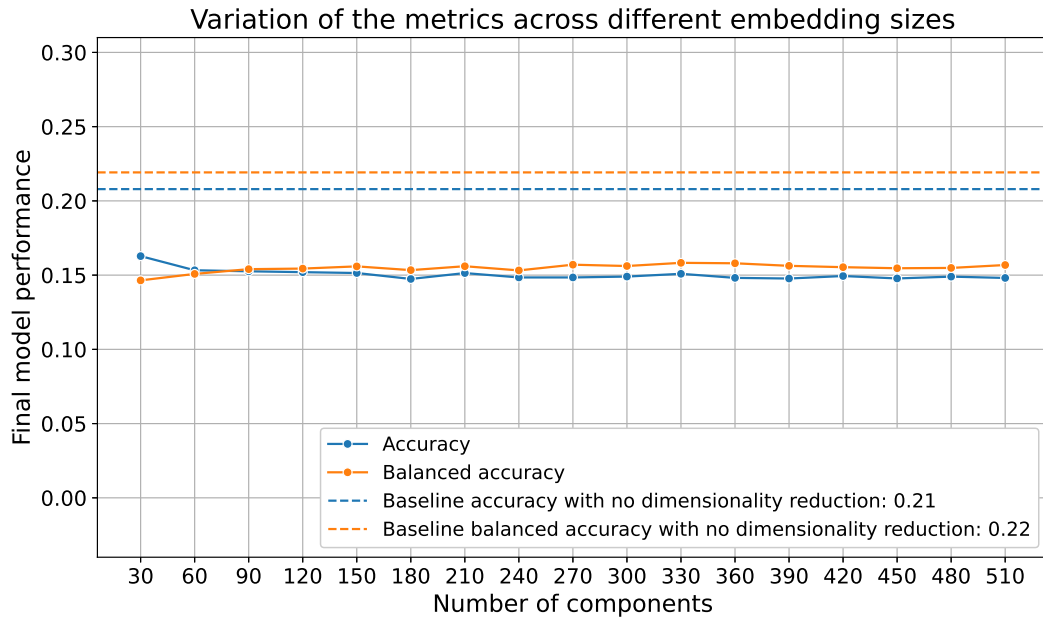


Figure A.6: Classification performance of Gaussian Naive Bayes method after feature extraction through ResNet18 and reduction with Batch-UMAP across different component size.

A.4. Softmax Regression

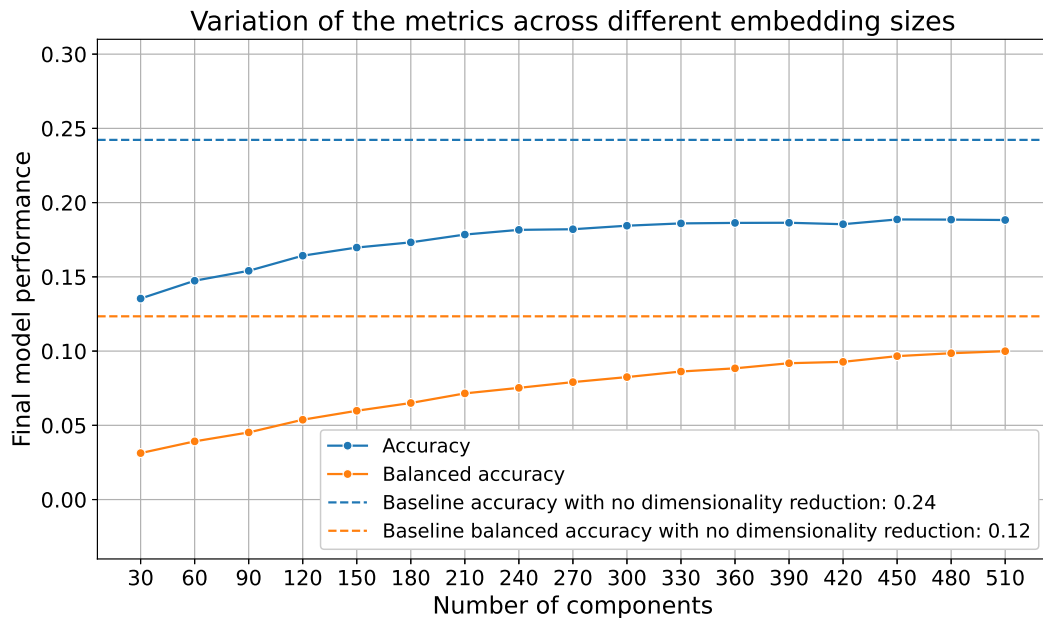


Figure A.7: Classification performance of Softmax Regression method after feature extraction through ResNet18 and reduction with Sparse Random Projection across different component size.

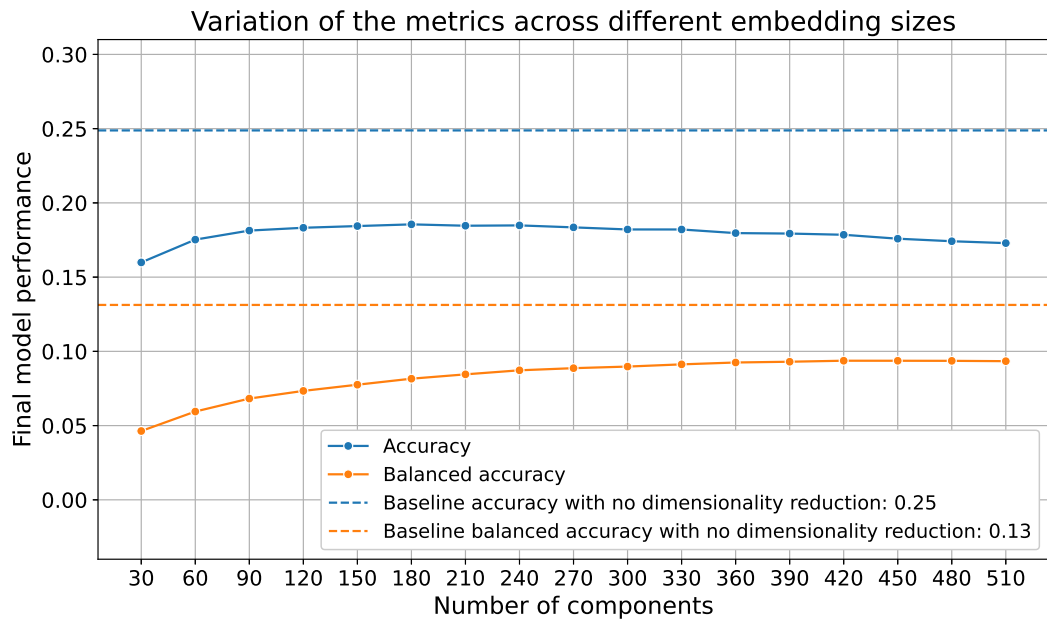


Figure A.8: Classification performance of Softmax Regression method after feature extraction through MobileNet V3 Small and reduction with Batch-UMAP across different component size.

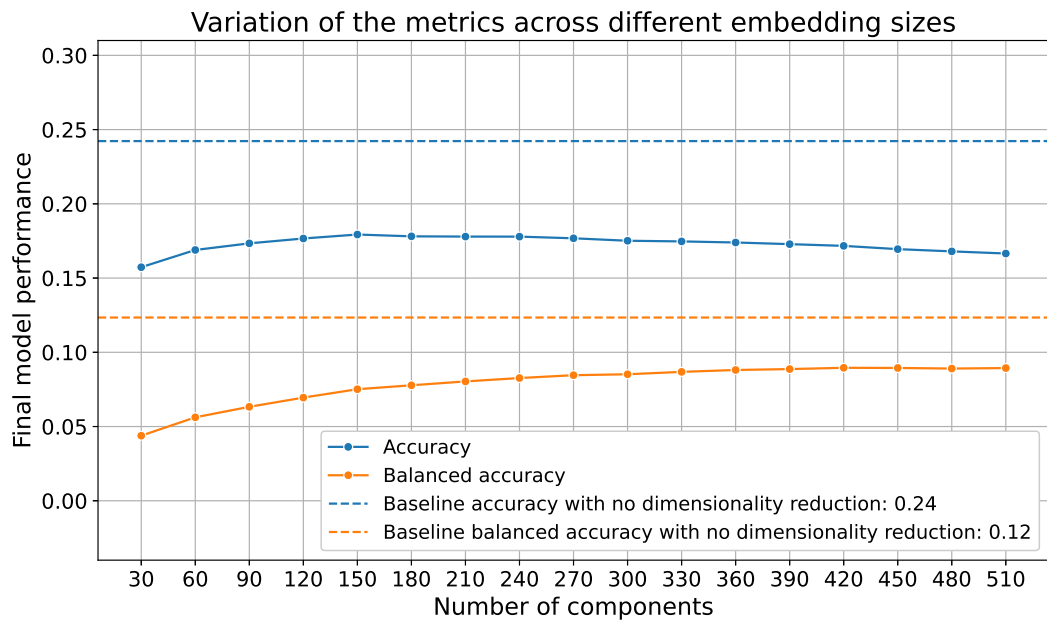


Figure A.9: Classification performance of Softmax Regression method after feature extraction through ResNet18 and reduction with Batch-UMAP across different component size.

List of Figures

2.1	Comparison of Online and Offline Learning	6
2.2	Types of Concept Drift	8
2.3	MobileNet V3 Small Architecture	15
2.4	ResNet 18 Architecture	15
2.5	Samples of images taken from the FMoW-Time dataset	17
2.6	Sparse Random Projection Algorithm	19
2.7	Uniform Manifold Approximation and Projection (UMAP) Algorithm . . .	20
2.8	Example of performance plots	26
3.1	Illustration of seasonal variations in satellite images	28
3.2	Example of concept drift in satellite images	29
4.1	SML Pipeline	32
4.2	Setup of data stream starting from batches	33
4.3	Distribution of the labels in the various years	34
4.4	Proportion of the labels in the various years	35
4.5	FMoW-Time distribution	36
4.6	Impact of concept drift on SLDA	37
4.7	Impact of concept drift on Kalman-SLDA	39
4.8	Iterative fitting of Streaming-UMAP on a bidimensional dataset	42
4.9	Iterative fitting of Batch-UMAP for a bidimensional dataset	44
5.1	Classification performance for Kalman-SLDA	48
5.2	Classification performance for SLDA	48
5.3	Classification performance for Gaussian Naive Bayes	49
5.4	Classification performance for Softmax Regression	49
5.5	Confusion matrix for Softmax Regression	50
5.6	Classification performance for Kalman-SLDA using MinMaxScaler	52
5.7	Classification performance for Kalman-SLDA without scaling features . . .	52
5.8	Performance for Softmax Regression with features from different CNNs . .	54
5.9	Confusion matrix for Softmax Regression with features from ResNet18 . .	55

5.10	Execution times for Sparse Random Projection method	57
5.11	Execution times for different SML classifiers after Sparse Random Projection	57
5.12	Classification performance of Kalman-SLDA after Sparse Random Projection	58
5.13	Execution times for Batch-UMAP method	60
5.14	Comparison of execution times for Batch-UMAP and SRP methods	61
5.15	Execution times for different SML classifiers after Batch-UMAP	61
5.16	Classification performance of Kalman-SLDA after Batch-UMAP	62
5.17	Performance of Softmax Regression with SRP on MobileNet	67
5.18	Performance of SLDA with SRP on MobileNet	68
5.19	Performance of SLDA with Batch-UMAP on MobileNet	69
5.20	Performance of Kalman-SLDA with SRP on MobileNet	70
5.21	Performance of Kalman-SLDA with Batch-UMAP on MobileNet	71
5.22	Performance of Gaussian Naive Bayes with SRP on MobileNet	73
5.23	Performance of Gaussian Naive Bayes with Batch-UMAP on MobileNet	74
5.24	Kalman-SLDA Performance Plot	76
5.25	Kalman-SLDA Confusion Matrix	77
A.1	Performance of Kalman-SLDA with SRP on ResNet	87
A.2	Performance of Kalman-SLDA with Batch-UMAP on ResNet	88
A.3	Performance of SLDA with SRP on ResNet	88
A.4	Performance of SLDA with Batch-UMAP on ResNet	89
A.5	Performance of Gaussian Naive Bayes with SRP on ResNet	89
A.6	Performance of Gaussian Naive Bayes with Batch-UMAP on ResNet	90
A.7	Performance of Softmax Regression with SRP on ResNet	90
A.8	Performance of Softmax Regression with Batch-UMAP on MobileNet	91
A.9	Performance of Softmax Regression with Batch-UMAP on ResNet	91

List of Tables

2.1	Example of confusion matrix	24
4.1	Label occurrences	35
5.1	Classification time without dimensionality reduction	47
5.2	Average classification performance for different SML classifiers	47
5.3	Average performance for Kalman-SLDA with and without a preliminary step using a Streaming Scaler	51
5.4	Time to perform feature extraction	53
5.5	Classification Time for different CNNs	53
5.6	Pipeline Execution Time without Dimensionality Reduction	53
5.7	Performance of different classifiers using features extracted from different CNNs	54
5.8	Time to execute the Pipeline with Sparse Random Projection	56
5.9	Performance of different classifiers after reducing features via SRP	59
5.10	Time to execute the Pipeline with Batch-UMAP	62
5.11	Performance of different classifiers after reducing features via Batch-UMAP	63
5.12	Distribution of Samples per Year of Observation	64
5.13	Performance of Softmax Regression using different pipelines	66
5.14	Time to execute the pipeline with Softmax Regression classifier	66
5.15	Performance of SLDA using different pipelines	68
5.16	Time to execute the pipeline with SLDA classifier	69
5.17	Performance of Kalman-SLDA using different pipelines	71
5.18	Time to execute the Pipeline with Kalman-SLDA classifier	72
5.19	Performance of Gaussian Naive Bayes using different pipelines	74
5.20	Time to execute the Pipeline with Gaussian Naive Bayes classifier	75
5.21	Performance of different classifiers for non-reduced features	76

Acknowledgements

I miei primi e più sentiti ringraziamenti vanno alla mia famiglia, che mi ha sostenuto in questi cinque anni di studi, offrendomi un'opportunità preziosa e permettendomi di affrontare questo percorso senza pressioni.

Desidero inoltre esprimere la mia gratitudine a tutte le persone che mi hanno accompagnato in questo cammino. Partito dal Friuli con tante speranze, ho trovato qui a Milano tante soddisfazioni.

Un ringraziamento speciale va al Professor Della Valle e al Dottor Ziffer, che mi hanno permesso di esplorare e approfondire gli argomenti affrontati in questa tesi.

Grazie di cuore a tutti.

