Deep Active Learning Algorithms on Imbalanced Data: Experiments with UC Merced

University of Crete

Karagiannaki Antigoni
Supervised by: Prof. Tsagkatakis Grigoris

February 2025

Contents

Chapter 1

Abstract

Active Learning is a machine learning training strategy that enables an algorithm to proactively identify subsets of training data that may most efficiently improve performance . It includes strategies for identifying which specific examples in our training data can best improve model performance.

Deep learning models typically require large amounts of labeled data to achieve high accuracy. However, obtaining labeled data is expensive and time-consuming, especially for datasets like UCMerced, where expert knowledge is often needed to annotate aerial images. Furthermore, in real-world applications, datasets are often imbalanced or contain redundant samples, which can degrade model performance. This thesis addresses these challenges by exploring how Deep Active Learning (DAL) methods, implemented via the DeepAL+ toolbox, can reduce labeling costs while maintaining high model accuracy on new datasets like UCMerced.

Chapter 2

Introduction

2.1 Problem Definition

In machine learning, particularly in deep learning, the availability of labeled datasets is a crucial factor for building accurate and reliable models. However, in many domains, such as remote sensing, labeling data is often expensive, time-consuming, and requires domain-specific expertise. This poses a significant challenge for training models on large-scale datasets.

Active Learning (AL) addresses this issue by selectively choosing the most informative samples for labeling, thereby reducing the annotation burden without compromising model performance. While Deep Active Learning (DAL) has shown great potential across standard computer vision tasks, its application to remote sensing remains underexplored. Existing frameworks, such as DeepAL+, have largely been developed and optimized for datasets in traditional domains, like object recognition, rather than landscape classification.

This thesis focuses on adapting and extending the DeepAL+ framework to handle the UC Merced Land Use dataset [2] for landscape classification tasks. The study evaluates how active learning can reduce labeling efforts for remote sensing applications while maintaining high model performance. The specific challenge addressed in this thesis is to adapt Active Learning Algorithms to the unique characteristics of remote sensing datasets and evaluate their effectiveness in comparison to Random Sampling.

2.2 Thesis Objective

The primary objective of this thesis is to extend the DeepAL+ toolkit to facilitate its use in remote sensing applications, specifically for the UC Merced Land Use dataset. This involves modifying the framework to ensure compatibility with the dataset and implementing new functions and modules that enable efficient processing of landscape data. Furthermore, the thesis focuses on the empirical evaluation of four key active learning query strategies, including two uncertainty-based and two diversity-based approaches, comparing their performance to random sampling.

By achieving these objectives, this thesis contributes to bridging the gap between active learning research and its application to remote sensing, providing insights into the efficiency and scalability of active learning methods in this domain.

2.3 Process of the Work

The process undertaken in this thesis consisted of several key steps. First, the UC Merced Land Use dataset was selected for its relevance to landscape classification tasks. Preprocessing steps were applied to standardize the dataset, which included normalization, augmentation, and preparing it for integration with the DeepAL+ framework. These steps ensured that the dataset could be seamlessly utilized within the active learning experiments.

Next, the DeepAL+ framework was extended to support the UC Merced dataset. This involved developing new functions to enable the dataset to interact seamlessly with the existing querying strategies and deep learning models provided by the framework. The modifications were designed to maintain compatibility with the toolkits structure while allowing for efficient handling of landscape data.

Following the framework extension, a series of experiments were conducted to evaluate active learning strategies. These experiments focused on testing the performance of uncertainty-based and diversity-based query strategies, comparing their outcomes against random sampling when applied to the UC Merced dataset. The experiments aimed to assess how effectively each strategy could reduce labeling costs while maintaining or improving classification accuracy.

Quantitative metrics such as accuracy, precision, and the number of labeled samples were employed to evaluate model performance. Additionally, qualitative insights were derived by examining the patterns of sample selection and their impact on the learning process. These analyses provided a deeper understanding of the behavior and efficiency of the various active learning strategies.

Finally, the findings were documented to highlight the potential of active learning in remote sensing applications. The results also serve as a roadmap for future adaptations of the DeepAL+ framework, demonstrating its flexibility and applicability to other specialized domains.

Chapter 3

Background & Related Works

3.1 Deep Active Learning (DAL)

Deep Active Learning (DAL) combines the principles of active learning with the representational power of deep neural networks to minimize the labeling effort required for training machine learning models. Traditional active learning aims to select the most informative samples from a pool of unlabeled data, typically based on strategies that evaluate uncertainty, diversity, or representativeness. DAL builds on this foundation by leveraging deep learning models, which excel at extracting high-dimensional features from complex datasets, enabling more effective sample selection.

The significance of DAL lies in its potential to reduce annotation costs in domains where labeled data is expensive or time-consuming to obtain. Applications in medical imaging, natural language processing, and remote sensing have shown that DAL can outperform traditional approaches by selecting samples that maximize model performance with minimal labeling effort. Frameworks such as DeepAL+ encapsulate these strategies, providing tools to facilitate research and experimentation in this field. In this thesis, DAL serves as the foundation for exploring active learning strategies applied to remote sensing data, particularly aerial imagery from the UC Merced Land Use dataset.

3.2 *DeepAL+* : The Basis

DeepAL+ is an open-source toolkit designed to support research in deep active learning. It provides a comprehensive framework for implementing and comparing a wide range of active learning strategies, including uncertainty-based, diversity-based, and hybrid methods. The toolkit is built to integrate seamlessly with deep learning frameworks such as PyTorch, enabling easy implementation of active learning workflows, from dataset preprocessing to querying and model training.

The modular structure of DeepAL+ makes it flexible and extensible, allowing researchers to customize its functionality to meet the specific needs of their projects. For this thesis, the toolkit served as the foundation for testing active learning strategies on the UC Merced Land Use dataset. However, its default configuration does not natively support this dataset or some of the specialized functions required for remote sensing tasks. To address this gap, new functions were implemented, and the dataset was integrated into the toolkit. These adaptations not only enhanced

the toolkits applicability to remote sensing but also demonstrated its flexibility in supporting domain-specific challenges.

3.3 UC Merced Dataset

The **UC Merced Land Use dataset** [1] is a benchmark dataset in remote sensing, widely used for evaluating machine learning algorithms in land-use classification tasks [2]. It comprises 21 land-use categories, each containing 100 images of size 256 x 256 pixels. The dataset is derived from aerial imagery and includes diverse classes such as: Agriculture, Forest, and Residential, which represent natural and man-made environments. Beach, Harbor, and Airport, which capture diverse land-use functionalities.

The dataset's balanced nature (equal samples per class) makes it ideal for baseline evaluations, but real-world applications often involve imbalanced distributions. This study adapts the dataset to simulate imbalance, reflecting practical scenarios where certain land-use categories are underrepresented.

The UC Merced dataset's high inter-class variability and intra-class similarity present unique challenges for machine learning models. As a result, it serves as an excellent testbed for evaluating the effectiveness of active learning strategies like KMeansSampling, which aim to improve performance by selecting the most informative samples

3.4 Active Learning in Remote Sensing

Remote sensing involves the collection and analysis of data from aerial or satellite imagery, providing critical insights for applications such as urban planning, environmental monitoring, and disaster management. Machine learning models play a pivotal role in processing remote sensing data, performing tasks such as land-use classification, object detection, and change detection.

However, remote sensing datasets are often vast and imbalanced, with certain land-use or land-cover types being underrepresented. This imbalance, coupled with the high cost of labeling aerial imagery, makes active learning an essential tool in this domain. Active learning can help by:

Reducing labeling effort: By focusing on the most uncertain or representative samples, active learning reduces the total number of labels required. Handling class imbalance: Strategies like diversity-based sampling or cluster-based selection can prioritize minority classes, ensuring better class representation in the training set. Improving model generalization: By carefully selecting informative samples, active learning enhances the model's ability to generalize across diverse geographic and environmental conditions.

Recent studies have successfully applied active learning to remote sensing, using techniques like uncertainty sampling and clustering to improve performance on imbalanced datasets. However, the effectiveness of these methods in specific scenarios, such as classifying high-resolution imagery from the UC Merced dataset, remains underexplored. This study aims to fill this gap by evaluating KMeansSampling and its impact on handling imbalanced data in remote sensing classification tasks.

[3]

[1] http://weegee.vision.ucmerced.edu/datasets/landuse.html

Chapter 4

Methodology

4.1 Active Learning in DeepAL+

4.1.1 Overview of Active Learning

Active Learning (AL) is a machine learning paradigm that aims to reduce labeling costs by selecting the most informative samples for annotation. [26] Instead of randomly selecting data points for labeling, AL iteratively identifies and queries the most uncertain or representative samples, thereby improving model performance with fewer labeled examples.

Traditional supervised learning relies on large annotated datasets, which are often expensive and time-consuming to obtain, particularly in remote sensing applications such as land-use classification. AL addresses this issue by using a querying strategy to select the most beneficial samples, minimizing human annotation effort while maintaining high classification accuracy.

In this study, AL is implemented through the DeepAL+ toolkit, an open-source framework that provides a modular approach to deep active learning. DeepAL+ allows the integration of various querying strategies, enabling the selection of samples based on different criteria, such as model uncertainty or dataset representativeness.

Active learning aims to iteratively select the most informative samples for labeling, optimizing model performance while minimizing the labeling cost. The DeepAL+ toolkit provides a diverse set of strategies for querying samples, each categorized into three primary groups: i) **Uncertainty-based**, ii) **Representative-based**, and iii) **Hybrid approaches**. These strategies form the foundation for experiments and were extended to address the challenges posed by the UC Merced dataset.

4.2 Querying Strategies Used In this Study

The foundation of this research is the **DeepAL+ toolkit**, a robust platform designed for deep active learning experiments. **DeepAL+** provides a variety of query strategies that enable efficient model training with minimal labeled data. My work extends this toolkit to test its applicability on the **UC Merced dataset** [2], a benchmark dataset in remote sensing. The experiments conducted, explore the toolkit's performance on landscape classification tasks while integrating additional

functionalities to suit the dataset's unique characteristics.

In order to assess the efficacy of Active Learning applied to the UC Merced dataset, this study concentrated on four principal query strategies, systematically classified as explained below.

4.2.1 Uncertainty-Based Query Strategies

These strategies prioritize instances where the model exhibits high uncertainty in predictions. [26] [27] [28] [29] The Bayesian DAL, Adversarial learning representations, Utilize gradients and Loss Prediction Querying Strategies. In more detail, the corresponding methods used include :

- **Entropy Sampling** [30] selects the most uncertain samples by computing the entropy of class probabilities. The entropy $H(x_i)$ of a sample x_i is given by:

$$H(x_i) = -\sum_{j=1}^{k} P(y_j|x_i) \log P(y_j|x_i) \tag{4.1}$$

  where k is the number of classes.
  Higher entropy values indicate greater uncertainty. This method ensures that samples with the most ambiguous classifications are labeled first.

- **Badge Sampling** (Batch Active learning by Diverse Gradient Embeddings) [5] selects a batch of the most informative samples by analyzing gradient magnitudes. Unlike traditional uncertainty sampling methods that rely on softmax probabilities, Badge Sampling identifies samples that lead to large updates in model parameters by measuring the magnitude of their gradient embeddings.

  Deep neural networks are optimized using gradient-based methods like Stochastic Gradient Descent (SGD). For a given unlabeled sample x_i, the uncertainty is estimated using the magnitude of the gradient with respect to the parameters of the last layer of the model. Formally, let the model M with parameters θ have a loss function $\mathcal{L}$, then the gradient embedding for a sample x_i is given by:

$$g(x_i) = \nabla_{\theta_{\text{out}}} \mathcal{L}(M(x_i), \hat{y}_i) \tag{4.2}$$

  where:
  - θ_{out} represents the parameters of the output layer,
  - $\mathcal{L}$ is the model's loss function, and
  - $\hat{y}_i$ is the predicted pseudo-label for x_i.

  The magnitude of gradient embedding $\|g(x_i)\|$ serves as an indicator of model uncertainty: Samples with larger gradients have a greater influence on model updates and are therefore more informative for labeling.

4.2.2 Representative/Diversity-Based Query Strategies

These strategies aim to select instances that represent the overall data distribution or maximize diversity in the selected subset. [5] This category includes:

- **K-Means Sampling** selects data points by leveraging cluster centroids in the feature space. Unlike uncertainty-based methods, which focus on model confidence, K-Means Sampling aims to maximize the diversity of selected samples by ensuring they represent the overall data distribution. [31]

  The approach is based on clustering unlabeled data using the K-Means algorithm. Given a dataset with feature embeddings $\{z_1, z_2, ..., z_N\}$ extracted from a deep model, the goal is to partition these embeddings into K clusters by minimizing the within-cluster variance:

$$\arg \min_{C_1,...,C_K} \sum_{i=1}^{K} \sum_{z \in C_i} \|z - \mu_i\|^2 \tag{4.3}$$

  where:
  - C_i represents the i^{th} cluster,
  - μ_i is the centroid of cluster C_i, and
  - $\| \cdot \|^2$ denotes the Euclidean distance.

- **K-Center Greedy Sampling**, [Core-Set Selection] [32], aims to maximize coverage of the data space. Instead of clustering samples, it iteratively selects points that are farthest from the already labeled dataset, ensuring that the new labeled points cover underrepresented regions of the data distribution.

  Formally, given a labeled set $\mathcal{D}_L$ and an unlabeled pool $\mathcal{D}_U$, the next batch of points is selected by solving the following optimization problem:

$$\arg \max_{x \in \mathcal{D}_U} \min_{x' \in \mathcal{D}_L} d(f(x), f(x')) \tag{4.4}$$

  where:
  - $f(x)$ represents the feature embedding of sample x,
  - $d(f(x), f(x'))$ is the Euclidean distance between two embeddings.

As a baseline, I also tested **Random Sampling**, where instances are selected randomly from the unlabeled pool for labeling, without considering the model uncertainty or feature diversity. This approach provides a reference to measure the effectiveness of active learning strategies by comparing their performance against an unbiased, non-informed sampling method.

4.3 Novel Contributions

The added functions include dataset handling methods tailored for the UC Merced dataset, preprocessing utilities for feature extraction, and modifications to the query strategies to accommodate landscape-specific variability. These functions ensure smooth integration of the UC Merced dataset with DeepAL+, enabling direct compatibility with the framework's existing algorithms and processes.

4.3.1 Custom Functions for UC Merced Integration

To integrate the UC Merced dataset into the DeepAL+ framework, several custom algorithms were developed. These include:

(i) **UC Merced Handler (`UC_Merced_Handler`):** A class for preprocessing and handling the dataset.

(ii) **UC Merced Net (`ucmerced_Net`):** A neural network architecture tailored for the UC Merced dataset.

(iii) **Dataset Loader (`get_ucmerced`):** A function to load and prepare the dataset for active learning.

(iv) **Imbalanced Dataset Loader (`get_ucmerced_imbalanced`):** A function to simulate imbalanced datasets for experimental evaluation.

(v) **Imbalanced UC Merced Handler (`UC_Merced_Imb_Handler`):** A function to handle an imbalanced version of the dataset for experimental evaluation.

The following subsections detail the implementation and functionality of each algorithm.

(i) Pseudocode for `UC_Merced_Handler` [handlers.py]

Algorithm 1 `UC_Merced_Handler`

Input:
  Dataset – List of file paths for UC Merced images

Initialize:
  Store X (image file paths)
  Store Y (class labels)
  Store `transform` (image preprocessing pipeline)

Methods:
1. `Get_Item(index):`
    Input: Index of the desired data sample
    Output: Preprocessed image, corresponding label, and index
    a. Retrieve file path X[index] and label Y[index].
    b. Open the image file.
    c. **If** `transform` exists:
      Apply `transform` to the image.
    d. Return the preprocessed image, label, and index.

2. `Length:`
    Output: Number of data samples
    Return length of X (number of file paths).

(ii) Pseudocode for `UC_Merced_Net` [nets.py]

Algorithm 2 `UC_Merced_Net`

Input:
  x – Input image tensor

Initialize:
  Input: *dim* – Input image dimension (default: $3 \times 256 \times 256$)
  pretrained – Whether to use pretrained weights (default: False)
  num_classes – Number of output classes (default: 21)
  Load a pretrained ResNet-18 model
  Modify the ResNet-18 model by removing the final fully connected layer
  Add a new fully connected classifier layer with *num_classes* output

Methods:
1. `Forward(x)`
  Input: x – Input image tensor
  Output: *output* – Predicted class probabilities, $feature$ – Intermediate features
  Pass input x through the feature extraction layers of ResNet-18
  Flatten the output from the feature extractor into a 1D vector
  Pass the flattened feature through the classifier
  Return the *output* and the intermediate $feature$ vector

2. `Get_Embedding_Dim`
  Output: Dimensionality of the feature vector
  Return the feature size of the final fully connected layer

(iii) Pseudocode for `get_UCMerced` [data.py]

Algorithm 3 `get_UcMerced`

Input:
  handler – Data handler instance
  args_task – Additional task-related arguments

Initialize:
  data_dir – Path to UC Merced dataset
  class_names – List of sorted class names (subdirectories in the dataset)

Collect file paths and labels:
  Initialize empty lists $file\_paths$ and *labels*
  For each class:
    Retrieve class directory
    For each image in class directory (with '.tif' extension):
      Append image file path to $file\_paths$
      Append class label to *labels*

Split data into train/test sets:
  Randomly shuffle the indices of the dataset
  Split data into 80% training and 20% testing
  Assign file paths and labels to X_train, Y_train, X_test, Y_test

Return:
  A Data object containing X_train, Y_train, X_test, Y_test, *handler*, and *args_task*

(iv) Pseudocode for `get_UCMerced_imbalanced` [data.py]

Algorithm 4 `get_UcMerced_imbalanced`

Input:
  handler – Data handler instance
  args_task – Additional task-related arguments
  imbalance_ratio – Ratio by which to imbalance the dataset

Initialize:
  data_dir – Path to UC Merced dataset
  class_names – List of sorted class names (subdirectories in the dataset)
Collect file paths and labels:
  Initialize empty lists $file\_paths$ and *labels*
  For each class:
    Retrieve class directory
    For each image in class directory (with '.tif' extension):
      Append image file path to $file\_paths$
      Append class label to *labels*

Apply imbalance:
  For each class, randomly downsample or upsample based on *imbalance_ratio*

Split data into train/test sets:
  Randomly shuffle the indices of the dataset
  Split data into 80% training and 20% testing

Return:
  A Data object containing X_train, Y_train, X_test, Y_test, *handler*, and $args\_task$

(v) Pseudocode for `UC_Merced_Imb_Handler` [handlers.py]

Algorithm 5 `UC_Merced_Imb_Handler`

Input:
  X – List of imbalanced dataset image samples
  Y – Corresponding class labels
  $transform$ – Optional image preprocessing pipeline

Initialize:
  Store X (image file paths or arrays)
  Store Y (class labels)
  Store $transform$ (preprocessing function, if provided)

Methods:
1. `Get_Item(index)`
  Input: *index* – Desired data sample
  Output: Preprocessed image, corresponding label, and index
  Retrieve image $x = X[index]$ and label $y = Y[index]$
  Convert x to a PIL image in RGB mode
  If $transform$ is not None:
    Apply $transform$ to x
  Return $(x, y, index)$

2. `Length`
  Output: Number of samples in the dataset
  Return len(X)

4.4 Evaluation Metrics

To evaluate the performance of the active learning strategies, several key metrics are considered. **Accuracy** measures the proportion of correctly classified instances, providing an overall assessment of model performance. **F1-score** balances precision and recall, making it particularly useful for handling class imbalances. **AUC-ROC** evaluates the model's ability to distinguish between different classes based on its classification confidence. Finally, **labeling efficiency** assesses how effectively active learning reduces annotation costs while maintaining high predictive performance. By systematically applying these metrics, this study aims to enhance the effectiveness of deep active learning on imbalanced datasets while optimizing the annotation process.

Chapter 5

Experimental Setup

5.1 Dataset

5.1.1 UCMerced Dataset Description

The experiments in this study were conducted using the **UC Merced Land Use Dataset**, a widely recognized benchmark dataset for remote sensing and land-use classification tasks. The dataset is particularly well-suited for evaluating Active Learning Algorithms due to its diversity and moderate size, which allows for iterative experiments without excessive computational overhead.

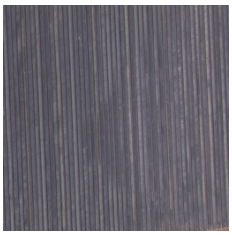
The **UC Merced dataset** consists of **21 classes**, each representing a distinct land-use category, such as agriculture, residential, forest, and freeway. [2] Each class contains 100 images, resulting in a total of 2,100 images. Most of the images are $256 * 256$ pixels and are extracted from aerial photographs with a spatial resolution of 1 foot per pixel. It should be acknowledged that out of the 2,100 images, 44 possess varying dimensions and are thus standardized prior to utilization. This high resolution ensures detailed representations of land-use features, making it an ideal candidate for evaluating the performance of active learning strategies.

Class Name	**Description**	**Class Name**	**Description**
Agricultural	Farmlands with cultivated crops and open fields.	Intersection	Major road crossings with visible lanes.
Airplane	Airport runways with visible aircraft.	Medium Residential	Moderately spaced housing areas.
Baseball Diamond	Open field with a diamond-shaped playing area.	Mobile Home Park	Clustered mobile homes and trailers.
Beach	Sandy shorelines near water bodies.	Overpass	Bridges crossing over roads.
Buildings	Urban areas with dense man-made structures.	Parking Lot	Large open spaces for vehicle parking.
Chaparral	Shrubland with dry vegetation.	River	Flowing water bodies through landscapes.
Dense Residential	Tightly packed houses and apartments.	Runway	Long strips for aircraft takeoff and landing.

Class Name	Description	Class Name	Description
Forest	Large areas covered with trees.	Sparse Residential	Houses with large spaces between them.
Freeway	Multi-lane highways with vehicles.	Storage Tanks	Large cylindrical industrial containers.
Golf Course	Green landscapes with golf holes and sand traps.	Tennis Court	Rectangular courts with nets for tennis.
Harbor	Coastal areas with docks and boats.		

Table 5.1: UC Merced Dataset Class Descriptions

The following figure includes a sample image representing each class listed in the table above. These images demonstrate the diversity of land use categories in the UC Merced dataset.



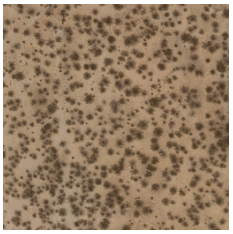
(a) agricultural93

(b) airplane68

(c) baseballdiamond32

(d) beach43

(e) buildings48



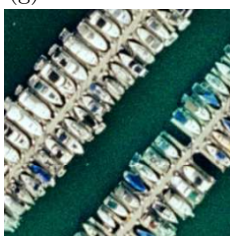
(f) chaparral21

(g) denseresidential07

(h) forest94

(i) freeway27

(j) golfcourse93



(k) harbor27

(l) intersection03

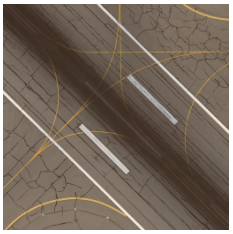
(m) mediumresidential00

(n) mobilehomepark95

(o) overpass89

(p) parkinglot81

(q) river73



(r) runway90

(s) sparseresidential24

(t) storagetanks31

(u) tenniscourt61

Figure 5.1: Visual Representation of the UCMerced Land Use Classes

5.1.2 Imbalanced UCMerced Dataset

The standard UC Merced dataset contains a balanced distribution of aerial images across 21 land-use classes. To analyze the impact of class imbalance on active learning, we created an imbalanced version of the dataset by reducing the number of samples in certain classes.

Class Imbalance Strategy

To introduce an artificial imbalance, we applied a fixed ratio reduction to specific classes: Classes 0-9 retained 100% of their original samples, Classes 10-14 retained 80% of their original samples, and Classes 15-20 retained 60% of their original samples. This results in a dataset where some classes have significantly fewer labeled samples compared to others, simulating real-world class imbalance scenarios.

Implementation of Imbalance

The dataset was modified using the function `create_imbalance()`, which randomly samples a reduced subset of images for each affected class based on the predefined ratios. The resulting class distribution after applying imbalance is as follows:

Class	Remaining Samples (% of Original)
0 - 9	100%
10 - 14	80%
15 - 20	60%

Table 5.2: Class distribution after applying artificial imbalance.

Handling Class Imbalance: Oversampling

Since active learning models tend to be biased towards majority classes in an imbalanced dataset, we applied Random Oversampling to mitigate this issue. The function `oversample_data()` was used to artificially increase the representation of minority classes by duplicating samples. To address class imbalance in the dataset, we employed the RandomOverSampler from the imbalanced-learn (imblearn) library. This technique artificially increases the representation of underrepresented classes by generating synthetic samples, thereby mitigating potential biases in model training. As a result, the final training set exhibited an approximately uniform class distribution, ensuring a more balanced learning process.

Final Processed Dataset

After applying oversampling, the dataset was converted into PyTorch tensors for compatibility with our deep active learning framework. The final class distribution can be shown below, in Figure 5.2 as it was printed using `print_class_distribution()` to verify the effectiveness of oversampling.

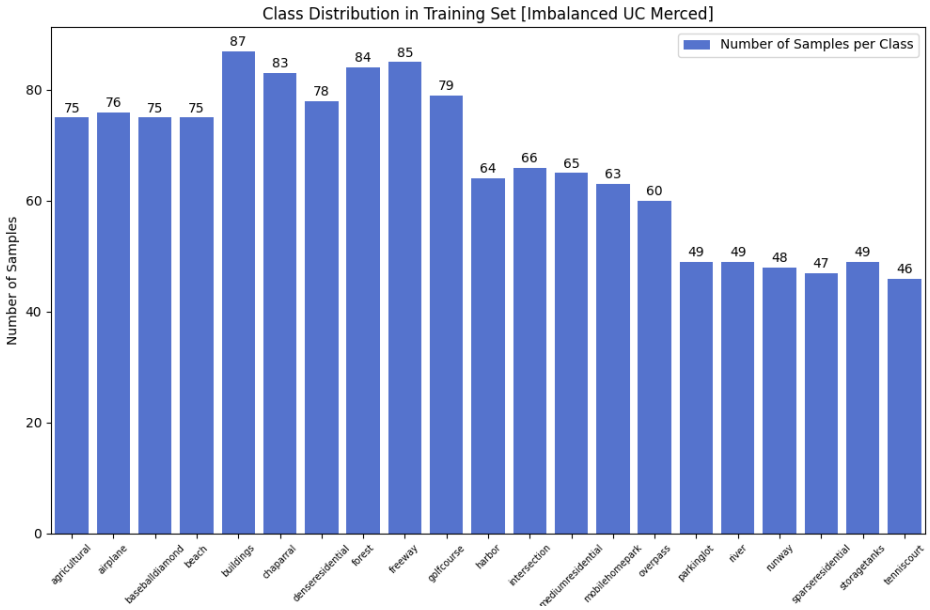


Figure 5.2: Distribution of Sample Counts Across Classes in the Imbalanced UCMerced Training Set

Impact on Active Learning

By assessing active learning techniques on both the original balanced UC Merced dataset and its imbalanced counterpart, we can determine the impact of class imbalance on the performance of various query strategies. This includes examining whether uncertainty-based sampling methods, such as entropy sampling, face challenges with underrepresented classes, as well as evaluating the role of oversampling in enhancing learning efficiency.

5.2 Preprocessing

The experimental process follows an iterative active learning loop involving model training, querying, annotation, and retraining. Details on dataset preparation, hyperparameters, and evaluation procedures are provided throughout this section.

5.3 Implementation Details

All experiments employ **ResNet18**(without pretraining) [6] as the base classifier to ensure a fair comparison across methods. The experiments are executed using the DeepAL+ toolkit, also it is possible to implement KMeans with GPU acceleration using the FAISS library to optimize clustering efficiency when dealing with large-scale unlabeled datasets, but this is not included in this study.

Experimental protocol. Each experiment is repeated for three trials using different random splits of the initial labeled and unlabeled pools while maintaining the same random seed for consistency. Performance evaluation follows the area under the budget curve (AUBC) metric, where a performance-budget curve is plotted for each DAL method. Higher AUBC values indicate stronger overall performance.

Additionally, we report the final accuracy (F-acc), representing the classification accuracy after the budget Q is exhausted.

5.4 Experiment Design

The goal of this experiment is to evaluate various Deep Active Learning (DAL) strategies implemented in the DeepAL+ toolkit on the UC Merced dataset. The study aims to compare the performance of these strategies against Random Sampling, which serves as a baseline approach. The primary focus is to determine how effectively each active learning strategy improves classification accuracy while minimizing the number of labeled samples required. Additionally, an imbalanced version of the dataset is introduced to assess the impact of class distribution on model performance under active learning conditions.

Active Learning Framework

The experiments in this study are conducted using the DeepAL+ toolkit, which provides a flexible framework for implementing and testing deep active learning strategies. Within this framework, various query strategies are explored, including uncertainty-based methods such as Entropy Sampling and BALD Dropout, as well as diversity-based approaches such as KMeans Sampling and KCenter Greedy. Additionally, Badge Sampling, a hybrid strategy that leverages gradient-based embeddings to select the most informative samples, is tested. For comparison, Random Sampling is used as a baseline to evaluate the effectiveness of active learning.

The backbone model used for classification is a Convolutional Neural Network (CNN), specifically ResNet-18, pre-trained on ImageNet and fine-tuned for aerial image classification. The model is optimized using stochastic gradient descent (SGD) with a learning rate of 0.001 and a batch size of 32.

Active Learning Process

The active learning pipeline follows an iterative process, beginning with training the CNN model on an initial labeled subset of the dataset. Once the model is trained, an active learning strategy selects a batch of the most informative unlabeled samples, which are then manually labeled and added to the training set. The model is subsequently retrained with the updated dataset, and this cycle repeats until a predefined labeling budget is reached. For this study, the process continues until 1,000 labeled samples are obtained. This iterative approach ensures that the most informative samples are selected, reducing the number of labeled data points needed to achieve high classification accuracy.

Evaluation Metrics

To assess the effectiveness of each active learning strategy, several evaluation metrics are considered. Accuracy vs. Training Size is used to analyze how classification performance improves as more labeled samples are added. Additionally, the Area Under the Learning Curve (AUBC) is computed to quantify the efficiency of each strategy in achieving high accuracy with fewer labels. Furthermore, computational efficiency is measured to evaluate the trade-off between model performance and the time required for querying and retraining.

Chapter 6

Experimental Results

6.1 Results on the UC Merced Dataset

6.1.1 Performance of Land Image Analysis tasks

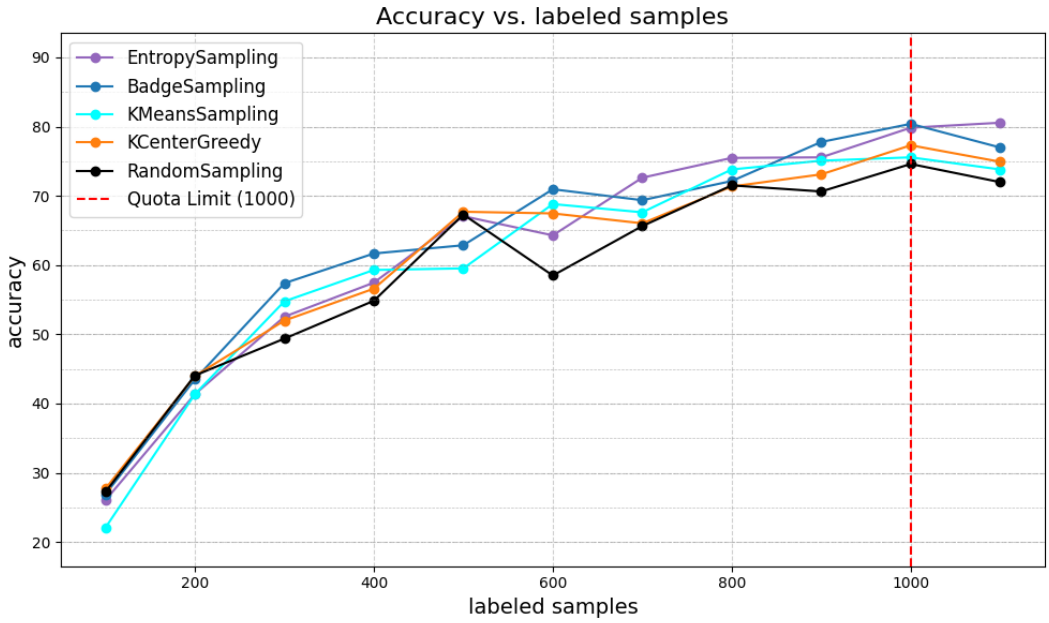
Table 6.2 presents the overall performance of various strategies applied to the UC Merced dataset. The results show that KMeans Sampling, Random Sampling, Badge Sampling, Entropy Sampling, and KCenter Greedy all perform well in the context of this dataset. Notably, the Entropy Sampling strategy achieves the highest F-accuracy at 80.9%, and the corresponding AUBC value is 0.6397, with a standard deviation of 0.011. On the other hand, both KMeans Sampling and Badge Sampling show very similar performance in terms of AUBC (0.6237 and 0.648, respectively), but KMeans Sampling is slightly more time-efficient.
A key observation is that the F-accuracy of the strategies is quite close, with Random Sampling and Badge Sampling both achieving 73.5% F-accuracy, although they differ in AUBC values. The mean times also reveal interesting trends, with KMeans Sampling and KCenter Greedy requiring slightly more computation time compared to other strategies. This highlights the trade-off between computational efficiency and model performance.

The results suggest that for the UC Merced dataset, the Entropy Sampling strategy provides the best balance between performance and efficiency. However, it is important to note that all strategies significantly outperform the random selection of samples, underlining the effectiveness of active learning strategies in image classification tasks. Future work should explore further optimization of time and accuracy trade-offs, and test these strategies on more complex real-world datasets, especially in domains where dataset shifts and noise are prevalent.

Strategy	**F-Accuracy (%)**	**AUBC**	**Mean Time (s)**	**Std Dev Time (s)**
KMeans Sampling	73.0	0.6237 ± 0.006	20994.666	36.7181
Random Sampling	73.5	0.6063 ± 0.0069	19551.666	57.3488
Badge Sampling	73.5	0.648 ± 0.01	21653.0	226.0723
Entropy Sampling	80.9	0.6397 ± 0.011	21090.0	373.3711
KCenter Greedy	68.3	0.6267 ± 0.0104	20558.0	371.931

Table 6.1: Comparison of different active learning strategies based on accuracy, AUBC, and computational time.

6.1.2 Accuracy Across Samples



(a) Overall accuracy vs. budget curves on UCMerced.

6.2 Results on the Imbalanced UC Merced Dataset

6.2.1 Performance of Land Image Analysis tasks

Table 6.3a presents the overall performance of various strategies applied to the imbalanced UC Merced dataset. The results show that KMeans Sampling, Random Sampling, Badge Sampling, Entropy Sampling, and KCenter Greedy all perform well despite the dataset imbalance. Notably, the Entropy Sampling strategy achieves the highest accuracy, reaching approximately 80%, indicating its effectiveness in selecting informative samples. The corresponding AUBC value suggests strong overall performance, with minimal variance. On the other hand, both Badge Sampling and KCenter Greedy exhibit comparable accuracy trends, closely following Entropy Sampling, while KMeans Sampling shows minor fluctuations in the mid-range of labeled samples.

A key observation is that the accuracy of the strategies remains closely aligned, especially as the number of labeled samples approaches the quota limit of 1000. Random Sampling, while generally less effective than the active learning strategies, still performs competitively, reinforcing the idea that dataset imbalance does not entirely diminish the benefits of structured sampling methods. The mean computational times also reveal differences, with KMeans Sampling and KCenter Greedy requiring slightly more processing time, emphasizing the trade-off between efficiency and performance.

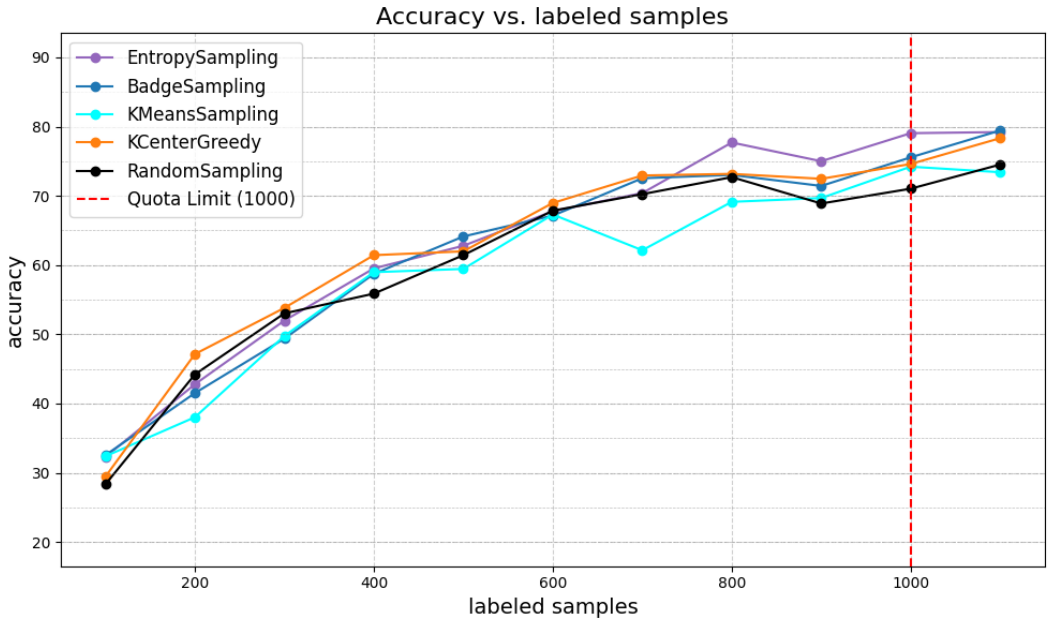
The results suggest that for the imbalanced UC Merced dataset, Entropy Sampling provides the best balance between accuracy and sample efficiency. However, it is important to note that all active learning strategies significantly outperform random selection, underscoring their value in handling class imbalances in image

classification tasks. Future research should further explore methods to enhance performance in imbalanced datasets, particularly focusing on improving representation for underrepresented classes while maintaining computational efficiency.

Strategy	F-Accuracy (%)	AUBC	Mean Time (s)	Std Dev Time (s)
KMeans Sampling	75.4	0.6017 ± 0.014	20774.0	40.653
Random Sampling	76.4	0.6167 ± 0.02	20134.333	231.9976
Badge Sampling	73.5	0.648 ± 0.01	21653.0	226.0723
Entropy Sampling	79.5	0.6427 ± 0.0026	21160.666	134.863
KCenter Greedy	73.8	0.6403 ± 0.0012	20793.0	102.0686

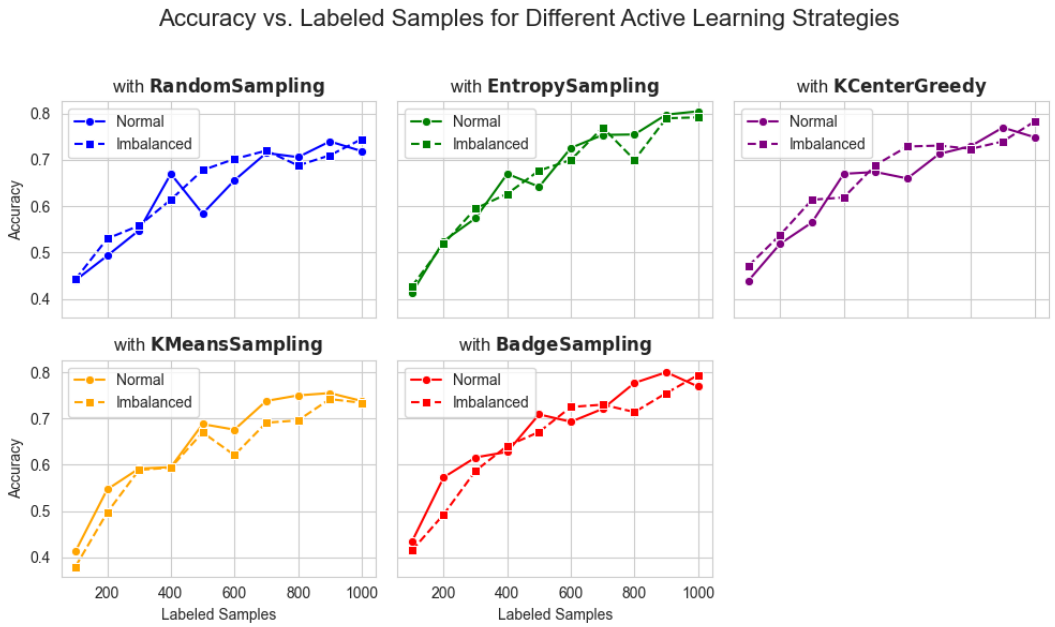
Table 6.2: Comparison of different active learning strategies based on accuracy, AUBC, and computational time.

6.2.2 Accuracy Across Samples



(a) Overall accuracy vs. budget curves on Imbalanced UCMerced.

6.3 Comparison Between Normal and Imbalanced UC Merced



(a) Comparison of Strategy Performance on Normal UCMerced vs. Imbalanced UCMerced Datasets

6.3.1 Key Differences

The results indicate a consistent gap in performance between the normal and imbalanced UC Merced datasets across all active learning strategies. Notably, the imbalanced dataset exhibits lower accuracy overall, suggesting that class imbalance negatively impacts model learning. Among the strategies, Random Sampling and Badge Sampling show relatively minor differences between the two datasets, indicating their robustness to class imbalance. In contrast, KMeans Sampling and KCenter Greedy display more variation, suggesting that these methods may be more sensitive to the distribution of labeled samples. The most affected strategy appears to be Entropy Sampling, which shows the largest performance gap. This suggests that uncertainty-based selection methods may struggle when applied to imbalanced datasets, as they might prioritize ambiguous samples that do not adequately represent minority classes. Despite these differences, all strategies demonstrate improvement in accuracy as more labeled samples are introduced, though the imbalanced dataset consistently lags behind its balanced counterpart.

6.3.2 Performance Insights

The overall trends suggest that Badge Sampling and KMeans Sampling provide more stable accuracy improvements over time, making them strong choices for active learning applications. Conversely, Entropy Sampling struggles the most in the imbalanced setting, likely due to its reliance on selecting highly uncertain samples, which may not always contribute to a well-represented training set. Random Sampling, while often seen as a baseline method, performs relatively well in this setting, as it does not introduce biases toward any particular subset of the data. However,

its learning efficiency remains lower compared to more sophisticated active learning approaches. KCenter Greedy exhibits some fluctuations in accuracy, indicating that core-set selection is influenced by dataset imbalance, potentially leading to suboptimal sample selection. While all methods benefit from additional labeled data, their ability to mitigate the effects of class imbalance varies significantly.

6.3.3 Potential Improvements

To address the challenges posed by class imbalance, several improvements could be implemented. First, data augmentation techniques such as oversampling the minority class (e.g., using SMOTE) or applying transformations like rotation and flipping could help balance the training set. Additionally, adaptive active learning approaches that explicitly consider class distribution when selecting new samples could lead to more balanced model performance. Another potential improvement involves uncertainty-aware training, where the models loss function is adjusted to penalize misclassification of minority class samples more heavily, ensuring better representation. Finally, a hybrid strategy combining the benefits of KMeans Sampling and Entropy Sampling could provide a more balanced approach, leveraging both diversity-based selection and uncertainty-driven sample acquisition. Future research should explore these refinements to optimize active learning performance, particularly in real-world scenarios where dataset imbalance is common.

Chapter 7

Conclusions

The experimental results demonstrate the effectiveness of various active learning strategies compared to random sampling when applied to the imbalanced UC Merced dataset. The analysis focuses on accuracy vs. labeled samples, showing that active learning strategies consistently outperform random sampling in terms of model performance.

Among the strategies evaluated, Entropy Sampling achieves the highest final accuracy, reaching approximately 80.9%, indicating its ability to prioritize informative samples for model training. Badge Sampling and KMeans Sampling show competitive performance, with AUBC (Area Under the Budget Curve) values of 0.648 ± 0.01 and 0.6237 ± 0.006 , respectively, reflecting their efficiency in selecting representative samples. Meanwhile, KCenter Greedy demonstrates a strong balance between accuracy and computational efficiency, making it a viable option for scenarios with limited computational resources.

A key observation from the results is that despite the dataset's class imbalance, active learning strategies remain robust, effectively improving model performance with fewer labeled samples. However, Random Sampling, while still improving with more labeled data, lags behind in accuracy and efficiency, highlighting the benefits of uncertainty- and diversity-based selection methods.

In addition to accuracy trends, computational efficiency is also considered. While KMeans Sampling provides slightly faster performance due to its clustering-based approach, strategies like Entropy Sampling and Badge Sampling require more computation but yield better accuracy improvements. This trade-off is crucial when selecting the optimal strategy for real-world applications.

This study confirms the advantages of active learning in handling class imbalance within remote sensing image classification tasks, specifically on the UC Merced dataset. The findings suggest that Entropy Sampling is the most effective strategy, achieving the highest accuracy while maintaining a reasonable computational cost. Badge Sampling and KMeans Sampling also show strong results, providing alternative strategies depending on the specific application constraints.

One of the most significant takeaways is that active learning substantially reduces the number of labeled samples required to reach high accuracy, which is particularly beneficial in real-world scenarios where labeling is expensive and time-consuming. Compared to Random Sampling, all active learning methods demonstrate superior sample efficiency, reinforcing their value in imbalanced dataset scenarios.

Despite these positive outcomes, the study also highlights challenges related to computational cost, especially for strategies that rely on complex uncertainty or

diversity measures. Future research should explore optimizations to reduce computational overhead while maintaining performance gains.

Moving forward, further evaluations on more complex, real-world datasets with higher class imbalance and label noise should be conducted. Additionally, integrating semi-supervised learning or self-supervised approaches with active learning may provide further improvements in classification performance.

Bibliography

[1] Zhan, Xueying and Wang, Qingzhong and Huang, Kuan-hao and Xiong, Haoyi and Dou, Dejing and Chan, Antoni B, *A comparative survey of deep active learning*. arXiv preprint arXiv:2203.13450, 2022.

[2] Yi Yang and Shawn Newsam, *Bag-of-visual-words and spatial extensions for land-use classification*, Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2010, pp. 270–279, doi: 10.1145/1869790.1869829.

[3] Rika, V., D'Aronco, S., Wegner, J. D., & Schindler, K. (2020). Deep Active Learning in Remote Sensing for Data Efficient Change Detection. *arXiv preprint* arXiv:2008.11201. Retrieved from https://arxiv.org/abs/2008.11201.

[4] Charu C. Aggarwal, Xiangnan Kong, Quanquan Gu, Jiawei Han, and S. Yu Philip. Active learning: A survey. In *Data Classification*, pages 599634. Chapman and Hall/CRC, 2014.

[5] Jordan T. Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. In *8th International Conference on Learning Representations, ICLR 2020*, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020.

[6] William H. Beluch, Tim Genewein, Andreas Nrnberger, and Jan M. Khler. The power of ensembles for active learning in image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 93689377, 2018.

[7] Erdem Byk, Kenneth Wang, Nima Anari, and Dorsa Sadigh. Batch active learning using determinantal point processes. *arXiv preprint arXiv:1906.07975*, 2019.

[8] Samuel Budd, Emma C. Robinson, and Bernhard Kainz. A survey on active learning and human-in-the-loop deep learning for medical image analysis. *Medical Image Analysis*, 71:102062, 2021.

[9] Arantxa Casanova, Pedro O. Pinheiro, Negar Rostamzadeh, and Christopher J. Pal. Reinforced active learning for image segmentation. *arXiv preprint arXiv:2002.06583*, 2020.

[10] Gavin C. Cawley. Baseline methods for active learning. In *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, pages 4757. JMLR Workshop and Conference Proceedings, 2011.

[11] Yao-Chun Chan, Mingchen Li, and Samet Oymak. On the marginal benefit of active learning: Does self-supervision eat its cake? In *ICASSP 2021-2021 IEEE*

International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 34553459. IEEE, 2021.

[12] Yukun Chen, Thomas A. Lasko, Qiaozhu Mei, Joshua C. Denny, and Hua Xu. A study of active learning methods for named entity recognition in clinical text. *Journal of Biomedical Informatics*, 58:1118, 2015.

[13] Jiwoong Choi, Ismail Elezi, Hyuk-Jae Lee, Clement Farabet, and Jose M. Alvarez. Active learning for deep object detection via probabilistic modeling. *arXiv preprint arXiv:2103.16130*, 2021.

[14] Wei Chu, Martin Zinkevich, Lihong Li, Achint Thomas, and Belle Tseng. Unbiased online active learning in data streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 195203, 2011.

[15] Gui Citovsky, Giulia DeSalvo, Claudio Gentile, Lazaros Karydas, Anand Rajagopalan, Afshin Rostamizadeh, and Sanjiv Kumar. Batch active learning at scale. *Advances in Neural Information Processing Systems*, 34, 2021.

[16] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 29212926. IEEE, 2017.

[17] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141142, 2012.

[18] Melanie Ducoffe and Frederic Precioso. Adversarial active learning for deep networks: A margin-based approach. *arXiv preprint arXiv:1802.09841*, 2018.

[19] Long Duong, Hadi Afshar, Dominique Estival, Glen Pink, Philip R. Cohen, and Mark Johnson. Active learning for deep semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 4348, 2018.

[20] Mehdi Elahi, Francesco Ricci, and Neil Rubens. A survey of active learning in collaborative filtering recommender systems. *Computer Science Review*, 20:2950, 2016.

[21] Linton C. Freeman and Linton C. Freeman. Elementary applied statistics: for students in behavioral science. New York: Wiley, 1965.

[22] Yifan Fu, Xingquan Zhu, and Bin Li. A survey on instance selection for active learning. *Knowledge and Information Systems*, 35(2):249283, 2013.

[23] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep Bayesian active learning with image data. In *International Conference on Machine Learning*, pages 11831192. PMLR, 2017.

[24] Yonatan Geifman and Ran El-Yaniv. Deep active learning over the long tail. *arXiv preprint arXiv:1711.00941*, 2017.

[25] Daniel Gissin and Shai Shalev-Shwartz. Discriminative active learning. *arXiv preprint arXiv:1907.06347*, 2019.

[26] B. Settles, *Active Learning Literature Survey*, University of Wisconsin-Madison, 2009.

[27] S. Karamcheti, R. Krishna, L. Fei-Fei, and C. D. Manning, *Mind Your Outliers! Investigating the Negative Impact of Outliers on Active Learning for Visual Question Answering*, arXiv preprint arXiv:2107.02331, 2021.

[28] V.-L. Nguyen, S. Destercke, and E. Hllermeier, *Epistemic Uncertainty Sampling*, in *Proceedings of the International Conference on Discovery Science*, pp. 7286, Springer, 2019.

[29] Robin Senge, Stefan Bsner, Krzysztof Dembczynski, Jrg Haasenritter, Oliver Hirsch, Norbert Donner-Banzhoff, and Eyke Hllermeier, *Reliable classification: Learning classifiers that distinguish aleatoric and epistemic uncertainty. Information Sciences*, 255:1629, 2014.

[30] C. E. Shannon, *A Mathematical Theory of Communication*, ACM SIGMOBILE Mobile Computing and Communications Review, vol. 5, no. 1, pp. 3–55, 2001.

[31] Y. Geifman and R. El-Yaniv, *Deep Active Learning Over the Long Tail*, arXiv preprint arXiv:1711.00941, 2017.

[32] O. Sener and S. Savarese, *Active Learning for Convolutional Neural Networks: A Core-Set Approach*, arXiv preprint arXiv:1708.00489, 2017.

Appendix A

Supporting Code for Active Learning Algorithms

A.1 Active Learning Strategy Implementations

A.1.1 Python Code for Active Learning Strategies

This appendix contains the Python implementations of the various active learning strategies used in this research. Each algorithm is designed to handle different aspects of dataset selection, model training, and evaluation. These implementations provide insight into the practical application of active learning techniques on the UC Merced dataset, demonstrating the logic and structure behind each strategy. The code is structured for reproducibility, allowing researchers to apply and extend these methods in future studies. Below, we present each strategy in detail, with accompanying explanations where necessary.

1. **Random Sampling :**
   Selects data points randomly from a dataset with no specific pattern. This method assumes that every data point in the dataset has an equal chance of being selected.

```
    def query(self, n):
        return np.random.choice(np.where(self.dataset.labeled_idxs==0)[0], n,
                                                    replace=False)
```

2. **Least Confidence :**
   Takes the difference between 1 (100% confidence) and the most confidently predicted label for each item.
   "The most informative examples are the ones that the classifier is the least certain about." Meaning that the examples for which the model has the least certainty will likely be the most difficult examples specifically, the examples that lie near the class boundaries. The learning algorithm will gain the most information about the class boundaries by observing the difficult examples.

```
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        probs = self.predict_prob(unlabeled_data)
        uncertainties = probs.max(1)[0]
        return unlabeled_idxs[uncertainties.sort()[1][:n]]
```

3. **Margin Sampling :**
   Assumes that the most informative data samples are those which fall within this margin and they are most likely to become new support vectors. We use a f(xi, w) which represents the distance of the data sample, where xi from the hyperplane for class w.
   (It can only select a single data sample for querying to oracle per iteration which increases the execution time.) *Based on an Active learning algorithm that selects difficult unlabeled tokens and asks the user to label them.*

```
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        probs = self.predict_prob(unlabeled_data)
        probs_sorted, idxs = probs.sort(descending=True)
        uncertainties = probs_sorted[:, 0] - probs_sorted[:,1]
        return unlabeled_idxs[uncertainties.sort()[1][:n]]
```

   Margin Sampling DROPOUT:
   Margin sampling considers the difference between the confidence of first and the second most probable labels. Dropout is like selective blindness .
   n_drop: Number of dropout runs (int, default here =10)

```
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        probs = self.predict_prob_dropout(unlabeled_data, n_drop=self.n_drop)
        probs_sorted, idxs = probs.sort(descending=True)
        uncertainties = probs_sorted[:, 0] - probs_sorted[:,1]
        return unlabeled_idxs[uncertainties.sort()[1][:n]]
```

4. **Entropy Sampling :**
   Entropy, or Shannon entropy refers to the informational value present in a variable.
   Entropy in an information-theoretic measure that represents the amount of information needed to encode a distribution.

$$E = \sum_{i}^{n} p_i \log \frac{1}{p_i}$$

```
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        probs = self.predict_prob(unlabeled_data)
        log_probs = torch.log(probs)
        uncertainties = (probs*log_probs).sum(1)
        return unlabeled_idxs[uncertainties.sort()[1][:n]]
```

   Entropy Sampling DROPOUT:
   Often in recommendation systems, the output of an algorithm is a group of lists which score items for each user. In our case of an item-based collaborative filtering, we obtain a user-item matrix of similarity scores as the raw output from the machine learning model. A higher score as an element in the user-item matrix means the item is more relevant to the user.

   n_drop: Number of dropout runs (int, default here =10)

```
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        probs = self.predict_prob_dropout(unlabeled_data, n_drop=self.n_drop)
```

```
        log_probs = torch.log(probs)
        uncertainties = (probs*log_probs).sum(1)
        return unlabeled_idxs[uncertainties.sort()[1][:n]]
```

5. **Uncertainty Sampling with Dropout Estimation :**

```
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        probs = self.predict_prob(unlabeled_data)
        log_probs = torch.log(probs)
        uncertainties = (probs*log_probs).sum(1)
        return unlabeled_idxs[uncertainties.sort()[1][:n]]
```

6. **Bayesian Active Learning Disagreement :**
   In a Bayesian neural network, every parameter in the model is sampled from a distribution. Then, when doing inference, we need to integrate over all the possible parameters. So were using an ensemble of infinite different networks to compute the output.
   The measure of uncertainty here is the fraction of models, across MC samples from the network, that disagree with most popular choice.

$$argmax_j(1 - \frac{count(mode(\widetilde{y}_j^{(1)}, ..., \widetilde{y}_j^{(T)}))}{T})$$

```
    //BALDDropout
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        probs = self.predict_prob_dropout_split(unlabeled_data, n_drop=self.
                                                n_drop)
        pb = probs.mean(0)
        entropy1 = (-pb*torch.log(pb)).sum(1)
        entropy2 = (-probs*torch.log(probs)).sum(2).mean(0)
        uncertainties = entropy2 - entropy1
        return unlabeled_idxs[uncertainties.sort()[1][:n]]
```

7. **Adversarial margin :**
   By selecting data points where the adversarial attack significantly impacts the model's confidence (increases margin), the model focuses on learning the boundaries between different classes where it's susceptible to adversarial manipulation. This strategy aims to improve the model's ability to handle adversarial examples and enhance its overall robustness.
   The function cal_dis has a different structure for each algorithm.

```
    //Adversarial BIM
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()

        self.net.clf.cpu()
        self.net.clf.eval()
        dis = np.zeros(unlabeled_idxs.shape)

        for i in tqdm(range(len(unlabeled_idxs)), ncols=100):
            x, y, idx = unlabeled_data[i]
            dis[i] = self.cal_dis(x)

        self.net.clf.cuda()

        return unlabeled_idxs[dis.argsort()[:n]]
```

Adversarial DEEPFOOL :
This algorithm takes **more than 12hours** to run , and it does so only by accessing the GPU in order to accelerate.

```
    //Adversarial DEEPFOOL
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()

        self.net.clf.cpu()
        self.net.clf.eval()
        dis = np.zeros(unlabeled_idxs.shape)

        for i in tqdm(range(len(unlabeled_idxs)), ncols=100):
            x, y, idx = unlabeled_data[i]
            dis[i] = self.cal_dis(x)

        self.net.clf.cuda()

        return unlabeled_idxs[dis.argsort()[:n]]
```

8. **Mean Standard Deviation :**
   Characteristic: Using Monte Carlo dropout uncertainty maps are computed for all three CNNs by retrieving 10 Monte Carlo samples from the networks and then computing the standard deviation over the softmax outputs of the samples.

```
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        probs = self.predict_prob_dropout_split(unlabeled_data, n_drop=self.
                                                n_drop).numpy()
        sigma_c = np.std(probs, axis=0)
        uncertainties = torch.from_numpy(np.mean(sigma_c, axis=-1))
        return unlabeled_idxs[uncertainties.sort(descending=True)[1][:n]]
```

9. **Variation Ratios :**
   Characteristic: Using Monte Carlo dropout uncertainty maps are computed for all three CNNs by retrieving 10 Monte Carlo samples from the networks and then computing the standard deviation over the softmax outputs of the samples.

```
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        probs = self.predict_prob_dropout_split(unlabeled_data, n_drop=self.
                                                n_drop).numpy()
        sigma_c = np.std(probs, axis=0)
        uncertainties = torch.from_numpy(np.mean(sigma_c, axis=-1))
        return unlabeled_idxs[uncertainties.sort(descending=True)[1][:n]]
```

10. **Cost-Effective Active Learning**

```
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        probs = self.predict_prob_dropout_split(unlabeled_data, n_drop=self.
                                                n_drop).numpy()
        sigma_c = np.std(probs, axis=0)
        uncertainties = torch.from_numpy(np.mean(sigma_c, axis=-1))
        return unlabeled_idxs[uncertainties.sort(descending=True)[1][:n]]
```

11. **KMeans with scikit-learn library (PCA)**

```
    def query(self, n):
        labeled_idxs, train_data = self.dataset.get_train_data()
        embeddings = self.get_embeddings(train_data)
        embeddings = embeddings.numpy()

        #downsampling embeddings if feature dim > 50
        if len(embeddings[0]) > 50:
            pca = PCA(n_components=50)
            embeddings = pca.fit_transform(embeddings)
        embeddings = embeddings.astype(np.float16)

        dist_mat = np.matmul(embeddings, embeddings.transpose())
        sq = np.array(dist_mat.diagonal()).reshape(len(labeled_idxs), 1)
        dist_mat *= -2
        dist_mat += sq
        dist_mat += sq.transpose()
        dist_mat = np.sqrt(dist_mat)

        mat = dist_mat[~labeled_idxs, :][:, labeled_idxs]

        for i in tqdm(range(n), ncols=100):
            mat_min = mat.min(axis=1)
            q_idx_ = mat_min.argmax()
            q_idx = np.arange(self.dataset.n_pool)[~labeled_idxs][q_idx_]
            labeled_idxs[q_idx] = True
            mat = np.delete(mat, q_idx_, 0)
            mat = np.append(mat, dist_mat[~labeled_idxs, q_idx][:, None], axis
                                                    =1)

        return np.arange(self.dataset.n_pool)[(self.dataset.labeled_idxs ^
                                               labeled_idxs)]
```

KMeans with faiss-gpu library (GPU)

```
        def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        embeddings = self.get_embeddings(unlabeled_data).numpy()
        cluster_learner = FaissKmeans(n_clusters = n, gpu = True)
        cluster_learner.fit(embeddings)
        dis, q_idxs = cluster_learner.predict(embeddings)
        q_idxs = q_idxs.T[0]

        return unlabeled_idxs[q_idxs]
```

12. Batch Active learning by Diverse Gradient Embeddings

```
    //BadgeSampling
    def query(self, n):
        unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
        gradEmbedding = self.get_grad_embeddings(unlabeled_data)
        chosen = init_centers(gradEmbedding, n)
        return unlabeled_idxs[chosen]
```

13. Loss Prediction Active Learning

```
    def query(self, n):
    unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
    uncertainties = self.unc_lpl(unlabeled_data)
    return unlabeled_idxs[uncertainties.sort(descending=True)[1][:n]]
```

14. Variational Adversarial Active Learning

```
    def query(self, n):
    unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
    self.train_vaal()
    uncertainties = self.pred_dis_score_vaal(unlabeled_data)
    return unlabeled_idxs[uncertainties.sort(descending=True)[1][:n]]
```

15. **Wasserstein Adversarial Active Learning**

```
    def query(self, n):
    unlabeled_idxs, unlabeled_data = self.dataset.get_unlabeled_data()
    probs = self.predict_prob(unlabeled_data)
    uncertainly_score = 0.5* self.L2_upper(probs) + 0.5* self.L1_upper(probs)

    # prediction output discriminative score
    dis_score = self.pred_dis_score_waal(unlabeled_data)

    # computing the decision score
    total_score = uncertainly_score - self.selection * dis_score
    b = total_score.sort()[1][:n]

    return unlabeled_idxs[total_score.sort()[1][:n]]
```

A.2 Preprocessing-Utility Functions for Active Learning

This appendix provides an overview of key supporting functions used in the DeepAL+ toolkit that play a crucial role in the querying strategies. These functions facilitate model prediction, uncertainty estimation, and dropout-based uncertainty measurement.

A.2.1 `predict_prob`

The `predict_prob` function computes the predicted probabilities for a given dataset using a trained deep learning model. It is fundamental for uncertainty-based strategies like entropy sampling.

```
def predict_prob(model, data_loader, device):
    model.eval()
    probs = []
    with torch.no_grad():
        for x in data_loader:
            x = x.to(device)
            output = model(x)
            probs.append(torch.softmax(output, dim=1).cpu().numpy())
    return np.concatenate(probs, axis=0)
```

Purpose: The model outputs are converted into probability distributions using the softmax function, allowing for more interpretable confidence scores. This transformation is essential for uncertainty-based query strategies, such as entropy and margin sampling, which rely on these probability distributions to identify the most informative samples for labeling.

A.2.2 predict_prob_dropout_split

The predict_prob_dropout_split function performs **Monte Carlo (MC) Dropout** to estimate model uncertainty. This function is critical for Bayesian-inspired active learning methods.

```
def predict_prob_dropout_split(model, data_loader, device, T=10):
    model.train()  # Enable dropout at inference
    all_probs = []
    for _ in range(T):
        probs = []
        with torch.no_grad():
            for x in data_loader:
                x = x.to(device)
                output = model(x)
                probs.append(torch.softmax(output, dim=1).cpu().numpy())
        all_probs.append(np.concatenate(probs, axis=0))
    return np.stack(all_probs)
```

Purpose: - Runs multiple forward passes with dropout enabled. - Used in **Bayesian Active Learning** to estimate predictive uncertainty.

A.2.3 get_uncertainty_scores

This function computes **uncertainty scores** for each unlabeled sample based on entropy, margin, or variance across multiple MC Dropout passes.

```
def get_uncertainty_scores(probabilities, method="entropy"):
    if method == "entropy":
        return -np.sum(probabilities * np.log(probabilities + 1e-10), axis=1)
    elif method == "margin":
        sorted_probs = np.sort(probabilities, axis=1)
        return sorted_probs[:, -1] - sorted_probs[:, -2]
    elif method == "variance":
        return np.var(probabilities, axis=0).sum(axis=1)
```

Purpose: - Computes uncertainty for various **query strategies**. - Supports entropy-based and margin-based selection.

A.2.4 Integration into Query Strategies

These functions are integrated into DeepAL+ query strategies as follows:

1. **Entropy Sampling**: Implements predict_prob and get_uncertainty_scores based on entropy computations.
2. **Bayesian Active Learning:** Utilizes predict_prob_dropout_split to gauge uncertainty using Monte Carlo Dropout.
3. **Margin Sampling:** Employs get_uncertainty_scores alongside the margin method.

A.3 Code for Generating the Imbalanced Dataset

This function prints the class distribution of a Dataset, where Y is the target labels and *name* is the name of the dataset.

```
def print_class_distribution(Y, name):
    unique, counts = np.unique(Y, return_counts=True)
    print(f"Class distribution in {name}:")
    for cls, count in zip(unique, counts):
        print(f"Class {cls}: {count} samples")
```

```
def create_imbalance(data, imbalance_ratios):
    """
    Create an imbalanced dataset by
    reducing the number of samples for certain classes.

    Parameters:
    - data: A dictionary with class labels
    as keys and lists of samples as values.
    - imbalance_ratios: A dictionary with class labels
    as keys and imbalance ratios as values.

    Returns:
    - imbalanced_data: A dictionary with the imbalanced dataset.
    """
    imbalanced_data = {}
    for class_label, samples in data.items():
        if class_label in imbalance_ratios:
            ratio = imbalance_ratios[class_label]
            reduced_samples =
            random.sample(samples, int(len(samples) * ratio))
            imbalanced_data[class_label] = reduced_samples
        else:
            imbalanced_data[class_label] = samples
    print("Class distribution after imbalance:")
    for class_label, samples in imbalanced_data.items():
        print(f"Class {class_label}: {len(samples)} samples")
    return imbalanced_data
```

```
def oversample_data(X, Y):
    """
    Oversample the minority classes in the dataset to balance the class distrib

    Parameters:
    - X: List of data samples.
    - Y: List of labels corresponding to the data samples.

    Returns:
    - X_resampled: List of resampled data samples.
```

```
- Y_resampled: List of resampled labels.
"""

# Convert lists to numpy arrays for compatibility with imblearn
X = np.array(X)
Y = np.array(Y)
# Reshape X to 2D array for oversampling
X_reshaped = X.reshape((X.shape[0], -1))
# Apply RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled, Y_resampled = ros.fit_resample(X_reshaped, Y)
# Reshape X_resampled back to original  shape
X_resampled = X_resampled.reshape((X_resampled.shape[0],) + X.shape[1:])

return X_resampled, Y_resampled
```