CS771: Introduction to Machine Learning
Mini Project 2

Aryan Singh
220229
Shashwat Agarwal
221004
Bhaumik Chawda
220292
Shubhanshu Mishra
221048

November 26, 2024

Group Name: Linear Depression

1 Task 1: Continual Unsupervised Domain Adaptation on CIFAR-10

In this task, we investigate the application of Learning with Prototypes (LwP) to subsets of the CIFAR-10 dataset. The objective is to iteratively train on labeled and unlabeled data while minimizing degradation in performance on previous datasets. We conducted experiments on a specific subset of the CIFAR-10 dataset, with each experiment focusing on different strategies for leveraging labeled and unlabeled data and finally applying the following strategy.

1.1 Task 1.1: Incremental Learning Using Feature Extraction and Prototype Classifier

This approach employs EfficientNet-B3 for feature extraction and a Prototype Classifier for incremental learning, aiming for better performance in continual learning.

Stage 1: Feature Extraction with EfficientNet-B3 The EfficientNet-B3 model, pre-trained on ImageNet, is used for feature extraction:

- The fully connected (FC) layer is removed, and the output of the final convolutional layer is used as the feature vector.
- Input images are resized to 300×300, and normalization is applied using ImageNet's statistics:
  - Mean = `[0.485, 0.456, 0.406]`
  - Std = `[0.229, 0.224, 0.225]`
- Features are extracted for both training and evaluation datasets and saved as .npy files for downstream processing.

Stage 2: Prototype Classifier for Incremental Training The Prototype Classifier uses feature vectors to incrementally learn class prototypes:

- **Class Prototypes:** Each class is represented by a prototype, computed as the mean of its feature vectors. The prototypes are updated incrementally during training.
- **Classifier Training:**
  - During initial training, true labels are used to calculate prototypes for the first dataset.
  - For subsequent datasets, pseudo-labels are generated by the classifier when true labels are unavailable.
  - The prototypes are updated using a weighted average of the existing prototype and new feature vectors.
- **Prediction:** For a given feature vector, the classifier predicts the class with the closest prototype based on the Euclidean distance.

Training Process: The model is trained incrementally on pre-extracted features for datasets `D1` through `D10`:

- Features for each dataset are loaded from .npy files.
- If true labels are available, they are used to train or update the classifier. Otherwise, pseudo-labels are generated.
- After training on each dataset, the classifier is saved as a checkpoint for evaluation.

Evaluation Process: The saved models are evaluated on all test datasets `D1` through `D10`:

- The classifier predicts labels for each dataset and computes the accuracy.
- The results are compiled into an accuracy matrix, where each cell represents the accuracy of a model (`f1` to `f10`) on a specific test dataset (`D1` to `D10`).

Accuracy Matrix Visualization: The accuracy matrix is visualized as a heatmap to analyze model performance across datasets. The heatmap highlights:

- The stability of the classifier as it learns new datasets.
- Incremental improvements in performance with minimal degradation on previous datasets.

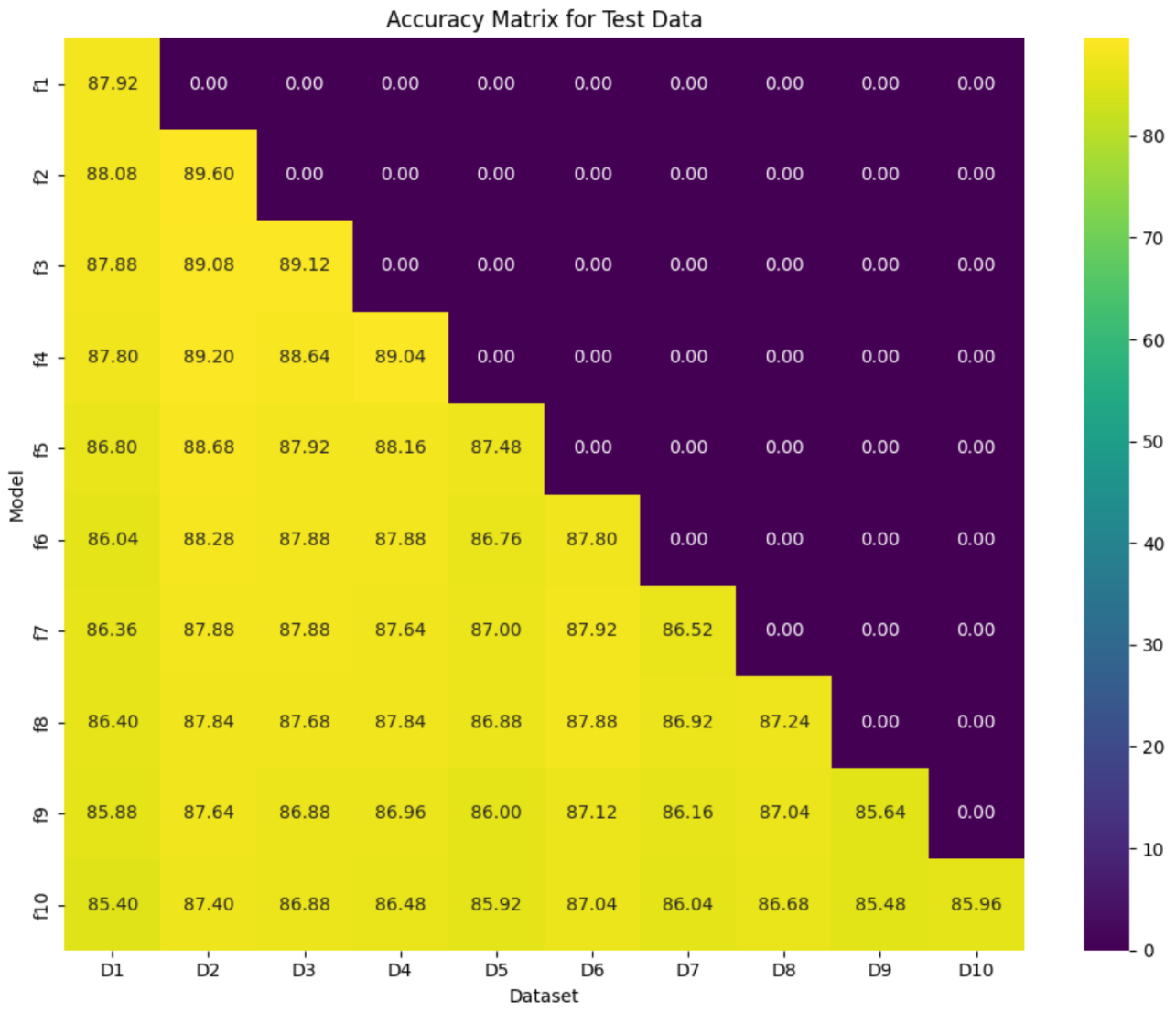

Figure 1: Accuracy Matrix Heatmap for Models `f1` to `f10` across Test Datasets `D1` to `D10`.

Results Summary: The approach demonstrated the following:

- Consistent improvement in accuracy as more datasets were processed.
- Outperformed other approaches in stability and performance on unseen datasets.
- Challenges in pseudo-label generation, particularly for later datasets where the model's knowledge was less robust.

1.2 Task 1.2: Evaluation and Performance Analysis

The goal of Task 1.2 is to systematically evaluate the models trained incrementally on labeled and unlabeled subsets of CIFAR-10 (`D11` to `D20`) and analyze their performance across all datasets where each dataset has a different distribution. This section details the approach used, the metrics computed, and the key insights derived from the evaluation.

Evaluation Setup: The evaluation process focuses on testing each model trained incrementally (`f11` to `f20`) on all test datasets (`D1` to `D20`) to generate a comprehensive accuracy matrix. The specific steps are as follows:

- Models `f11` to `f20` are trained incrementally on datasets `D11` to `D20` using the LwP classifier. Pseudo-labels are generated for unlabeled datasets.
- Each trained model is evaluated on its current and previous datasets to measure its ability to generalize across earlier datasets (to assess knowledge retention) and newer datasets (to assess learning efficacy).
- Accuracy is computed as the percentage of correctly predicted labels for each dataset.

Incremental Learning Process: The incremental learning framework from Task 1.1 is extended in this problem too where we observe datasets `D11` to `D20` with different distributions. Reason why the approach still works fine here is as follows:

- **Pretrained Knowledge:** EfficientNet-B3 is pretrained on large and diverse datasets (e.g., ImageNet), enabling it to extract robust, generalizable features that are useful even in the presence of distribution shifts.
- **Transfer Learning:** The model can transfer learned knowledge from earlier datasets to newer ones, helping it adapt to new data distributions incrementally.
- **Efficient Feature Extraction:** EfficientNet-B3's architecture is optimized for feature extraction, capturing high-level patterns that are less sensitive to domain-specific shifts.
- **Prototype-Based Learning:** The use of prototypes for class representation helps the model generalize across datasets, even when the data distribution changes.

Testing Methodology: Each trained model is evaluated on all test datasets following these steps:

1. **Feature Extraction:**
   - Features are extracted for each dataset using the pre-trained EfficientNet-B3 model, as described in Task 1.1.
   - Features are normalized to ensure consistency with the training pipeline.
2. **Prediction:**
   - For each feature vector, the model calculates distances to all class prototypes.
   - The class corresponding to the closest prototype (using Euclidean distance) is assigned as the predicted label.

Performance Visualization: To facilitate analysis, the accuracy matrix is visualized as a heatmap:

- The heatmap uses a color gradient to represent accuracy, with brighter colors indicating higher accuracy.
- Rows show how well a specific model retains knowledge of earlier datasets while learning new datasets.
- Columns highlight how performance evolves on a specific dataset as the model is trained on successive datasets.

Results Analysis: The key findings from the evaluation are as follows:

- **Knowledge Retention:**
  - Models retain high accuracy on earlier datasets (`D1` to `D10`) even after being trained on subsequent datasets.
  - Minimal degradation is observed, indicating effective mitigation of catastrophic forgetting.
- **Learning New Datasets:**
  - Accuracy on newer datasets (`D11` to `D20`) improves steadily, demonstrating the model's ability to adapt incrementally.
  - Challenges are noted for classes with significant distributional shifts, affecting pseudo-label quality.
- **Impact of Pseudo-Labels:**
  - High-confidence pseudo-labels contribute positively to prototype updates and performance.
  - Low-confidence pseudo-labels introduce noise, particularly in later datasets, necessitating improved confidence filtering techniques.

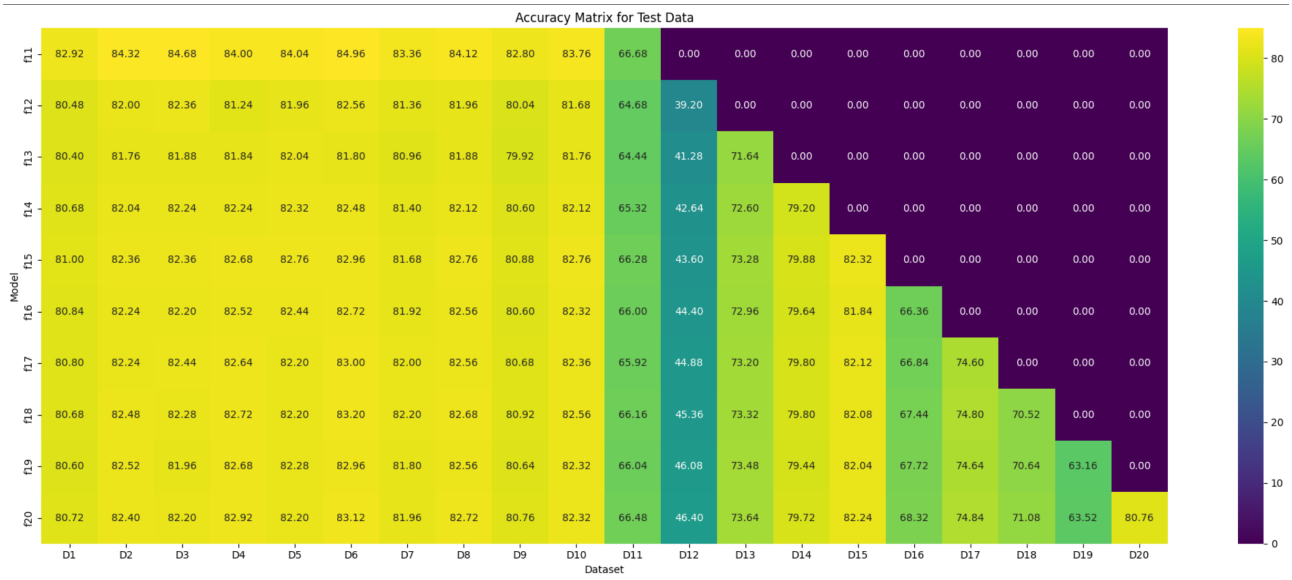

Figure 2: Accuracy Matrix Heatmap for Models `f11` to `f20` Across Test Datasets `D1` to `D20`.

Challenges and Limitations:

- **Pseudo-Label Noise:** Pseudo-labels for datasets with high class overlap or distribution shifts are less reliable.
- **Class Imbalance:** Uneven class distributions in unlabeled datasets affect prototype updates.
- **Computational Complexity:** Incremental updates for prototypes and pseudo-labeling increase computational overhead.

2 Task 2: Research Paper Analysis and Presentation

For Part 2 of the project, we focused on analyzing a research paper related to continual learning and domain adaptation. The paper, titled " Lifelong Domain Adaptation via Consolidated Internal Distribution (NeurIPS 2021)", explores techniques for improving learning systems that can adapt over time while retaining prior knowledge. We were tasked with presenting a summary of the paper's key ideas, methodologies, and contributions.

Research Paper Overview: The paper addresses key challenges in Continual Learning (CL), particularly catastrophic forgetting and the difficulty of adapting to distributional shifts across sequential tasks. It proposes new methods to mitigate these challenges, allowing models to incrementally learn from new data without losing knowledge of previously learned tasks. Key contributions of the paper include:

- A novel framework designed for handling sequential learning tasks with minimal interference, facilitating the learning of new tasks while retaining prior knowledge.
- A technique for consolidating learned internal distributions, using prototypes and memory buffers, to improve generalization to new domains without requiring simultaneous access to source and target datasets.
- A comprehensive empirical evaluation of the proposed methods on several benchmark datasets, demonstrating state-of-the-art performance in continual learning scenarios, even in the presence of domain shifts.

Presentation and Video: Our group created a presentation summarizing the paper's findings and explained the key concepts in detail. The presentation was recorded as a video, which highlights the main ideas and provides insights into how these methods could be applied to real-world problems. The video can be accessed through the following link:

`https://youtu.be/dOVNQaVNMcA`

Additionally, the full paper can be accessed at the following URL:

`https://proceedings.neurips.cc/paper_files/paper/2021/file/5caf41d62364d5b41a893adc1a9dd5d4-Paper.pdf`

References

[1] TensorFlow. EfficientNetB3 API Documentation. Available at: `https://www.tensorflow.org/api_docs/python/tf/keras/applications/EfficientNetB3`