CS771 MINI PROJECT-2

Riyanshu Kumar
Roll no: 220904

Atharva Singh
Roll no: 220251

Kawaljeet Singh
Roll no: 220514

Nikhil Jain
Roll no: 220709

Introduction

This project explores the application of **Learning with Prototypes (LwP)** in the context of *continual learning* and *domain adaptation* for image classification on the CIFAR-10 dataset. The work is divided into two primary tasks:

Task 1: Continual Learning with Homogeneous Input Distributions

A prototype-based model (f_1) is trained in the first task using a labeled dataset (D_1). This model is then iteratively updated (f_1 to f_{10}) by utilizing predictions on additional datasets (D_2 to D_{10}), which share similar input distributions. The updated models are evaluated on corresponding held-out labeled datasets ($\hat{D}_1$ to $\hat{D}_{10}$), and the accuracies are recorded in a matrix. The primary objective is to ensure that the models maintain high performance on both newly added datasets and previously evaluated datasets.

Task 2: Continual Learning with Heterogeneous Input Distributions

The final model from Task 1 (f_{10}) is the starting point in the second task. The model is further updated (f_{11} to f_{20}) using datasets (D_{11} to D_{20}), which are drawn from distinct input distributions. The models are evaluated on all held-out datasets ($\hat{D}_1$ to $\hat{D}_{20}$), emphasizing the ability to adapt to new distributions while retaining performance on earlier datasets.

1 Task 1

1.1 Feature Extraction

We have used **Vision Transformer (ViT)** for feature extraction. ViT, introduced by Google in 2020, applies the Transformer architecture to image data. It leverages the self-attention mechanism to encode spatial relationships and content within an image, enabling it to extract meaningful feature representations.

1. **Feature Extractor:**
   - The `ViTFeatureExtractor` is utilized for preprocessing input images. This involves:
     - Resizing images to 224×224 pixels, which is the required input size for ViT models.
     - Normalizing pixel values to match the model's training distribution.
2. **Transformer Backbone (`ViTModel`):**
   - The pretrained model `vit-base-patch16-224-in21k`, trained on ImageNet-21k, is used.
   - The model architecture includes:
     - 12 transformer encoder layers.
     - Each encoder layer has 12 attention heads.
   - Outputs of the model:
     - `last_hidden_state`: A sequence of embeddings for all tokens, including the `[CLS]` token.

3. **Feature Pooling:**
    - Applies **mean pooling** on the `last_hidden_state`.
    - This operation averages the embeddings across all tokens to produce a global feature vector representing the entire image.
4. **Output:**
    - The final output is a NumPy array containing feature vectors.
    - Each feature vector corresponds to a specific input image, ready for downstream tasks.

Pretraining: ViT models are pretrained on the **ImageNet** dataset. This can either be:

- **ImageNet-1k:** Contains 1.2 million images with 1,000 classes.
- **ImageNet-21k:** Contains 14 million images with 21,843 classes.

Pretraining on ImageNet helps ViT models learn fundamental visual features, such as edges, textures, and shapes, which are crucial for effective feature extraction.

1.2 Approach

The code implements a sequential learning approach to handle datasets D_1 to D_{10} in Task 1. Initially, features are extracted from raw images using a pre-trained Vision Transformer (ViT), providing compact and meaningful representations of input images. The process starts by training an LwP classifier on the labeled dataset D_1, where the mean feature vector of each class is computed to form the initial model f_1.
For subsequent datasets (D_2 to D_{10}), pseudo-labels for unlabeled data are generated based on the nearest class mean, determined by computing the Euclidean distance between the input features and the current class means. To update the model f_i to f_{i+1}, the pseudo-labeled data from D_{i+1} is used to recompute the class means. Specifically, the features of all pseudo-labeled samples belonging to a class are averaged to compute the new mean for that class. This iterative update ensures that the model adapts to the characteristics of the new dataset without revisiting the previous datasets.

1.3 Accuracy Matrix

```
Accuracy Matrix for Task-1:
[[0.9436 0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.9124 0.9108 0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.8972 0.8972 0.9004 0.     0.     0.     0.     0.     0.     0.    ]
 [0.8964 0.8968 0.8996 0.9016 0.     0.     0.     0.     0.     0.    ]
 [0.8944 0.8912 0.8948 0.8976 0.896  0.     0.     0.     0.     0.    ]
 [0.892  0.8912 0.8956 0.894  0.8948 0.8972 0.     0.     0.     0.    ]
 [0.8912 0.8932 0.8952 0.898  0.8936 0.898  0.8964 0.     0.     0.    ]
 [0.8908 0.892  0.8944 0.8972 0.894  0.8984 0.8976 0.8928 0.     0.    ]
 [0.8912 0.8888 0.892  0.896  0.896  0.8976 0.8936 0.8916 0.8904 0.    ]
 [0.8892 0.886  0.8896 0.8892 0.8876 0.892  0.8888 0.8872 0.8852 0.8928]]
```

Figure 1: Accuracy Matrix for Task-1

2 Task 2

2.1 Feature Extraction

We have also used **VIT** for feature extraction for this task.

2.2 Approach

The code extends the sequential learning framework from Task 1 to handle datasets D_{11} to D_{20}, accounting for their potentially diverse input distributions. Starting with the class means derived from the final model (f_{10}) of Task 1, the process involves iteratively updating the model to $f_{11}, f_{12}, \ldots, f_{20}$. For each dataset d_i, features are extracted using a pre-trained Vision Transformer (ViT). Pseudo-labels are then generated for the unlabeled data by assigning each input to the nearest class mean, determined using Euclidean distance. To enhance robustness, a confidence-based filtering mechanism retains only high-confidence samples for further processing. The class means are updated iteratively using these high-confidence samples, employing an exponential moving average to consolidate new and existing means.

To aid in better adaptation to evolving distributions, a memory buffer is maintained to store a fixed number of representative samples from each dataset. This buffer ensures that the model retains critical information about previously seen data while staying within a fixed storage budget. When adding new samples to the buffer, the oldest samples are replaced if the budget is exceeded, ensuring that the buffer remains relevant and efficient. The memory buffer plays a crucial role in mitigating catastrophic forgetting and supports consistent performance across both new and previous datasets.

2.3 Accuracy Matrix

```
Accuracy Matrix for Task 2:
[[0.8924 0.8928 0.8944 0.8964 0.892  0.898  0.8948 0.8948 0.8936 0.8964
  0.6988 0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.8972 0.8988 0.898  0.8956 0.8988 0.9012 0.8992 0.896  0.8992 0.8988
  0.7136 0.5972 0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.8988 0.8984 0.9004 0.8988 0.9    0.902  0.9012 0.8952 0.9    0.898
  0.7128 0.5936 0.7764 0.     0.     0.     0.     0.     0.     0.    ]
 [0.896  0.8968 0.8988 0.8988 0.8996 0.9032 0.8992 0.8964 0.8996 0.8988
  0.7112 0.5848 0.7724 0.8152 0.     0.     0.     0.     0.     0.    ]
 [0.8936 0.8964 0.8968 0.8976 0.8964 0.9    0.9004 0.8948 0.8992 0.9
  0.7064 0.574  0.7652 0.812  0.872  0.     0.     0.     0.     0.    ]
 [0.8968 0.8988 0.8992 0.8992 0.8996 0.8988 0.902  0.8964 0.8996 0.8988
  0.7096 0.578  0.7744 0.8136 0.8712 0.7596 0.     0.     0.     0.    ]
 [0.8996 0.9016 0.9008 0.904  0.9052 0.9044 0.9052 0.9016 0.9024 0.9028
  0.7132 0.5808 0.7784 0.8196 0.8748 0.7644 0.6764 0.     0.     0.    ]
 [0.9032 0.9012 0.9024 0.9    0.9028 0.904  0.9056 0.8992 0.902  0.9036
  0.7148 0.5912 0.7812 0.8196 0.8756 0.762  0.6784 0.7404 0.     0.    ]
 [0.8976 0.8976 0.9008 0.8972 0.8988 0.9028 0.9024 0.8964 0.8964 0.8964
  0.7092 0.5844 0.7708 0.8148 0.8716 0.7556 0.6736 0.7324 0.6924 0.    ]
 [0.8952 0.898  0.8992 0.8984 0.8992 0.9012 0.9016 0.8944 0.898  0.8996
  0.7112 0.5756 0.7668 0.814  0.8716 0.7536 0.668  0.728  0.6812 0.8064]]
```

Figure 2: Accuracy Matrix for Task-2

Additional Resources

For the video presentation, please click on this link: **Video Presentation**
We have extracted features for all the datasets and uploaded them to GitHub. You can access it using the following link: **Mini Project 2**