CS771 - Intro to ML : Mini Project 2
Group 36 - CerebroX

Ashvani Kumar Yadav	Khushwant Kaswan	Sangharsh Nagdevte	Souvik Sarkar
241110013	241110035	241110064	231160402

Contents

1 Problem 1: LwP Classifier on CIFAR-10 Dataset

1.1 Feature Extraction Using EfficientNet_B3

In our implementation, EfficientNet_B3 [2], a widely recognized and efficient convolutional neural network architecture trained on ImageNet-1k at resolution 300x300, is employed as the feature extractor. EfficientNet_B3, like other EfficientNet models, is initially trained on large-scale image classification datasets like ImageNet-1K and ImageNet-22K.

Key features :

- **Pretrained Weights:** EfficientNet_B3 is typically initialized with weights from models pre-trained on large datasets such as ImageNet1K. This initialization allows the model to leverage general image features learned from a wide range of objects and scenes.
- **Parameter Efficiency:** EfficientNet_B3 strikes a balance between accuracy and computational speed, making it ideal for tasks where resources are constrained.
- **Feature Extraction:** The core layers of EfficientNet_B3 act as general feature extractors that are transferable to new datasets. These layers extract useful patterns from the images, which can be fine-tuned for various downstream tasks such as classification, segmentation, and detection.

1.1.1 Feature Extraction Pipeline

- EfficientNet_B3 is loaded with pre-trained weights [1] (IMAGENET1K_V1) and set to evaluation. We have used the Pytorch torchvision implementation.
- Before passing data to EfficientNet_B3, we have resized to 224x224, normalized, and converted to tensors.
- Features are extracted in batches to optimize memory usage. Using the `torch.no_grad()` context, the model processes each batch, and the resulting features are flattened and stored.

1.1.2 Feature Precomputation

- The datasets are processed to extract and store features for both training and evaluation. For each training dataset and evaluation dataset, the features are extracted and cached, and stored as files using Pytorch: `features_cache.pth` and `eval_features_cache.pth`.
- These files are later loaded in our Task 1 and Task 2.
- Link to these precomputed features: `https://drive.google.com/drive/folders/1iGbY-n8NhaeQw54OHYYztDX5sMSFg17o?usp=sharing`

1.2 Task 1 Implementation

1. **Feature Representation**
   EfficientNet is used to precompute feature representations for all the datasets $D_1, \ldots, D_{10}$ and also for all evaluation datasets $\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_{10}$. After the feature extraction pepeline, the features are saved as `features_cache1.pth` and `eval_features_cache1.pth` and loaded for the task.
2. **LwP Classifier**
   Works by calculating the distance of test features from class prototypes (mean feature vectors for each class) created during training. Classification is done by assigning the label of the nearest prototype (e.g. Euclidean distance).
3. **Training and Update**
   - f_1 is trained using extracted features and labels of D_1.
   - The model f_i is updated to f_{i+1} by averaging the existing prototype and the new mean:

$$updated\_prototype = \frac{old\_prototype + new\_mean}{2}$$

where *new_mean* is calculated on the current dataset features with the help of predicted labels.

- After predicting labels for $D_2, \ldots, D_{10}$:
  - Recompute class prototypes using the predicted labels to update the LwP.

4. **Evaluation**
   - For each f_i, evaluate performance on the held-out dataset $\hat{D}_i$ and all prior held-out datasets $\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_{i-1}$ and create a tabular representation 10x10.
5. **Save the Final Model**
   Save the final f_{10} model to used later in Task 1.2.

1.2.1 Task 1 Results

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
f1	0.8856									
f2	0.8772	0.8896								
f3	0.8744	0.8856	0.8816							
f4	0.8744	0.8860	0.8808	0.8868						
f5	0.8740	0.8816	0.8776	0.8868	0.8888					
f6	0.8736	0.8800	0.8740	0.8840	0.8860	0.8804				
f7	0.8732	0.8836	0.8736	0.8840	0.8872	0.8792	0.8736			
f8	0.8696	0.8792	0.8716	0.8828	0.8852	0.8768	0.8732	0.8728		
f9	0.8700	0.8796	0.8716	0.8824	0.8856	0.8784	0.8740	0.8744	0.8720	
f10	0.8688	0.8796	0.8716	0.8796	0.8824	0.8764	0.8720	0.8744	0.8716	0.8844

Our results show that, despite the updates, the performance on earlier datasets showed no significant degradation. The degradation in accuracy is very minimal, demonstrating that the model strikes a balance between learning on new datasets while preserving its learned knowledge from prior datasets.

1.3 Task 2 Implementation

1. **Feature Representation**
   Like Task 1, EfficientNet is used to precompute feature representations for training datasets $D_{11}, \ldots, D_{20}$ and also for all held-out datasets datasets $D_1, \ldots, D_{20}$. After the feature extraction pipeline, the features are saved as `features_cache2.pth` and `eval_features_cache2.pth`, and loaded for the task.
2. **LwP Classifier**
   The classifier works by calculating the distance from class prototypes (mean feature vectors for each class) created during training. Classification is done by assigning the label of the nearest prototype (e.g. Euclidean distance). We are also maintaining a count of samples in each class as a part of our LwP model ie (prototype,count) for each class.
3. **Training and Update**
   - f_{10} of task 1 is loaded from `f10.pth` file and is used to predict labels of D_{11}.
   - **Weighted Features:** We used `compute_class_weight` from scikit-learn to compute the class weights using the predicted labels. These weights are used to address class imbalance in the dataset by assigning a higher weight to underrepresented classes. We then weighted each feature vector by its class weight.
   - The model f_i is updated to f_{i+1} by weighted averaging of the existing prototype and the new mean based on the class distribution. The old prototype is weighted by the number of samples in each class seen so far using the maintained counter. The new mean is weighted by the number of samples in each class in the current dataset using predicted labels. Along with the prototype we also update the class count by adding the new count of samples for the each class to old counter.

$$updated\_prototype = \frac{(old\_prototype \times old\_count) + (new\_mean \times new\_count)}{old\_count + new\_count}$$

where *new_mean* is calculated on the features of current dataset using the predicted labels. *old_count* and *new_count* are respective sample counts for each class in *old_prototype* and newly predicted labels in the current dataset.

- **Evaluation**
  - Similar to Task 1, For each f_i, evaluate performance on the held-out dataset D_i and all prior held-out datasets $\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_{i-1}$ and create a tabular representation of size 10×20.

Other methods involving calculating **distribution shift** with **Wasserstein Distance** and then **dymamic weighted updates with the measured drifts** were explored and resulted in almost similar accuracy.

1.3.1 Task 2 Results

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	\
f11	0.8696	0.8792	0.8712	0.8772	0.8824	0.8752	0.8716	0.8720	0.8712	0.8860	0.7240	
f12	0.8656	0.8780	0.8676	0.8772	0.8792	0.8736	0.8684	0.8700	0.8676	0.8820	0.7212	
f13	0.8640	0.8780	0.8660	0.8760	0.8776	0.8716	0.8676	0.8680	0.8660	0.8800	0.7196	
f14	0.8636	0.8780	0.8668	0.8760	0.8772	0.8712	0.8676	0.8676	0.8656	0.8808	0.7212	
f15	0.8632	0.8760	0.8680	0.8764	0.8772	0.8712	0.8672	0.8672	0.8652	0.8808	0.7208	
f16	0.8624	0.8748	0.8660	0.8756	0.8756	0.8704	0.8656	0.8656	0.8644	0.8804	0.7204	
f17	0.8628	0.8748	0.8664	0.8752	0.8748	0.8692	0.8660	0.8644	0.8644	0.8780	0.7192	
f18	0.8612	0.8720	0.8652	0.8744	0.8744	0.8692	0.8644	0.8624	0.8636	0.8772	0.7188	
f19	0.8604	0.8712	0.8652	0.8756	0.8744	0.8688	0.8636	0.8616	0.8628	0.8768	0.7164	
f20	0.8600	0.8708	0.8656	0.8756	0.8736	0.8680	0.8636	0.8612	0.8624	0.8764	0.7184	

	D12	D13	D14	D15	D16	D17	D18	D19	D20
f11									
f12	0.4752								
f13	0.4744	0.7800							
f14	0.4748	0.7792	0.8528						
f15	0.4760	0.7796	0.8528	0.8636					
f16	0.4752	0.7776	0.8520	0.8616	0.7284				
f17	0.4752	0.7760	0.8516	0.8608	0.7284	0.7928			
f18	0.4736	0.7756	0.8512	0.8620	0.7276	0.7920	0.7556		
f19	0.4736	0.7748	0.8524	0.8616	0.7272	0.7928	0.7548	0.6544	
f20	0.4740	0.7752	0.8524	0.8608	0.7260	0.7924	0.7532	0.6536	0.8440

Our results show that, despite the updates, the performance on earlier datasets showed no significant concern worthy degradation when compared to previous accuracies on same dataset.

Thus, both our Task1 and Task2 LwP model successfully achieved the goal of updating for new data without substantial degradation of accuracy on previously encountered datasets wrt previous models.

2 Problem 2: Paper Presentation

Selected Paper : Lifelong Domain Adaptation via Consolidated Internal Distribution

YouTube Video Link (earphones recommended) : `https://youtu.be/LqY7CQMpMCY?si=9BTjhvNDABolbQb2`

Acknowledgments

We would like to express our sincere gratitude to our course instructor, **Dr. Piyush Rai**, for their invaluable guidance throughout the course of this project. Their feedback were crucial in shaping the direction of our work.

We would also like to extend our appreciation to our team members. Each member played an integral role in the successful completion of this project.

Additionally, we are grateful for the online resources, including scikit-learn documentation and Stack Overflow , which provided helpful solutions to some of the challenges we faced. The tools and libraries, including Google Colab, PyTorch, Keras, and scikit-learn were instrumental in the completion of our mini machine learning project.

References

[1] PyTorch Contributors. EfficientNetB3 Weights - IMAGENET1KV1.

[2] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *ArXiv*, abs/1905.11946, 2019.