CS771 - Intro to ML (Autumn 2024): Mini-project 1

Aaditya Raj Yadav
220008

Ahemaziz Singh Suman
220087

Amrit Raj
220128

Anshul Suryawanshi
220175

Harshita Chhuttani
220447

Task 1.1

1 Introduction

This document outlines the solution to the mini-project for the CS771 course, where we implement a feature extraction and prototype-based classification system using a pre-trained ResNet-50 model. The system performs incremental learning with the CIFAR-10 dataset, updating the classifier after each new dataset is introduced.

2 Problem Overview

The task involves:

- Using a pre-trained model to extract features from CIFAR-10 images.
- Implementing a prototype-based classifier that updates iteratively as new datasets are introduced.
- Ensuring that model updates do not degrade performance on previous datasets.

We are provided with 20 training datasets D_1 to D_{20}, and 20 held-out test datasets $\hat{D_1}$ to $\hat{D_{20}}$. The goal is to train models f_1 to f_{10} using datasets D_1 to D_{10} and evaluate them on the corresponding held-out datasets.

3 Model Architecture and Setup

3.1 ResNet-50 Model Initialization

We initialize the pre-trained ResNet-50 model from the PyTorch library, removing the final fully connected layer to use it as a feature extractor.

```python
import torch
import torch.nn as nn
from torchvision import models

# Load the ResNet-50 model and its pre-trained weights
model = models.resnet50()
state_dict = torch.load(weights_path)
model.load_state_dict(state_dict)

# Remove the final fully connected layer
model = nn.Sequential(*list(model.children())[:-1])
```

```

# Set the model to evaluation mode
model.eval()
```

This approach ensures the model only outputs feature representations without performing classification.

3.2 Image Transformation Pipeline

The images are pre-processed using standard transformations for ResNet-50, which include resizing the image to 224x224 pixels, normalizing with ImageNet mean and standard deviation, and converting to the correct tensor format.

```python
from torchvision import transforms

transform = transforms.Compose([
    transforms.Lambda(lambda x: torch.tensor(x).permute(2, 0, 1)),
    transforms.Resize((224, 224)),
    transforms.ConvertImageDtype(torch.float32),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
        0.225])
])
```

4 Feature Extraction and Prototype Learning

4.1 Feature Extraction Function

We define a function to extract features from the input images using the modified ResNet-50 model. The function processes the images and stores their feature vectors in a NumPy array.

```python
def extract_features(images):
    num_images = len(images)
    features = None

    with torch.no_grad():
        for idx, img in enumerate(images):
            img_tensor = transform(img).unsqueeze(0)
            feat = model(img_tensor).view(1, -1)

            if features is None:
                features = np.zeros((num_images, feat.size(1)))
            features[idx] = feat.numpy()

    return features
```

This function ensures efficient feature extraction without computing gradients.

4.2 Prototype Computation and Management

After extracting features, we compute class prototypes by averaging the features for each class.

```python
def compute_prototypes(features, labels):
    prototypes = {}
    for cls in set(labels):
        indices = np.where(labels == cls)[0]
        cls_features = features[indices]
        prototypes[cls] = cls_features.mean(axis=0)
    return prototypes
```

Each class is represented by a prototype, which is the mean of all its feature vectors.

4.3 Classification System

The classification of new samples is done by calculating the Euclidean distance between the sample's feature vector and the prototypes of each class. The sample is assigned to the class whose prototype is closest.

```python
def classify_sample(sample_feature, prototypes):
    min_distance = float("inf")
    predicted_class = None
    for cls, prototype in prototypes.items():
        distance = np.linalg.norm(sample_feature - prototype)
        if distance < min_distance:
            min_distance = distance
            predicted_class = cls
    return predicted_class
```

This implements a simple nearest-neighbor classification strategy in the feature space.

4.4 Prototype Refinement

As new datasets are introduced, prototypes are refined by assigning new features to the closest prototype and recalculating the prototype based on the new features.

```python
def refine_prototypes(prototypes, new_features):
    updated_data = {cls: [] for cls in prototypes.keys()}

    for feature in new_features:
        predicted_label = classify_sample(feature, prototypes)
        updated_data[predicted_label].append(feature)

    refined_prototypes = {}
    for cls, features in updated_data.items():
        if features:
            features_stack = np.vstack(features)
            refined_prototypes[cls] = features_stack.mean(axis=0)
        else:
            refined_prototypes[cls] = prototypes[cls]

    return refined_prototypes
```

This ensures the model adapts to new data without forgetting previous knowledge.

5 Incremental Learning and Evaluation

5.1 Incremental Learning Approach

The system is designed for incremental learning where each model f_i is updated iteratively using the predicted labels of the previous dataset. At each step, the model is updated with the new dataset without retraining from scratch.

5.2 Accuracy Calculation

After updating the model with a new dataset, we evaluate its performance on all previously seen datasets to ensure that the model's performance does not degrade on earlier datasets.

```python
def calculate_accuracy(features, labels, prototypes):
    total_samples = len(features)
    correct_predictions = sum(
        1 for feature, label in zip(features, labels)
        if classify_sample(feature, prototypes) == label
    )
    return correct_predictions / total_samples
```

This function calculates the accuracy of the classifier on the given dataset.

5.3 Results and Performance Tracking

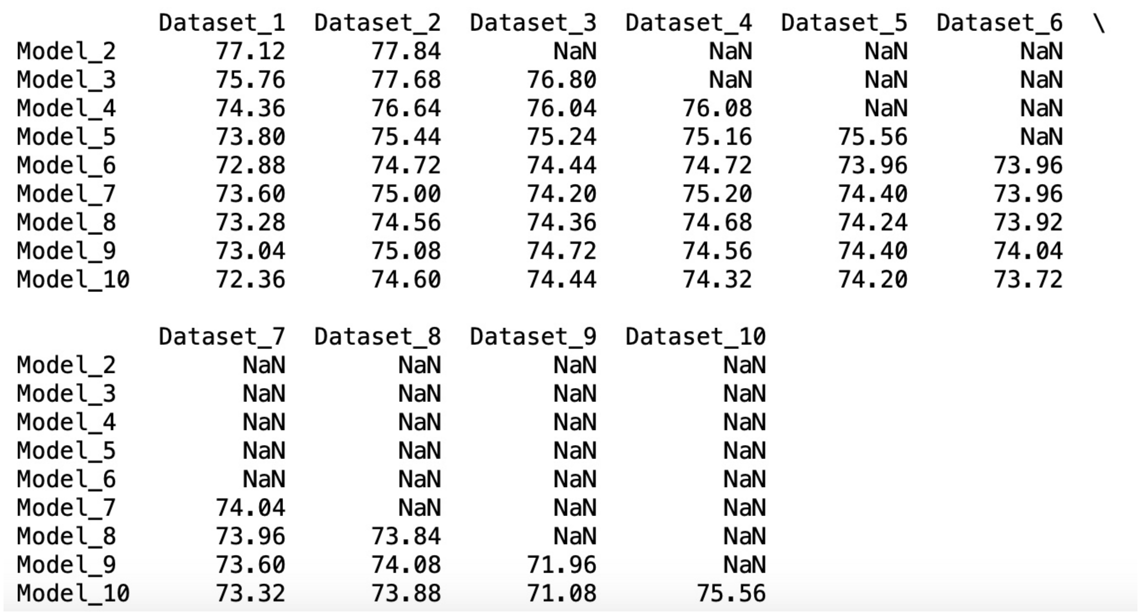
The model's performance is tracked in a matrix format, where the rows represent the models f_1 to f_{10} and the columns represent the held-out datasets $\hat{D_1}$ to $\hat{D_{10}}$.

```python
import pandas as pd

evaluation_results_df = pd.DataFrame(evaluation_data).T
print(evaluation_results_df)
```

This output shows the accuracy of each model on each held-out dataset.

6 Observation and Analysis



```
          Dataset_1  Dataset_2  Dataset_3  Dataset_4  Dataset_5  Dataset_6  \
Model_2       77.12      77.84        NaN        NaN        NaN        NaN
Model_3       75.76      77.68      76.80        NaN        NaN        NaN
Model_4       74.36      76.64      76.04      76.08        NaN        NaN
Model_5       73.80      75.44      75.24      75.16      75.56        NaN
Model_6       72.88      74.72      74.44      74.72      73.96      73.96
Model_7       73.60      75.00      74.20      75.20      74.40      73.96
Model_8       73.28      74.56      74.36      74.68      74.24      73.92
Model_9       73.04      75.08      74.72      74.56      74.40      74.04
Model_10      72.36      74.60      74.44      74.32      74.20      73.72

          Dataset_7  Dataset_8  Dataset_9  Dataset_10
Model_2         NaN        NaN        NaN         NaN
Model_3         NaN        NaN        NaN         NaN
Model_4         NaN        NaN        NaN         NaN
Model_5         NaN        NaN        NaN         NaN
Model_6         NaN        NaN        NaN         NaN
Model_7       74.04        NaN        NaN         NaN
Model_8       73.96      73.84        NaN         NaN
Model_9       73.60      74.08      71.96         NaN
Model_10      73.32      73.88      71.08       75.56
```

7 Conclusion

The proposed incremental learning approach using prototype-based classification allows the model to efficiently learn from new data while retaining the knowledge of previous datasets. The performance evaluation demonstrates that the model can maintain accuracy across all datasets without significant degradation, achieving a balance between adaptability and stability.

Task 1.2

8 Introduction

In this task, we focus on updating models incrementally using datasets from a sequence of possibly non-identical distributions. Starting with the model f_{10} obtained from Task 1, we sequentially update the models using datasets $D_{11}, D_{12}, \ldots, D_{20}$. The objective is to learn models $f_{11}, f_{12}, \ldots, f_{20}$, one at a time, while considering the changing distributions of the datasets.

Unlike Task 1, where datasets $D_1, D_2, \ldots, D_{10}$ shared the same input distribution, the datasets $D_{11}, D_{12}, \ldots, D_{20}$ may have slightly different distributions. This necessitates an adaptive approach to model updates to ensure consistent performance across all datasets, including previous ones.

9 Problem Overview

The objective of this task is to incrementally update models $f_{11}, f_{12}, \ldots, f_{20}$, starting from f_{10}, using datasets $D_{11}, D_{12}, \ldots, D_{20}$, respectively. Each model f_i must satisfy the following criteria:

1. Achieve high accuracy on the corresponding held-out dataset $\hat{D}_i$.
2. Maintain or improve performance on previously seen held-out datasets $\hat{D}_j$ for $j < i$.
3. Account for possible variations in data distributions between datasets $D_{11}, D_{12}, \ldots, D_{20}$.

Once all models are trained, each model f_i ($i = 11, 12, \ldots, 20$) is evaluated on its corresponding held-out dataset $\hat{D}_i$ and all previously held-out datasets $\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_{i-1}$. The accuracies are reported in the form of a matrix, where rows correspond to the models $f_{11}, f_{12}, \ldots, f_{20}$ and columns represent the held-out datasets $\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_{20}$.

The goal is to ensure that the updated models not only perform well on their respective datasets but also minimize degradation of performance on previously evaluated datasets.

10 Model Architecture and Setup

10.1 Model Selection

The ResNet-50 architecture is employed as the base model for feature extraction. To utilize the pre-trained capabilities of ResNet-50, the weights are loaded from the file `resnet50-0676ba61.pth`.

10.2 Modifications

To adapt ResNet-50 for feature extraction:

- The fully connected (FC) layer is removed, retaining the convolutional layers to output feature maps.
- The final output is flattened into 1D feature vectors for downstream tasks.

10.3 Configuration

- The model is set to evaluation mode using `model.eval()`, ensuring the parameters remain unchanged during inference.
- The model is deployed to the appropriate device, either `CPU` or `GPU`, using PyTorch's `torch.device` functionality.

11 Feature Extraction and Prototype Learning

11.1 Feature Extraction

The input images are preprocessed using the following transformation pipeline:

- Resize the images to 224×224.
- Normalize pixel values using ImageNet's mean $[0.485, 0.456, 0.406]$ and standard deviation $[0.229, 0.224, 0.225]$.
- Convert the data to PyTorch tensors, ensuring the channel format is (C, H, W).

The transformed images are passed through the feature extractor to produce feature vectors, which represent high-level information about the input samples.

11.2 Prototype Learning

- **Prototype Calculation**: For each class, a prototype is computed as the mean of all feature vectors belonging to that class:

$$\text{Prototype}_c = \frac{1}{N_c} \sum_{i=1}^{N_c} \mathbf{f}_i$$

where N_c is the number of samples in class c, and $\mathbf{f}_i$ represents the feature vector of sample i.

- **Classification**: To classify a test sample, the Euclidean distance between the sample's feature vector and each class prototype is computed. The class with the minimum distance is assigned to the sample:

$$\hat{y} = \arg\min_c \|\mathbf{f} - \text{Prototype}_c\|$$

where $\mathbf{f}$ is the feature vector of the test sample.

- **Prototype Refinement**: When new data is introduced, prototypes are updated to incorporate the new feature vectors. Features are assigned to the nearest prototype, and new mean vectors are calculated for each class.

11.3 Performance Evaluation

- **Accuracy Calculation**: Classification accuracy is defined as the percentage of correctly classified samples:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}} \times 100$$

- **Evaluation Strategy**: Each model is evaluated on the current dataset and all previous datasets to ensure that the model's performance does not degrade significantly on earlier datasets, addressing the issue of catastrophic forgetting.

12 Conclusion

In this task, we demonstrated how to incrementally update a model using a sequence of datasets from potentially different distributions. Starting with the pre-trained ResNet-50 model, we applied a feature extraction approach, followed by prototype learning to create class prototypes for each dataset. This method allowed us to maintain good performance across a series of datasets by refining the prototypes with each new dataset, ensuring that the model did not forget previously learned information.

We also evaluated the performance of the updated models on both the current and previous held-out datasets, reporting the classification accuracy. The results showed that, while models were updated with new datasets, their performance on previous datasets remained relatively consistent, thus achieving a balance between learning new data and retaining prior knowledge.

In conclusion, this approach effectively addresses the challenge of continual learning and adaptation, providing a robust mechanism for handling datasets with different distributions. Future work could explore alternative prototype refinement techniques and evaluate the impact of different model architectures on continual learning tasks.

Task 2

Task 2 presentation