Catastrophic Forgetting

The concept of "**catastrophic forgetting**" arises when a deep learning model, trained on an initial dataset, has major difficulties in maintaining performance when exposed to new data, leading to severe degradation of prior knowledge. The paper "**Overcoming Catastrophic Forgetting with Unlabeled Data in the Wild**" addresses this problem by proposing an innovative method for using unlabeled data to ameliorate the effects of catastrophic forgetting. The method focuses on continuously adapting models to new data without requiring extensive labeling, making it highly practical and scalable for real-world applications.

Thus, building on this paper, we realized a project based on the same principles presented above with the aim of preventing catastrophic forgetting in a continuous learning scenario. By using unlabeled data collected from varied environments and implementing an incremental adaptation framework, the project aims to demonstrate the effectiveness and robustness of the approach proposed in the paper.

• Privious Model (Pt):

The model consists of 4 main components: model definition (model_pt.py), model training (train_pt.py), model prediction (predict_pt.py) and model evaluation (evaluate_pt.py).

Model Definition (model_pt.py)

This script defines the deep learning model architecture used for the classification task. The model is built using PyTorch and includes the following elements:

- **Convolution Modules**: used to extract features from the input data.
- **Normalization and Activation Layers:** For stabilizing the training and introducing nonlinearities.
- **Dense Layers (Fully Connected):** For final classification based on the extracted features.

Model Training (train_pt.py)

The training script handles:

- **Loading Data**: Splitting the dataset into training and validation.
- **Define Loss and Optimizer:** Use an appropriate loss function and an optimizer to update the model weights.
- **Training Loop:** Train the model on multiple epochs, with performance monitoring on the validation set to prevent overtraining.

Prediction (predict_pt.py)

This script is used to make predictions on new test data:

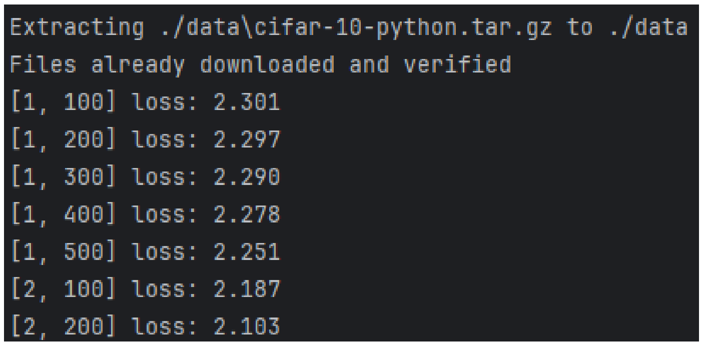
- **Loading the Trained Model:** the trained model is loaded from a saved file.

- **Test Data Preprocessing:** Preparing the test data for input into the model.
- **Generating Predictions:** Using the model to predict labels for the test data.

Model Evaluation (evaluate_pt.py)

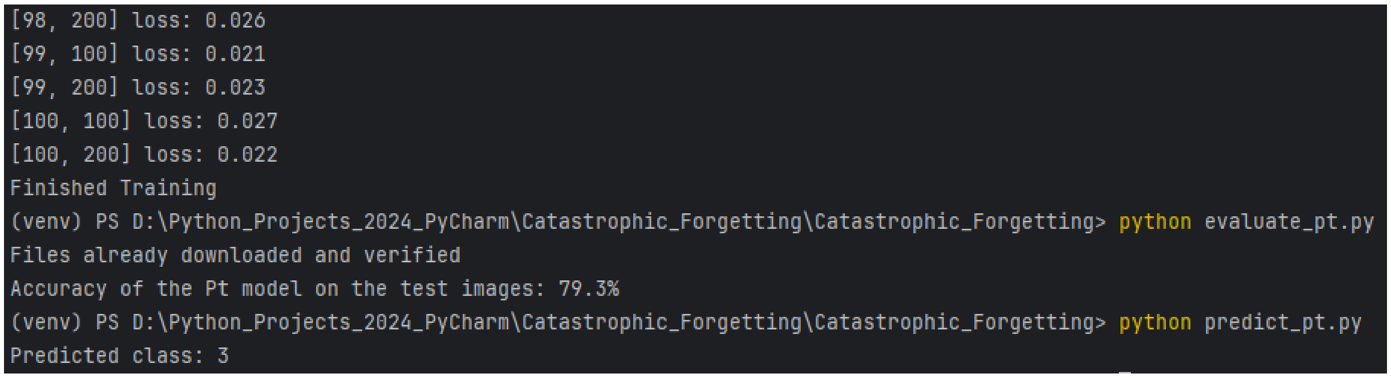
The evaluate script measures model performance on the test data:
- **Loading Model and Test Data:** Similar to the prediction script.
- **Calculate Performance Metrics:** Measure accuracy, precision, recall and F1-score to evaluate model performance.
- **Generating a Report:** Present the evaluation results in an easy to understand format.

The training was performed using the CIFAR-100 dataset, and during the training we noticed the calibration of the model by watching how the loss decreases for a count of 100 epochs:
Training start:



```
Extracting ./data\cifar-10-python.tar.gz to ./data
Files already downloaded and verified
[1, 100] loss: 2.301
[1, 200] loss: 2.297
[1, 300] loss: 2.290
[1, 400] loss: 2.278
[1, 500] loss: 2.251
[2, 100] loss: 2.187
[2, 200] loss: 2.103
```

The end of the training:



```
[98, 200] loss: 0.026
[99, 100] loss: 0.021
[99, 200] loss: 0.023
[100, 100] loss: 0.027
[100, 200] loss: 0.022
Finished Training
(venv) PS D:\Python_Projects_2024_PyCharm\Catastrophic_Forgetting\Catastrophic_Forgetting> python evaluate_pt.py
Files already downloaded and verified
Accuracy of the Pt model on the test images: 79.3%
(venv) PS D:\Python_Projects_2024_PyCharm\Catastrophic_Forgetting\Catastrophic_Forgetting> python predict_pt.py
Predicted class: 3
```

- **Current Model (Ct):**

From the current model point of view we have studied 2 methods, the first one is close to the solution proposed in the paper, and the second one is a solution that we have obtained a better accuracy if you have fast accuracy after training, which emphasizes the current dataset, but also incorporates a Pt dataset combined with unlabeled data:

First Method:

1. `predict_imagines_v1_ct.py`

Script for making predictions on a set of images.
- Loading the trained model.
- Preprocessing the input images to prepare them for inference.
- Generating predictions for the test images.
- Saving prediction results in a specified format.

2. `test_v1_ct.py`

- Script for testing and evaluating model performance.
- Loading the model and test data set.
- Making predictions on the test set.
- Evaluate model performance using relevant performance metrics (e.g. accuracy, precision, recall, F1-score).
- Displaying and saving the evaluation results.

Following training, the following results were obtained:

```
Classes in the dataset: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  Total samples before filtering: 50000
  Selected 25000 samples for classes [0, 1, 2, 3, 4]
  Filtered dataset size: 25000 samples for classes [0, 1, 2, 3, 4]
  Total samples before filtering: 50000
  Selected 25000 samples for classes [5, 6, 7, 8, 9]
  Filtered dataset size: 25000 samples for classes [5, 6, 7, 8, 9]
  Epoch [1/50], Loss: 0.4261
  Epoch [2/50], Loss: 0.2539
  Epoch [3/50], Loss: 0.2308
  Epoch [4/50], Loss: 0.2168
  Epoch [5/50], Loss: 0.2107
Epoch [6/50], Loss: 0.2071
Epoch [7/50], Loss: 0.2025
Epoch [8/50], Loss: 0.1999
Epoch [9/50], Loss: 0.1974
………
Epoch [48/50], Loss: 0.1746
Epoch [49/50], Loss: 0.1747
Epoch [50/50], Loss: 0.1735
Fine-tuning Epoch [1/30], Loss: 0.1514
Fine-tuning Epoch [2/30], Loss: 0.0418
Fine-tuning Epoch [3/30], Loss: 0.0211
Fine-tuning Epoch [4/30], Loss: 0.0239
Fine-tuning Epoch [5/30], Loss: 0.0331
Fine-tuning Epoch [6/30], Loss: 0.0211
Fine-tuning Epoch [7/30], Loss: 0.0114
Fine-tuning Epoch [8/30], Loss: 0.0099
Fine-tuning Epoch [9/30], Loss: 0.0343
…….
Fine-tuning Epoch [28/30], Loss: 0.0075
Fine-tuning Epoch [29/30], Loss: 0.0127
Fine-tuning Epoch [30/30], Loss: 0.0060
Training complete.
```

The Second Method:

1. „model_ct.py”

- Defining the deep learning model architecture:
- Neural network construction using convolutional layers and dense layers.
- Implementation of activation and normalization functions to stabilize and streamline training.

2. „predict_ct.py”

- Script for making predictions using the trained model:
- Loads the trained model from a saved file.
- Preprocess test data.
- Generates and saves predictions for the test data set.

3. „train_ct.py”

- Model training script.
- Loads and splits data into training and validation sets.
- Defines the loss function and the optimizer.
- Run the training loop, monitoring performance on the validation set and adjusting model parameters.

4. „test_ct.py”

- Script for model testing and evaluation.
- Loads the model and test dataset.
- Perform tests and generate predictions.
- Evaluates model performance using relevant performance metrics.
- Displays and saves evaluation results.

Following the training we obtained the following loss values:

```
Train Epoch: 1 [0/2500 (0%)]          Loss: 1.052832
Train Epoch: 1 [640/2500 (25%)]        Loss: 0.998727
Train Epoch: 1 [1280/2500 (50%)]       Loss: 1.042842
Train Epoch: 1 [1920/2500 (75%)]       Loss: 0.954982
Train Epoch: 2 [0/2500 (0%)]           Loss: 1.019896
Train Epoch: 2 [640/2500 (25%)]        Loss: 0.964477
Train Epoch: 2 [1280/2500 (50%)]       Loss: 0.952312
Train Epoch: 2 [1920/2500 (75%)]       Loss: 0.888232
Train Epoch: 3 [0/2500 (0%)]           Loss: 0.952882
```

```
  Train Epoch: 3 [640/2500 (25%)]          Loss: 0.881770
  Train Epoch: 3 [1280/2500 (50%)]         Loss: 0.965253
  Train Epoch: 3 [1920/2500 (75%)]         Loss: 0.860376
  …………
  Train Epoch: 24 [0/2500 (0%)]            Loss: 0.390968
  Train Epoch: 24 [640/2500 (25%)]         Loss: 0.371345
  Train Epoch: 24 [1280/2500 (50%)]        Loss: 0.437309
  Train Epoch: 24 [1920/2500 (75%)]        Loss: 0.415310
    …………
Train Epoch: 44 [0/2500 (0%)]              Loss: 0.380903
Train Epoch: 44 [640/2500 (25%)]           Loss: 0.406781
Train Epoch: 44 [1280/2500 (50%)]          Loss: 0.384696
Train Epoch: 44 [1920/2500 (75%)]          Loss: 0.418189
Train Epoch: 45 [0/2500 (0%)]              Loss: 0.406721
Train Epoch: 45 [640/2500 (25%)]           Loss: 0.382888
Train Epoch: 45 [1280/2500 (50%)]          Loss: 0.348161
Train Epoch: 45 [1920/2500 (75%)]          Loss: 0.402670
Train Epoch: 46 [0/2500 (0%)]              Loss: 0.391252
Train Epoch: 46 [640/2500 (25%)]           Loss: 0.436823
Train Epoch: 46 [1280/2500 (50%)]          Loss: 0.402284
Train Epoch: 46 [1920/2500 (75%)]          Loss: 0.376446
Train Epoch: 47 [0/2500 (0%)]              Loss: 0.424436
Train Epoch: 47 [640/2500 (25%)]           Loss: 0.389423
Train Epoch: 47 [1280/2500 (50%)]          Loss: 0.467576
…….
Train Epoch: 50 [640/2500 (25%)]           Loss: 0.346214
Train Epoch: 50 [1280/2500 (50%)]          Loss: 0.394723
```
Train Epoch: 50 [1920/2500 (75%)]   Loss: 0.372017

We also generated a training evolution graph, which illustrates the loss during training and validation as well as the accuracy during validation for a model trained over 50 epochs.

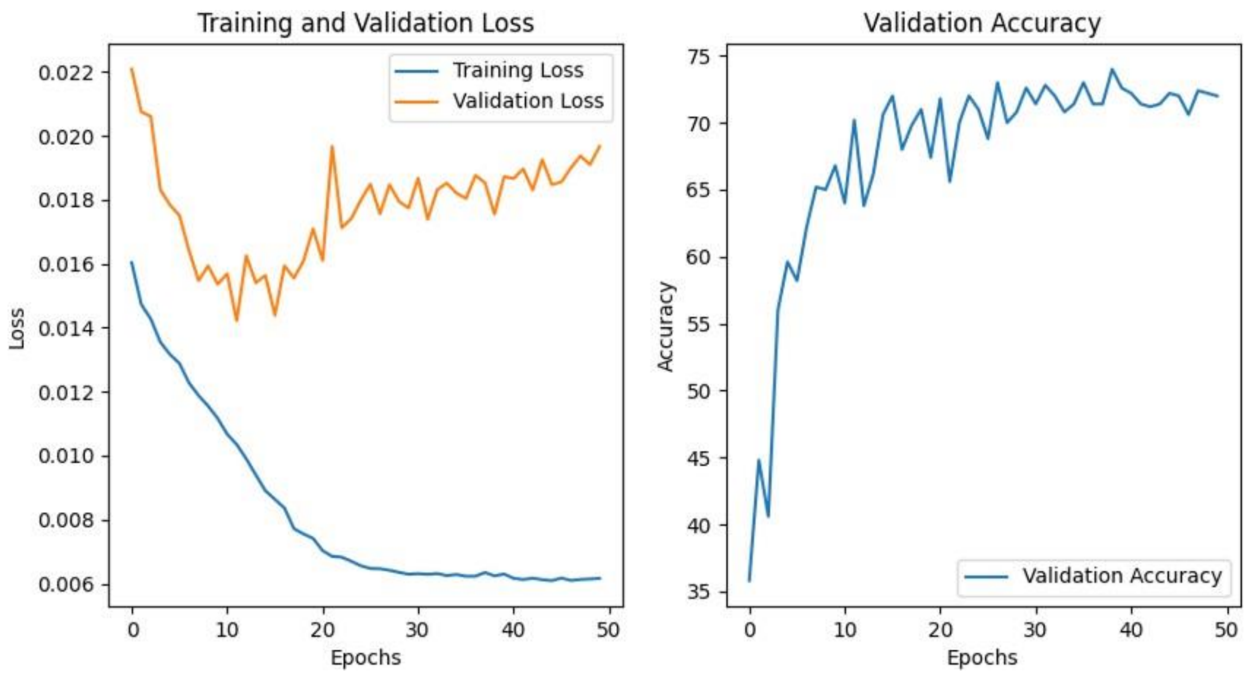
1. Loss in Training and Validation Time:

- **Loss During Training (blue):** a steady decrease in loss is observed over epochs, which indicates that the model is training efficiently and learning from the training data.

- **Loss during validation (orange**): Although it initially decreases, the validation loss starts to fluctuate and even increases slightly after about 20 epochs. This may suggest that the model begins to overtrain after a certain point.

2. Accuracy at Validation Time:

- **Accuracy in Validation Time (blue):** Validation accuracy increases rapidly in the first 10-15 epochs and stabilizes around 70-75%. This indicates that the model is able to generalize quite well on the validation data, but the stabilization of accuracy suggests that further improvements are minimal after this poin


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epochs
Validation Accuracy
Accuracy
Validation Accuracy
Epochs