Experiment 1: Case Study on PyTorch

# AIM: The aim of this assignment is to explore PyTorch in detail, understand its architecture, ecosystem, and applications.

**Problem Definition:**

1. Choose one application area of PyTorch (e.g., Computer Vision, NLP, Reinforcement Learning, or another relevant domain).
2. Justify why you selected this application.
3. Define the software requirements (Python version, PyTorch version, libraries, etc.).
4. Define the hardware requirements (minimum and recommended specs).
5. Identify the dataset you will use and explain why.
6. Specify the evaluation metrics you plan to use (accuracy, F1 score, loss curves, etc.).

# Report of the Case Study:

Your report should include the following sections:

* Introduction: Overview of PyTorch and motivation for the chosen application.
* Objectives: Clearly stated aims of your study.
* Requirements: Software, hardware, dataset, and evaluation metrics.
* Implementation: A step-by-step outline of how you would implement your model in PyTorch.
* Results: Present expected or actual outcomes (training curves, accuracy, screenshots, etc.).
* Discussion: Analyze the strengths, weaknesses, and challenges faced.
* Conclusion & Future Scope: Suggest improvements or research extensions.

**Introduction**

PyTorch is an open-source deep learning framework developed by Facebook’s AI Research lab (FAIR). It is widely used for research and production due to its dynamic computational graph, ease of debugging, and strong community support. PyTorch has a rich ecosystem with libraries like torchvision, torchaudio, and torchtext for domain-specific tasks.  
  
In this study, we focus on Computer Vision using PyTorch, specifically image classification with Convolutional Neural Networks (CNNs). Computer Vision is chosen because it has a wide range of applications (autonomous vehicles, medical imaging, face recognition, etc.), and PyTorch provides strong support for image-related tasks through torchvision datasets and pretrained models.

**Objectives**

• To understand the architecture and workflow of PyTorch.

• To implement an image classification task using CNNs.

• To train and evaluate the model on a benchmark dataset.

• To analyze the performance using accuracy and loss curves.

• To explore the strengths, limitations, and possible extensions of PyTorch in Computer Vision.

**Requirements**

**Software Requirements**

• Python Version: 3.9 or later  
• PyTorch Version: 2.0 or later  
• Libraries:  
 - torch (core PyTorch)  
 - torchvision (datasets, transforms, pretrained models)  
 - matplotlib (visualization)  
 - numpy (numerical operations)

**Hardware Requirements**

• Minimum:  
 - CPU: Dual-core processor  
 - RAM: 4 GB  
 - Storage: 2 GB free space  
  
• Recommended:  
 - GPU: NVIDIA CUDA-enabled (e.g., GTX 1050 or higher)  
 - RAM: 8 GB or more  
 - Storage: 10 GB free space

**Dataset**

• CIFAR-10 Dataset (60,000 32×32 color images in 10 classes).  
• Justification: It is a standard benchmark dataset for image classification, small enough for quick training, yet complex enough to demonstrate deep learning techniques.

**Evaluation Metrics**

• Accuracy (percentage of correctly classified images).  
• Loss Curves (training & validation loss).  
• Confusion Matrix (optional, to analyze per-class performance).

**Implementation (Step-by-Step)**

• Import Libraries – Load torch, torchvision, torch.nn, torch.optim, etc.

• Load Dataset – Use torchvision.datasets.CIFAR10 with data augmentation.

• Define CNN Model – Build a CNN with convolutional, pooling, and fully connected layers.

• Define Loss & Optimizer – Use CrossEntropyLoss and Adam optimizer.

• Training Loop – Forward pass, compute loss, backpropagation, update weights.

• Validation – Evaluate on the test set after each epoch.

• Visualization – Plot training/validation loss and accuracy curves.

**Results**

Expected Outcomes:  
• Training accuracy gradually increases, reaching ~80–85% on CIFAR-10.  
• Validation accuracy around ~75–80% depending on hyperparameters.  
• Loss decreases over epochs, indicating effective learning.  
  
Visualizations:  
• Loss vs. Epoch curve.  
• Accuracy vs. Epoch curve.  
• Example predictions with ground truth labels.

**Discussion**

Strengths of PyTorch:  
• Dynamic computation graph for flexibility.  
• Strong GPU acceleration.  
• Easy-to-use APIs for datasets and transforms.  
• Large community and pretrained models available.  
  
Weaknesses:  
• Higher memory usage compared to some frameworks.  
• Slower on CPU-only systems for large models.  
  
Challenges Faced:  
• Hyperparameter tuning for better accuracy.  
• Overfitting on small datasets if regularization is not applied.

**Conclusion & Future Scope**

In this experiment, we successfully explored PyTorch for Computer Vision using CNNs on the CIFAR-10 dataset. The framework’s simplicity and flexibility make it a strong choice for both research and deployment.  
  
Future Scope:  
• Implement advanced architectures like ResNet or DenseNet.  
• Use transfer learning with pretrained models from torchvision.models.  
• Deploy the trained model using TorchServe or ONNX.  
• Extend the work to real-world datasets (e.g., medical images, self-driving car datasets).