Project Mid Report- CSE445.8 Spring2025

Group 7- Soleman Hossain 2021682042, Nafis Mahmud 2022396642, Shahriar Ratul 2111514642, Durjoy Barua 2131766642.

Abstract

This project aims to develop a Convolutional Neural Network (CNN) for image classification using PyTorch, leveraging GPU acceleration to enhance training efficiency. The dataset is organized into training, validation, and test subsets, with appropriate data augmentation techniques applied to improve model robustness. A custom CNN architecture (**SimpleCNN**) was designed and implemented, and initial training epochs have been completed. This report provides an overview of the project setup, methodology, preliminary results, and future work.

Overview

Environment Setup

The development environment was configured by installing Python 12.6 and creating a virtual environment to ensure dependency isolation. Jupyter Notebook was integrated into Visual Studio Code to facilitate interactive development. To utilize GPU acceleration, the NVIDIA driver was updated via the NVIDIA app, and the CUDA toolkit (version 12.6) was installed. PyTorch was installed following the official guide at https://pytorch.org/ , ensuring compatibility with the CUDA toolkit. Additional dependencies such as **scikit-learn**, **matplotlib**, **seaborn**, and **tqdm** were installed to support data processing, visualization, and model evaluation.

Flowchart-

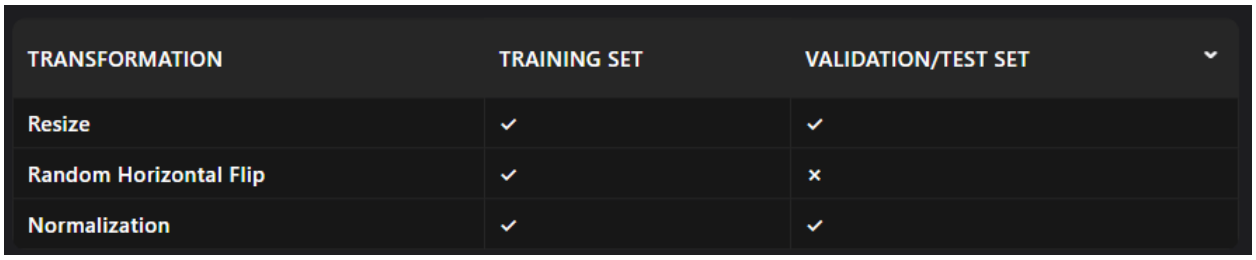
1. Install Python
2. Create Virtual Environment
3. Install Jupyter Notebook in VSCode
4. Update NVIDIA GPU Driver
5. Install CUDA Toolkit
6. Install Dependencies (e.g., PyTorch)

Methodology

Data Preprocessing and Augmentation

The dataset was stored in the project directory under separate folders for training, validation, and testing. Data transformations were applied to resize images to 128x128 pixels, normalize pixel values, and augment training data with random horizontal flips. These steps ensure consistent input dimensions and reduce overfitting during training.

Table: Data Transformations



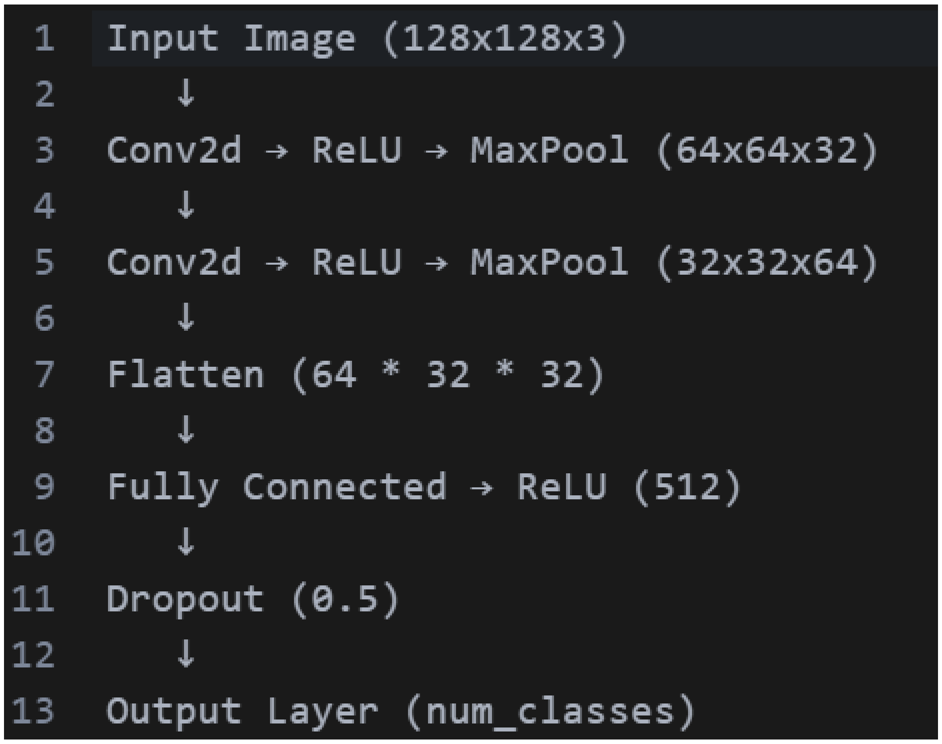
TRANSFORMATION	TRAINING SET	VALIDATION/TEST SET
Resize	✓	✓
Random Horizontal Flip	✓	×
Normalization	✓	✓

A visualization function was implemented to display sample images from the dataset, providing insights into the data distribution and preprocessing effects.

Model Architecture

A simple yet effective CNN architecture (**SimpleCNN**) was designed, consisting of two convolutional layers followed by max-pooling layers, fully connected layers, and dropout for regularization. The model was initialized with the number of output classes corresponding to the dataset's categories. The architecture was moved to the GPU for accelerated computation, and its summary was printed to confirm layer configurations.

Diagram: SimpleCNN Architecture



```
Input Image (128x128x3)
   ↓
Conv2d → ReLU → MaxPool (64x64x32)
   ↓
Conv2d → ReLU → MaxPool (32x32x64)
   ↓
Flatten (64 * 32 * 32)
   ↓
Fully Connected → ReLU (512)
   ↓
Dropout (0.5)
   ↓
Output Layer (num_classes)
```

Training Process

The model was trained using the Adam optimizer and cross-entropy loss, with a learning rate scheduler to adjust the learning rate dynamically. Training progress was monitored using **tqdm** for batch-wise updates, displaying metrics such as batch loss and average loss per epoch. GPU utilization was confirmed using the **nvidia-smi** command.

Preliminary Result

Visualisation

Sample images from the training dataset were visualized to verify the correctness of data loading and preprocessing. The images were displayed with their corresponding class labels, confirming proper organization and transformation.

Initial Training

Three epochs of training were completed, and the average loss per epoch was recorded. While detailed performance metrics are yet to be evaluated, the training process demonstrated smooth convergence, indicating the feasibility of the current approach.

Future Work

Performance Evaluation

The next steps involve evaluating the model's performance on the validation set using metrics such as accuracy, precision, recall, F1-score, and confusion matrix. Precision-recall curves and ROC-AUC will also be analyzed to assess classification effectiveness.

Hyperparameter Tuning

Further experiments will focus on fine-tuning hyperparameters, including learning rate, batch size, and dropout rate, to optimize model performance. Additionally, more advanced architectures or transfer learning techniques may be explored to enhance classification accuracy.

Testing and Deployment

Once satisfactory performance is achieved on the validation set, the model will be tested on the unseen test dataset to evaluate its generalization capabilities. The final model will then be prepared for deployment in a real-world application.

[The End]