**CSYE 7105**

**High Performance Parallel Machine Learning and Artificial Intelligence**

****

**Team 13**

**Car Image Classification Using Convolutional Neural Networks**

****

**Team Member**

Atulya Sharma (001306609)

**Introduction**

**Background**

Cars come in a host of different body styles like sedans, Crossovers, SUVs, Pickups. Each of these styles are slightly different from each other and can get confusing for people who aren’t car fanatic, and this can be a big hurdle when deciding on your next car. Deep Learning have been extremely helpful in classifying cars into various categories and help the general public to identify cars with as much ease as possible.

This project deals with Stanford Car dataset which uses CNN to classify images into 8 classes based on different car body styles.

**Motivations / Problem Statement**

The primary problem people face is that there is no such solution where one can simply upload a picture and get what kind of a car it that. Neural Networks like CNN are very efficient in image classification and distinguish different categories of cars with ease if trained properly.

Another problem from a computing standpoint is that Serial computing takes more time to perform the same task when compared to Parallel Computing and thus there is a significant speedup in performance when using Parallel Machine Learning over Serial.

**Objectives**

1. Image Classification of cars using Convolutional Neural Networks on a test dataset on a Python Framework to distinguish between 8 different Car Styles.
2. Implementing Data Parallelism on the code and then running on single and multiple GPUs using Nvidia’s CUDA.
3. Running the code on High Performance Computing (HPC) Cluster called Discovery to compare the Speedup in performance for Parallel V/S Serial Deep Learning.

**Methodology**

**Libraries**

The following Libraries have been used to perform the Image Classification on our test Dataset:

* Numpy
* Pytorch
* Matplotlib
* Torchvision

1. **NumPy**

NumPy is the fundamental package for scientific computing with Python. NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

1. **PyTorch**

PyTorch is an open-source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab. It is free and open-source software released under the Modified BSD license. In this project, PyTorch has been used to run Convolutional Neural Networks to find the training and testing accuracy of the algorithm when asked to classify the images.

1. **Matplotlib**

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits. Matplotlib has been used in this project to visualize the accuracy of the algorithm on the image dataset.

1. **Torchvision**

This library is a part of the PyTorch project. Torchvision package consists of popular datasets, model architectures and common image transformations for computer vision. This project uses torchvision to perform image transformation like resizing, horizontal flipping and normalization.

**Compute Unified Device Architecture (CUDA)**

CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs.

In GPU-accelerated applications, the sequential part of the workload runs on the CPU – which is optimized for single-threaded performance – while the compute intensive portion of the application runs on thousands of GPU cores in parallel. When using CUDA, developers’ program in popular languages such as C, C++, Fortran, Python and MATLAB and express parallelism through extensions in the form of a few basic keywords.

CUDA has several advantages over traditional general-purpose computation on GPUs using graphics APIs (GPGPU):

1. Scattered reads
2. Unified virtual memory
3. Shared memory
4. Faster downloads and readback to and from the GPU
5. Full support for integer and bitwise operations, including integer texture lookups

**Flow Diagram**

Graphical user interface, text, application, chat or text message

Description automatically generated

**Algorithm**

The Image Classification done in this project has been made possible due to the following Deep Learning Algorithm:

**Convolutional Neural Network:** or CNN is a class of Deep Neural Networks (DNN), most commonly applied to analyze visual imagery like image classification and identification, video recognition, medical image analysis, Natural Language Processing, etc.

Convolutional networks are inspired from the biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual neurons respond to only a specific stimulus and the overall outcome is a result of multiple neurons giving their responses.

CNN Architecture is made up of the following stack of distinct layers to function properly:

1. Convolutional Layer
2. Pooling Layer
3. ReLU Layer
4. Fully Connected Layer
5. Loss Layer

Diagram

Description automatically generated

Application of Convolutional Neural Networks

1. Natural Language Processing
2. Computer Vision
   1. Face Recognition
   2. Scene Labelling
   3. Image Classification
   4. Action Recognition
   5. Human Pose Estimation
   6. Document Analysis

**Image Classification:** CNN is widely used for classifying images. Compared to other methods CNNs achieve better classification accuracy on large scale datasets due to their capability of joint feature and classifier learning.

**Parallel Computing**

Large Problem can take longer times to process, thus parallel computing is a type of computation where many calculations or the execution of processes are carried out simultaneously. Large problems can often be divided into smaller ones, which can then be solved simultaneously, thus the name parallel.

Parallelism has long been employed in High-Performance Computing but has also become the dominant paradigm in computer architecture, mainly in the form of multi-core processors. It uses either multiple CPUs or multiple GPUs or both depending on the type and degree of problems.

Some of the major applications of parallel computing are:

1. Data Mining
2. Networked imaging and Multimedia Technologies
3. Medical Imaging and diagnosis
4. Augmented and Virtual Reality

**Data Parallelism**

It is parallelization across multiple processors in parallel computing environments. It focuses on distributing the data across different nodes, which operate on the data in parallel. It can be applied on regular data structures like arrays and matrices by working on each element in parallel. It contrasts to task parallelism as another form of parallelism.

A data parallel job on an array of n elements can be divided equally among all the processors. Same model is used for every thread, but the data given to each of them is divided and shared

The main difference between Data Parallelism and Model Parallelism is that data parallelism is ideal for array and matrix computations and Convolutional Neural Networks (CNN) and model parallelism is more suitable for deep learning.

**Diagram

Description automatically generated**

Another difference between the two is that data parallelism is fast for small networks but very slow for large networks whereas model parallelism is the exact opposite.

In our case, data parallelism is achieved by the use of torch.nn package.

**Dataset**

**Source**

The dataset used for the project is a cars image dataset provided by Stanford University. It is split into training and testing images, where each class is split roughly 50:50 split. It contains 196 classes of cars.

<http://ai.stanford.edu/~jkrause/cars/car_dataset.html>

The initial Dataset was 1.85 Gigabytes in size

**Specification of the Dataset**

The Stanford car image dataset consists of training and testing data split into separate folders including class labels for all the images. The statistics of the dataset are mentioned in the following table.

|  |  |
| --- | --- |
| **Group** | **Number of Images** |
| cars\_train | 8,144 |
| cars\_test | 8,041 |
| **Total** | **16,185** |

**Final Dataset**

Before the dataset could be used in the Algorithm, there were a few changes that were made to it. 8 subdirectories were created inside both train and test. Out of 196 classes of cars 111 were used and were split into the new subdirectories based on their body style which are as follows:

* Sedan
* Coupe
* Coupe\_Convertible
* Crossover
* Hatchback
* Pickup
* Van
* SUV

The statistics of the final dataset are shown in the table below.

|  |  |
| --- | --- |
| **Group** | **Number of Images** |
| train | 4,635 |
| test | 4,528 |
| **Total** | **9,163** |

The final dataset is 991.4 Megabytes in size.

Below are some of the sample images from the dataset

**Coupe SUV**

A picture containing outdoor, road, car, transport

Description automatically generated A picture containing road, outdoor, car, street

Description automatically generated

**Van Sedan**

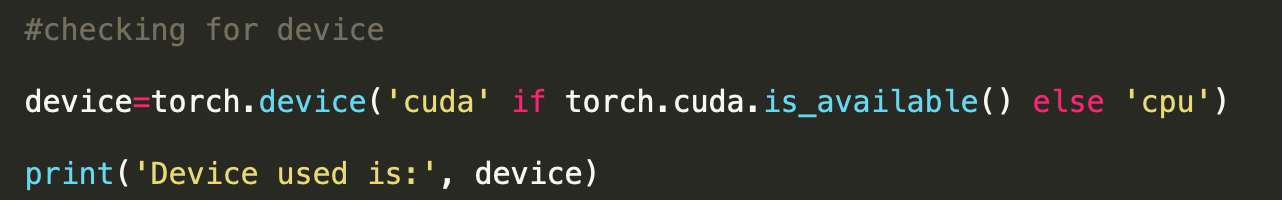
**A picture containing text, car, parked, transport

Description automatically generated **

**Analysis**

**Checking for available devices**

One Systems that do not have Nvidia GPU, CPU is used as a device. The discovery cluster has dedicated Nvidia GPU with CUDA. When Code is run on the cluster, the device used is CUDA to perform the image classification.



Output screen shows

Logo

Description automatically generated

**Transformation**

In order to maintain consistency, we resize the image to 150 x 150, flip the images horizontally randomly, convert numpy into tensors and normalize the overall dataset using the formula (X-mean) / std).

Following is the code snippet to do the same.

Text

Description automatically generated

**Data Loading**

Both train and test data are being loaded using DataLoader after fixing the Path.

Text

Description automatically generated

**Classes**

The final goal of using CNN is to classy images into the set 8 classes. Following code snippet shows that how the labels of the images are being read and then displayed.

Text

Description automatically generated

Output is



**Defining CNN**

PyTorch is used to construct the Neural Network. All the layers including Convolutional Layer, Normalized Layer, ReLU Layer, Pooling Layer, Connected Layer, Optimizer and Loss Layer are defined in this step.

3 Convolutional Layer, 3 ReLU Layer and 1 Pooling Layer has been implemented to get the final output matrix of shape (256, 32, 75, 75). The loss function and Optimizer used are “CrossEntropy” and “Adam” respectively.

The learning rate for the optimizer is 0.0001

Text

Description automatically generated

Text

Description automatically generated

**Calculating the size of test and train data**

The following code snippet shows the number of images in train and test dataset being read using glob package.

Text

Description automatically generated

Output is



**Model Training and Evaluating on training dataset**

The model is run for 10 epochs

Text

Description automatically generated

**Evaluating the test dataset and getting the accuracies**

Text

Description automatically generated

**Running on Discovery Cluster**

Following are the steps to run the code on discovery:

1. Transfer dataset and Python File to /scratch/sharma.at using scp
2. Install miniconda on discovery
3. Module load anaconda3 and CUDA
4. Start a GPU node using srun
5. Create a PyTorch Virtual Environment using conda create
6. Activate virtual environment using conda activate
7. Conda install pytorch and torchvision on the environment
8. Run the code using “python car\_image\_classification\_project.py”

**Running on GPU node of Discovery Cluster with 1 GPU**

srun -p reservation --reservation=csye7105-gpu --gres=gpu:v100-sxm2:1 --mem=16Gb --time=05:00:00 --export=ALL --pty /bin/bash

A picture containing calendar

Description automatically generated



The elapsed time for single GPU computing is **3134.994 Seconds**

**Upon running the code on 2 GPUs, following snippet the output and the elapsed time.**

srun -p reservation --reservation=csye7105-gpu --gres=gpu:v100-sxm2:2 --mem=16Gb --time=05:00:00 --export=ALL --pty /bin/bash

A picture containing text, newspaper, screenshot

Description automatically generated

The elapsed time for 2 GPUs is **2479.674 Seconds**

**Visualization of Accuracies**

Serial Computing / Single GPU

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Parallel Computing / 2 GPUs

**Chart, line chart

Description automatically generated**

**Chart

Description automatically generated**

**Calculating the Speedup**

Speedup in Computer Architecture is a relative measure of the performance of two systems processing the same problem. In Parallel Computing, Speedup is the ratio of serial runtime of the best sequential or serial algorithm to solve a problem to the time taken by the parallel algorithm to solve the same problem on n number of processors.

The wall clock time of serial execution is expected to be more than that of parallel execution. The final step of this project is to calculate the speedup, which is done using the formula

wall-clock time of serial execution

Speedup = ---------------------------------------------------------

wall-clock time of parallel execution

**Result**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of GPUs** | **Number of Epochs** | **Best Train Accuracy** | **Best Test Accuracy** | **Minimum Loss** | **Elapsed Time** |
| **1** | 10 | 95.44% | 27.42% | 0.1459 | 3134.994 |
| **2** | 10 | 93.00% | 26.87% | 0.2465 | 2479.674 |

The Speedup from Single GPU to 2 GPUs is **3134.994 / 2479.674**

Finally, **Speedup = 1.2642**

Conclusion

1. Images were successfully classified into the 8 classes based on the body style
2. Data Parallelism was successfully implemented on the code and the code was run using single as well as 2 GPUs to get the elapsed time.
3. We find that there is a significant benefit in using multiple GPUs as the speedup of **1.2642** is quite noticeable, thus pointing towards the fact that parallel computing has performance advantages.

Reference

* <https://developer.nvidia.com/cuda-zone>
* <https://pytorch.org/tutorials/beginner/blitz/data_parallel_tutorial.html>
* <https://www.cs.uky.edu/~jzhang/CS621/chapter7.pdf>
* <https://www.slideshare.net/mobile/burhanAhmed14/parallel-computing-and-its-applications>
* <https://en.wikipedia.org/wiki/Speedup>
* <https://pytorch.org/docs/stable/torchvision/index.html>
* <https://en.wikipedia.org/wiki/Data_parallelism>
* <https://en.wikipedia.org/wiki/Convolutional_neural_network>
* <https://en.wikipedia.org/wiki/PyTorch>
* <https://rc-docs.northeastern.edu/en/latest/using-discovery/workingwithgpu.html>
* <https://github.com/gaurav67890/Pytorch_Tutorials/blob/master/cnn-scratch-training.ipynb>