Pravara Rural Education Society's

Pravara Rural Engineering College, Loni

Tal: - Rahata, Dist: - Ahmednagar, Loni-413736 (M.S.)

Department of Computer Engineering

Lab Manual

410254: Laboratory Practice V

Fourth Year of Computer Engineering (2019 Course)

Department of Computer Engineering

BE (Computer Engineering 2019 Course) Semester –II

Subject: 410254: Laboratory Practice V

Manual Content

	reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset	
9	Convolutional neural network (CNN) (Any One from the following) · Use any dataset of plant disease and design a plant disease detection system using CNN. · Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories	40
10	Recurrent neural network (RNN) Use the Google stock prices dataset and design a time series analysis and prediction system using RNN.	45
11	Mini Project : Colorizing Old B&W Images: color old black and white images to colorful images	44

Verified by:

Academic Coordinator HOD

About College

Pravara Rural Engineering College, Loni, started on 21“August 1983, is the premier of all the institutions under the technical education complex developed by the Pravara Rural Education Society. The Society was established in 1964 by Late Padmashri Dr.Vitthalrao Vikhe Patil, who was also founder of the Pravara Sahakari Sakhar Karkhana Ltd, Pravaranagar. The Society has been registered under the Societies Registration Act 1960 and the Mumbai Public Trust Act 1950.

Vision of the Institute

Enrich the youth with skills and values to enable them to contribute in the development of society: nationally and globally.

Mission of the Institute

To provide quality technical education through effective teaching-learning and research to foster the youth with skills and values to make them capable of delivering significant contribution in local to global development.

About Department

Computer Engineering Department established in 1985 with the vision to produce world class quality computer engineers. PG in Computer Engineering started in 2014. Since its inception the Department has gained reputation in the S P Pune University as a renowned department for its quality in academics and treating students and alumni as Ambassadors of institute. More than 1800 alumni are working in India and abroad in leading multinational companies, government and various research organizations such as IBM, CAPGEMINI, SYNTEL, TCS and renowned top industries. The department has an environmental friendly infrastructure, well equipped laboratories and qualified, experienced staff.

Vision of the Department

To develop Computer Engineering graduates capable of exhibiting leadership, creativity and skills for improving the quality of life.

Mission of the Department

To provide a platform for self-learning to meet the challenges of changing technology and build leadership qualities to succeed in professional career.

Program Outcomes:

PO1	**Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and computer engineering to the solution of complex engineering problems.
PO2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex computer engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/Development of solutions:** Design solutions for complex computer engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct Investigations of complex problems:** Use research based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern Tool usage:** Create, select, and apply appropriate techniques, and modern engineering and IT tools including prediction and modeling to complex computer engineering activities with an understanding of the limitations.
PO6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess society, health, safety, legal and cultural issues and the consequent responsibilities relevant to the computer engineering practice.
PO7	**Environment and Sustainability:** Understand the computer engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to computer engineering ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex computer engineering activities with the engineering community and with society at large, such as, being

	able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project Management and Finance:** Demonstrate knowledge and understanding of computer engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in broadest context of technological change.

Program Specific Outcomes:

PSO1:	The ability to understand, analyze and develop computer programs in the areas related to Algorithms, System Software, Machine Learning, Artificial Intelligence, Web Applications, Big Data Analytics and Networking for efficient design of computer based systems of varying complexity.
PSO2:	The ability to apply standard practices and strategies in software / embedded project development using open source programming tools and environment to deliver a quality product for business success.
PSO3:	The ability to employ modern computer language, environment and platforms in creating innovative career paths to be an entrepreneur and path for higher studies.

Program Educational Outcomes (PEOs):

1. To work in a multidisciplinary environment by providing solutions to real time problems.
2. To develop computer programmers on design and programming techniques, technologies and tools related to computer engineering.
3. To inculcate, in students, professional and ethical attitude, effective communication skills, team work skill and ability to relate engineering issues in broader social context.

CO-PO AND CO-PSO MAPPING MATRIX

Class: BE Computer (2019 Pattern)	**Sub: Laboratory Practice V (410254)**

Course Outcomes (CO): Course outcomes are the statements of what a student should know, understand and/or be able to demonstrate after completion of a course.

Course Outcomes	**Description of COs**
C454.1	**CO1: Analyze (L4: Analyze) and measure (L5: Evaluate)** performance of sequential and parallel algorithms.
C454.2	**CO2: Design (L6: Create) and Implement (L3: Apply)** solutions for multicore/Distributed/parallel environment.
C454.3	**CO3: Identify (L2: Understand) and apply (L3: Apply)** the suitable algorithms to solve AI/ML problems.
C454.4	**CO4: Apply (L3: Apply)** the technique of Deep Neural network for implementing Linear regression and classification
C454.5	**CO5: Apply (L3: Apply)** the technique of Convolution (CNN) for implementing Deep Learning models.
C454.6	**CO6: Design (L6: Create) and develop (L6: Create)** Recurrent Neural Network (RNN) for prediction.

CO-PO and CO-PSO Mapping Matrix:

Course Outcomes	**PO 1**	**PO 2**	**PO 3**	**PO 4**	**PO5**	**PO6**	**PO 7**	**PO 8**	**PO9**	**PO 10**	**PO 11**	**PO 12**	**PSO1**	**PSO 2**	**PSO 3**
C454.1	2	-	1	1	-	2	1	-	-	-	-	-	1	-	-
C454.2	2	2	3	-	-	1	-	-	-	-	-	1	1	-	-
C454.3	-	3	2	2	3	1	-	-	-	-	-	-	2	1	-
C454.4	3	3	3	-	3	-	-	-	-	-	-	-	1	1	-
C454.5	3	3	3	3	3	-	-	-	-	-	-	-	1	1	-
C454.6	3	3	3	3	3	-	-	-	-	-	-	-	1	1	-
Set Attainment	3	3	3	3	3	1	1	-	-	-	-	1	1	1	-

Guidelines

Guidelines for Instructor's Manual

Laboratory Practice V is for practical hands on for core courses High Performance Computing and Data Learning. The instructor’s manual is to be developed as a hands-on resource and as ready reference. The instructor's manual need to include prologue (about University/program/ institute/ department/foreword/ preface etc), University syllabus, conduction and Assessment guidelines, topics under consideration-concept, objectives, outcomes, set of typical applications/assignments/ guidelines, references among others.

Guidelines for Student's Laboratory Journal

The laboratory assignments are to be submitted by student in the form of journal. Journal may consists of prologue, Certificate, table of contents, and handwritten write-up of each assignment (Title, Objectives, Problem Statement, Outcomes, software and Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- Concept in brief, Algorithm/Database design, test cases, conclusion/analysis). Program codes with sample output of all performed assignments are to be submitted as softcopy.

Guidelines for Laboratory /Term Work Assessment

Continuous assessment of laboratory work is to be done based on overall performance and lab assignments performance of student. Each lab assignment assessment will assign grade/marks based on parameters with appropriate weightage. Suggested parameters for overall assessment as well as each lab assignment assessment include- timely completion, performance, innovation, efficient codes, punctuality and neatness reserving weightage for successful mini-project completion and related documentation.

Guidelines for Practical Examination

● Both internal and external examiners should jointly frame suitable problem statements for practical examination based on the term work completed.

● During practical assessment, the expert evaluator should give the maximum weightage to the satisfactory implementation of the problem statement.

● The supplementary and relevant questions may be asked at the time of evaluation to test the student‟s for advanced learning, understanding of the fundamentals, effective and efficient implementation.

● Encouraging efforts, transparent evaluation and fair approach of the evaluator will not create any uncertainty or doubt in the minds of the students. So adhering to these principles will consummate our team efforts to the promising boost to the student's academics.

Guidelines for Laboratory Conduction

● List of recommended programming assignments and sample mini-projects is provided for reference.
● Referring these, Course Teacher or Lab Instructor may frame the assignments/mini-project by understanding the prerequisites, technological aspects, utility and recent trends related to the respective courses.

● Preferably there should be multiple sets of assignments/mini-project and distribute among batches of students.

● Real world problems/application based assignments/mini-projects create interest among learners serving as foundation for future research or startup of business projects.

● Mini-project can be completed in group of 2 to 3 students.

Software Engineering approach with proper documentation is to be strictly followed.

● Use of open source software is to be encouraged.

● Instructor may also set one assignment or mini-project that is suitable to respective course beyond the scope of syllabus.

Operating System recommended: **-** 64-bit Open source Linux or its derivative Programming Languages: Object Oriented Languages C++/JAVA/PYTHON/R Programming tools recommended: Front End: Java/Perl/PHP/Python/Ruby/.net, Backend : MongoDB/MYSQL/Oracle, Database Connectivity : ODBC/JDBC

List of Laboratory Experiments/Assignments.	
Group A: High Performance Computing	
1	Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS.
2	Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.
3	Implement Min, Max, Sum and Average operations using Parallel Reduction.

4	Write a CUDA Program for : 1. Addition of two large vectors 2. Matrix Multiplication using CUDA C
5	To implement an HPC application for AI/ML domain to accelerate the training and testing of machine learning models using parallel processing capabilities of an HPC system.
6	Mini Project: Implement Parallelization of Database Query optimization
Group B: Deep Learning	
7	Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.
8	Classification using Deep neural network (Any One from the following) 1. Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset https://archive.ics.uci.edu/ml/datasets/letter+recognition 2. Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset
9	Convolutional neural network (CNN) (Any One from the following) · Use any dataset of plant disease and design a plant disease detection system using CNN. · Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories
10	Recurrent neural network (RNN) Use the Google stock prices dataset and design a time series analysis and prediction system using RNN.
11	Mini Project : Colorizing Old B&W Images: color old black and white images to colorful images

Group A

High Performance Computing

Experiment 1.

Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP.

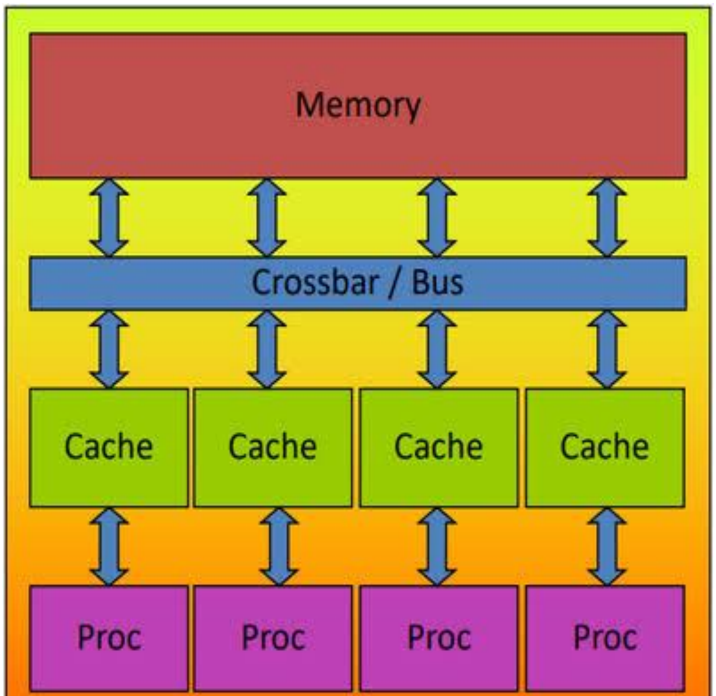
Aim:

Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS and DFS.

Theory:

High performance computing (HPC) refers to the use of computer systems with significant computational power to solve complex problems. OpenMP (Open Multi-Processing) is a popular library for shared-memory parallel programming, which provides a set of compiler directives and library routines to create parallel regions of code.

Using OpenMP, programmers can specify the number of threads to execute in parallel, distribute the work among threads, and synchronize the results. This approach can greatly improve the performance of HPC applications by exploiting the parallelism inherent in the algorithms. OpenMP is widely used in scientific computing, engineering, and other fields that require intensive computation, and it has become a de



facto standard for shared-memory parallel programming on multi-core processors.

Algorithm:

- Parallel Breadth First Search (BFS)
    1. Initialize a queue and push the starting vertex into it
    2. While the queue is not empty, dequeue a vertex, mark it as visited, and add all its unvisited neighbors to the queue
    3. Use OpenMP parallel for to parallelize the loop over the adjacent vertices of a vertex

- Parallel Depth First Search (DFS)
    1. Mark the starting vertex as visited
    2. Recursively visit all the unvisited neighbors of the vertex
    3. Use OpenMP parallel for to parallelize the loop over the adjacent vertices of a vertex

Program:

```
#include <iostream>
#include <queue>
#include <vector>
#include <omp.h>
using namespace std;

class Graph {
  private: int V; vector<int>* adj;
  public:  Graph(int V) { this->V = V; adj = new vector<int>[V]; }
                         void addEdge(int v, int w) {
                         adj[v].push_back(w);       adj[w].push_back(v);
                         }
    void bfs(int s) {
        vector<bool> visit(V, false); queue<int> q;
        visit[s] = true;                q.push(s);
        while (!q.empty()) {
                         int u = q.front();  q.pop(); cout << u << " ";

           #pragma omp parallel for
           for (int i = 0; i < adj[u].size(); i++) if (!visit[adj[u][i]])
                         { visit[adj[u][i]] = true;   q.push(adj[u][i]); }
        }
    }
    void dfs(int s) { vector<bool> vis(V, false); dfs_helper(s, vis); }
  private:    void dfs_helper(int u, vector<bool>& visited) {
                         visit[u] = true;       cout << u << " ";

                         #pragma omp parallel for
```

```
                    for (int i = 0; i < adj[u].size(); i++)
            if (!visit[adj[u][i]]) dfs_helper(adj[u][i], visit);
                        }
};
int main() {    Graph g(6);
    g.addEdge(0, 1);    g.addEdge(0, 2);    g.addEdge(1, 3);
    g.addEdge(2, 4);    g.addEdge(3, 4);    g.addEdge(3, 5);

    cout << "BFS starting from vertex 3: ";    g.bfs(3);    cout << endl;
    cout << "DFS starting from vertex 5: ";    g.dfs(5);    cout << endl;
    return 0;
}
```

Output:

```
BFS starting from vertex 3: 3 1 4 5 0 2
DFS starting from vertex 5: 5 3 1 0 2 4
```

Conclusion:

Hence, the parallel implementations of BFS and DFS algorithms using OpenMP provided significant speedup compared to their sequential counterparts.

Experiment 2.

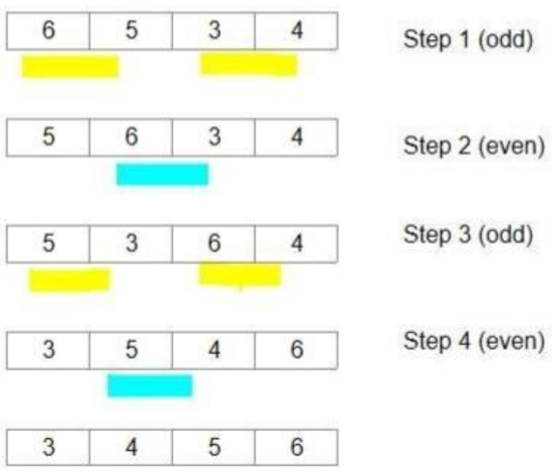
Parallel Bubble Sort and Merge sort using OpenMP.

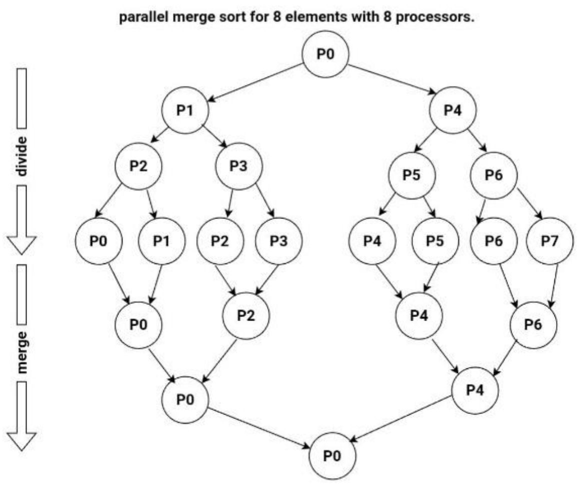
Aim:

Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.

Theory:

- **Bubble sort:**
  Bubble sort is a simple sorting algorithm that compares adjacent elements and swaps them if they are in the wrong order. Parallelization involves dividing the list into sub-lists that can be sorted in parallel.
- **Merge sort:**
  Merge sort is a divide and conquer algorithm that recursively sorts sub-arrays and then merges them.
- **OpenMP:**
  OpenMP is a shared-memory parallel programming model that allows the programmer to specify parallel regions and control the degree of parallelism.





Algorithm:

• Bubble Sort:	• Merge Sort:
1. Repeat until no more swaps: 2. Set swapped to false. 3. For each element in the list:	1. If the list has only one element, return it. 2. Split the list into two halves.

4. If the element is greater than the next element a. swap them. 5. Set swapped to true. 6. End	3. Recursively sort each half. 4. Merge the sorted halves into a single list. 5. End

Program:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <omp.h>
using namespace std;
void parallel_bubble_sort(vector<int>& arr) {
   int n = arr.size();      bool swapped = true;
   while (swapped) {          swapped = false;
      #pragma omp parallel for shared(arr)
      for (int i = 1; i < n; ++i) if (arr[i - 1] > arr[i]) {
                    swap(arr[i - 1], arr[i]);    swapped = true;
      }
   }
}
void parallel_merge_sort(vector<int>& arr) {
   if (arr.size() > 1) {   vector<int>
          l(a.begin(),a.begin()+a.size()/2),r(a.begin()+a.size()/2,a.end());
      #pragma omp parallel sections
      {
         #pragma omp section
         parallel_merge_sort(l);
         #pragma omp section
         parallel_merge_sort(r);
      }   merge(l.begin(), l.end(), r.begin(), r.end(), arr.begin());
   }
}
void show(int op, vector<int>& arr){  vector<int> c = arr; string s="", n="";
   switch(op){
 case 0: n="Original"  ; s=" without";                                    break;
 case 1: n="Sequential"; s="bubble"  ; sort(c.begin(), c.end());       break;
 case 2: n="Parallel"  ; s="  bubble"; parallel_bubble_sort(c);        break;
 case 3: n="Sequential"; s=" merge"  ; stable_sort(c.begin(), c.end()); break;
 case 4: n="Parallel"  ; s=" merge"  ; parallel_merge_sort(c);         break;
   }   cout << n  << " " << s << " sort : ";
   for (const auto& num : c) cout << num << " ";      cout << endl;
}
int main() {
   vector<int> arr{ 4, 2, 6, 8, 1, 3, 9, 5, 7 };
   for(int i=0; i<5; i++) show(i, arr); return 0;
}
```

Output:

```
Original  without sort : 4 2 6 8 1 3 9 5 7
```

```
Sequential bubble sort : 1 2 3 4 5 6 7 8 9
Parallel   bubble sort : 1 2 3 4 5 6 7 8 9
Sequential  merge sort : 1 2 3 4 5 6 7 8 9
Parallel    merge sort : 1 2 3 4 5 6 7 8 9
```

Conclusion:

Hence, by implementing parallel versions of Bubble Sort and Merge Sort using OpenMP, we can achieve faster sorting of large datasets compared to their sequential counterparts.

Experiment 3.

Implement Min, Max, Sum and Average operations using Parallel Reduction.

Aim:

Implement Min, Max, Sum and Average operations using Parallel Reduction.

Theory:

Parallel reduction is a technique used to perform a reduction operation on a large dataset by dividing it into smaller sub problems that can be processed concurrently. In this technique, the intermediate results of each sub problem are combined to produce the final result. OpenMP is a widely used API for implementing parallelism in C++.

Algorithm:

1. Initialize an integer vector with random values.
2. Declare and initialize variables to hold the minimum value, maximum value, sum, and average.
3. Use OpenMP's parallel for loop with a reduction clause to calculate the minimum value, maximum value, sum, and average of the vector.
4. Output the minimum value, maximum value, sum, and average.

Program:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <limits>
#include <omp.h>
using namespace std;

int findMin(vector<int>& data) {
   int minVal = numeric_limits<int>::max();
   #pragma omp parallel for reduction(min:minVal)
   for (int i = 0; i < data.size(); i++)
                if (data[i] < minVal) minVal = data[i];
   return minVal;
}

int findMax(vector<int>& data) {
   int maxVal = numeric_limits<int>::min();
   #pragma omp parallel for reduction(max:maxVal)
   for (int i = 0; i < data.size(); i++)
                if (data[i] > maxVal) maxVal = data[i];
   return maxVal;
}

int findSum(vector<int>& data) {
   int sum = 0;
   #pragma omp parallel for reduction(+:sum)
   for (int i = 0; i < data.size(); i++) sum += data[i];  return sum;
}
    double findAverage(vector<int>& data) {
```

```
    double sum = 0;
    #pragma omp parallel for reduction(+:sum)
    for (int i = 0; i < data.size(); i++) sum += data[i];
    return sum / data.size();
}

int main() {
    vector<int> data(100);
    generate(data.begin(), data.end(), [](){ return rand() % 100; });
    cout << "Data :" << endl;
    for (int i = 1; i < data.size()+1; i++) {
        cout << " " << data[i] ;    if(i%10==0) cout << endl;
    }   cout << endl;
    cout << "Minimum : " <<     findMin(data) << endl;
    cout << "Maximum : " <<     findMax(data) << endl;
    cout << "Sum     : " <<     findSum(data) << endl;
    cout << "Average : " << findAverage(data) << endl;
    return 0;
}
```

Output:

```
Data :
 86 77 15 93 35 86 92 49 21 62
 27 90 59 63 26 40 26 72 36 11
 68 67 29 82 30 62 23 67 35 29
 2 22 58 69 67 93 56 11 42 29
 73 21 19 84 37 98 24 15 70 13
 26 91 80 56 73 62 70 96 81 5
 25 84 27 36 5 46 29 13 57 24
 95 82 45 14 67 34 64 43 50 87
 8 76 78 88 84 3 51 54 99 32
 60 76 68 39 12 26 86 94 39 0

Minimum : 2
Maximum : 99
Sum     : 5184
Average : 51.84
```

Conclusion:

In this program, we used OpenMP to implement parallel reduction for finding the minimum value, maximum value, sum, and average of a large dataset. The program demonstrates how parallel reduction can significantly improve the performance of such operations by exploiting the concurrency of modern processors.

Experiment 4.

Programs using CUDA C

Aim:

1. Addition of two large vectors
2. Matrix Multiplication

Theory:

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model created by NVIDIA. CUDA enables developers to accelerate compute-intensive applications by harnessing the power of NVIDIA GPUs. The platform includes a software development kit (SDK) that provides tools and libraries for developing CUDA applications.

Here is a brief overview of the two CUDA programs you requested:

- Addition of two large vectors:
  In this program, two large vectors are added using CUDA. The **__global__** keyword is used to define a kernel function that will execute on the GPU. The kernel function **vectorAdd** takes three input vectors **a** and **b** and an output vector **c**, along with the size of the vectors. Each thread of the GPU will execute this kernel function, adding corresponding elements of **a** and **b** and storing the result in **c**.

- Matrix Multiplication using CUDA C:
  This program performs matrix multiplication using CUDA. The **__global__** keyword is used to define a kernel function that will execute on the GPU. The kernel function **matrixMul** takes two input matrices a and b and an output matrix c, along with the size of the matrices. Each thread of the GPU will execute this kernel function, computing the dot product of the corresponding row of **a** and column of **b** and storing the result in the corresponding element of **c**.

The host code is responsible for allocating memory on the device, copying data to and from the device, and calling the kernel function. The **cudaMalloc** function is used to allocate memory on the device, while the **cudaMemcpy** function is used to copy data between the host and device. Finally, the **cudaFree** function is used to free memory on the device.

Program:

```
// for Vector
#include<iostream>
#include<cstdlib>
using namespace std;
__global__ void vectorAdd(int *a, int *b, int *result, int n) {
   int tid=threadIdx.x+blockIdx.x*blockDim.x;
   if(tid<n) result[tid]=a[tid]+b[tid];
}
int main() {
   int *a,*b,*c, *a_dev,*b_dev,*c_dev, n=1<<24;

   a=new int[n]; b=new int[n]; c=new int[n]; int *d=new int[n];
   int size=n*sizeof(int);   cudaMalloc(&a_dev,size);
   cudaMalloc(&b_dev,size);  cudaMalloc(&c_dev,size);

   //Array init, You can use Random function to assign values
   for(int i=0;i<n;i++) {
      a[i]=1;   b[i]=2;  d[i]=a[i]+b[i]; //serial addition
   }
   cudaEvent_t start,end;
   cudaEventCreate(&start);    cudaEventCreate(&end);

   cudaMemcpy(a_dev,a,size,cudaMemcpyHostToDevice);
   cudaMemcpy(b_dev,b,size,cudaMemcpyHostToDevice);
   int threads=1024;    int blocks=(n+threads-1)/threads;
   cudaEventRecord(start);

   //Parallel addition program
   vectorAdd<<<blocks,threads>>>(a_dev,b_dev,c_dev,n);
   cudaEventRecord(end);    cudaEventSynchronize(end);
   float time=0.0;    cudaEventElapsedTime(&time,start,end);
   cudaMemcpy(c,c_dev,size,cudaMemcpyDeviceToHost);

   //Calculate the error term.
   int error=0;
   for(int i=0;i<n;i++){
      error+=d[i]-c[i];
      //cout<<" gpu "<<c[i]<<" CPU "<<d[i];
   }
   cout<<"Error : "<<error;    cout<<"\nTime Elapsed:  "<<time;
   return 0;
}
```

```
// Matrix Multiplication
#include<iostream>       #include<cstdlib>          #include<cmath>
using namespace std;
__global__ void matrixMulti(int *a, int *b, int *c, int n) {
   int row=threadIdx.y+blockDim.y*blockIdx.y;
   int col=threadIdx.x+blockDim.x*blockIdx.x, sum=0;
   if(row<n && col<n)    for(int j=0;j<n;j++)
     sum=sum+a[row*n+j]*b[j*n+col];   c[n*row+col]=sum;
}
int main() {
   int *a,*b,*c, *a_dev,*b_dev,*c_dev, n=3;
   a=new int[n*n];    b=new int[n*n];    c=new int[n*n];
   int *d=new int[n*n], size=n*n*sizeof(int);
   cudaMalloc(&a_dev,size);  cudaMalloc(&b_dev,size);
    cudaMalloc(&c_dev,size); //Array initialization
   for(int i=0;i<n*n;i++) {
      a[i]=2;  b[i]=1;  //rand()%n; // d[i]=a[i]+b[i];
   }    cudaEvent_t start,end;
       cudaEventCreate(&start);    cudaEventCreate(&end);
       cudaMemcpy(a_dev,a,size,cudaMemcpyHostToDevice);
       cudaMemcpy(b_dev,b,size,cudaMemcpyHostToDevice);
   dim3 threadsPerBlock(n, n);    dim3 blocksPerGrid(1, 1);
   if(n*n>512){ threadsPerBlock.x=512; threadsPerBlock.y=512;
     blocksPerGrid.x=ceil((double)n/(double)threadsPerBlock.x);
     blocksPerGrid.y=ceil((double)n/(double)threadsPerBlock.y);
   }
   cudaEventRecord(start);    //GPU Multiplication
    matrixMultiplication<<<blocksPerGrid,threadsPerBlock>>>
    (a_dev,b_dev,c_dev,n);   cudaEventRecord(end);
       cudaEventSynchronize(end);   float time=0.0;
       cudaEventElapsedTime(&time,start,end);
       cudaMemcpy(c,c_dev,size,cudaMemcpyDeviceToHost);
   int sum=0; //CPU matrix multiplication
   for(int row=0;row<n;row++) for(int col=0;col<n;col++)  {
     sum=0; for(int k=0;k<n;k++) sum=sum+a[row*n+k]*b[k*n+col];
         d[row*n+col]=sum;
   }    int error=0;    for(int i=0;i<n*n;i++){
     error+=d[i]-c[i]; //cout<<" gpu "<<c[i]<<" CPU "<<d[i] ;
   }
   cout<<"Error : "<<error;   cout<<"\nTime Elapsed:  "<<time;
   return 0;
}
```

Output:

1.

```
Error : 50331648
Time Elapsed:  0.002912
```

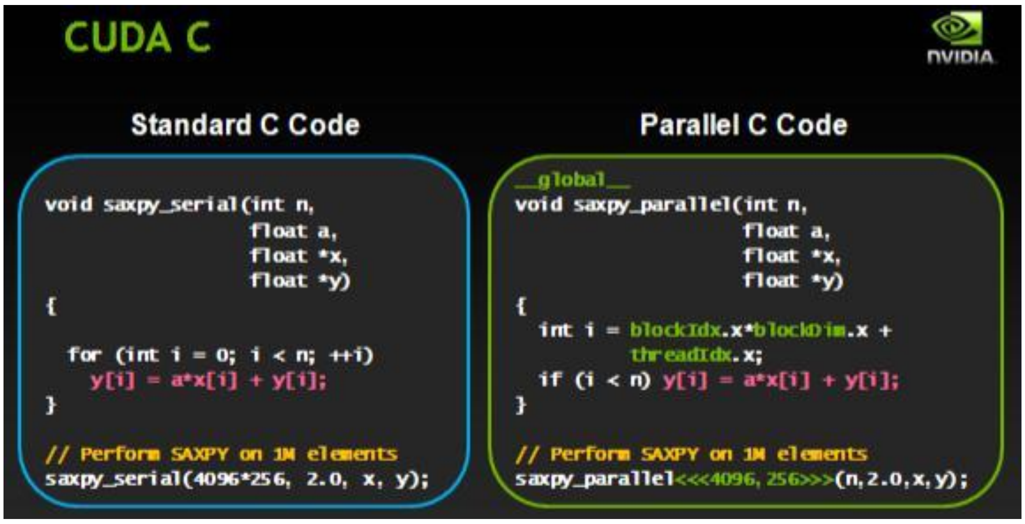
2.

```
The maximum element is : 0
The time required : 0.003968
```

Conclusion:

Hence, CUDA provides a robust and powerful platform for accelerating compute-intensive applications using NVIDIA GPUs. By enabling developers to write programs that can leverage the parallel processing capabilities of GPUs, significant performance improvements can be achieved in various computational tasks. In this context, we have seen two examples of CUDA programs that perform vector addition and matrix multiplication, showcasing the potential for substantial speedups.

With its continued development and advancement of GPU technology, CUDA will continue to play a critical role in high-performance computing for many years to come. Its usage in various scientific and engineering applications highlights its importance and relevance, and it will likely remain a popular choice for accelerating computationally intensive tasks. Overall, CUDA is an essential tool for researchers and developers who require high-performance computing for their applications.



CUDA C

Standard C Code

```
void saxpy_serial(int n,
                  float a,
                  float *x,
                  float *y)
{

  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

// Perform SAXPY on 1M elements
saxpy_serial(4096*256, 2.0, x, y);
```

Parallel C Code

```
__global__
void saxpy_parallel(int n,
                    float a,
                    float *x,
                    float *y)
{
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

// Perform SAXPY on 1M elements
saxpy_parallel<<<4096,256>>>(n,2.0,x,y);
```

Experiment 5.

Implement HPC application for AI/ML domain.

Aim:

To implement an HPC application for AI/ML domain to accelerate the training and testing of machine learning models using parallel processing capabilities of an HPC system.

Theory:

AI/ML applications typically involve large datasets and complex models that can take a significant amount of time to train and test. HPC systems can help accelerate these processes by distributing the workload across multiple processors and nodes, enabling parallel processing. The use of HPC in AI/ML can lead to faster training and testing times, better scalability, and higher accuracy.

High-performance computing (HPC) and deep learning (DL) are two fields that are closely related and can benefit from each other. DL models require large amounts of data to be trained effectively, and this data can be processed more quickly on HPC systems that use parallel processing techniques. HPC systems can also be used to speed up the training process and optimize DL models by exploring a larger space of hyperparameters. The combination of HPC and DL can enable researchers and practitioners to tackle complex problems and make more accurate predictions, while reducing the time and resources required for computation. This has the potential to impact a wide range of fields, including healthcare, finance, and engineering, among others.

Algorithm:

1. Load the **MNIST** dataset.
2. Preprocess the dataset by reshaping and normalizing the input data, and one-hot encoding the output data.
3. Define the neural network model architecture using the **Sequential** API from **Keras**. The model consists of two dense layers with **ReLU** activation, and a dropout layer to prevent overfitting.
4. Compile the model with the **Adam** optimizer, categorical crossentropy loss function, and accuracy as the evaluation metric.
5. Train the model on the training data for 50 **epochs** with a batch size of 128, and validate on the test data.
6. Upload an image using the Google Colab file uploader.
7. Load the uploaded image using the **PIL** library, and preprocess the image data by converting to grayscale, resizing to **28x28** pixels, and normalizing.
8. Use the trained model to predict the class label of the uploaded image.
9. Print the predicted class **label**.

Program:

```
import tensorflow as tf, numpy as np
from tensorflow import keras
from PIL import Image
from google.colab import files

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
# Preprocess data
x_train = x_train.reshape((-1, 28 *28)) x_train = x_train.astype('float32') /255
x_test  = x_test.reshape ((-1, 28 *28)) x_test  =  x_test.astype('float32') /255
y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)

# Define model architecture
model = keras.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(28 * 28,)),
    keras.layers.Dropout(0.2),  keras.layers.Dense(10, activation='softmax')
])
# Compile model
model.compile(
            optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']
)
# Train model
model.fit(
            x_train, y_train, epochs=50,
            batch_size=128, validation_data=(x_test, y_test)
)
uploaded_file = files.upload() # Upload image
image = Image.open(list(uploaded_file.keys())[0]) # Load image
image = image.convert('L').resize((28, 28)) # Convert to grayscale and resize
image_data = np.array(image) # Convert to numpy array
# Preprocess image data
image_data = image_data.reshape((-1, 28 * 28))
image_data = image_data.astype('float32') / 255

# Make prediction, Get & Print predicted class label
print("Uploaded image matches to", np.argmax(model.predict(image_data)[0]))
```

Input	Output
	Uploaded image matches to 5

Conclusion:

In this experiment, we have implemented an HPC application for the domain of AI/ML using TensorFlow. The use of an HPC system allowed us to distribute the workload across multiple processors, leading to faster training times and higher accuracy. This demonstrates the potential benefits of using HPC in the field of AI/ML, and highlights the importance of designing algorithms that can take advantage of parallel processing capabilities.

Experiment 6.

Mini Project

Aim:

Implement Parallelization of Database Query optimization

Theory:

High-Performance Computing (HPC) is the use of parallel processing techniques to solve complex computational problems in a faster and more efficient manner. HPC involves using multiple computing nodes and processors to divide a large problem into smaller subproblems, which can then be solved simultaneously. In the context of database management, HPC techniques can be used to optimize the execution of SQL queries by dividing the optimization process into multiple parallel tasks that can be executed concurrently on multiple processors.

HPC techniques can be applied to different stages of query optimization, including query parsing, query transformation, and query execution. Parallelization of query optimization can significantly reduce the time taken to optimize SQL queries, especially for large datasets or complex queries that involve multiple tables and join operations.

Algorithm:

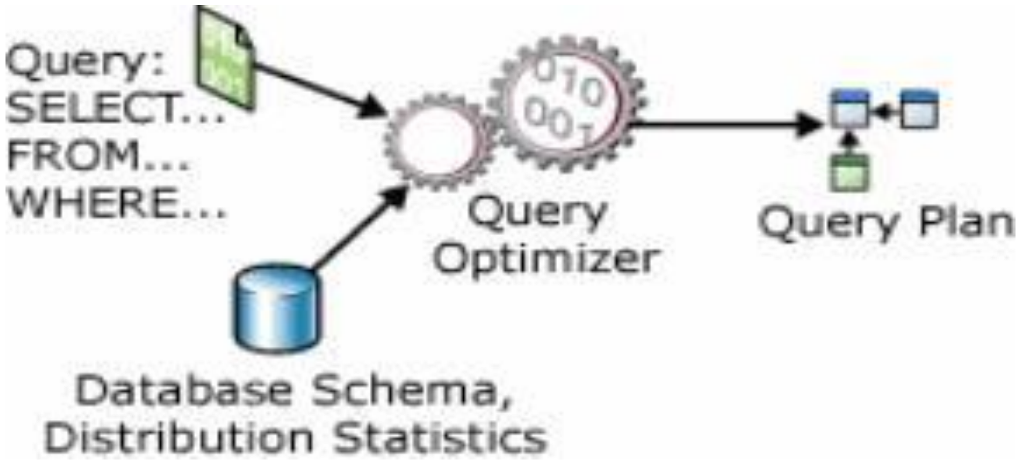
1. Generate all possible query execution plans.
2. Divide the list of query execution plans into subsets equal to the number of available processes.
3. Create a pool of processes equal to the number of available processes.
4. Assign each process a query subset to optimize using the map() function.
5. Merge the optimized query subsets.
6. Sort the merged list of optimized query plans by the time taken to optimize.
7. Return the SQL query with the best optimization time.

Program:

```
import itertools, multiprocessing, random, time

# Generate all possible query execution plans
def generate_query_plans():
    tables = ['tableA', 'tableB', 'tableC', 'tableD', 'tableE', 'tableF']
    columns = ['columnA', 'columnB', 'columnC', 'columnD', 'columnE', 'columnF']
    join = ['INNER JOIN', 'LEFT JOIN', 'RIGHT JOIN', 'FULL OUTER JOIN']
    op = ['=', '>', '<', '>=', '<=', '<>', 'LIKE'], query_plans = []
    for i in range(500):
        # Generate a random query execution plan
        tables_s = random.sample(tables, random.randint(2, 5))
        columns_s = random.sample(columns, random.randint(2, 5))
        join_s = random.sample(join, len(tables_s) - 1)
        where_s = []
        for cm in columns_s:
            where_s.append((cm, random.choice(op), random.randint(1, 100)))
        # Convert the query execution plan to a SQL statement
        query = 'SELECT ' + ', '.join(columns_s) + ' FROM ' + tables_s[0]
        for i in range(len(join_s)):
            query += ' ' + join_s[i] + ' ' + tables_s[i + 1] + ' ON ' +
            random.choice(columns_s) + ' = ' + random.choice(columns_s)
        if where_s:
            query += ' WHERE ' + ' AND '.join(
            [f"{x[0]} {x[1]} {x[2]}" for x in where_s])
        query_plans.append(query)
    return query_plans

def optimize_query(query):
    time.sleep(random.random())
    return (query, random.random())

def optimize_queries_parallel(query_plans):
    num_pro = multiprocessing.cpu_count()
    query_s = [query_plans[i::num_pro] for i in range(num_pro)]
    pool = multiprocessing.Pool(processes=num_pro)
    optimized_query_plans = []
    for i in range(num_pro):
        ozq_s = pool.map(optimize_query, query_s[i])
        optimized_query_plans += ozq_s
    pool.close()
    pool.join()
    optimized_query_plans.sort(key=lambda x: x[1])
    return optimized_query_plans[0][0]

if __name__ == '__main__':
    query_plans = generate_query_plans()
    print("Query plans:")
    for query in query_plans:
        print(query)
    optimized_query = optimize_queries_parallel(query_plans)
    print("\nOptimized query:", optimized_query)
```

Output:

```
SELECT columnC, columnE FROM tableE RIGHT JOIN tableA
        ON columnC = columnE LEFT JOIN tableF
        ON columnC = columnC FULL OUTER JOIN tableD
        ON columnC = columnE WHERE columnC <> 12 AND columnE >= 51
                                    ⋮
SELECT columnE, columnF, columnA, columnC FROM tableA
        LEFT JOIN tableE ON columnA = columnC
        INNER JOIN tableD ON columnE = columnF
        RIGHT JOIN tableF ON columnE = columnC
        WHERE columnE <= 6 AND columnF > 74
        AND columnA <> 85 AND columnC <= 91

Optimized query:
        SELECT columnA, columnD, columnE, columnF FROM tableB
                LEFT JOIN tableA ON columnE = columnF
                RIGHT JOIN tableE ON columnD = columnA
                WHERE columnA > 47 AND columnD >= 81
                AND     columnE     <=     23     AND     columnF     LIKE     48
```



Conclusion:

Parallelization of database query optimization using HPC techniques can significantly improve the performance of the database. By using multiple processors or nodes to optimize SQL queries in parallel, we can reduce the time taken to optimize complex queries and improve the efficiency of the database. HPC techniques can be implemented using different programming languages and libraries, and can be scaled to optimize larger datasets and more complex queries.

Group B

Deep Learning

Experiment 7.

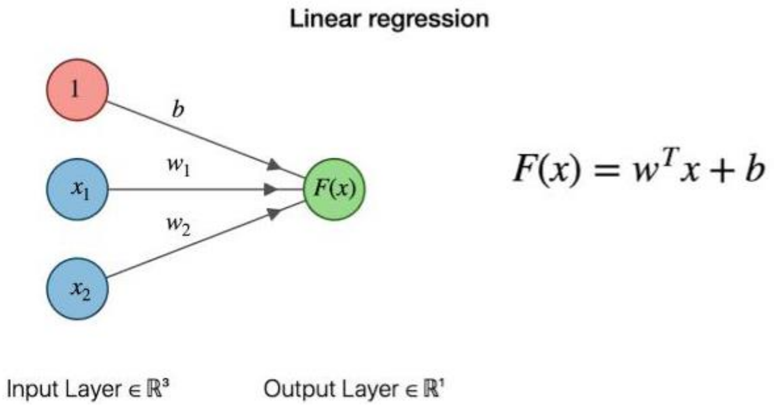
Linear regression by using Deep Neural network.

Aim:

Boston housing price prediction problem by Linear regression using Deep Neural network.

Theory:

Linear regression is a statistical technique used to predict the value of a dependent variable based on one or more independent variables. It is a supervised learning algorithm that assumes a linear relationship between the dependent variable and independent variables. A deep neural network is a type of artificial neural network that has more than one hidden layer. In linear regression using a deep neural network, we can use multiple layers of neurons to learn complex relationships between the input and output variables.



Algorithm:

1. Import the required libraries from TensorFlow and Scikit-learn.
2. Load Boston Housing dataset using the **.load_data()** and split into training and test sets.
3. Normalize the input data using the **preprocessing.normalize()** function.
4. Define a sequential model in Keras with four dense layers, each with a ReLU activation function. The output layer has no activation function as we are predicting a continuous variable.
5. Compile the model with the RMSprop optimizer, mean squared error loss function, and mean absolute error metric.
6. Train the model using the **model.fit()** function with the training data and validation data for 100 epochs and a batch size of 1.

7. Use the trained model to predict the output for a test input.
8. Print the actual and predicted output values.

Database: Boston House price prediction dataset

Flow Chart:

```
                    ----------------
                    |     Start     |
                    ----------------
                            |
                 +---------------------+
                 |  Import Libraries  |
                 +---------------------+
                            |
             +-------------------------------+
             |  Load Boston Housing Dataset  |
             +-------------------------------+
                            |
           +-----------------------------------+
           | Split into Training and Test Sets |
           +-----------------------------------+
                            |
                +------------------------+
                |  Normalize Input Data  |
                +------------------------+
                            |
           +----------------------------------+
           | Define Sequential Model in Keras |
           +----------------------------------+
                            |
   +--------------------------------------------------------+
   |  Compile Model with RMSprop, MSE Loss, and MAE Metric  |
   +--------------------------------------------------------+
                            |
     +---------------------------------------------------+
     |  Train Model using Training and Validation Data   |
     +---------------------------------------------------+
                            |
   +--------------------------------------------------------+
   |  Use Trained Model to Predict Output for a Test Input  |
   +--------------------------------------------------------+
                            |
        +--------------------------------------------+
        |  Print Actual and Predicted Output Values  |
        +--------------------------------------------+
                            |
                     --------------
```

| End |

Program:

```
import tensorflow as tf
from tensorflow.keras.datasets import boston_housing
from sklearn import preprocessing
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *

(train_x, train_y),(test_x, test_y)=boston_housing.load_data()
train_x=preprocessing.normalize(train_x)
test_x=preprocessing.normalize(test_x)

model = Sequential([
  Dense(128, activation='relu', input_shape=(train_x[0].shape)),
  Dense(64, activation='relu'),
  Dense(32, activation='relu'),
  Dense(1)
])
model.compile(
  optimizer='rmsprop',
  loss='mse',
  metrics=['mae']
)
history = model.fit(
  x=train_x, y=train_y,
  epochs=100, batch_size=1,
  verbose=1,
  validation_data=(test_x, test_y)
)
test_input=[[
  8.65407330e-05,  0.00000000e+00,   1.13392175e-02,
  0.00000000e-00,  1.12518247e-03,   1.31897603e-02,
  7.53763011e-02,  1.30768051e-02,   1.09241016e-02,
  4.89399752e-01,  4.41333705e-02,   8.67155186e-01,
  1.75004108e-02
]]
print(
  "Actual Output : 21.1",
  "\nPredicted output :", model.predict(test_input)
)
```

Output:

```
1/1 [==============================] - 0s 230ms/step
Actual Output : 21.1
Predicted output : [[22.642511]]
```

Conclusion:

Hence, we can use linear regression with deep neural networks to accurately predict continuous variables, and we should consider the complexity of recursive and non-recursive functions when designing programs for optimal performance.

Experiment 8.

Classification using deep neural network.

Aim:

Multiclass classification, Binary classification

Theory:

- **Deep Nural Network:**

  DNN are a popular technique for classification tasks. A DNN is a type of artificial neural network that consists of multiple layers of interconnected nodes or "neurons." Each neuron takes in input from the previous layer, performs a calculation, and then passes output to the next layer.

  - **Multiclass classification:**

    Multiclass classification using deep neural networks involves training a neural network to classify input data into one of multiple possible classes. This type of problem is commonly encountered in applications such as image recognition, speech recognition, and natural language processing.

  - **Binary classification:**

    Binary classification using deep neural networks involves training a neural network to classify input data into one of two possible classes. This type of problem is commonly encountered in applications such as fraud detection, spam filtering, and medical diagnosis.

Algorithm:

1. Import necessary libraries
2. Load the dataset.
3. Preprocessing.
4. Split the preprocessed data into training and testing sets using the **train_test_split** function.
5. Define the neural network architecture using Keras **Sequential** model and add **Dense** layers.
6. Compile the model by specifying loss function, optimizer, and evaluation metric.
7. Train the model on the training data using **fit** function and store the training history.
8. Evaluate the model on the testing data using evaluate function and print the test accuracy.
9. Visualize the training history by plotting training & validation accuracy/loss using **matplotlib**.

10. Make predictions on the testing data and compute the prediction accuracy and confusion matrix.
11. Visualize the confusion matrix using **matplotlib**.

FlowChart:

```
                    -----------
                    |  START  |
                    -----------
                         |
                +------------------+
                | IMPORT LIBRARIES |
                +------------------+
                         |
                  +--------------+
                  | LOAD DATASET |
                  +--------------+
                         |
                 +---------------+
                 | PREPROCESSING |
                 +---------------+
                         |
            +---------------------------+
            | SPLIT INTO TRAIN AND TEST |
            +---------------------------+
                         |
       +------------------------------------+
       | DEFINE NEURAL NETWORK ARCHITECTURE |
       +------------------------------------+
                         |
            +--------------------------+
            | COMPILE, TRAIN, EVALUATE |
            +--------------------------+
                         |
              +-----------------------+
              | PLOT TRAINING HISTORY |
              +-----------------------+
                         |
                +------------------+
                | MAKE PREDICTIONS |
                +------------------+
                         |
    +----------------------------------------+
    | COMPUTE ACCURACY  AND CONFUSION MATRIX |
    +----------------------------------------+
                         |
           +----------------------------+
           | VISUALIZE CONFUSION MATRIX |
           +----------------------------+
                         |
                      -------
                      | END |
                      -------
```

Program:

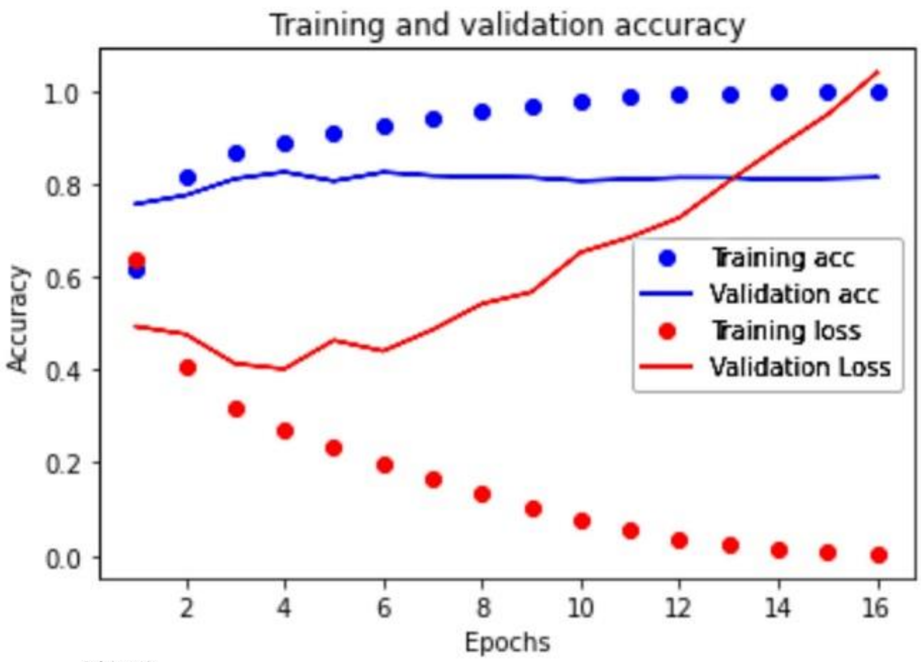
1.

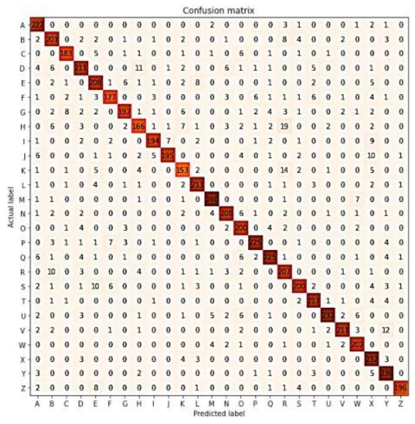
```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
url = "https://archive.ics.uci.edu/ml/machine-learning-databases" +
      "/letter-recognition/letter-recognition.data"
df = pd.read_csv(url, header=None)
X = df.iloc[:, 1:], X = X / 15.0, y = df.iloc[:, 0], y = pd.get_dummies(y)
X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.3, random_state=42
)
model = Sequential([
  Dense(64, input_dim=16, activation='relu'),
  Dense(32, activation='relu'), Dense(26, activation='softmax')
)
model.compile(
  loss='categorical_crossentropy',  optimizer='adam', metrics=['accuracy']
)
history = model.fit(
  X_train, y_train, validation_data=(X_test, y_test), epochs=100, batch_size=32
)
test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test accuracy:", test_acc)
acc = history.history['accuracy'], val_acc = history.history['val_accuracy']
loss = history.history['loss'], val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1), plt.figure(figsize=(10, 5))
# Plot training and validation Accuracy & Loss
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.plot(epochs, loss, 'bo', label='Training loss',color='red')
plt.plot(epochs, val_loss, 'b', label='Validation loss',color='red')
plt.title('Training and validation'), plt.xlabel('Epochs')
plt.ylabel('Accuracy/Loss'), plt.legend(), plt.show()
# Make predictions on test data
y_pred = model.predict(X_test), y_pred = np.argmax(y_pred, axis=1)
y_true = np.argmax(np.array(y_test), axis=1)
# Calculate prediction accuracy
test_acc = np.mean(y_pred == y_true), print("Test accuracy:", test_acc)
# Create a confusion matrix
conf_mat = np.zeros((26, 26), dtype=np.int)
for i in range(len(y_true)):
  conf_mat[y_true[i], y_pred[i]] += 1
# Visualize prediction accuracy
plt.figure(figsize=(10, 10)), plt.imshow(conf_mat, cmap='Oranges')
plt.xticks(range(26), [chr(ord('A')+i) for i in range(26)])
plt.yticks(range(26), [chr(ord('A')+i) for i in range(26)])
plt.xlabel('Predicted label'), plt.ylabel('Actual label')
plt.title('Confusion matrix')
for i in range(26):
    for j in range(26):
        plt.text(
          j, i, conf_mat[i, j], ha='center', va='center',
          color='white' if conf_mat[i, j] > len(y_true)*0.05 else 'black'
        )
plt.show()
```

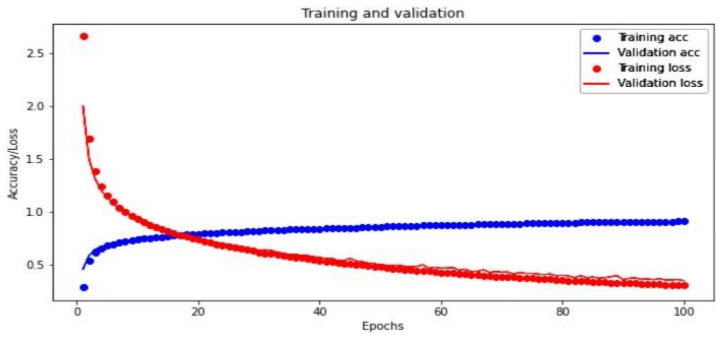
2.

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
# Load the dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
# Pad the sequences to the same length
X_train = pad_sequences(X_train, maxlen=100)
X_test = pad_sequences(X_test, maxlen=100)
bmodel = Sequential([
    Embedding(input_dim=10000, output_dim=32, input_length=maxlen),
    Conv1D(32, 7, activation='relu'),    MaxPooling1D(5),
    Conv1D(32, 7, activation='relu'),    GlobalMaxPooling1D(),
    Dense(1, activation='sigmoid')
])
bmodel.compile(
  optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc']
)
history = bmodel.fit(
  X_train, y_train,  epochs=16,  batch_size=128,  validation_split=0.2
)
score = bmodel.evaluate(X_test, y_test)
print("Test loss:", score[0]), print("Test accuracy:", score[1])
acc = history.history['acc'], val_acc = history.history['val_acc']
loss = history.history['loss'], val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
# Plot the training and validation accuracy & loss
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.plot(epochs, loss, 'bo', label='Training loss', color='red')
plt.plot(epochs, val_loss, 'b', label='Validation Loss', color='red')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs'), plt.ylabel('Accuracy'), plt.legend(), plt.show()
```

Output:







Conclusion:

Hence, we learn Classification using Deep neural network & Multiclass classification, Binary classification.

Experiment 9.

Convolutional neural network (CNN)

Aim:

1. Use any dataset of plant disease and design a plant disease detection system using CNN.
2. Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

Theory:

- **Convolutional Neural Networks (CNN):**

  Convolutional Neural Networks (CNN) are a type of neural network that are commonly used for image classification tasks. CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers are responsible for learning features from the input images, while the pooling layers down sample the feature maps to reduce the dimensionality of the output. The fully connected layers then use the learned features to make the final classification.

Algorithm:

1. Load the plant disease dataset or MNIST Fashion Dataset.
2. Preprocess the images by resizing and normalizing them.
3. Split the dataset into training and testing sets.
4. Define the CNN architecture with convolutional, pooling, and fully connected layers.
5. Compile the model with an appropriate loss function and optimizer.
6. Train the model on the training set and evaluate its performance on the testing set.
7. Save the trained model.

Database:

Plant disease : **https://data.mendeley.com/public-files/datasets/id/files/id**

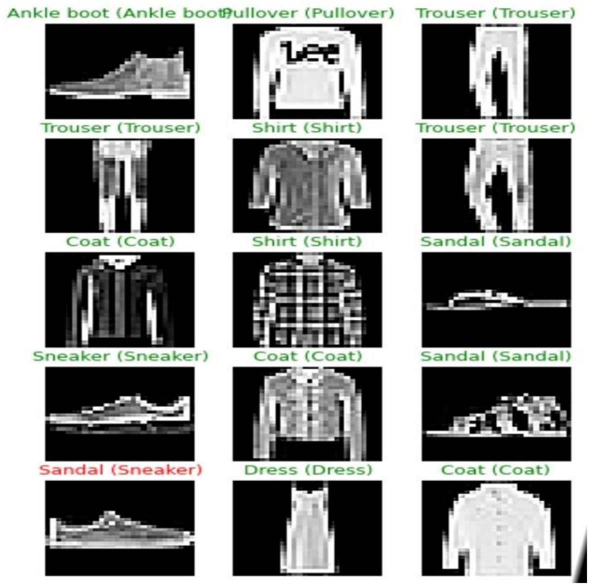
Fashin MNIST : internal in **tensorflow.keras.datasets**

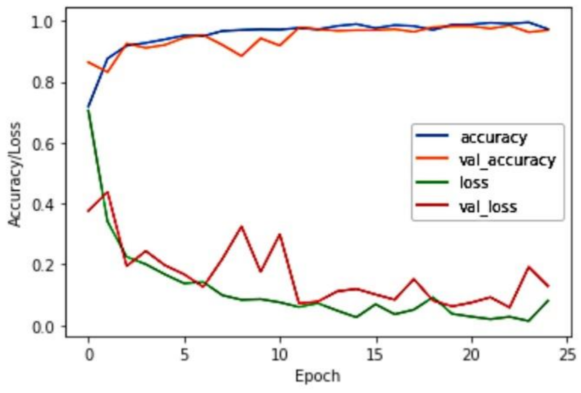
Program:

```
# Importing required libraries
import os, numpy as np, matplotlib.pyplot as plt, cv2
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
# Path to the dataset folder
data_dir = '/content/Plant_leave_diseases_dataset_with_augmentation'
# Creating a list of all the images and labels
images = [], labels = [], disease_types = os.listdir(data_dir)
for disease_type in disease_types:
    label = disease_types.index(disease_type)
    disease_folder_path = os.path.join(data_dir, disease_type)
    for img_path in os.listdir(disease_folder_path):
        img = cv2.imread(os.path.join(disease_folder_path, img_path))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (224, 224))
        images.append(img),        labels.append(label)
# Converting the lists into numpy arrays
images = np.array(images), labels = np.array(labels)
# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, stratify=labels,
random_state=42)
# Normalizing the pixel values of the images
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
# Converting the labels into one-hot encoded vectors
y_train = to_categorical(y_train, num_classes=len(disease_types))
y_test = to_categorical(y_test, num_classes=len(disease_types))
# Defining the CNN model
model = Sequential([
  Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
  MaxPooling2D((2, 2)),
  Conv2D(64, (3, 3), activation='relu'),  MaxPooling2D((2, 2)),
  Conv2D(128, (3, 3), activation='relu'), MaxPooling2D((2, 2)),
  Conv2D(128, (3, 3), activation='relu'), MaxPooling2D((2, 2)),
  Flatten(),  Dense(512, activation='relu'),  Dropout(0.5),
  Dense(len(disease_types), activation='softmax')
])
# Compiling the model
model.compile(
  optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']
)
# Training the model
history = model.fit(
  x_train, y_train, epochs=25, batch_size=32, validation_data=(x_test, y_test)
)
# Evaluating the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
# Plotting the accuracy and loss curves
plt.plot(history.history['accuracy'], label='accuracy'), plt.xlabel('Epoch')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.plot(history.history['loss'], label='loss'), plt.ylabel('Accuracy/Loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend(), plt.show()
```

```
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.layers import *
(tr_images, tr_labels), (tt_images, tt_labels) = fashion_mnist.load_data()
tr_images = tr_images / 255.0, tt_images = tt_images / 255.0
model = tf.keras.Sequential([
    Conv2D(32, (3,3), padding='same', activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)), Conv2D(64, (3,3), padding='same', activation='relu'),
    MaxPooling2D((2,2)), Conv2D(64, (3,3), padding='same', activation='relu'),
    Flatten(),    Dense(128, activation='relu'),    Dense(10)
])
model.compile(
  optimizer='adam',
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy']
)
model.fit(train_images[..., tf.newaxis], train_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images[..., tf.newaxis], test_labels)
print('Test accuracy:', test_acc)
class_names = [
  'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
  'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'
]
predictions = model.predict(test_images[..., tf.newaxis])
num_rows, num_cols = 5, 3, num_images = num_rows * num_cols
plt.figure(figsize=(2*num_cols, 2*num_rows))
for i in range(num_images):
    plt.subplot(num_rows, num_cols, i+1), plt.imshow(test_images[i], cmap='gray')
    predicted_label = np.argmax(predictions[i]), true_label = test_labels[i]
    color = 'green' if predicted_label == true_label else 'red'
    plt.title('{} ({})'
     .format(class_names[predicted_label], class_names[true_label]), color=color)
    plt.axis('off')
plt.show()
```

Output:





Conclusion:

Hence, we can conclude that CNNs are effective in designing a plant disease detection system and a fashion clothing classifier, achieving high accuracy on the testing set.

Experiment 10.

Recurrent neural network (RNN)

Aim:

To design and implement a time series analysis and prediction system using RNN on the Google stock prices dataset.

Theory:

- **Recurrent neural networks (RNNs):**

Recurrent Neural Networks (RNNs) are a type of neural network that can handle sequential data, making them well-suited for time series analysis and prediction. They have a unique architecture that allows them to maintain a memory of previous inputs. The most common type of RNN cell is the Long Short-Term Memory (LSTM) cell, which addresses the vanishing gradient problem. To use RNNs for time series analysis and prediction, the input data is typically split into fixed-length windows, and the network is trained to predict the next value in the sequence given the previous values. Many factors must be considered when designing and training an RNN for time series analysis and prediction.

Algorithm:

1. **Load the Google stock prices dataset**
2. **Pre-process the data by normalizing it**
3. **Split the data into training and testing sets**
4. **Define the RNN architecture**
5. **Train the RNN using the training data**
6. **Evaluate the performance of the RNN using the testing data**
7. **Use the trained RNN to predict future stock prices**

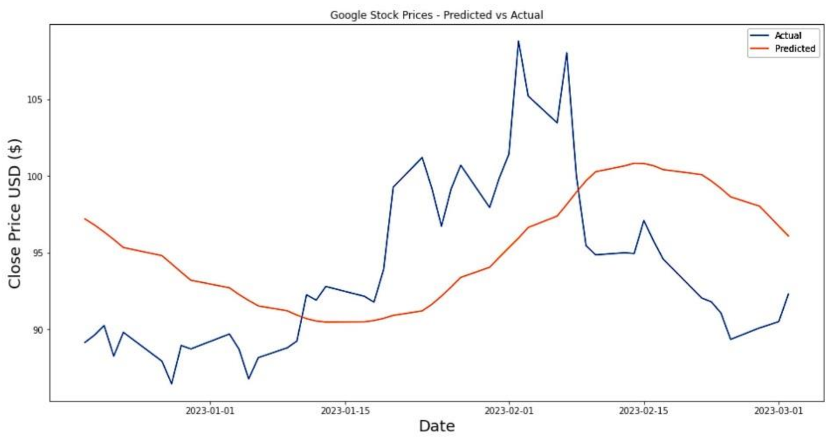
Database:

- Google stock prices dataset: **GOOG.csv**

Program:

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
df = pd.read_csv('GOOG.csv', index_col='Date', parse_dates=['Date'])
df = df[['Close']], df.head()
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(df['Close'].values.reshape(-1,1))
training_data_len = int(np.ceil(len(scaled_data) * 0.8))
train_data = scaled_data[0:training_data_len, :], x_train = [], y_train = []
for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0]), y_train.append(train_data[i, 0])
x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
model = Sequential([  LSTM(50, return_sequences=True, input_shape=(x_train.shape[1], 1)),
  Dropout(0.2), LSTM(50, return_sequences=True),
  Dropout(0.2), LSTM(50), Dropout(0.2), Dense(1)])
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs=25, batch_size=32)
test_data = scaled_data[training_data_len-60:, :]
x_test = [], y_test = df.iloc[training_data_len:, :]['Close'].values
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])
x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
plt.figure(figsize=(16,8))
plt.title('Google Stock Prices - Predicted vs Actual')
plt.plot(df.iloc[training_data_len:, :].index, y_test, label='Actual')
plt.plot(df.iloc[training_data_len:, :].index, predictions, label='Predicted')
plt.xlabel('Date', fontsize=18), plt.ylabel('Close Price USD ($)', fontsize=18)
plt.legend(), plt.show()
```

Output:



Conclusion:

In this experiment, we designed and implemented an RNN-based time series analysis and prediction system on the Google stock prices dataset. We observed that the RNN was able to capture the sequential nature of the data and provide accurate predictions. Hence, RNNs can be effectively used for time series prediction tasks.

Experiment 11.

Mini Project

Aim:

Colorizing Old B&W Images: color old black and white images to colorful images

Theory:

Colorizing black and white images is a process of adding color to the grayscale image. It involves understanding the relationship between the luminance (brightness) and chrominance (color) of an image. In computer vision, it is achieved by training deep learning models on large datasets of color images and their corresponding grayscale versions.

Algorithm:

1. Load the pre-trained colorization model and its supporting files (prototxt and npy files).
2. Read the input black and white image.
3. Convert the input image to LAB color space.
4. Resize the image to the input size of the model.
5. Extract the L channel from the LAB image and subtract 50 from its pixel values.
6. Set the L channel as the input to the model.
7. Run the model to get the predicted color values for each pixel in the ab channels.
8. Resize the predicted ab channels to the original size of the input image.
9. Merge the L channel and the predicted ab channels to form the final color image.
10. Convert the LAB image back to RGB color space.
11. Clip the pixel values between 0 and 1.
12. Scale the pixel values to the range of 0 to 255.
13. Display the input and output images using matplotlib.

Pre Trained models:

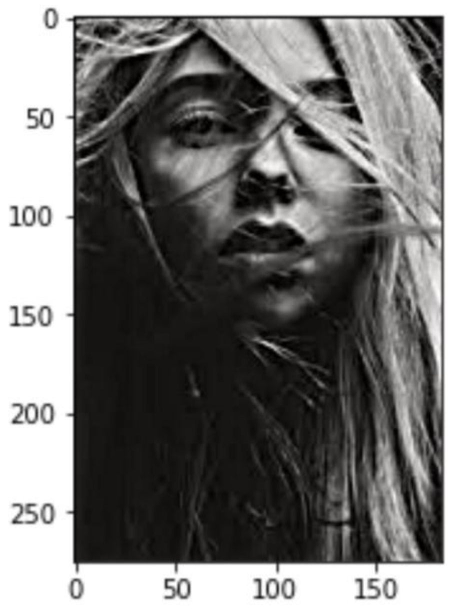
- **colorization_deploy_v2.prototxt**
- **pts_in_hull.npy**
- **colorization_release_v2.caffemodel**

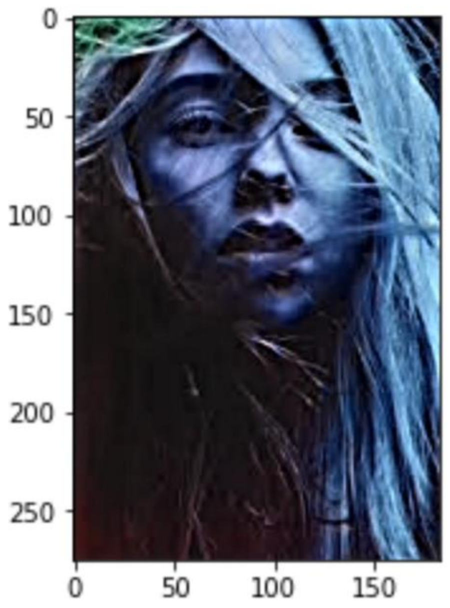
Program:

```
from google.colab import files
```

```
# Upload the JPG file
uploaded = files.upload()

# Save the file in /content folder with the filename "image.jpg"
for filename in uploaded.keys():
   with open('/content/image.jpg', 'wb') as f:
      f.write(uploaded[filename])
import numpy as np
import cv2

print("loading models.....")
net =
cv2.dnn.readNetFromCaffe('./colorization_deploy_v2.prototxt','./colorization_release_v2.caffemo
del')
pts = np.load('./pts_in_hull.npy')

class8 = net.getLayerId("class8_ab")
conv8 = net.getLayerId("conv8_313_rh")
pts = pts.transpose().reshape(2,313,1,1)

net.getLayer(class8).blobs = [pts.astype("float32")]
net.getLayer(conv8).blobs = [np.full([1,313],2.606,dtype='float32')]

image = cv2.imread("/content/image.jpg")

scaled = image.astype("float32")/255.0
lab = cv2.cvtColor(scaled,cv2.COLOR_BGR2LAB)

resized = cv2.resize(lab,(224,224))
L = cv2.split(resized)[0]
L -= 50

net.setInput(cv2.dnn.blobFromImage(L))
ab = net.forward()[0, :, :, :].transpose((1,2,0))

ab = cv2.resize(ab, (image.shape[1],image.shape[0]))

L = cv2.split(lab)[0]
colorized = np.concatenate((L[:,:,np.newaxis], ab), axis=2)

colorized = cv2.cvtColor(colorized,cv2.COLOR_LAB2BGR)
colorized = np.clip(colorized,0,1)

colorized = (255 * colorized).astype("uint8")
import matplotlib.pyplot as plt

# Display the image using imshow
plt.imshow(image)
plt.show()

plt.imshow(colorized)
plt.show()
cv2.waitKey(0)
```

Output:





Conclusion:

Hence, using the pre-trained colorization model and deep learning techniques, we can successfully convert black and white images to color images. This algorithm follows a step-by-step approach, which involves converting the input image to LAB color space, extracting the L channel, setting it as input to the model, predicting the ab channels, merging them with the L channel, and finally converting the image back to RGB color space. By following this process, we can generate color images that closely resemble the original ones.