PROJECT 1. Convolutional Neural Networks for Object Recognition: Forward Propagation
Topic: Multi-dimensional arrays
ESE 344 Software Techniques for Engineers
SUNY at Stony Brook, Dept. of ECE, Murali Subbarao

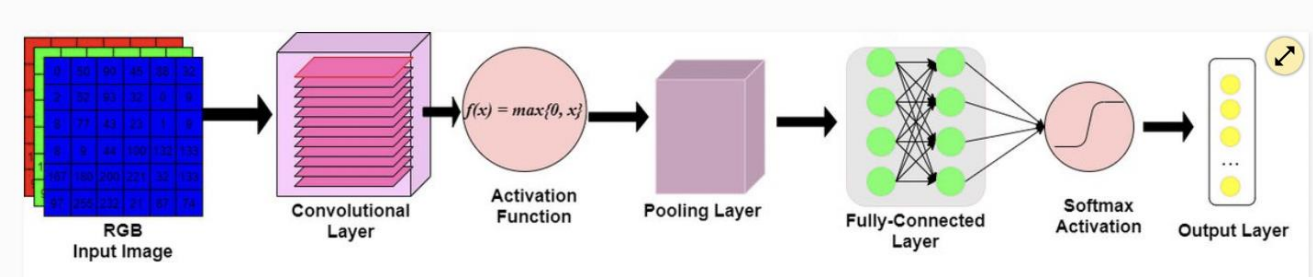
DRAFT 1.0 Subject to changes

A Convolutional Neural Net (CNN) that can identify 10 class of objects in 32x32x3 color images is found at the website ConvNetJS CIFAR-10 demo (stanford.edu). It was implemented by Andrej Karpathy, and we will refer to it as CNN-AK. You do not need to understand how/why a CNN works to complete this project, but some knowledge of what this website is demonstrating would make this project more interesting to you. In particular, this project can read the filter coefficients of this CNN-AK and classify the images into one of 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Estimated accuracy is 90% while it is 94% for humans. See CIFAR-10 and CIFAR-100 datasets (toronto.edu) for more details.

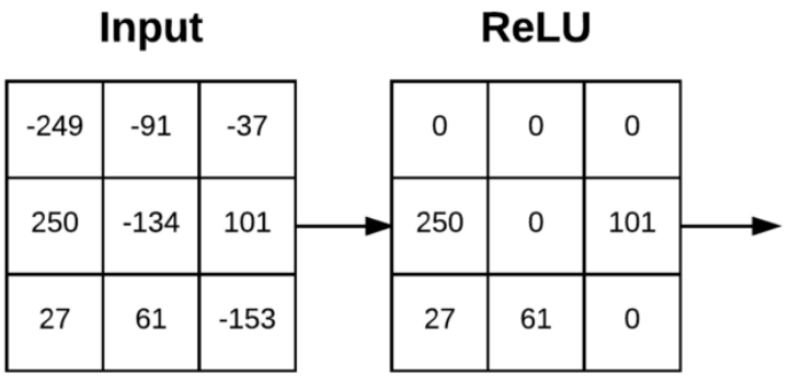
In this project, you will not be implementing the training part of the CNN, but only the classification part, i.e. only the forward propagation (no backpropagation and training). Also, randomly generated synthetic data will be used for filter weights/coefficients instead of data corresponding to the result of training the CNN (if you find the trained weights, then you may use them in the project). Therefore your program cannot classify images (unless you read in real trained weight data instead of synthetic data). Your project will include all the default layers of CNN-AK specified by

```
layer_defs = [];
layer_defs.push({type:'input', out_sx:32, out_sy:32, out_depth:3});
layer_defs.push({type:'conv', sx:5, filters:16, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'conv', sx:5, filters:20, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'conv', sx:5, filters:20, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'softmax', num_classes:10});
```

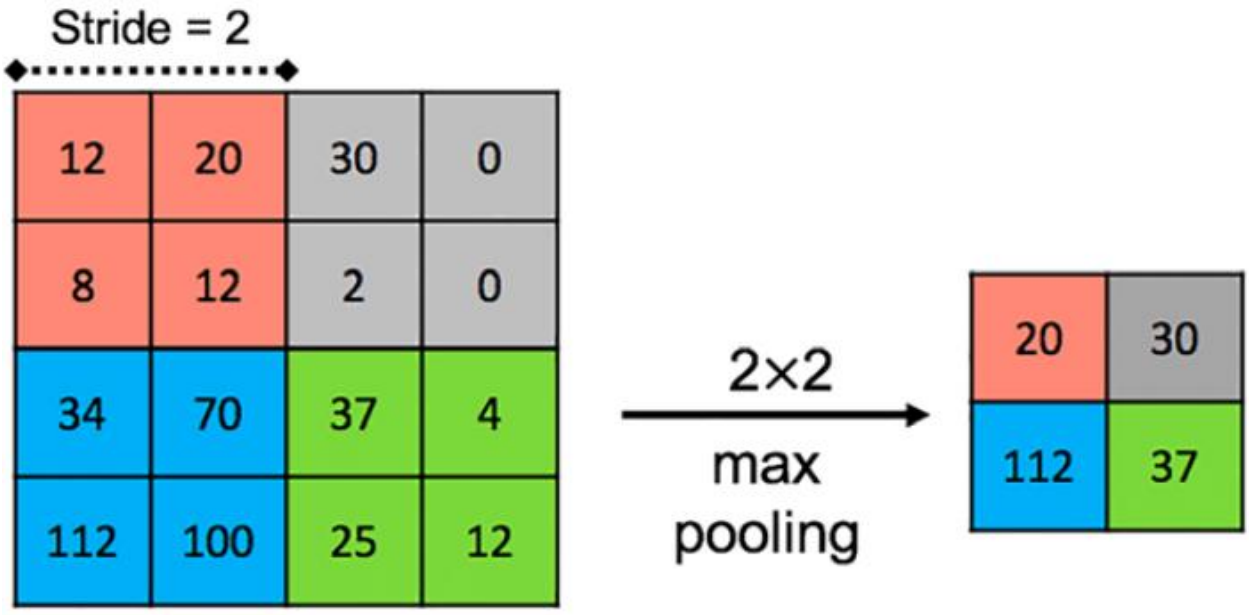
An Abbreviated diagram of CNN with only one Convolution layer



Rectified Linear Unit (ReLU) example: output z, input x: z=max(0,x).



Max pooling example: Filter size 2x2, stride 2



An example of Zero padding for a 3x3 filter (but this project uses 5x5 filters; so the width of padding will be 2 in the project instead of 1 shown here. In the example code given at the end, zero padding is achieved by checking for boundary conditions instead of adding bands of elements with zeros.)

15	20	90	120	160	200	10	90
77	63	58	98	72	112	245	255
68	93	95	94	78	151	54	84
51	84	53	84	65	81	20	30
48	86	87	15	35	65	45	59
65	98	87	160	140	198	200	145
21	45	87	55	32	21	65	87
12	54	8	21	45	87	2	36

Input Matrix (8x8x[0])

0	0	0	0	0	0	0	0	0	0
0	15	20	90	120	160	200	10	90	0
0	77	63	58	98	72	112	245	255	0
0	68	93	95	94	78	151	54	84	0
0	51	84	53	84	65	81	20	30	0
0	48	86	87	15	35	65	45	59	0
0	65	98	87	160	140	198	200	145	0
0	21	45	87	55	32	21	65	87	0
0	12	54	8	21	45	87	2	36	0
0	0	0	0	0	0	0	0	0	0

Input Matrix (10x10x[0]) after zero padding

Each layer and its operation will be described below. There are 5 main stages where the first stage is the input part, followed by 3 stages that are similar and each includes a Convolution layer, a Rectified Linear Unit (ReLU) layer, and a Max pooling layer. The last stage includes a fully connected layer and a softmax layer.

You can define a ConvNet class with the following class members: (1) one data member for storing the output of each layer where the first layer stores the result of reading the input image data (32x32x3 array of floating point numbers) and the last layer stores the output of the ConvNet (which is the probability of the 10 classes), (2) one method/function member to compute the output of each of the layers from 2 to the last layer. Read reference material to understand the operation of Convolution, ReLu, Max Pool, Fully Connected, and the Softmax layers. In your implementation, you can fix the size of all arrays and some parameters like stride (instead of making them variables read from an input file; making them variable is optional).

When a ConvNet object is created in the main() function, by default, all memory is allocated but the data members are not initialized. The methods for reading inputs initialize all the data members, and then the methods compute the output of each layer in sequence. This is one way to do it.

You are provided with an example of implementing multi-dimensional arrays and operations on them including the convolution operation. You are expected to understand the example and use parts of the code there with some changes in your source code. ReLu and Max pooling are easy and straightforward to implement.

Reference: ConvNetJS CIFAR-10 demo

Here is an outline for ConvNet architecture. See references for more details.

Input: floating point arrays as follows:

D1: 32x32x3, D3: 5x5x3x16 , D4: 1x16, D8: 5x5x16x20, D9: 1x20, D13: 5x5x20x20, D14: 1x20, D18: 4x4x20x10, D19: 1x10

The program to generate the input data is given.
You can modify this program to read the generated data from a file in the same order.

Output: floating point array: D20: 1x10

Stage 1 Input

1) Layer L1 : input layer

a) Data member D1: float 3D matrix array of size 32x32x3 to store input image.
b) Method member M1: reads input file and initializes the input layer in 1a.

Stage 2: First Conv+ReLU+Max pool

2) **Layer L2 : Convolution, Stride 1**

a. Data member D2: 32x32x16 array of floating point numbers to store the output of L2
b. Data member D3: 5x5x3x16 array for storing the 16 convolution filters with zero padding.
c. Data member D4: 16x1 array for storing the bias vector
d. Method member M2: for reading input file to initialize the data member D3 in L2.
e. Method member M3: for reading input file to initialize the data member D4 in L2.
f. Method M4 : for computing the data member D2 by convolving D1 with D3 and adding D4, and storing the output in D2. Set Stride=1.

3) Layer L3: ReLu activation function
    a. Data member D5: 32x32x16 array for storing the output of this layer L3.
    b. Method member M5: Computes D5 using D2. Z= max(0,x).
4) Layer L4: Maxpooling: filter size 2x2, stride=2.
    a. Data member D6: 16x16x16 array for storing the output of this layer L4.
    b. Method member M6: Process D5 to compute and store D6. Each element is the maximum of 4 elements in a 2x2 block, with stride 2.

The way this CNN works is such that it considers max-pooling of 2x2 regions in 2D cross-sections of 3D arrays, for each cross-section separately. So, you don't need 3D max pooling in this project. Only 2D max pooling is used. For example, the first max pooling in layer L4 does the following:

Input: D5: 32x32x16 say D5[m][n][k]
Output: D6: 16x16x16, say D6[i][j][k]
Stride: 2
The code could be:

```
for(k=0;k<16;k++) {  // for each cross section k
   for(m=0, i=0;m<32;m += stride ,i++) { //for row m
      for(n=0, j=0;n<32;n +=stride, j++) { // for column n
          D6[i][j][k] = maximum(D5[m][n][k], D5[m+1][n][k],D5[m][n+1][k],D5[m+1][n+1][k]);
      }
   }
}
```

Stage 3: Second Conv+ReLU+Max pool

5) **Layer L5 : Convolution, stride 1**
    a. Data member D7: 16x16x20 array of floating point numbers to store the output of L5
    b. Data member D8: 5x5x16x20 array for storing the 20 convolution filters with zero padding.
    c. Data member D9: 20x1 array for storing the bias vector
    d. Method member M7: for reading input file to initialize the data member D8 in L5.
    e. Method member M8: for reading input file to initialize the data member D9 in L5.
    f. Method M9: for computing the data member D7 by convolving D6 with D8, stride=1, and adding D9, and storing the output in D7.
6) Layer L6: ReLu activation function
    a. Data member D10: 16x16x20 array for storing the output of this layer L6.
    b. Method member M10: Computes and stores D10 using D7. Z= max(0,x).

7) Layer L7: Maxpooling: filter size 2x2, stride=2.
    a. Data member D11: 8x8x20 array for storing the output of this layer L7.
    b. Method member M11: Process D10 to compute and store D11. Each element is the maximum of 2x2 block, with stride 2.

Stage 4: Third Conv+ReLU+Max pool

8) **Layer L8 : Convolution, stride 1**
    a. Data member D12: 8x8x20 array of floating point numbers to store the output of L8
    b. Data member D13: 5x5x20x20 array for storing the 20 convolution filters with zero padding.
    c. Data member D14: 20x1 array for storing the bias vector
    d. Method member M12: for reading input file to initialize the data member D13 in L8.
    e. Method member M13: for reading input file to initialize the data member D14 in L8.
    f. Method M14: for computing the data member D12 by convolving D11 with D13, stride=1, and adding D14, and storing the output in D12.
9) Layer L9: ReLu activation function
    a. Data member D15: 8x8x20 array for storing the output of this layer L9.
    b. Method member M15: Computes and stores D15 using D12. Z= max(0,x).
10) Layer L10: Maxpooling: filter size 2x2, stride=2.
    a. Data member D16: 4x4x20 array for storing the output of this layer L10.
    b. Method member M16: Process D15 to compute and store D16. Each element is the maximum of 4 elements in a 2x2 block, with stride 2.

Stage 5: Last layer: Fully connected + Softmax

11) Layer L11 : Fully Connected Layer
    a. Data member D17: array of size 10 to store the output of this layer L11
    b. Data member D18: 4x4x20x10 array for storing 10 full connection filters (dot product).
    c. Data member D19: array of size 10 for storing a bias vector.
    d. Method member M17: for reading input file to initialize the data member D18 in L11.
    e. Method member M18: for reading input file to initialize the data member D19 in L11.
    f. Method M19: for computing the data member D17 by taking dot-product of D16 with the 10 different 4x4x20 filters stored in D18, and adding D19, and storing the output in D17.
12) **Layer L12: Softmax layer**
    a. Data member D20: array of size 10 to store the output of this layer L12
    b. Method member M20: In order to avoid taking the exponent of large numbers that cause overflow, normalize the contents of D17, by dividing each element by the square-root of the sum of the squares of each element in D17. Use this result in computing probabilities in the next method M21.
    c. Method M21: for each element x of normalized D17, compute the corresponding softmax function of x that gives it's probability: (exp(x)/(sum of exp(xi) for all i).
    d. Method M22: Method that prints the output of this CNN stored in D20.

Write a main function that creates a ConvNet object defined above, and calls methods to initialize the input, filters, and bias vectors, and then calls the methods to compute the output of layers in sequence, and prints the output to a file. At first, you may use synthetic data generated randomly. In order to avoid numerical overflow due to potentially large numbers, make the input numbers to be in the range 0.0 to 1.0.

Instructions

An example of a program that demonstrates the creation of multi-dimensional arrays and various operations on them is provided below. Compile and run this program, understand various parts in this program, and adapt parts of the code here to implement the CNN defined above.

```
#ifndef MATRIX_H
#define MATRIX_H
#include <iostream>
#include <vector>
#include <string>
#include <complex>
#include <math.h>
#include <cassert>

#include <cstdlib>   // for rand(), srand()
#include <ctime>     // for time()

// Udated 2/3/2022 Added Fully connected layer computation and softmax

// Modified by Prof. Murali Subbarao, ESE 344

using namespace std;

template <typename Object>
class matrix
{
public:
      matrix(int rows, int cols) : array(rows)
      {
            for (auto & thisRow : array)
                  thisRow.resize(cols);
      }

      matrix(vector<vector<Object>> v) : array{ v }
      { }
      matrix(vector<vector<Object>> && v) : array{ std::move(v) }
      { }

      const vector<Object> & operator[](int row) const
      {
            return array[row];
      }
      vector<Object> & operator[](int row)
```

```
      {
            return array[row];
      }

      int numrows() const
      {
            return array.size();
      }
      int numcols() const
      {
            return numrows() ? array[0].size() : 0;
      }

      void matprt() const // print matrix
      {
            for (int i = 0; i < numrows(); ++i) {
                  for (int j = 0; j < numcols(); ++j) {
                        cout << array[i][j] << "   ";
                  }
                  cout << endl;
            }
            cout << endl <<endl;
      }

      /**
* Standard matrix multiplication. Version 1
* Arrays start at 0.
*/

      matrix<double> operator*(const matrix<double>& b) const
      {
            int nra = numrows();
            int nca = numcols();
            int nrb = b.numrows();
            int ncb = b.numcols();

            if (nca != nrb) {
                  cerr << "nca != nrb in matrix multiplication.
Exiting." << endl;
                  exit(0);
            }

            matrix<double> c(nra, ncb);

            for (int i = 0; i < nra; ++i)
                  for (int j = 0; j < ncb; ++j) {
                        c[i][j] = 0.0;
                        for (int k = 0; k < nca; ++k)
                              c[i][j] += array[i][k] * b[k][j];
                  }

            return c;
```

```
      }



private:
      vector<vector<Object>> array;
};
#endif


/**  Global function
* Standard matrix multiplication.
* Arrays start at 0.
* Return value by copying.
*/
/*
matrix<double> operator*(const matrix<double> & a, const
matrix<double> & b)
{
      int nra = a.numrows();
      int nca = a.numcols();
      int nrb = b.numrows();
      int ncb = b.numcols();

      if (nca != nrb) {
            cerr << "nca != nrb in matrix multiplication. Exiting." <<
endl;
            exit(0);
      }

      matrix<double> c( nra, ncb );

      for (int i = 0; i < nra; ++i)
            for (int j = 0; j < ncb; ++j) {
                  c[i][j] = 0.0;
                  for (int k = 0; k < nca; ++k)
                        c[i][j] += a[i][k] * b[k][j];
            }

      return c; // return by copying
}

*/

/**
* Standard matrix multiplication.
* Arrays start at 0.
* No copying. Pass by reference.
*/
void matmul(const matrix<double>& a, const matrix<double>& b,
matrix<double>& c)
```

```
{
      int nra = a.numrows();
      int nca = a.numcols();
      int nrb = b.numrows();
      int ncb = b.numcols();

      if (nca != nrb) {
            cerr << "nca != nrb in matrix multiplication. Exiting." <<
endl;
            exit(0);
      }

      for (int i = 0; i < nra; ++i)
            for (int j = 0; j < ncb; ++j) {
                  c[i][j] = 0.0;
                  for (int k = 0; k < nca; ++k)
                        c[i][j] += a[i][k] * b[k][j];
            }

      return;
}

// Standard matrix print.

void matprint(const matrix<double> & a)
{
      for (int i = 0; i < a.numrows(); ++i) {
            for (int j = 0; j < a.numcols(); ++j) {
                  cout << a[i][j] << "   ";
            }
            cout<< endl;
      }
      cout << endl<<endl;
}

int main()
{
      int nr1, nc1, nr2, nc2;
      char c;

      srand((unsigned int)time(NULL));// seed rand() with system time

      cout << "Enter number of rows nr1 : (1 to 20) : "  << endl;
      cin >> nr1;
      cout << "nr1 : " << nr1 << endl;

      if ( (nr1<1) || (nr1>20) ) {
            cout << "nr1 is out of range . " << endl;

            assert((1 <= nr1) && (nr1 <= 20));
            return 1;
```

```
//          exit(1);
      }
      cout << "Enter number of columns nc1 : (1 to 20) : " << endl;
      cin >> nc1;
      cout << "nc1 : " << nc1 << endl;

      if (!((1 <= nc1) && (nc1 <= 20))) {
            cout << "nc1 is out of range . " << endl;
            return 1;
      }

      cout << "Enter number of rows nr2 : (1 to 20) : " << endl;
      cin >> nr2;
      cout << "nr2 : " << nr2 << endl;
      if (!((1 <= nr2) && (nr2 <= 20))) {
            cout << "nr2 is out of range . " << endl;
            return 1;
      }
      cout << "Enter number of columns nc2 : (1 to 20) : " << endl;
      cin >> nc2;
      cout << "nc2 : " << nc2 << endl;
      if (!((1 <= nc2) && (nc2 <= 20))) {
            cout << "nc2 is out of range . " << endl;
            return 1;
      }

      matrix<double> mat1(nr1, nc1), mat2(nr2, nc2);

      for (int i = 0; i < nr1; ++i) { // Initialization
            for (int j = 0; j < nc1; ++j) {
      //          mat1[i][j] = (double)(i + j +5.5);
                mat1[i][j] = (double)(rand() % 100);
            }
      }
      for (int i = 0; i < nr2; ++i) { // Initialization
            for (int j = 0; j < nc2; ++j) {
//                mat2[i][j] = (double)(i*j +7.9);
                  mat2[i][j] = (double)(rand() % 100);
            }
      }

      //matrix<double> mat3(nr1, nc2);
      // mat3 = mat1 * mat2; //copy assignment

      //matrix<double> mat3(nr1, nc2);
      //mat3 = std::move(mat1 * mat2); // rvalue reference

      matrix<double>&& mat3{ mat1 * mat2 };
      //matrix<double> && mat3 = mat1*mat2; // rvalue reference

      //matrix<double> mat3(nr1, nc2);
      //matmul(mat1, mat2, mat3);
```

```
      cout <<endl<< "mat1 : " << endl;
      mat1.matprt();
      //matprint(mat1);
      mat2.matprt();
      //matprint(mat2);
      cout << endl << "mat3 : " << endl;
      //matprint(mat3);
      mat3.matprt();

      cout << endl << "Enter any char to continue : ";
      cin >> c;

      complex<double> p(1.0,2.0) , q(3.0 ,4.0),r;
      cout << "p = " << p << ",   q =  "<< q << endl;
      p = complex<double>(5.0, 1.0);
      cout << "p = " << p << ",   q =  " << q << endl;
      cout << p.real() << "  " << p.imag()<<endl;
      r = p + q;
      cout << "r = p+q = " << r<<endl;
      r = p * q;
      cout << "r = p*q = " << r << endl;
      r = p /q;
      cout << "r = p/q =  " << r << endl;

      cout << endl << "Enter any char to continue : ";
      cin >> c;

      matrix<complex <double>> mat4(nr1, nc1);
      for (int i = 0; i < nr1; ++i) { // Initialization
            for (int j = 0; j < nc1; ++j) {
//                mat4[i][j] =
complex<double>((double)(i+1.5),(double)(j+1.4));
                mat4[i][j] = complex<double>((double)(rand() %
100),(double)(rand() % 100));
            }
      }

      cout << endl << "mat4 : " << endl;
      mat4.matprt();

      cout << endl << "Enter any char to continue : ";
      cin >> c;

      cout << "exp(2.5)  : " << exp(2.5) <<endl;
      cout << "sqrt(5.0) : " << sqrt(5.0) <<endl;
      cout << "exp(r) : " << exp(r) <<endl;
      cout << "sin(2.5) : " << sin(2.5) << endl;
      cout << "cos(2.5) : " << cos(2.5) << endl;

      cout << endl << "Enter any char to continue : ";
```

```
	cin >> c;

	// complex matrices
	matrix<complex <double>> mat11(nr1, nc1), mat12(nr2, nc2),
mat13(nr1, nc2);

	cout << endl << "mat11 :" << endl;
	for (int i = 0; i < nr1; ++i) { // Initialization
		for (int j = 0; j < nc1; ++j) {
//			mat11[i][j] = complex<double>
//				( (double)(i + j + 1.5) , (double)(i-j + 3.5));
			mat11[i][j] = complex<double>
				((double)(rand()%100), (double)(rand()%100));
			cout << mat11[i][j] << "  ";
		}
		cout << endl;
	}

	cout << endl << "mat12 :" << endl;
	for (int i = 0; i < nr2; ++i) { // Initialization
		for (int j = 0; j < nc2; ++j) {
//			mat12[i][j] = complex<double>( (double)(i * j + 1) ,
(double)(i * j - 5.7) );
			mat12[i][j] = complex<double>((double)(rand()%100),
(double)(rand()%100));
			cout << mat12[i][j] << "  ";
		}
		cout << endl;
	}


	cout<<endl<< "mat13 :" << endl;

	for (int i = 0; i < nr1; ++i) {
		for (int j = 0; j < nc2; ++j) {
			mat13[i][j] = complex<double>(0.0, 0.0);
			for (int k = 0; k < nc1; ++k)
			{
				mat13[i][j] += mat11[i][k] * mat12[k][j];
			}
			cout << mat13[i][j] << "  ";
		}
		cout << endl;
	}

	cout << endl << "Enter any char to continue : ";
	cin >> c;

	vector<vector<vector <double>>> a1;
	vector<vector<vector <vector <double>>>> a2;
	int s1 = 5, s2 = 6, s3 = 7, s4 = 4;
```

```
	a1.resize(s1);
	a2.resize(s1);

	for (int i1 = 0; i1 < s1; i1++) {
		a1[i1].resize(s2);
		a2[i1].resize(s2);
	}

	for (int i1 = 0; i1 < s1; i1++)
		for (int i2 = 0; i2 < s2; i2++) {
			a1[i1][i2].resize(s3);
			a2[i1][i2].resize(s3);
		}

	for (int i1 = 0; i1 < s1; i1++) {
		for (int i2 = 0; i2 < s2; i2++) {
			for (int i3 = 0; i3 < s3; i3++)
			{
				a2[i1][i2][i3].resize(s4);

//				a1[i1][i2][i3] = 1 + i3 + i2 * s3 + i1 * s2 * s3;
				a1[i1][i2][i3] = (double)(rand()%100);
				cout << a1[i1][i2][i3] <<"   ";
			}
			cout << endl;
		}
		cout << endl << endl;
	}
	cout << endl << endl;

	cout << endl << "Enter any char to continue : ";
	cin >> c;
	for (int i1 = 0; i1 < s1; i1++) {
		for (int i2 = 0; i2 < s2; i2++) {
			for (int i3 = 0; i3 < s3; i3++) {
				for (int i4 = 0; i4 < s4; i4++) {
			//		a2[i1][i2][i3][i4] = 1 + i4 + i3 * s4 + i2
* s3 * s4 + i1 * s2 * s3 * s4;
					a2[i1][i2][i3][i4] = (double)(rand()%100);
					cout << a2[i1][i2][i3][i4] << "   ";
				}
				cout << endl;
			}
			cout << endl<< endl;
		}
		cout << endl << endl;
	}
	cout << endl << endl;


	cout << endl << "Enter any char to continue : " << endl;
	cin >> c;
```

```
	vector<vector<vector< double>>> tn1, tn3;
	vector<vector<vector< vector< double >>>> tn2, tn4;
	// "Convolve" tn1 with tn2 with stride 2 to obtain tn3
	int stride = 2;
	int tn1s1 = 32, tn1s2 = 32, tn1s3 = 16;
	int tn2s1 = 5, tn2s2 = 5, tn2s3 = tn1s3, tn2s4 = 20;
	int tn2s1by2 = tn2s1 / 2;  // 2
	int tn2s2by2 = tn2s2 / 2;  // 2
	int tn3s1 = tn1s1 / stride, tn3s2 = tn1s2 / stride, tn3s3 =
tn2s4;
	int tn4s1 = tn1s1, tn4s2 = tn1s2, tn4s3 = tn1s3, tn4s4 = 10;

	tn1.resize(tn1s1);
	tn4.resize(tn1s1);

	for (int i1 = 0; i1 < tn1s1; i1++) {
		tn1[i1].resize(tn1s2);
		tn4[i1].resize(tn1s2);
	}

	for (int i1 = 0; i1 < tn1s1; i1++) {
		for (int i2 = 0; i2 < tn1s2; i2++) {
			tn1[i1][i2].resize(tn1s3);
			tn4[i1][i2].resize(tn1s3);
		}
	}

	cout << "tn1 :";
	for (int i1 = 0; i1 < tn1s1; i1++) {
		for (int i2 = 0; i2 < tn1s2; i2++) {
			for (int i3 = 0; i3 < tn1s3; i3++)
			{
				tn1[i1][i2][i3] = ((double)(rand() % 100))/100.0;
				cout << tn1[i1][i2][i3] << "   ";
				tn4[i1][i2][i3].resize(tn4s4);
			}
			cout << endl;
		}
		cout << endl << endl;
	}
	cout << endl << endl;

	cout << "Fully connected filter tn4" << endl;
	for (int i1 = 0; i1 < tn1s1; i1++) {
		for (int i2 = 0; i2 < tn1s2; i2++) {
			for (int i3 = 0; i3 < tn1s3; i3++) {
				for (int i4 = 0; i4 < tn4s4; i4++) {
					tn4[i1][i2][i3][i4] = ((double)(rand() %
100)) / 100.0;
					cout << tn4[i1][i2][i3][i4] << "   ";
				}
```

```
                    cout << endl;
               }
               cout << endl << endl;
          }
          cout << endl << endl;
     }
     cout << endl << endl;


     tn2.resize(tn2s1);

     for (int i1 = 0; i1 < tn2s1; i1++) {
          tn2[i1].resize(tn2s2);
     }

     for (int i1 = 0; i1 < tn2s1; i1++) {
          for (int i2 = 0; i2 < tn2s2; i2++) {
               tn2[i1][i2].resize(tn2s3);
          }
     }

     for (int i1 = 0; i1 < tn2s1; i1++) {
          for (int i2 = 0; i2 < tn2s2; i2++) {
               for (int i3 = 0; i3 < tn2s3; i3++) {
                    tn2[i1][i2][i3].resize(tn2s4);
               }
          }
     }

     cout << "tn2 :" << endl;
     for (int i1 = 0; i1 < tn2s1; i1++) {
          for (int i2 = 0; i2 < tn2s2; i2++) {
               for (int i3 = 0; i3 < tn2s3; i3++) {
                    for (int i4 = 0; i4 < tn2s4; i4++) {
                         tn2[i1][i2][i3][i4] = ((double)(rand() %
100)) / 100.0;
                         cout << tn2[i1][i2][i3][i4] << "   ";
                    }
                    cout << endl;
               }
               cout << endl <<endl;
          }
          cout << endl << endl;
     }
     cout << endl << endl;

     tn3.resize(tn3s1);

     for (int i1 = 0; i1 < tn3s1; i1++) {
          tn3[i1].resize(tn3s2);
     }
```

```
	for (int i1 = 0; i1 < tn3s1; i1++) {
		for (int i2 = 0; i2 < tn3s2; i2++) {
			tn3[i1][i2].resize(tn3s3);
		}
	}

	cout << "tn3: " << endl;
	for (int i1 = 0; i1 < tn3s1; i1++) {
		for (int i2 = 0; i2 < tn3s2; i2++) {
			for (int i3 = 0; i3 < tn3s3; i3++)
			{
				tn3[i1][i2][i3] = ((double)(rand() % 100))/100.0;
				cout << tn3[i1][i2][i3] << "   ";
			}
			cout << endl;
		}
		cout << endl << endl;
	}
	cout << endl << endl;

		cout << endl << "Enter any char to continue : ";
	cin >> c;
	vector<double> bias; // bias vector
	bias.resize(tn3s3); // note: tn3s3=tn2s4
	cout << "Bias vector:  " << endl;
	for (int i3 = 0; i3 < tn3s3; i3++)
	{
		bias[i3] = ((double)(rand() % 100)) / 100.0;
		cout << bias[i3] << "   ";
	}
	cout << endl;

	cout << endl << "Enter any char to continue : ";
	cin >> c;
	cout << "Output of Convolution layer: tn3" << endl;
	for (int tn2i4 = 0, tn3i3 = 0; tn2i4 < tn2s4; tn2i4++, tn3i3++) {
		for (int tn1i1 = 0, tn3i1 = 0; tn1i1 < tn1s1; tn1i1 +=
stride, tn3i1++) {
			for (int tn1i2 = 0, tn3i2 = 0; tn1i2 < tn1s2; tn1i2 +=
stride, tn3i2++) {
				double tmpsum = 0.0;
				for (int tn2i3 = 0; tn2i3 < tn2s3; tn2i3++) {
					// note tn1s3=tn2s3
					for (int tn2i1 = -tn2s1by2; tn2i1 <=
tn2s1by2; tn2i1++) {
						for (int tn2i2 = -tn2s2by2; tn2i2 <=
tn2s2by2; tn2i2++) {
							if (((tn1i1 + tn2i1) >= 0) &&
((tn1i1 + tn2i1) < tn1s1) && ((tn1i2 + tn2i2) >= 0) && ((tn1i2 +
tn2i2) <tn1s1)) { // zero padding of tn1
```

```
                                                tmpsum += tn2[tn2i1 +
tn2s1by2][tn2i2 + tn2s2by2][tn2i3][tn2i4] * tn1[tn1i1 + tn2i1][tn1i2 +
tn2i2][tn2i3];
                                          }
                                     }
                                }
                           }
                           tn3[tn3i1][tn3i2][tn3i3] = tmpsum + bias[tn3i3];
                           cout << tn3[tn3i1][tn3i2][tn3i3] << "  ";
                     }
                     cout << endl;
               }
               cout << endl << endl;
          }

          cout << endl << "Enter any char to continue : ";
          cin >> c;

          vector<double> bias2; // bias vector
          bias2.resize(tn4s4); // note: tn5 has the same size= tn4s4
          cout << "Bias vector 2:  " << endl;
          for (int i4 = 0; i4 < tn4s4; i4++)
          {
               bias2[i4] = ((double)(rand() % 100)) / 100.0;
               cout << bias2[i4] << "   ";
          }
          cout << endl;

          cout << endl << "Enter any char to continue : ";
          cin >> c;

          cout << "Output of fully connected layer: tn4s4=10 fully
connected filters to tn1 produce the output tn5 " <<endl;

          vector<double> tn5(tn4s4);

          for (int tn4i4 = 0; tn4i4 < tn4s4; tn4i4++) {
               // note tn1s1=tn4s1 tn1s2=tn4s2, tn1s3=tn4s3
               double tmpsum = 0.0;
               for (int tn4i3 = 0; tn4i3 < tn4s3; tn4i3++) {
                    for (int tn4i1 = 0; tn4i1 < tn4s1; tn4i1++) {
                         for (int tn4i2 = 0; tn4i2 < tn4s2; tn4i2++) {
                              tmpsum += tn4[tn4i1][tn4i2][tn4i3][tn4i4] *
tn1[tn4i1][tn4i2][tn4i3];
                         }
                    }
               }
               tn5[tn4i4] = tmpsum+bias2[tn4i4];
               cout << tmpsum << "  ";
          }
          cout << endl;
```

```
	cout << endl << "Enter any char to continue : ";
	cin >> c;

	// Computing softmax of tn5 after normalizing
	double tmpsum = 0.0;
	for (int i = 0; i < tn4s4; i++) {
		tmpsum += ( tn5[i] * tn5[i] );
	}
	tmpsum = sqrt(tmpsum);
	for (int i = 0; i < tn4s4; i++) {
		tn5[i] /= tmpsum;
	}
// compute softmax
	tmpsum = 0.0;
	for (int i = 0; i < tn4s4; i++) {
		tmpsum += (exp(tn5[i])) ;
	}

	cout << "Softmax probabilities : " << endl;
	for (int i = 0; i < tn4s4; i++) {
		tn5[i] = (exp(tn5[i]))/tmpsum;
		cout << tn5[i] << "   ";
	}

	cout << endl << "Enter any char to continue : ";
	cin >> c;

	return 0;
}
```

```
#include <iostream>
#include <cstdlib>   // for rand(), srand()
#include <ctime>     // for time()
#include <fstream>
using namespace std;

//Program for generating input data for Project 1
// by Prof. Murali Subbarao, ESE 344

void data3d(int s1, int s2, int s3) {
	srand((unsigned int)time(NULL));// seed rand() with system time
	//Generating input for project 1
	for (int i3 = 0; i3 < s3; i3++) {
		for (int i2 = 0; i2 < s2; i2++) {
			for (int i1 = 0; i1 < s1; i1++) {
				cout << (double)((rand() % 100) / 100.0) << "   ";
			}
			cout << endl;
		}
		cout << endl << endl;
	}
```

```
	cout << endl << endl;
}

	void data4d(int s1, int s2, int s3, int s4) {

		srand((unsigned int)time(NULL));// seed rand() with system time

		//Generating input for project 1
		for (int i4 = 0; i4 < s4; i4++) {
			for (int i3 = 0; i3 < s3; i3++) {
				for (int i2 = 0; i2 < s2; i2++) {
					for (int i1 = 0; i1 < s1; i1++) {
						cout << (double)((rand() % 100) / 100.0) << "
";
					}
					cout << endl;
				}
				cout << endl <<endl;
			}
			cout << endl << endl;
		}
	cout << endl << endl;
}

	void data1d(int s1) {
		srand((unsigned int)time(NULL));// seed rand() with system time
		//Generating input for project 1
		for (int i1 = 0; i1 < s1; i1++) {
			cout << (double)((rand() % 100) / 100.0) << "   ";
		}
		cout << endl<<endl;
	}


int main()
{	char c;
// Generate input data for project 1

	data3d(32, 32, 3); //D1
	data4d(5, 5, 3, 16); //D3
	data1d(16); //D4
	data4d(5, 5, 16, 20); //D8
	data1d(20); //D9
	data4d(5, 5, 20,20);  //D13
	data1d(20); //D14
	data4d(4,4,20,10); //D18
	data1d(10);  //D19

//	cout << endl << "Enter any char to continue : ";
//	cin >> c;

	return 0;
}
//Program for generating and reading input data for Project 1
// by Prof. Murali Subbarao, ESE 344

#include <iostream>
#include <cstdlib>   // for rand(), srand()
```

```cpp
#include <ctime>     // for time()
#include <fstream>
#include <vector>
using namespace std;

//Program for generating input data fore Project 1
// by Prof. Murali Subbarao, ESE 344

// generate random data for a 3d tensor
void data3d(int s1, int s2, int s3) {
	srand((unsigned int)time(NULL));// seed rand() with system time
	//Generating input for project 1
	for (int i3 = 0; i3 < s3; i3++) {
		for (int i2 = 0; i2 < s2; i2++) {
			for (int i1 = 0; i1 < s1; i1++) {
				cout << (double)((rand() % 100) / 100.0) << "   ";
			}
			cout << endl;
		}
		cout << endl << endl;
	}
	cout << endl << endl;
}

// generate random data for a 2d tensor
void data2d(int s1, int s2) {
	srand((unsigned int)time(NULL));// seed rand() with system time
	//Generating input for project 1
	for (int i2 = 0; i2 < s2; i2++) {
		for (int i1 = 0; i1 < s1; i1++) {
			cout << (double)((rand() % 100) / 100.0) << "   ";
		}
		cout << endl;
	}
	cout << endl << endl;
}


// generate random data for a 4d tensor
void data4d(int s1, int s2, int s3, int s4) {

	srand((unsigned int)time(NULL));// seed rand() with system time

	//Generating input for project 1
	for (int i4 = 0; i4 < s4; i4++) {
		for (int i3 = 0; i3 < s3; i3++) {
			for (int i2 = 0; i2 < s2; i2++) {
				for (int i1 = 0; i1 < s1; i1++) {
					cout << (double)((rand() % 100) / 100.0) << "   ";
				}
				cout << endl;
			}
			cout << endl << endl;
		}
		cout << endl << endl;
	}
	cout << endl << endl;
}
```

```
// generate random data for a 1d vector
void data1d(int s1) {
	srand((unsigned int)time(NULL));// seed rand() with system time
	//Generating input for project 1
	for (int i1 = 0; i1 < s1; i1++) {
		cout << (double)((rand() % 100) / 100.0) << "   ";
	}
	cout << endl << endl;
}



// allocate memory for a 1d vector
void alloc1d(vector<double>& tn1d, int s1) {
	// allocate memory for a 1d tensor tn1d of size s1
	tn1d.resize(s1);
}

// read data for a 1d tensor
void read1d(vector<double>& tn1d, int s1) {
	for (int i1 = 0; i1 < s1; i1++)
	{
		cin >> tn1d[i1];
	}
}
// print data for a 1d tensor
void print1d(vector<double>& tn1d, int s1) {
	cout << endl << endl;
	for (int i1 = 0; i1 < s1; i1++)
	{
		cout << tn1d[i1] << "   ";
	}
	cout << endl << endl;
}

// allocate memory for a 2d tensor
void alloc2d(vector<vector<double>>& tn2d, int s1, int s2) {
	// allocate memory for a 2d tensor tn2d of size s1, s2
	tn2d.resize(s1);
	for (int i1 = 0; i1 < s1; i1++) {
		tn2d[i1].resize(s2);
	}
}

// read data for a 2d tensor
void read2d(vector<vector<double>>& tn2d, int s1, int s2) {

	for (int i2 = 0; i2 < s2; i2++) {
		for (int i1 = 0; i1 < s1; i1++)
		{
			cin >> tn2d[i1][i2];
		}
	}
}

// print data for a 2d tensor
void print2d(vector<vector<double>>& tn2d, int s1, int s2) {
```

```
	cout << endl << endl;
	for (int i2 = 0; i2 < s2; i2++) {
		for (int i1 = 0; i1 < s1; i1++)
		{
			cout << tn2d[i1][i2] << "   ";
		}
		cout << endl;
	}
	cout << endl << endl;
}

// allocate memory for a 3d tensor
void alloc3d(vector<vector<vector<double>>>& tn3d, int s1, int s2, int s3) {
	// allocate memory for a 3d tensor tn3d of size s1, s2, s3
	tn3d.resize(s1);
	for (int i1 = 0; i1 < s1; i1++) {
		tn3d[i1].resize(s2);
		for (int i2 = 0; i2 < s2; i2++) {
			tn3d[i1][i2].resize(s3);
		}
	}
}

// read data for a 3d tensor
void read3d(vector<vector<vector<double>>>& tn3d, int s1, int s2, int s3) {
	for (int i3 = 0; i3 < s3; i3++) {
		for (int i2 = 0; i2 < s2; i2++) {
			for (int i1 = 0; i1 < s1; i1++)
			{
				cin >> tn3d[i1][i2][i3];
			}
		}
	}
}

// print data for a 3d tensor
void print3d(vector<vector<vector<double>>>& tn3d, int s1, int s2, int s3) {
	cout << endl << endl;
	for (int i3 = 0; i3 < s3; i3++) {
		for (int i2 = 0; i2 < s2; i2++) {
			for (int i1 = 0; i1 < s1; i1++)
			{
				cout << tn3d[i1][i2][i3] << "  ";
			}
			cout << endl;
		}
		cout << endl << endl;
	}
	cout << endl << endl;
}

// allocate memory for a 4d tensor tn4d of size s1, s2, s3, s4
void alloc4d(vector<vector<vector<vector<double>>>>& tn4d, int s1, int s2, int s3, int
s4) {

	tn4d.resize(s1);

	for (int i1 = 0; i1 < s1; i1++) {
```

```
                tn4d[i1].resize(s2);
                for (int i2 = 0; i2 < s2; i2++) {
                        tn4d[i1][i2].resize(s3);
                        for (int i3 = 0; i3 < s3; i3++)
                        {
                                tn4d[i1][i2][i3].resize(s4);
                        }
                }
        }
}

// read a 4d tensor tn4d of size s1, s2, s3, s4
void read4d(vector<vector<vector<vector<double>>>>& tn4d, int s1, int s2, int s3, int s4)
{

        for (int i4 = 0; i4 < s4; i4++) {
                for (int i3 = 0; i3 < s3; i3++) {
                        for (int i2 = 0; i2 < s2; i2++) {
                                for (int i1 = 0; i1 < s1; i1++) {
                                        cin >> tn4d[i1][i2][i3][i4];
                                }
                        }
                }
        }
}

// print a 4d tensor tn4d of size s1, s2, s3, s4
void print4d(vector<vector<vector<vector<double>>>>& tn4d, int s1, int s2, int s3, int
s4) {
        cout << endl << endl;
        for (int i4 = 0; i4 < s4; i4++) {
                for (int i3 = 0; i3 < s3; i3++) {
                        for (int i2 = 0; i2 < s2; i2++) {
                                for (int i1 = 0; i1 < s1; i1++) {
                                        cout << tn4d[i1][i2][i3][i4] << "    ";
                                }
                                cout << endl;
                        }
                        cout << endl << endl;
                }
                cout << endl << endl;
        }
        cout << endl << endl;
}


int main()
{
        char c;

        /*
        //   Part 1 is to generate data.
        //   Part 2 is to read and print data.

        // Part 1: Generate input data for project 1

                data3d(32, 32, 3); //D1
                data4d(5, 5, 3, 16); //D3
```

```
            data1d(16); //D4
            data4d(5, 5, 16, 20); //D8
            data1d(20); //D9
            data4d(5, 5, 20,20);  //D13
            data1d(20); //D14
            data4d(4,4,20,10); //D18
            data1d(10);  //D19

        //      cout << endl << "Enter any char to continue : ";
        //      cin >> c;

        */

        // Comment this part when generating input data

        //Part 2:  Allocate memory and,  Read and print input data generated above
        vector<vector<vector<double>>> D1;
        vector<vector<vector<vector<double>>>> D3, D8, D13, D18;
        vector<double> D4, D9, D14, D19;

        alloc3d(D1, 32, 32, 3); //D1
        read3d(D1, 32, 32, 3);
        print3d(D1, 32, 32, 3);

        alloc4d(D3, 5, 5, 3, 16); //D3
        read4d(D3, 5, 5, 3, 16); //D3
        print4d(D3, 5, 5, 3, 16); //D3

        alloc1d(D4, 16); //D4
        read1d(D4, 16); //
        print1d(D4, 16); //

        alloc4d(D8, 5, 5, 16, 20); //D8
        read4d(D8, 5, 5, 16, 20); //D8
        print4d(D8, 5, 5, 16, 20); //D8

        alloc1d(D9, 20); //D9
        read1d(D9, 20); //
        print1d(D9, 20); //

        alloc4d(D13, 5, 5, 20, 20); //D13
        read4d(D13, 5, 5, 20, 20); //
        print4d(D13, 5, 5, 20, 20); //

        alloc1d(D14, 20); //D14
        read1d(D14, 20); //
        print1d(D14, 20); //

        alloc4d(D18, 4, 4, 20, 10); //D18
        read4d(D18, 4, 4, 20, 10); //D18
        print4d(D18, 4, 4, 20, 10); //D18

        alloc1d(D19, 10); //D19
        read1d(D19, 10); //
        print1d(D19, 10); //

        return 0;
}
```