Benha University
Faculty of Science

Department of Mathematics
and computer science

Scientific Computing (2)
452 MC

Table of Contents

Chapter 1

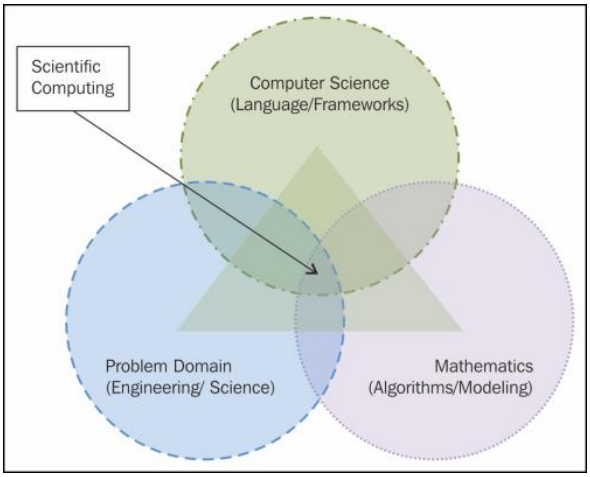
Introduction

Chapter 1

Introduction

What is Scientific Computing?

Scientific Computing is the collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in Science and Engineering.



Computation becomes crucially important in situations such as:

- The problem at hand cannot be solved by traditional experimental or theoretical means
- Experimentation may be dangerous, e.g., characterization of toxic materials
- The problem would be too expensive or time-consuming to try to solve by other avenues, e.g. determination of the structure of proteins

In terms of computer science, scientific computing can be considered a numerical simulation of a mathematical model and domain data/information. The objective behind a simulation depends on the domain of the application under simulation. The objective can be to understand the cause behind an event, reconstruct a specific situation, optimize the process, or predict the occurrence of an event. There are several situations where numerical simulation is the only choice, or the best choice. There are some phenomena or situations where performing experiments are almost impossible, for example, climate research, astrophysics, and weather forecasts. In some other situations, actual experiments are not preferable, for example, to check the stability or strength of some material or product. Some experiments are very costly in terms of time/economy, such as car crashes or life science experiments. In such scenarios, scientific computing helps users analyze and solve problems without spending much time or cost.

What is MATLAB?

MATLAB is a software package for high-performance mathematical computation, visualization, and programming environment. It provides an interactive environment with hundreds of built-in functions for technical computing, graphics, and animations.

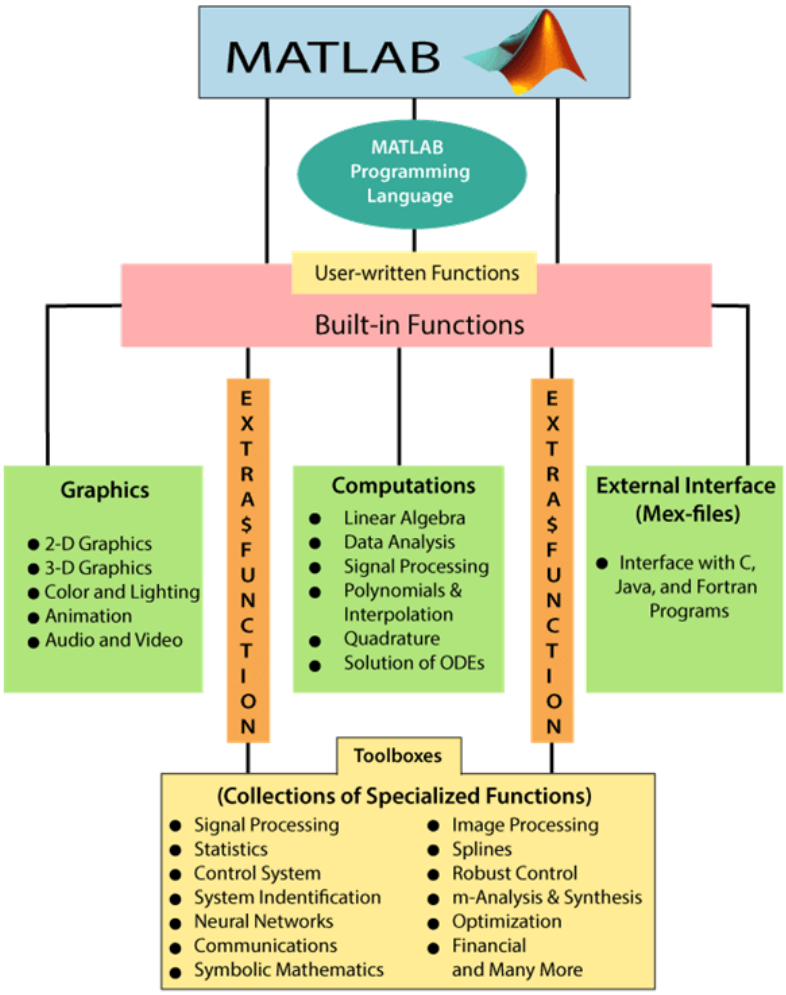
MATLAB stands for Matrix Laboratory. MATLAB was written initially to implement a simple approach to matrix software **developed by the LINPACK (Linear system package)** and **EISPACK** (Eigen system package) projects.

MATLAB Features

As there are numerous features to describe, but here, we will focus on some of the key features:

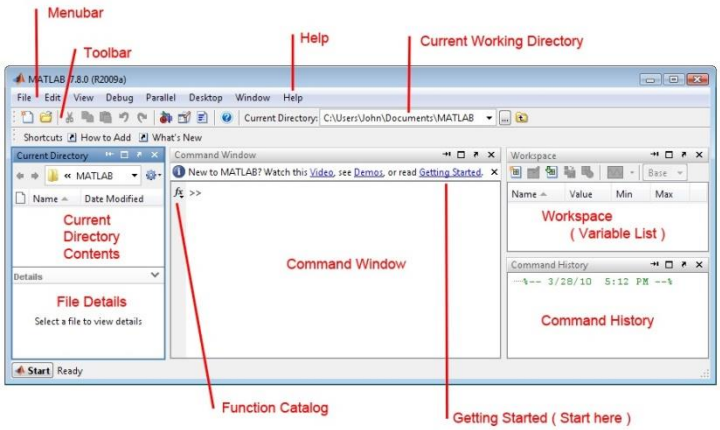
- It is designed for numerical as well as symbolic computing.
- It's a high-level language used mainly for engineering and scientific computing.
- It works within a Desktop environment providing full features for iterative exploration, design, and problem-solving.
- Creation of custom plots for visualizing data and tools, with the help of built-in Graphics.
- Specific applications are designed to work with any particular type of problems, such as data classification, control system design and tuning, signal analysis.
- Provide interfaces to work with other programming languages such as C, C++, Java, .NET, Python, SQL, Hadoop.

The diagram in the figure shows the main features and capabilities of MATLAB.



A schematic diagram of MATLAB's main features.

Workspace and Variables in MATLAB

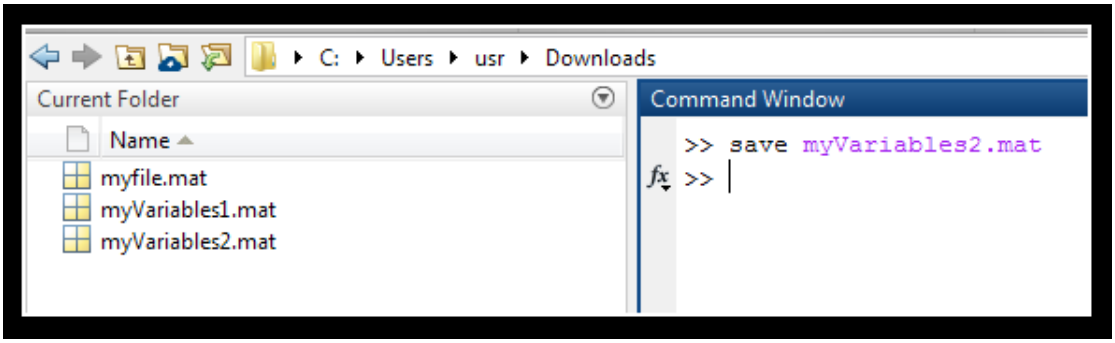


Workspace

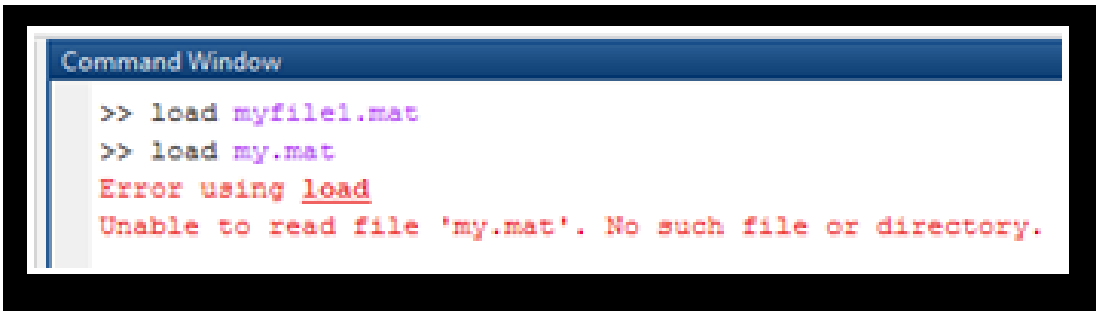
- The workspace contains all variables we create while working in MATLAB.
- Whenever we assign a value to a variable, it automatically gets space in the workspace

MAT-file in MATLAB

- The file saved with .mat extension is called a MAT-file in MATLAB.
- As variables in the workspace no longer exist after the closing of the environment, so these variables are saved in MAT-file for later us
- We can change the current working folder by using **the Browse For Folder** button located above the current folder pane.
- After running the save command, the file saved with .mat extension becomes visible in the Current Folder Pane.
- After saving the workspace, we can clear all the contents of the workspace by using **the clear** command at the command line.



Before loading the file, ensure the parent folder is selected as the current folder; otherwise, it will show error:


Command Window
>> load myfile1.mat
>> load my.mat
Error using load
Unable to read file 'my.mat'. No such file or directory.

Chapter 2

Basic use of Matlab

2.1 Introduction

2.2 Entering a vector

2.3 Dealing with matrices

2.4 Matrix indexing

2.5 Array Commands

2.6 Generating a Numeric Sequence

2.7 Relational and logical operators

2.8 Commands for Managing a Session

2.9 Creating simple plots

Chapter 2

Basic use of Matlab

2.1 Introduction in Matlab

Let's start at the very beginning. For example, let's suppose you want to calculate the expression, 1 + 2 * 3. You type it at the prompt command (>>) as follows,

>> 1+2*3

ans = 7

You will have noticed that if you do not specify an output variable, MATLAB uses a default variable ans, short for answer, to avoid this, you may assign a value to a variable or output argument name. For example,

>> x = 1+2*3

x =7

Therefore, computing 4x will result in

>> 4*x

ans =28.0000

Note that for the output of 11/7 the result was only given to a few decimal places. In fact, Matlab does all its calculations internally to double precision. However, the default display format is to use only eight decimal places. We can change this by using the format function. For example:

>> format long

>> 11/7

ans =1.57142857142857

Table 1 gives the partial list of arithmetic operators.

Table 1: Basic arithmetic operators

Symbol	Operation	Example
+	Addition	a + b
-	Subtraction	a-b
*	Multiplication	a*b
/	Division	a/b
.*	Element by element Multiplication	a.*b

Matlab has all the elementary mathematical functions built in:

>> **sqrt(2)**

ans =1.4142

>> **sin(pi/8)**

ans =0.3827

>> log(10)

ans =2.3026

>> log10(2)

ans = 0.3010

>>exp(10)

ans = 2.2026e+004

2.2 Entering a vector

A vector is a special case of a matrix. The purpose of this section is to show how to create vectors and matrices in MATLAB. As discussed earlier, an array of dimension 1£n is called a row vector, whereas an array of dimension m£ 1 is called a column vector. The elements of vectors in MATLAB are enclosed by square brackets and are separated by spaces or by commas. For example, to enter a row vector, v, type

>> v = [1 4 7 10 13]

v = 1 4 7 10 13

Column vectors are created in a similar way, however, semicolon (;) must separate the components of a column vector,

>> w = [1;4;7;10;13]
w =
1
4
7
10
13

On the other hand, a row vector is converted to a column vector using the transpose operator. The transpose operation is denoted by an apostrophe or a single quote (').

>> w = v'
w =
1
4
7
10
13

For example, to access the first three elements of v, we write,

>> v(1:3)

ans =1 4 7

Or, all elements from the third through the last elements,

>> v(3,end)

ans =7 10 13

2.3 Dealing with matrices

There are several ways to build a matrix: One way to do it is to declare a matrix as if you wrote it by hand

```
>> A=[1 2 3
     4 5 6
     7 8 9]
```

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

Another way is to separate rows with

```
>> A=[1 2 3;4 5 6; 7 8 9]
```

Another way is to do it element by element

```
>> A(1,1)=1;
>> A(1,2)=2;
>> A(1,3)=3;
>> A(2,1)=4;
>> A(2,2)=5;
```

>> A(2,3)=6;

2.4 Matrix indexing

Here we substitute A(3,3)=9 by A(3,3)=0. The result is

>> A(3,3) = 0

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{matrix}$$

>> A(2,:)

ans =

4 5 6

is the second-row elements of A.

The colon operator can also be used to extract a sub-matrix from a matrix A.

>> A(:,2:3)

ans =

2 3

5 6

8 0

A(:,2:3) is a sub-matrix with the last two columns of A.

A row or a column of a matrix can be deleted by setting it to a null vector.

>> A(:,2)=[]
ans =
1 3
4 6
7 0

Matrix arithmetic operations

A+B or B+A is valid if A and B are of the same size

A*B is valid if A's number of columns equals B's number of rows

A^2 is valid if A is square and equals A*A

∝*A or A*∝ multiplies each element of A by ∝

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

$$B = \begin{matrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{matrix}$$

>> C = A.*B

$$C = \begin{matrix} 10 & 40 & 90 \\ 160 & 250 & 360 \\ 490 & 640 & 810 \end{matrix}$$

>> A.^2

$$\text{ans} = \begin{matrix} 1 & 4 & 9 \\ 16 & 25 & 36 \\ 49 & 64 & 81 \end{matrix}$$

2.5 Array Commands

MATLAB has several functions that create different kinds of matrices. The zero (or Null) matrix: zero(m,n) creates a m-by-n matrix of zeros. Thus,

>> B=zeros (3,2)

B = 0 0
0 0
0 0

The ones matrix: zero(m,n) creates a m-by-n matrix of zeros. Thus,

>> C=ones(2,3)

results is

C = 1 1 1
1 1 1

The identity matrix: eye(n) creates a n-by-n matrix with ones along the diagonal and zeros everywhere else.

>> D = eye (3)

$$D = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

The following functions change the shape of a matrix:

Reshape 3-by-4 matrix A to have dimensions 2-by-6:

A = [1 4 7 10; 2 5 8 11; 3 6 9 12]

$$A = \begin{matrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{matrix}$$

>>B = reshape(A, 2, 6)

B = 1 3 5 7 9 11

2 4 6 8 10 12

Transposing a Matrix. Transpose A so that the row elements become columns. You can use either the transpose function or the transpose operator (.') to do this:

>> B = A.'

$$B = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{matrix}$$

Rotating a Matrix. Rotate the matrix by 90 degrees:

>>B = rot90(A)

$$B = \begin{matrix} 10 & 11 & 12 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{matrix}$$

Flipping a Matrix. Flip A in a left-to-right direction:

>>B = fliplr(A) Left to Right

$$\begin{matrix} 10 & 7 & 4 & 1 \\ 11 & 8 & 5 & 2 \\ 12 & 9 & 6 & 3 \end{matrix}$$

The following table shows various commands used for working with arrays, matrices and vectors:

Command	**Purpose**
Cat	Concatenates arrays.

Find	Finds indices of nonzero elements.
Length	Computes number of elements.
Linspace	Creates regularly spaced vector.
Max	Returns largest element
Min	Returns smallest element.
Size	Computes array size.
Sort	Sorts each column
Sum	Sums each column
det	Computes determinant of an array.
inv	Computes inverse of a matrix.
rand	Create random numbers
horzcat	Horizontally concatenate matrices
vertcat	Vertically concatenate matrices
magic	Create a square matrix with rows, columns, and diagonals that add up to the same number.

2.6 Generating a Numeric Sequence

The colon operator (first : last) generates a 1-by-n matrix (or vector) of sequential numbers from the first value to the last. The default sequence is made up of incremental values, each 1 greater than the previous one:

>>A = 10:15

A = 10 11 12 13 14 15

The numeric sequence does not have to be made up of positive integers. It can include negative numbers and fractional numbers as well:

>>A = -2.5:2.5

A = -2.5000 -1.5000 -0.5000 0.5000 1.5000 2.5000

By default, MATLAB always increments by exactly 1 when creating the sequence, even if the ending value is not an integral distance from the start:

>>A = 1:6.3

A = 1 2 3 4 5 6

To generate a series of numbers from 10 to 50, incrementing by 5, use

>>A = 10:5:50

A = 10 15 20 25 30 35 40 45 50

This example increments by 0.2:

>>A = 3:0.2:3.8

A = 3.0000 3.2000 3.4000 3.6000 3.8000

To create a sequence with a decrementing interval,

>>A = 9:-1:1

A = 9 8 7 6 5 4 3 2 1

Data Type Conversion

MATLAB provides various functions for converting a value from one data type to another. The following table shows the data type conversion functions:

types	Description
char	Convert to character array (string)
int2str	Convert integer data to string
mat2str	Convert matrix to string
str2double	Convert string to double-precision value
str2num	Convert string to number
dec2bin	Convert decimal to binary number in string
cell2mat	Convert cell array to numeric array
hex2dec	Convert hexadecimal number string to decimal number

2.7 Relational and logical operators

A relational operator compares two numbers by determining whether a comparison is true or false.

Relational operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
= =	Equal to
~ =	Not Equal to

Example:

```
A = [2 4 6; 8 10 12]
B = [5 5 5; 9 9 9]
A < B
ans =
     1     1     0
     1     0     0
```

Similarly, you can compare one of the arrays to a scalar.

```
A > 7
ans =
     0     0     0
     1     1     1
```

Logical Operators and Functions: MATLAB has three Logical Operators (or Boolean Operators).

Operator	Name
&	And
|	Or
~	Not

Example:

Assume if A = 60; and B = 13; Now in binary format they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

2.8 Commands for Managing a Session

MATLAB provides various commands for managing a session. The following table provides all such commands:

command	purpose

clc	Clears command window.
clear	Removes variables from memory.
exist	Checks for existence of file or variable.
help	Searches for a help topic
who	Lists current variables.
whos	Lists current variables (long display)

Functions to Find the Structure or Shape of a Matrix

function	description
isempty	Return true for 0-by-0 or 0-by-n matrices.
isscalar	Return true for 1-by-1 matrices.
isvector	Return true for 1-by-n matrices
ndims	Return the number of dimensions in a matrix
numel	Return the number of elements in a matrix.

2.9 Creating simple plots

The general form of the plot instruction is plot (x, y, S) where x and y are vectors or matrices and S is a one, two or three charter string specifying color, marker symbol, or line style.

The S string is optional and is made of the following (and more) characters:

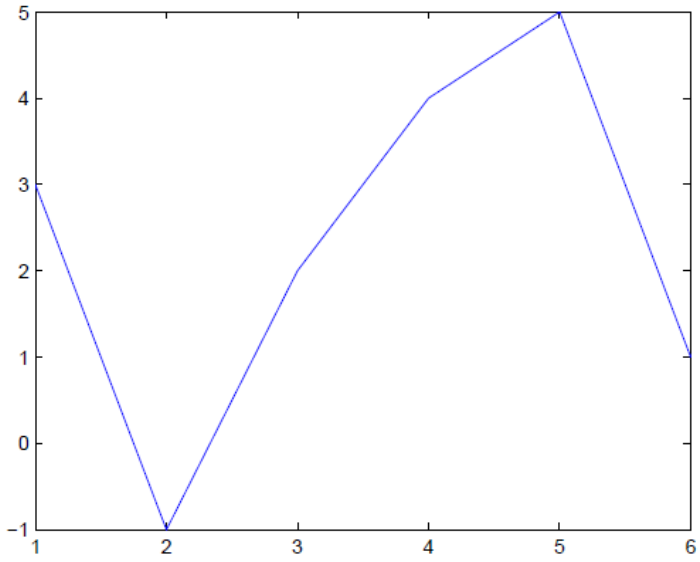
symbol	color	symbol	color
Y	yellow	.	point
M	magenta	0	circle
C	cyan	+	plus
R	red	:	dotted
G	green	-	solid
B	blue	-.	dashdot
W	white	x	x-mark
K	black	*	asterisk

Example:

The basic MATLAB graphing procedure, for example in 2D, is to take a vector of x-coordinates, x = (x_1; ... ; x_N), and a vector of y-coordinates, y = (y_1; ... ; y_N)), locate the points (x_i; y_i), with $i = 1; 2; \dots ; n$ and then join them by straight lines. You need to prepare x and y in an identical array form; namely, x and y are both row arrays or column arrays of the same length.

The vectors x = (1; 2; 3; 4; 5; 6) and y = (3;¡1; 2; 4; 5; 1)

```
>> x = [1 2 3 4 5 6];
>> y = [3 -1 2 4 5 1];
>> plot(x,y)
```



For example, to plot the function sin (*x*) on the interval [0*;* 2¼], we ¯rst create a vector of *x* values ranging from 0 to 2¼, then compute the *sine* of these values, and finally plot the result:

```
>> x = 0:pi/100:2*pi;
>> y = sin(x);
>> plot(x,y)
```

Notes:

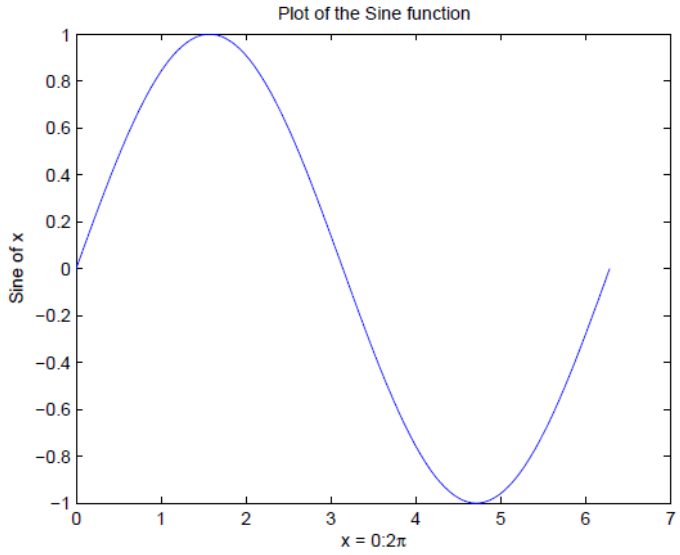
0:pi/100:2*pi yields a vector that

- starts at 0,
- takes steps (or increments) of ¼=100,
- stops when 2π is reached.

Adding titles, axis labels, and annotations

MATLAB enables you to add axis labels and titles. For example, using the graph from the previous example, add an *x*- and *y*-axis labels.

```
>> xlabel('x = 0:2pi')
>> ylabel('Sine of x')
>> title('Plot of the Sine function')
```

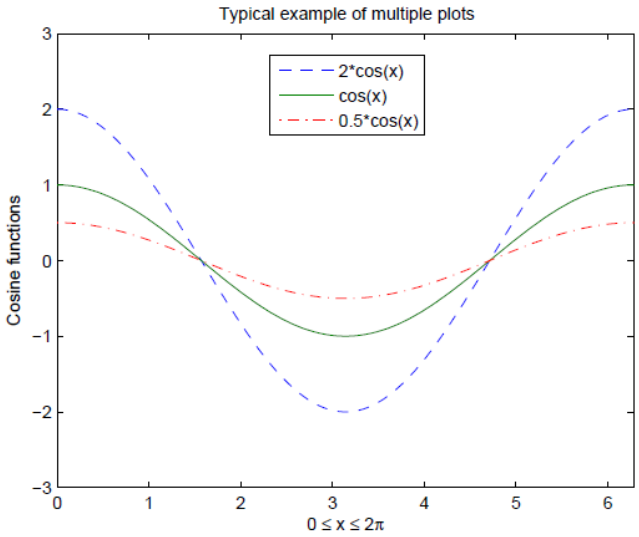


Multiple data sets in one plot

Multiple (x; y) pairs arguments create multiple graphs with a single call to plot. For example, these statements plot three related functions of x: y1 = 2 cos(x), y2 = cos(x), and y3 = 0:5 * cos(x), in the interval [0, 2π].

```
>> x = 0:pi/100:2*pi;
>> y1 = 2*cos(x);
```

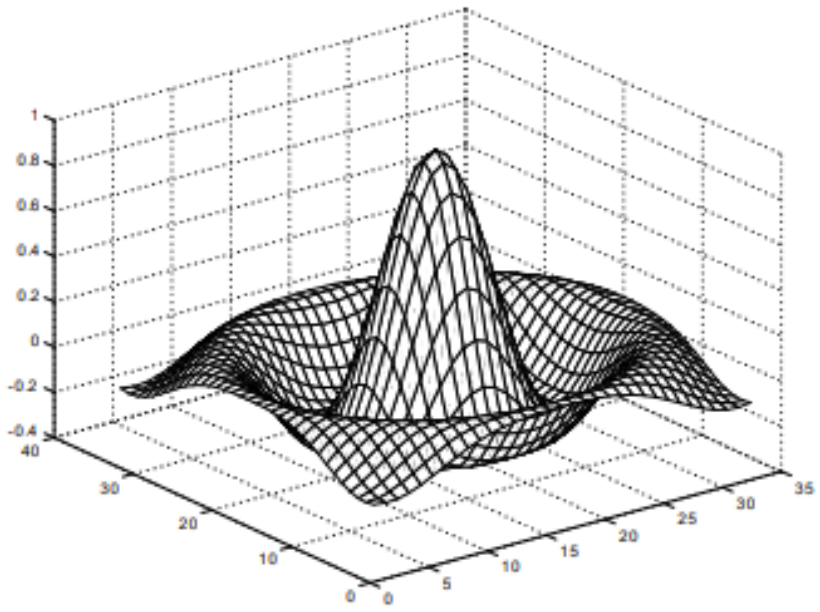
```
>> y2 = cos(x);
>> y3 = 0.5*cos(x);
>> plot(x,y1,'--',x,y2,'-',x,y3,':')
>> xlabel('0 \leq x \leq 2\pi')
>> ylabel('Cosine functions')
>> legend('2*cos(x)','cos(x)','0.5*cos(x)')
>> title('Typical example of multiple plots')
>> axis([0 2*pi -3 3])
```



Matlab defines a mesh surface by the z coordinates of points above a rectangular grid in the x-y plane. It forms a plot by joining adjacent points with straight lines

```
>> x = -8:0.5:8;
>> y = x;
>> [X,Y] = meshgrid (x,y);
```

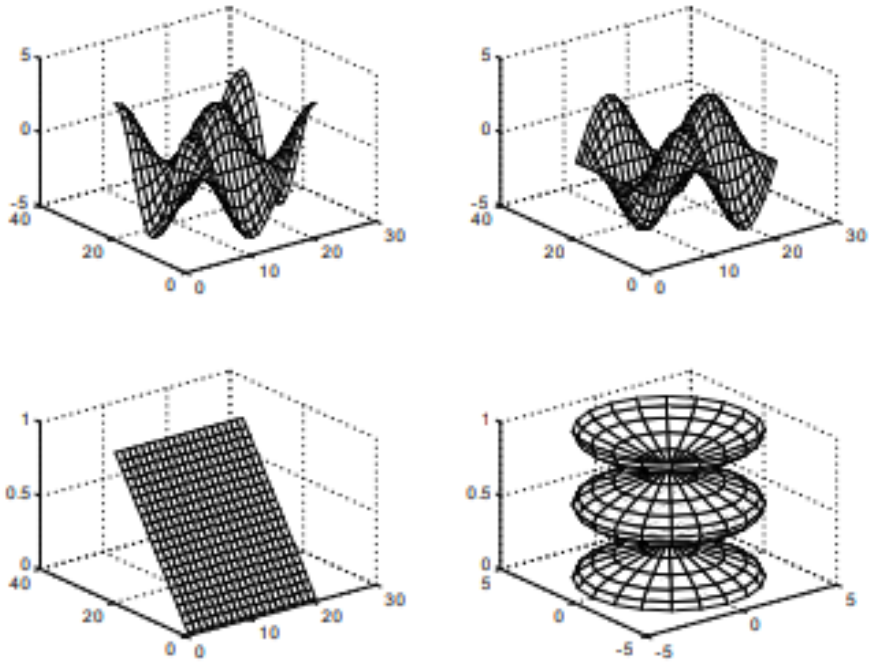
```
>> R = sqrt(X.^2 + Y.^2) + eps;
 >> Z = sin(R)./R;
>> mesh(Z)
```



We can display multiple plots in the same window or print them on the same piece of paper with the subplot function. subplot(m,n,p) breaks the figure window into an m-by-n matrix of small subplots and select the p-th subplot for the current plot.

```
>> t = 0:pi/10:2*pi;
>> [X,Y,Z] = cylinder(4*cos(t));
>> subplot(2,2,1)
>> mesh(X)
 >> subplot(2,2,2)
>> mesh(Y)
 >> subplot(2,2,3)
```

```
>> mesh(Z)
>> subplot(2,2,4)
>> mesh(X,Y,Z)
```



Matlab 2D plot

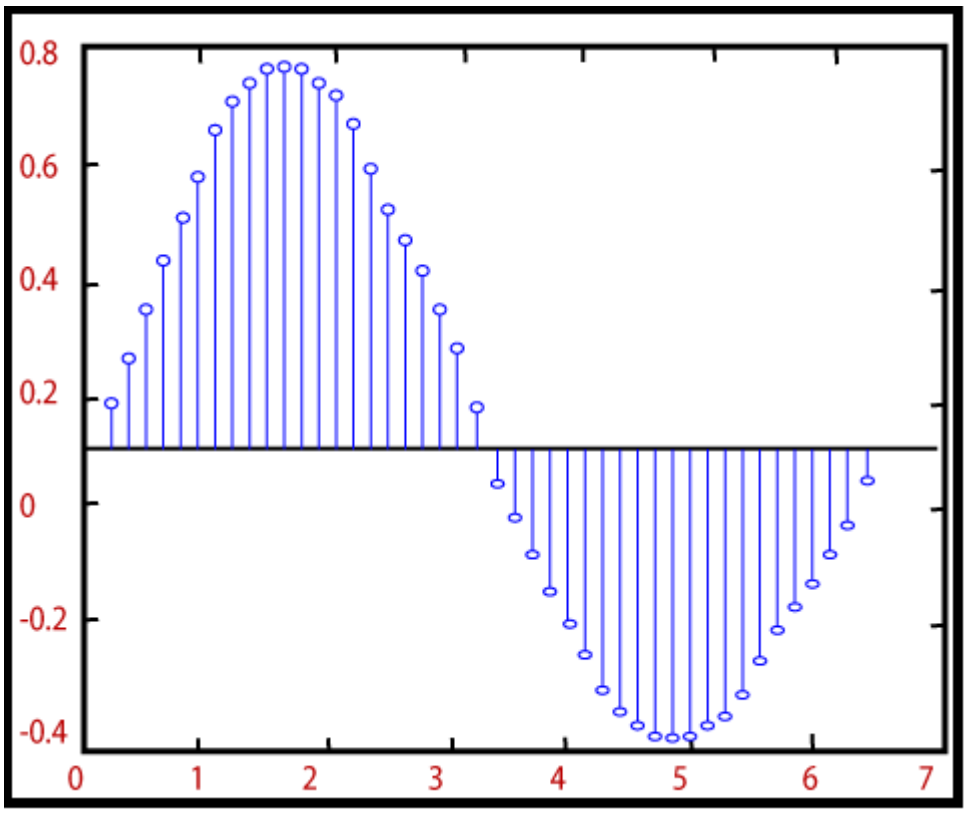
Name	Description
errorbar	Plot error bars along a curve
Semilogy	plot of the values of x and y, using a linear scale for x and a logarithmic scale for y
Semilogx	lot of the values of x and y, using the logarithmic scale for x and the linear scale for y.
hist	presenting the distribution of values within a data set.
Pie	determines the percentage of the total pie corresponding to each value of x, and plots pie slices

	of that size.
Bar	each point is represented by a vertical bar or horizontal bar.
stem	plot shows data as lines extending from a baseline along the x-axis

Example:

f=e^-t/5 sint,0≤t≤2π

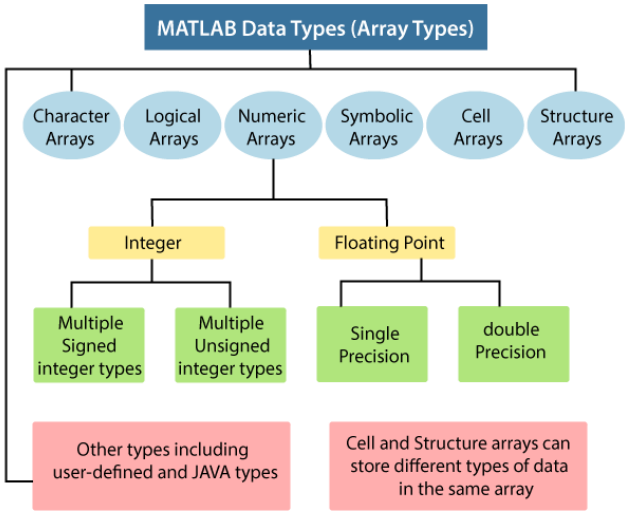
```
t=linspace (0, 2*pi, 200);
f=exp (-.2*t).*sin(t);
stem(t, f)
```



MATLAB 3D Plots

Name	Description
Plot3	shows a three-dimensional plot of a set of data points.
Fill3	creates flat-shaded and Gouraud-shaded polygons.
contour3	generates a three-dimensional contour plot of a surface defined on a rectangular grid.
surf	creates a surface plot.
sphere	develops the x-, y-, and z-coordinates of a unit sphere
cylinder	creates x, y, and z coordinates of the unit cylinder.
ellipsoid	It generates an ellipsoid function

MATLAB Data Types



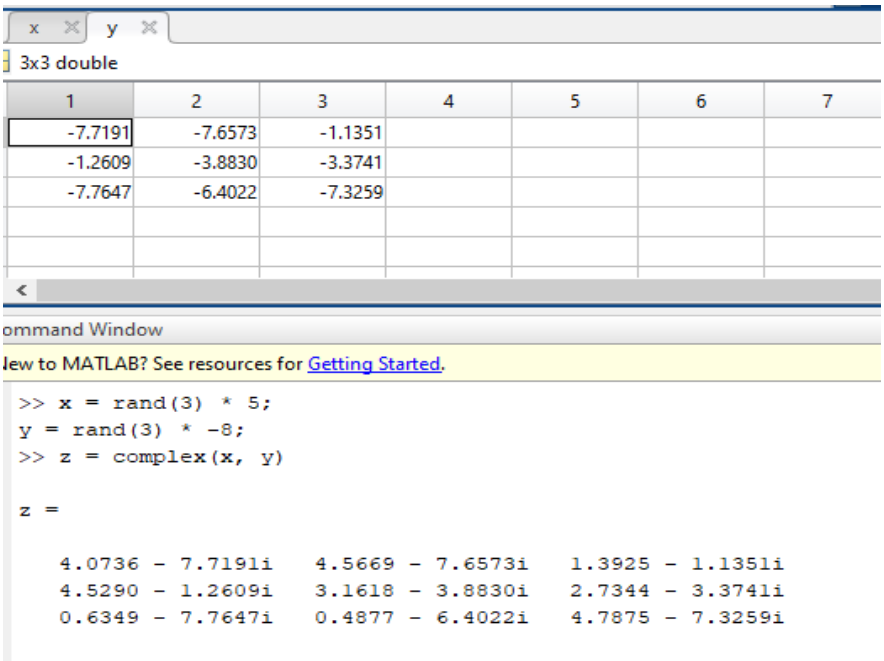
There are eight numeric data type

Data Type	Range of Values	Conversion Function
Signed 8-bit integer	-2^7 to 2^7-1	int8
Signed 16-bit integer	-2^{15} to $2^{15}-1$	int16
Signed 32-bit integer	-2^{31} to $2^{31}-1$	int32
Signed 64-bit integer	-2^{63} to $2^{63}-1$	int64
Unsigned 8-bit integer	0 to 2^8-1	uint8
Unsigned 16-bit integer	0 to $2^{16}-1$	uint16
Unsigned 32-bit integer	0 to $2^{32}-1$	uint32
Unsigned 64-bit integer	0 to $2^{64}-1$	uint64

Creating Complex Numbers

The following statement shows one way of creating a complex value in MATLAB. The variable x is assigned a complex number with a real part of 2 and an imaginary part of 3:

```
x = 2 + 3i;
```



You can separate a complex number into its real and imaginary parts **using the real and imag functions**:

```
>> real(z)

ans =

    4.0736    4.5669    1.3925
    4.5290    3.1618    2.7344
    0.6349    0.4877    4.7875

>> imag(z)

ans =

   -7.7191   -7.6573   -1.1351
   -1.2609   -3.8830   -3.3741
   -7.7647   -6.4022   -7.3259
```

Infinity and NaN

MATLAB represents infinity by the special value inf. Infinity results from operations like division by zero and overflow

`x = 1/0`
`x =`
`Inf`
	`x = 1.e1000`
	`x =`
	`Inf`

`x = exp(1000)`
`x =`
`Inf`
	`x = log(0)`
	`x =`
	`-Inf`

- MATLAB represents values that are not real or complex numbers with a special value called NaN, which stands for Not a Number.
- Expressions like 0/0 and inf/inf result in NaN,

Chapter 3

Controlling the Flow

Chapter 3

Controlling the Flow

So far in these lab sessions, all the commands were executed in the Command Window. The problem is that the commands entered in the Command Window cannot be saved and executed again for several times. Therefore, a different way of executing repeatedly commands with MATLAB is:

1. to create a file with a list of commands,
2. save the file, and
3. run the file.

If needed, corrections or changes can be made to the commands in the file. The files that are used for this purpose are called script ¯les or scripts for short. This section covers the following topics:

- M-File Scripts
- M-File Functions

3.1 M-File Scripts

A *script file* is an external file that contains a sequence of MATLAB statements. Script files have a filename extension .m and are often called M-files. M-files can be *scripts* that simply execute a series of MATLAB statements, or they can be *functions* that can accept arguments and can produce one or more outputs.

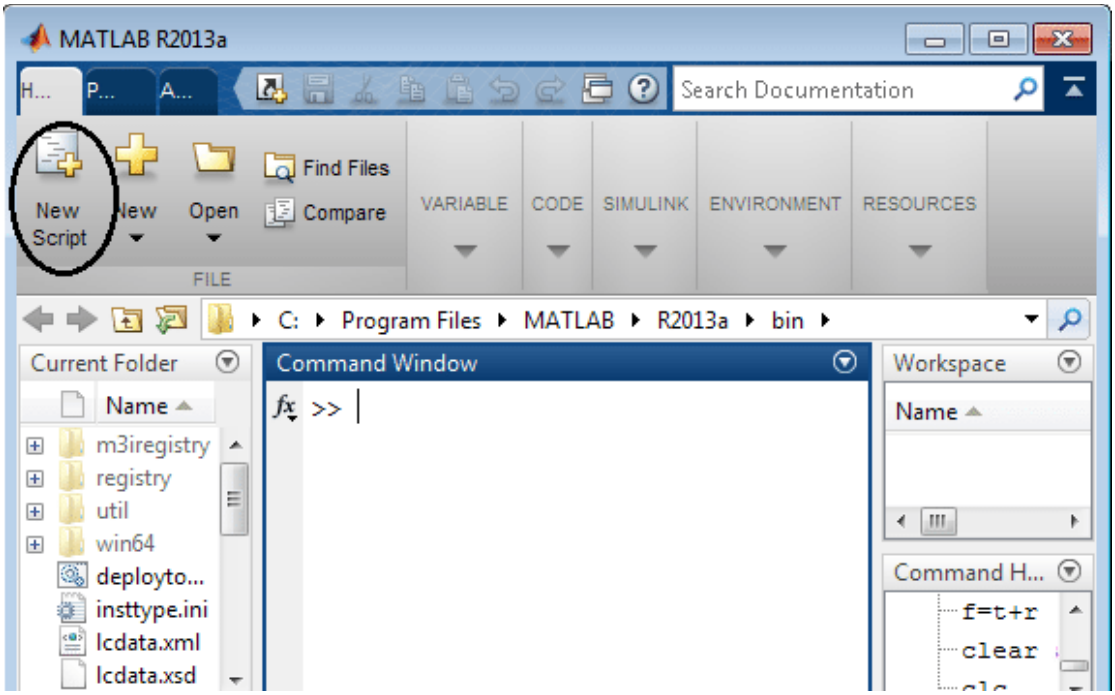
Matlab Scripts Advantages of M-files

- Easy editing and saving of work
- Undo changes

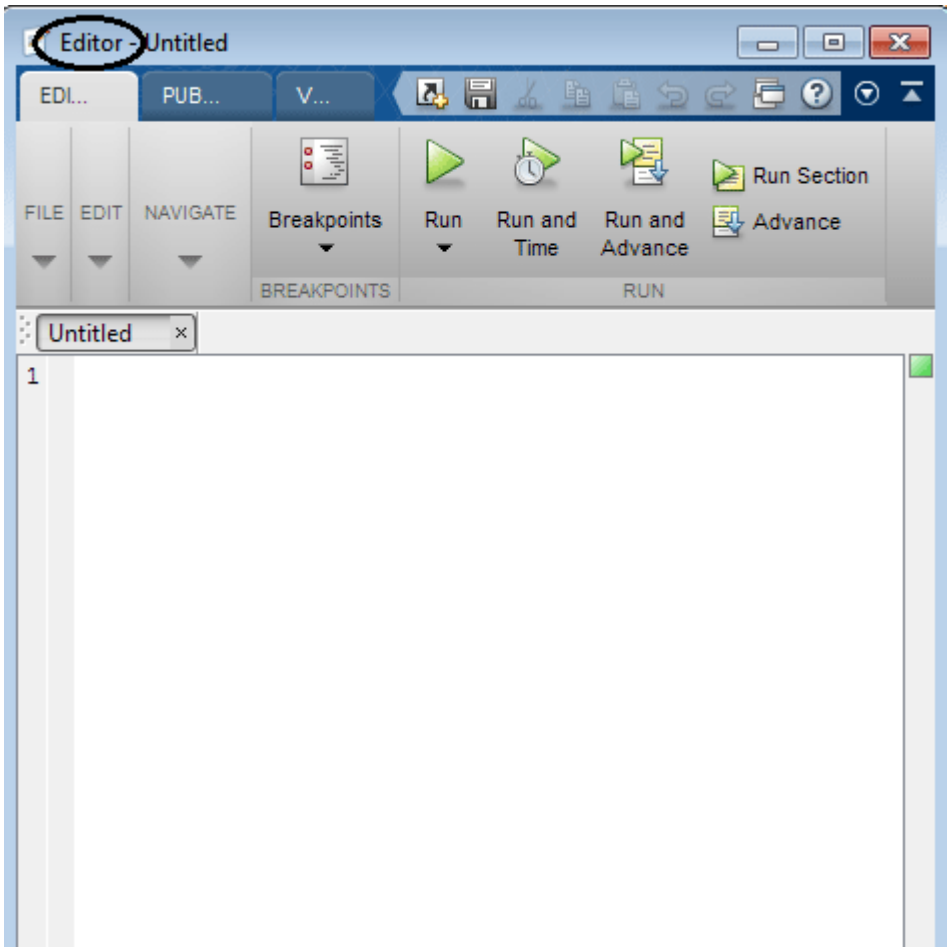
- non executable comments can be added using the '%' symbol to make commands easier to understand
- Saving M-files is far more memory efficient than saving a workspace

How to Create m File in MATLAB

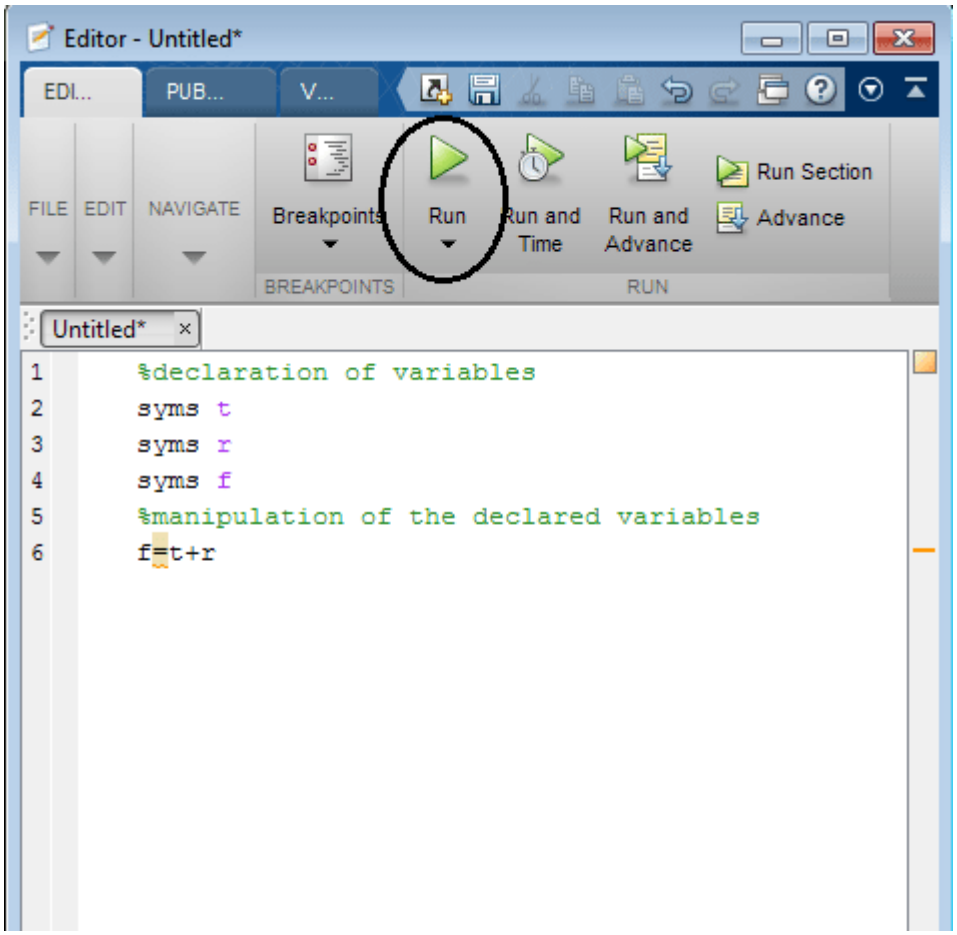
1.Go to the top left corner of the front window of the MATLAB and click on New Script.



2.A new window will be appeared on your screen right after clicking on the New Script.



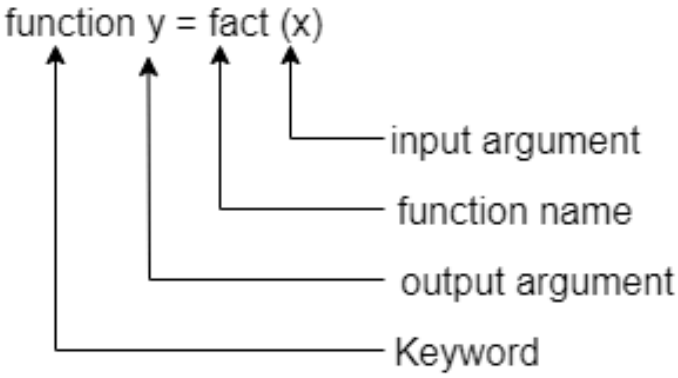
3.A very simple code written on the editor by declaring three different kinds of variables and their manipulation is shown in the figure below.



3.2 M-File functions

The general syntax of a Matlab function is as follows:

function [output variables] =function_name(input variables)



- Matlab functions must begin with the keyword "function"
- Matlab supports multiple output arguments (i.e. multiple "return" values), listed as shown in square brackets.
    - If a function only has a single output argument, then the square brackets are not required.
    - If a function does not have any output arguments, then neither the square brackets nor the equals sign that follows are used.
    - where function_name must be the same as the filename (without .m extension) in which the functions are written. For example, if the name of the functions is projectile, it must be written and stored in a file with the name projectile.m.

Example

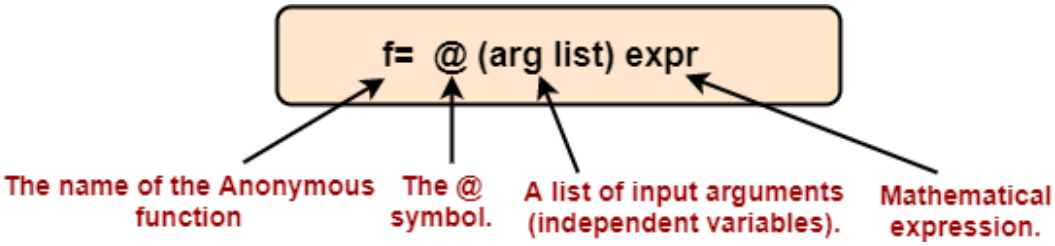
	Function Definition Line	**File Name**
1.	function [rho,H,F]=motion(x,y,t);	motion.m
2.	function [theta] = angleTH(x, y);	angleTH.m

3. function theta=THETA(x,y,z); THETA.m
4. function []=circle (r); circle.m
5. function circle (r); circle.m

Anonymous Function

An anonymous function is a simple (one-line) user-defined function that is defined without creating a separate function file (M-file). Anonymous functions can be defined in the Command Window, within a script file, or inside a user-defined function.

An anonymous function is generated by typing the following command:



For example, create an anonymous function that evaluates and return the area of a circle:

```
>> cirarea = @ (radius) pi * radius .^ 2;
>> cirarea(4)
ans =
     50.2655
>> cirarea(1:4)
ans =
    3.1416   12.5664   28.2743   50.2655
```

3.3 Control flow

MATLAB has four control °ow structures: the if statement, the for loop, and the while loop.

1. The if-else... end statement

if expression
Statements;
else
Statements;
end

Example:

```
% program to check the number is even or odd
a = randi(100,1);
if rem(a,2) == 0
   disp('an even number')
else
   disp('an odd number')
end
```

MATLAB provides different types of loops to handle looping requirements, including while loops, for loops, and nested loops.

Loop types	Description
While loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
For loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop

	variable
Nested loop	You can use one or more loops inside any another loop.

2. The while Loop

The syntax of a while loop in MATLAB is:

while (condition)
statements;
end

Example:

```
a = 10;
% while loop execution
 While (a < 20)
fprintf('value of a: %d\n', a);
 a = a + 1;
end
```

3. The for Loop

The syntax of a for loop in MATLAB is:

for index = values
statements;
end

values have one of the following forms:

Format	Description
initval:endval **for a = 10:20** **disp(a)** **end**	increments the index variable from initval to endval by 1, and repeats execution of program statements until index is greater than endval
initval:step:endval **for a = 1.0: -0.1: 0.0** **disp(a)** **end**	increments index by the value step on each iteration, or decrements when step is negative.
valArray **for a =[24,18,17,23,28]** **disp(a)** **end**	creates a column vector index from subsequent columns of array valArray on each iteration.

The Nested Loops

The syntax for a nested for loop statement in MATLAB is as follows:

```
for m = 1: j
for n = 1: k
statements;
end
```

3.4 Solving Equations

we discuss how to solve algebraic equations using MATLAB. We will discuss both linear and nonlinear algebraic equations. We will also discuss systems of simultaneous linear and nonlinear algebraic equations.

Types of Equations

- Linear equation
- Equation with one variable
- Equation with two variables
- Equation with three variables
- Quadratic equation
- Exponential equation

Linear Equation

A linear equation is an algebraic equation. In linear equation, each term is either a constant or the product of a constant and a single variable

General form of the linear equation with two variables is given below-:

$$y = mx + c$$

Equation with one Variable: An equation who have only one variable, e.g.

$$12\,x - 10 = 0$$

Equation with two Variables: An equation who have two variables, e.g.

$$x + 10y - 10 = 0$$

$$12x + 23y = 20$$

First, let us solve the following simple linear equation for the variable x:

$$2x - 3 = 0$$

We enter the coefficients of the above polynomial in a vector in MATLAB then use the roots command to find the root which is the solution to the above linear equation in x.

Example:

```
>> p = [2 -3]
p =
2     -3
>> roots(p)
ans =
1.5000
```

Example:

```
>> A = [1 -3 ; 4 6]
A =
1      -3
4       6
>> b = [5 ; 3]
b =
5
3
>> x = inv(A)*b
x =
2.1667
-0.9444
```

Solve System of Linear Equations

Solve System of Linear Equations Using linsolve

Solve System of Linear Equations Using solve

Example

$$2x+y+z=2$$

$$-x+y-z=3$$

$$x+2y+3z=-10$$

If you do not have the system of linear equations in the form AX = B ، use **equationsToMatrix** to convert the equations into this form.

```
syms x y z
eqn1 = 2*x + y + z == 2;
eqn2 = -x + y - z == 3;
eqn3 = x + 2*y + 3*z == -10;
[A,B] = equationsToMatrix([eqn1, eqn2, eqn3], [x, y, z])
A =
[  2, 1,  1]
[ -1, 1, -1]
[  1, 2,  3]

B =
   2
   3
 -10
```

```
X = linsolve(A,B)
X =
  3
  1
 -5
```

From X, x = 3, y = 1 and z = -5.

Solve System of Linear Equations Using solve

$2x+y+z=2$

$-x+y-z=3$

$x+2y+3z=-10$

```
syms x y z
eqn1 = 2*x + y + z == 2;
eqn2 = -x + y - z == 3;
eqn3 = x + 2*y + 3*z == -10;
sol = solve([eqn1, eqn2, eqn3], [x, y, z]);
```

Quadratic equation

It is the second degree equation in which one variable contains the variable with an exponent of 2. Its general form is

For $ax^2+bx+c=0$ where $a\neq 0$:

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Create The m-file

```
function x = rqe(a,b,c)
x(1) = (-b + sqrt(b^2 - 4 * a * c))/(2*a);
x(2) = (-b - sqrt(b^2 - 4 * a * c))/(2*a);
end
```

If we want to compute the roots of the following expression:

$$2x^2 + x - 1 = 0$$

We can call our function (first code) like this:

```
x = rqe(2, 1, -1)
```

and we get from Matlab:

```
x =     0.5000   -1.0000
```

Example:

let us show similarly how to find the solution of a quadratic equation in x by solving the following quadratic equation:

$$5x^2 + 3x - 4 = 0$$

Here are the needed MATLAB commands:

```
>> p = [5 3 -4]
p =
5        3        -4
>> roots(p)
ans =
-1.2434
0.6434
```

Example:

Consider the following simple system of two linear algebraic equations in the variables x and y:

$$x - 3y = 5$$

$$4x + 6y = 3$$

To solve the above system of equations, we need to re-write the system as a matrix equation. The coefficients of x and y on the left-hand-side of the equations are considered as the elements of a square matrix.

```
>> A = [1 -3 ; 4 6]
A =
1      -3
4       6
>> b = [5 ; 3]
b =
5
3
>> x = inv(A)*b
x =
2.1667
-0.9444
```

Fibonacci Numbers

In mathematics, Fibonacci numbers are the numbers in the following sequence: 0, 1, 1, 2 ,3, 5, 8, 13, 21, 34, 55, 89, 144, …

• By definition, the first two Fibonacci numbers are 0 and 1, and each subsequent number is the sum of the previous two.

$$f_n = f_{n-1} + f_{n-2}$$

- with seed values:

$$f_0 = 0, f_1 = 1$$

```
function f = fibonacci(N)

f=zeros(N,1);
f(1)=0;
f(2)=1;

for k=3:N
   f(k)=f(k-1)+f(k-2);
end
```

Chapter 4

String

Strings in Matlab are vectors of characters always use single quotes to define strings

```
>> name = 'Jonas'
name =
Jonas
>> name(1)
ans =
J
>> name(1:3)
ans =
Jon
```

Working with strings

str = "Hello, world"

str =

"Hello, world"

As an alternative, you can convert a character vector to a string using the string function. chr is a 1-by-17 character vector. str is a 1-by-1 string that has the same text as the character vector.

```
chr = 'Greetings, friend'

chr =
'Greetings, friend'

str = string(chr)

str =
"Greetings, friend"
```

Create a string array containing multiple strings using the [] operator. str is a 2-by-3 string array that contains six strings.

```
str = ["Mercury","Gemini","Apollo";
      "Skylab","Skylab B","ISS"]

str = 2x3 string
   "Mercury"    "Gemini"     "Apollo"
   "Skylab"     "Skylab B"   "ISS"
```

Find the length of each string in str with the strlength function. Use strlength, not length, to determine the number of characters in strings.

```
L = strlength(str)

L = 2×3

    7     6     6
    6     8     3
```

As an alternative, you can convert a cell array of character vectors to a string array using the string function. MATLAB displays strings in string arrays with double quotes, and displays characters vectors in cell arrays with single quotes.

```
C = {'Mercury','Venus','Earth'}

C = 1x3 cell
    {'Mercury'}    {'Venus'}    {'Earth'}


str = string(C)

str = 1x3 string
    "Mercury"    "Venus"    "Earth"
```

In addition to character vectors, you can convert numeric, date, time duration, and categorical values to strings using the string function.

Convert a numeric array to a string array.

```
X = [5 10 20 3.1416];
string(X)

ans = 1x4 string
    "5"    "10"    "20"    "3.1416"
```

Convert a datetime value to a string.

```
d = datetime('now');
string(d)
```

```
ans =
"01-Sep-2021 10:20:02"
```

Also, you can read text from files into string arrays using the readtable, textscan, and fscanf functions.

Create Empty and Missing Strings

String arrays can contain both empty and missing values. An empty string contains zero characters. When you display an empty string, the result is a pair of double quotes with nothing between them (""). The missing string is the string equivalent to NaN for numeric arrays. It indicates where a string array has missing values. When you display a missing string, the result is <missing>, with no quotation marks.

Create an empty string array using the strings function. When you call strings with no arguments, it returns an empty string. Note that the size of str is 1-by-1, not 0-by-0. However, str contains zero characters.

```
str = strings
str =
""
```

Create an empty character vector using single quotes. Note that the size of chr is 0-by-0.

```
chr = ''
chr =
  0x0 empty char array
```

Create a string array where every element is an empty string. You can preallocate a string array with the strings function.

str = strings(2,3)

str = *2x3 string*

"" "" ""

"" "" ""

Creating Character Arrays

Specify character data by placing characters inside a pair of single quotes. For example, this line creates a 1-by-13-character array called name:

name = 'Thomas R. Lee';

You can also join two or more-character arrays together to create a new character array

name = 'Thomas R. Lee';

title = ' Sr. Developer';

strcat(name, ',' ,title)

ans = Thomas R. Lee, Sr. Developer

To concatenate strings vertically, use strvcat

When creating a two-dimensional character array, be sure that each row has the same length.

For example, this line is legal because both input rows have exactly 13 characters:

name = ['Thomas R. Lee' ; 'Sr. Developer’]

name =

Thomas R. Lee

Sr. Developer

When creating character arrays from strings of different lengths, you can pad the shorter strings with blanks to force rows of equal length :

name = ['Thomas R. Lee ; 'Senior Developer'];

Comparing Strings for Equality

- strcmp determines if two strings are identical.
- strncmp determines if the first n characters of two strings are identical.
- strcmpi and strncmpi are the same as strcmp and strncmp, except that they ignore case.

Consider the two strings str1 = 'hello'; str2 = 'help‘’

C = strcmp(str1,str2) C = 0

The first three characters of str1 and str2 are identical, so invoking strncmp with any value up to 3 returns 1:

C = strncmp(str1, str2, 2)

C = 1

These functions work cell-by-cell on a cell array of strings. Consider the two cell arrays of strings

A = {'pizza'; 'chips'; 'candy'};

B = {'pizza'; 'chocolate'; 'pretzels'}; Now apply the string comparison functions:

strcmp(A,B)

ans =

1

0

0

Categorizing Characters within a String

There are three functions for categorizing characters inside a string:

• isletter determines if a character is a letter

• isspace determines if a character is white space (blank, tab, or new line)

• isstrprop checks characters in a string to see if they match a category, you specify

mystring = 'Room 401';

A = isletter(mystring)

1 1 1 1 0 0 0 0

chr = '123 Main St.'

TF = isspace(chr)

TF = 1x12 logical array

0 0 0 1 0 0 0 0 1 0 0 0

- TF = isstrprop(str, category)

determines if characters in the input text are of the specified category, such as letters, numbers, or whitespace.

isstrprop('ABC123','alpha') returns a

[1 1 1 0 0 0]

Searching and Replacing

strrep function performs the standard search-and-replace operation.

label = 'Sample 1, 10/28/95’

newlabel = strrep(label, '28', '30)

newlabel = Sample 1, 10/30/95

findstr returns the starting position of a substring within a longer string

label = 'Sample 1, 10/28/95’;

position = findstr('amp', label)

position = 2

Converting from a Character Equivalent

- name = 'Thomas R. Lee’;
- name = double(name)

name = 84 104 111 109 97 115 32 82 46 32 76 101 101

name = char(name)

name = Thomas R. Lee

Types of Date Formats

Date Format	Example
Date string	02-Oct-1996
Serial date number	729300
Date vector	1996 10 2 0 0 0

Conversions between Date Formats

Function	Description
datenum	Convert a date string to a serial date number.
datestr	Convert a serial date number to a date string.
datevec	Split a date number or date string into individual date elements.

Formatting strings

- disp() is almost the same as above, except it does not print out the variable name
- fprintf() is for formatting text and printing out to a file or other device, such as the workspace
- sprintf() is for formatting text in order to create new string variables

Example

```
>> employee = 'Fred';
>> age = 32;
>> score = 88.432;
>> fprintf('Employee: %s is %d years old and scored %f',employee,age,score);
Employee: Fred is 32 years old and scored 88.432000>>
```

%s	string
%d	integer/digit
%i	integer/digit
%f	floating point number
%c	single character

```
>> fprintf('%s\t%d\n',employee,age)
```

```
Fred    32
```

There are many special characters to control formatting that begin with the backslash:

\t	tab
\n	newline
\v	vertical tab

Example

```
>> fprintf('Score: %f\n',score);
Score: 88.432000
>> fprintf('Score: %.2f\n',score);
Score: 88.43
>> fprintf('Score: %.0f\n',score);
Score: 88
>> fprintf('Score: %.5f\n',score);
Score: 88.43200
```

Specifies the number of decimal places in a floating point number

Reverse Strings

- str = ["airport", "control tower", "radar","runway"]
- newStr = reverse(str)

  "tropria" "rewot lortnoc" "radar" "yawnur"
- tf = (newStr == str)

  0 0 1 0

strlength – Lengths of strings

str = "Hello, World"

L = strlength(str)

L = 12

insertBefore – Insert strings before specified substrings

newStr = insertBefore(str, endPos, newText) inserts the text specified by newText into str before the position specified by endPos.

```
str = ["The quick fox jumps"; "over the dog"]

newStr = insertBefore(str,[" fox";" dog"],[" brown";" lazy"])

newStr =

    "The quick brown fox jumps"

    "over the lazy dog"
```

Insert Substring before Position

```
str = "James Maxwell"

newStr = insertBefore(str,7,"Clerk ")

newStr =

"James Clerk Maxwell"
```

insertAfter – Insert strings after specified substrings

newStr = insertAfter(str,startStr,newText) inserts newText into str after the substring specified by startStr and returns the result as newStr.

```
str = "The quick fox"

newStr = insertAfter(str,"quick"," brown")

newStr =

"The quick brown fox"
```

Insert Text after Position in Character Vector

```
chr = 'mushrooms and onions'
```

newChr = insertAfter(chr,9,', peppers,')

newChr =

'mushrooms, peppers, and onions'

Structures

Structures can be used to organize and group information

```
>> patient.name = 'John Doe';
>> patient.billing = 127.00;
>> patient.test = [79, 75, 73; 180, 178, 177.5; 220, 210, 205];
>> patient
patient =
      name: 'John Doe'
   billing: 127
      test: [3x3 double]
```

Saving variables

- save() to save the workspace to disk
- load() to load a .mat file that contains variables from disk

- clear() to remove a variable from memory
- who, whos to list variables in memory

```
>> who

Your variables are:

age      employee  mycell    patient   score

>> whos
  Name         Size          Bytes  Class     Attributes

  age          1x1               8  double
  employee     1x4               8  char
  mycell       1x2             304  cell
  patient      1x2            1056  struct
  score        1x1               8  double
```

Exercises

Define

A = 'Hello Peter'; B = 'The year is: '; C = 2010; D = 1.3333;

Using only the variables A-D and the **sprintf()** function make the following text display on the screen:

Hello Peter
The year is: 2011
1.33%
001.3

```
sprintf('%s \n%s%d \n%.2f %% \n00%.1f \n ',A,B,C+1,D,D)
```

Exercises

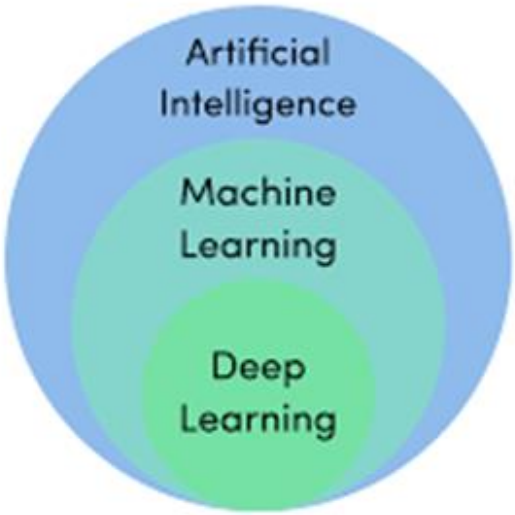
- Define
- >> s1 = 'I love Matlab'
- >> s2 = [true false]
- >> s3 = [10 13 NaN Inf]
- Combine arrays s1, s2, s3 into a 1x3 (row) cell array, K `cellRow = {s1,s2,s3};`
- Combine arrays s1, s2, s3 into a 3x1 (column) cell array, L `cellCol = {s1;s2;s3};`
- Expand the cell array K to be of size 1x10 by adding the numeric array [100 200 300; 7 8 9] as the 10th element of K
```
x= [100 200 300;7 8 9];
cellRow{10} =x;
```
- Remove element 5 of the cell array K.
```
cellRow(5) = [];
```
- Confirm K is now of size 1x9 `size(cellRow)`

Chapter 5: Transfer Learning for Deep Learning

What is deep learning?

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks.

Deep learning is a specialized subset of machine learning which, in turn, is a subset of artificial intelligence.



What's the difference between DL and ML?

Traditional machine learning algorithms have a rather simple structure, such as linear regression or a decision tree, deep learning is based on an artificial neural network

The most important difference between deep learning and traditional machine learning is its performance as the scale of data increases. When the data is small, deep learning algorithms don’t perform that well. This is because deep learning algorithms need a large amount of data to understand it perfectly. On the other hand, traditional machine learning algorithms with their handcrafted rules prevail in this scenario. Below image summarizes this fact.

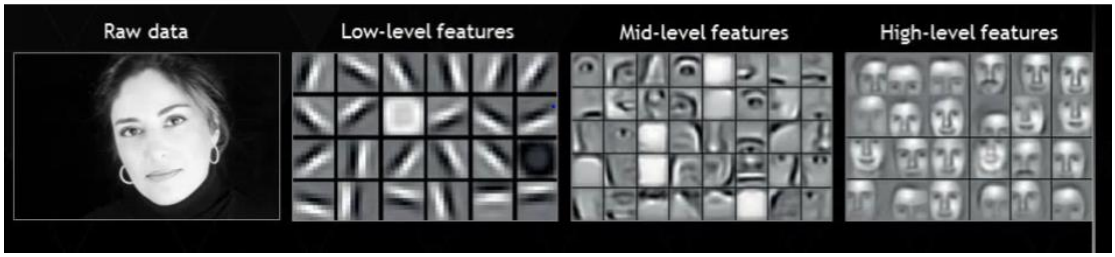
Deep learning algorithms heavily depend on high-end machines, contrary to traditional machine learning algorithms, which can work on low-end machines. This is because the requirements of deep learning algorithm include GPUs which are an integral part of its working. Deep learning algorithms inherently do a large amount of matrix multiplication operations. These operations can be efficiently optimized using a GPU because GPU is built for this purpose.

Feature engineering is a process of putting domain knowledge into the creation of feature extractors to reduce the complexity of the data and make patterns more visible to learning algorithms to work. This process is difficult and expensive in terms of time and expertise.

In Machine learning, most of the applied features need to be identified by an expert and then hand-coded as per the domain and data type.

For example, features can be pixel values, shape, textures, position and orientation. The performance of most of the Machine Learning algorithm depends on how accurately the features are identified and extracted.

Deep learning algorithms try to learn high-level features from data. This is a very distinctive part of Deep Learning and a major step ahead of traditional Machine Learning. Therefore, deep learning reduces the task of developing new feature extractor for every problem. Like, Convolutional NN will try to learn low-level features such as edges and lines in early layers then parts of faces of people and then high-level representation of a face.

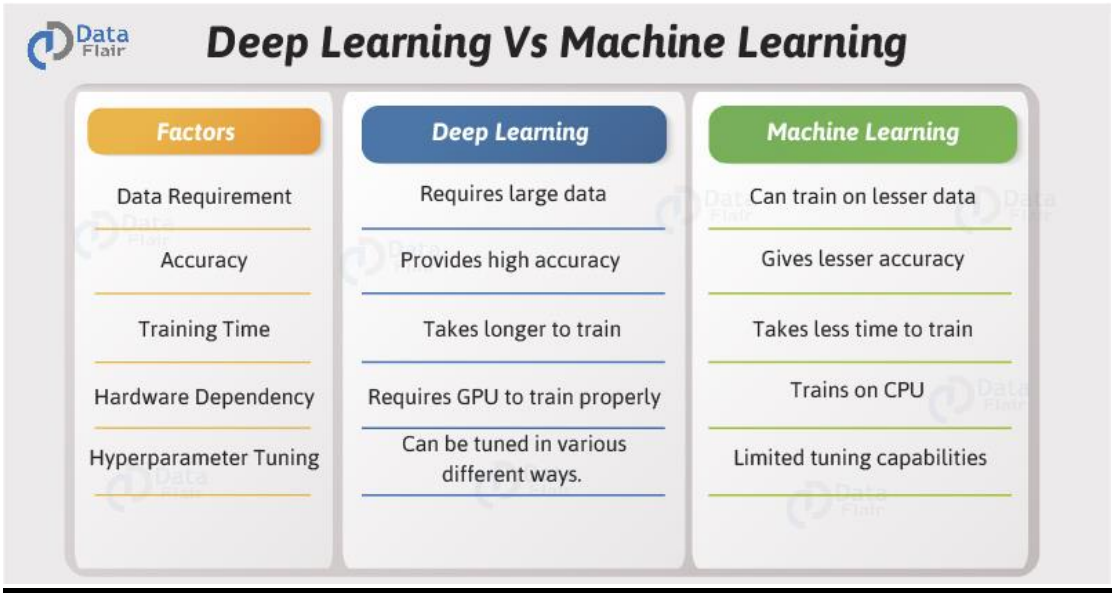


Usually, a deep learning algorithm takes a long time to train. This is because there are so many parameters in a deep learning algorithm that training them takes longer than usual. State of the art deep learning algorithm ResNet takes about two weeks to train completely from

scratch. Whereas machine learning comparatively takes much less time to train, ranging from a few seconds to a few hours.

This is turn is completely reversed on testing time. At test time, deep learning algorithm takes much less time to run. Whereas, if you compare it with k-nearest neighbors (a type of machine learning algorithm), test time increases on increasing the size of data. Although this is not applicable on all machine learning algorithms, as some of them have small testing times too.

- Computer Vision: for applications like vehicle number plate identification and facial recognition.
- Information Retrieval: for applications like search engines, both text search, and image search.
- Marketing: for applications like automated email marketing, target identification
- Medical Diagnosis: for applications like cancer identification, anomaly detection
- Natural Language Processing: for applications like sentiment analysis, photo tagging



Some Real Applications

Facebook

it has had great success with identifying faces in photographs by using Deep Learning

Information Retrieval

We use ML and DL for applications like search engines, both text search, and image search.

Medical Diagnosis

It has very wide usage in the medical field also. Applications like cancer identification, anomaly detection

Marketing

In the field of Marketing, DL based time-series forecasting are being used for predicting sales.

Types of deep learning algorithms

- **Convolutional Neural Networks (CNNs)**
- Long Short Term Memory Networks (LSTMs)
- Recurrent Neural Networks (RNNs)
- Generative Adversarial Networks (GANs)
- Radial Basis Function Networks (RBFNs)
- Multilayer Perceptrons (MLPs)
- Self Organizing Maps (SOMs)
- Deep Belief Networks (DBNs)

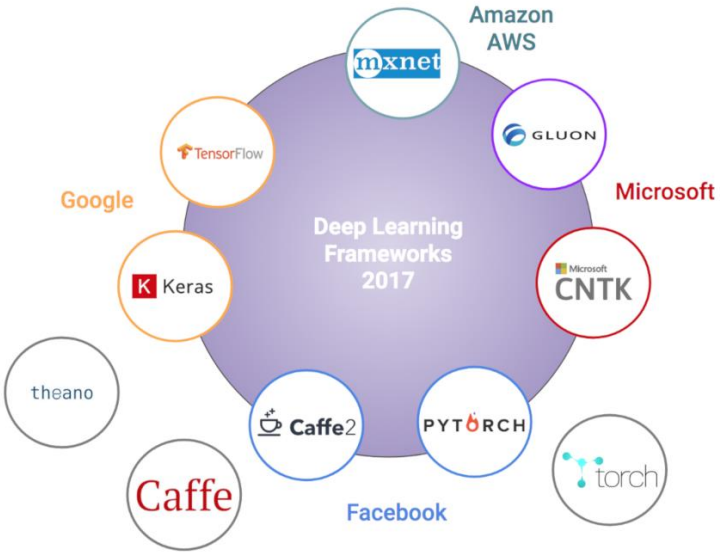
Frameworks

1.**Tenserflow**

It is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

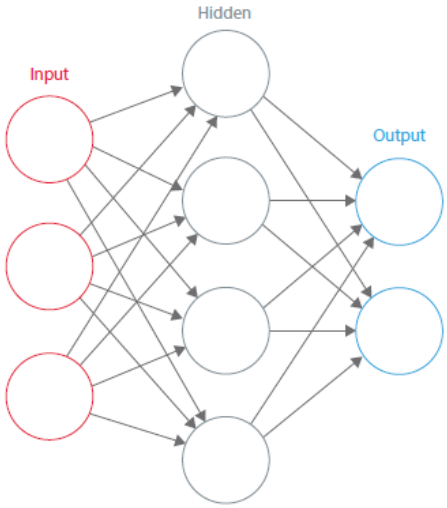
2.Coffe

It aims to provide an easy and straightforward way for you to experiment with deep learning and leverage community contributions of new models and algorithms.

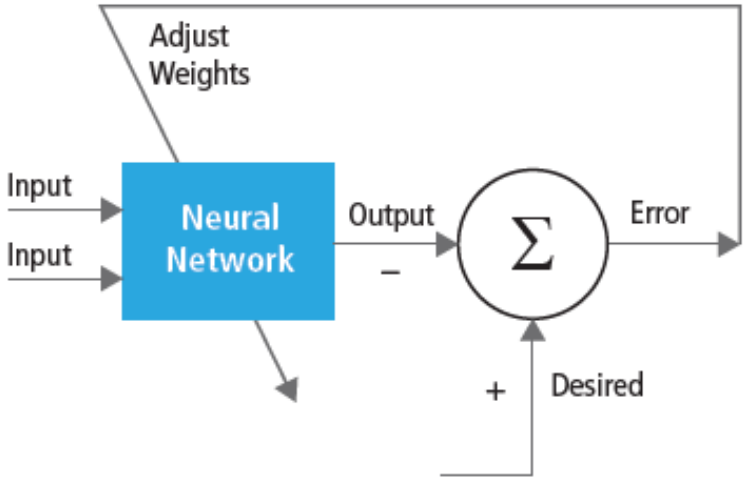


Convolutional Neural Networks

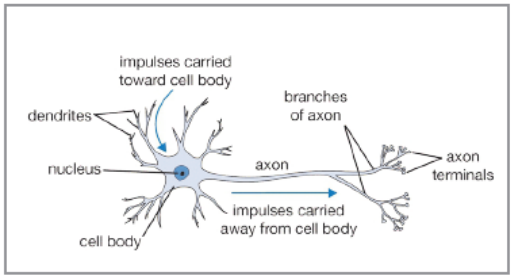
A neural network is a system of interconnected artificial “neurons” that exchange messages between each other. The connections have numeric weights that are tuned during the training process, so that a properly trained network will respond correctly when presented with an image or pattern to recognize. The network consists of multiple layers of feature-detecting “neurons”. Each layer has many neurons that respond to different combinations of inputs from the previous layers. The layers are built up so that the first layer detects a set of primitive patterns in the input, the second layer detects patterns of patterns, the third layer detects patterns of those patterns, and so on. Typical CNNs use 5 to 25 distinct layers of pattern recognition.

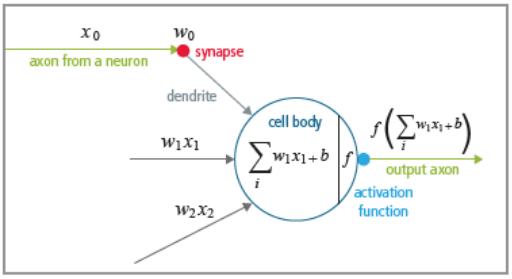


Training is performed using a “labeled” dataset of inputs in a wide assortment of representative input patterns that are tagged with their intended output response. Training uses general-purpose methods to iteratively determine the weights for intermediate and final feature neurons. The following figure demonstrates the training process at a block level



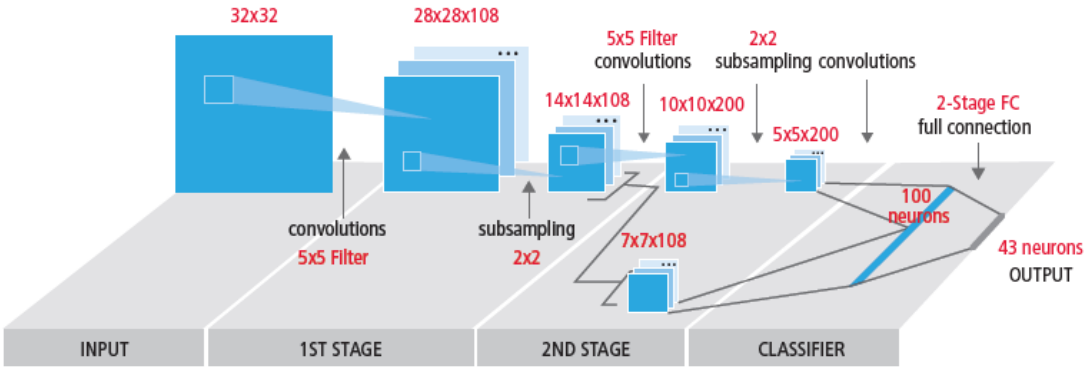
Neural networks are inspired by biological neural systems. The basic computational unit of the brain is a neuron and they are connected with synapses





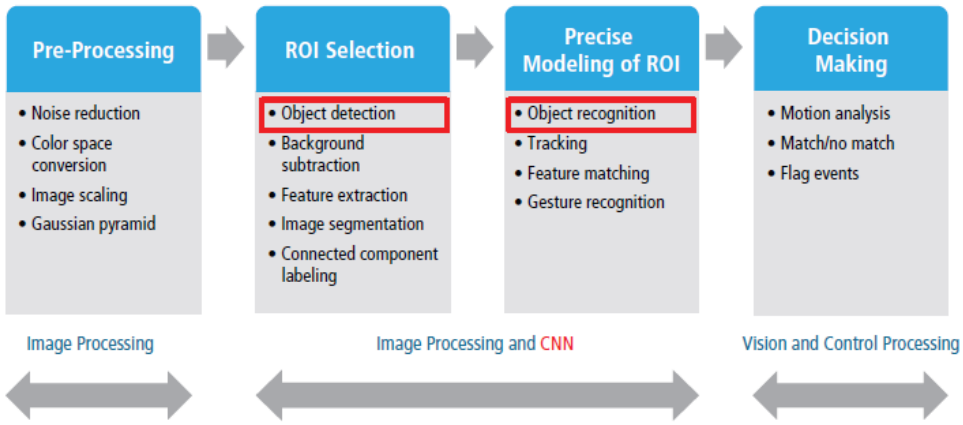
In the traditional model of pattern/image recognition, a hand-designed feature extractor gathers relevant information from the input and eliminates irrelevant variabilities. The extractor is followed by a trainable classifier, a standard neural network that classifies feature vectors into classes.

In a CNN, convolution layers play the role of feature extractor. But they are not hand designed. Convolution filter kernel weights are decided on as part of the training process. Convolutional layers are able to extract the local features because they restrict the receptive fields of the hidden layers to be local.



The following figure shows a typical vision algorithm pipeline, which consists of four stages: pre-processing the image, detecting regions of interest (ROI) that contain likely objects, object recognition, and vision decision making. The pre-processing step is usually dependent on the details of the input, especially the camera system, and is often implemented in a hardwired unit outside the vision subsystem. The

decision making at the end of pipeline typically operates on recognized objects—It may make complex decisions, but it operates on much less data, so these decisions are not usually computationally hard or memory-intensive problems. The big challenge is in the object detection and recognition stages, where CNNs are now having a wide impact.

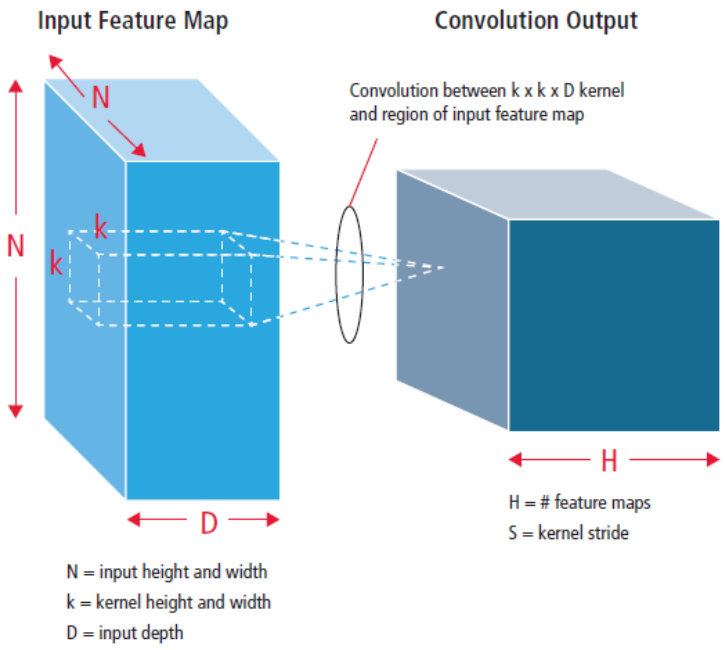


Layers of CNNs

Convolution layers

The convolution operation extracts different features of the input. The first convolution layer extracts low-level features like edges, lines, and corners. Higher-level layers extract higher-level features. The following Figure illustrates the process of 3D convolution used in CNNs. The input is of size N x N x D and is convolved with H kernels, each of size k x k x D separately. Convolution of an input with one kernel produces one output feature, and with H kernels independently produces H features. Starting from top-left corner of the input, each kernel is moved from left

to right, one element at a time. Once the top-right corner is reached, the kernel is moved one element in a downward direction, and again the kernel is moved from left to right, one element at a time. This process is repeated until the kernel reaches the bottom-right corner. For the case when N = 32 and k = 5 , there are 28 unique positions from left to right and 28 unique positions from top to bottom that the kernel can take. Corresponding to these positions, each feature in the output will contain 28x28 (i.e., (N-k+1) x (N-k+1)) elements. For each position of the kernel in a sliding window process, k x k x D elements of input and k x k x D elements of kernel are element-byelement multiplied and accumulated. So to create one element of one output feature, k x k x D multiply-accumulate operations are required.

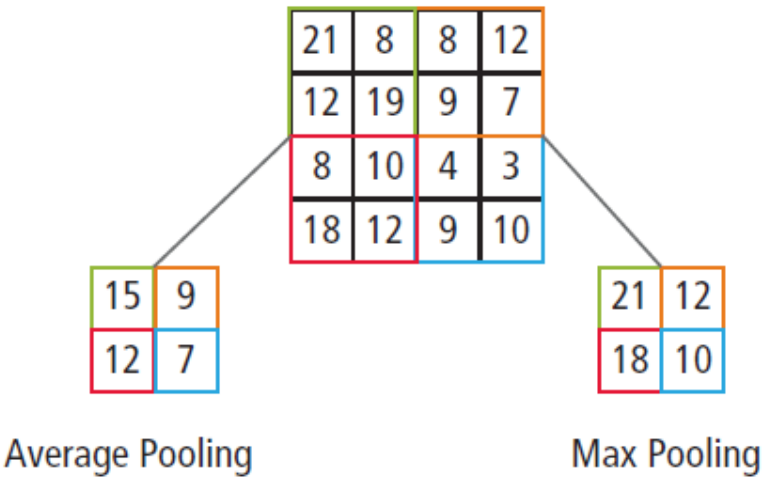


Pooling/subsampling layers

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.The input is of size 4x4. For 2x2 subsampling, a 4x4 image is divided into four non-overlapping matrices of size 2x2. In the case of max pooling, the maximum value of the four values in the 2x2 matrix is the output. In case of average pooling, the average of the four values is the output.



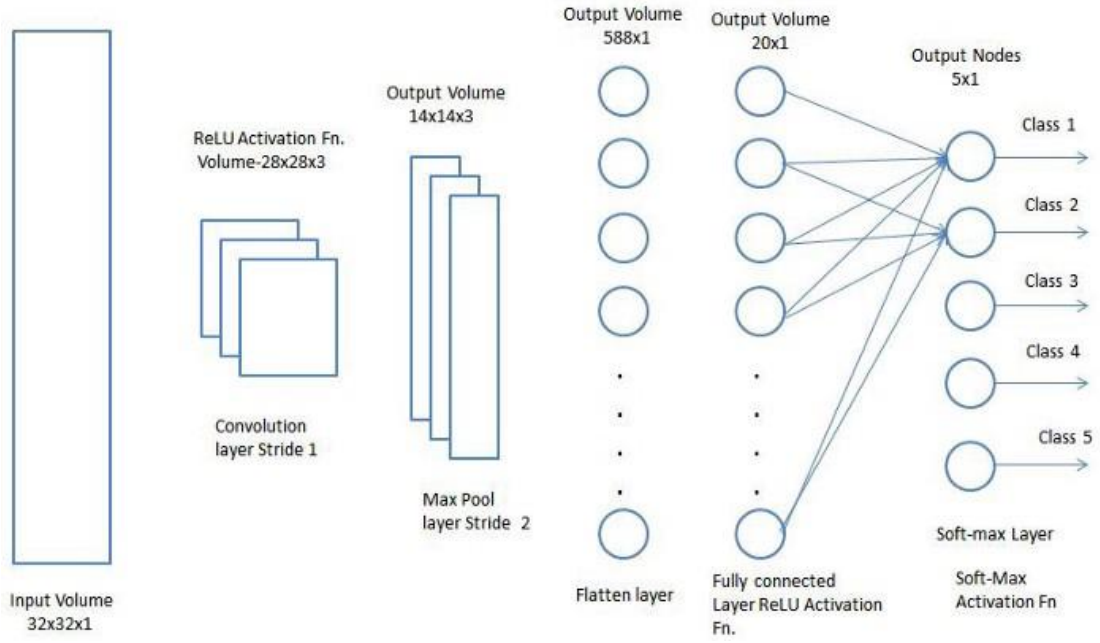
Fully connected layers

Fully connected layers are often used as the final layers of a CNN. These layers mathematically sum a weighting of the previous layer of features, indicating the precise mix of "ingredients" to determine a specific target output result. In case of a fully connected layer, all the elements of all the features of the previous layer get used in the calculation of each element of each output feature.

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and back propagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level

features in images and classify them using the Softmax Classification technique.



There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future. Some of them have been listed below:

1. LeNet
2. AlexNet
3. VGGNet

4. GoogLeNet
5. ResNet
6. ZFNet

AlexNet

AlexNet is the name of a convolutional neural network which has had a large impact on the field of machine learning, specifically in the application of deep learning to machine vision. It famously won the 2012 ImageNet LSVRC-2012 competition by a large margin (15.3% VS 26.2% (second place) error rates). The network had a very similar architecture as LeNet by Yann LeCun et al but was deeper, with more filters per layer, and with stacked convolutional layers. It consisted of 11×11, 5×5,3×3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum. It attached ReLU activations after every convolutional and fully-connected layer. AlexNet was trained for 6 days simultaneously on two Nvidia Geforce GTX 580 GPUs which is the reason for why their network is split into two pipelines.

Key Points

- Relu activation function is used instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
- Use dropout instead of regularisation to deal with overfitting. However, the training time is doubled with the dropout rate of 0.5.

- Overlap pooling to reduce the size of the network. It reduces the top-1 and top-5 error rates by 0.4% and 0.3%, respectively.

DataSet

ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories. The images were collected from the web and labeled by human labelers using Amazon's Mechanical Turk crowd-sourcing tool. Starting in 2010, as part of the Pascal Visual Object Challenge, an annual competition called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held. ILSVRC uses a subset of ImageNet with roughly 1000 images in each of 1000 categories. In all, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images. ImageNet consists of variable-resolution images. Therefore, the images have been down-sampled to a fixed resolution of 256×256. Given a rectangular image, the image is rescaled and cropped out the central 256×256 patch from the resulting image.

The Architecture

The architecture depicted in Figure 1, the AlexNet contains eight layers with weights; the first five are convolutional and the remaining three are fully connected. The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels. The network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average across training cases of the log-probability of the correct label under the prediction

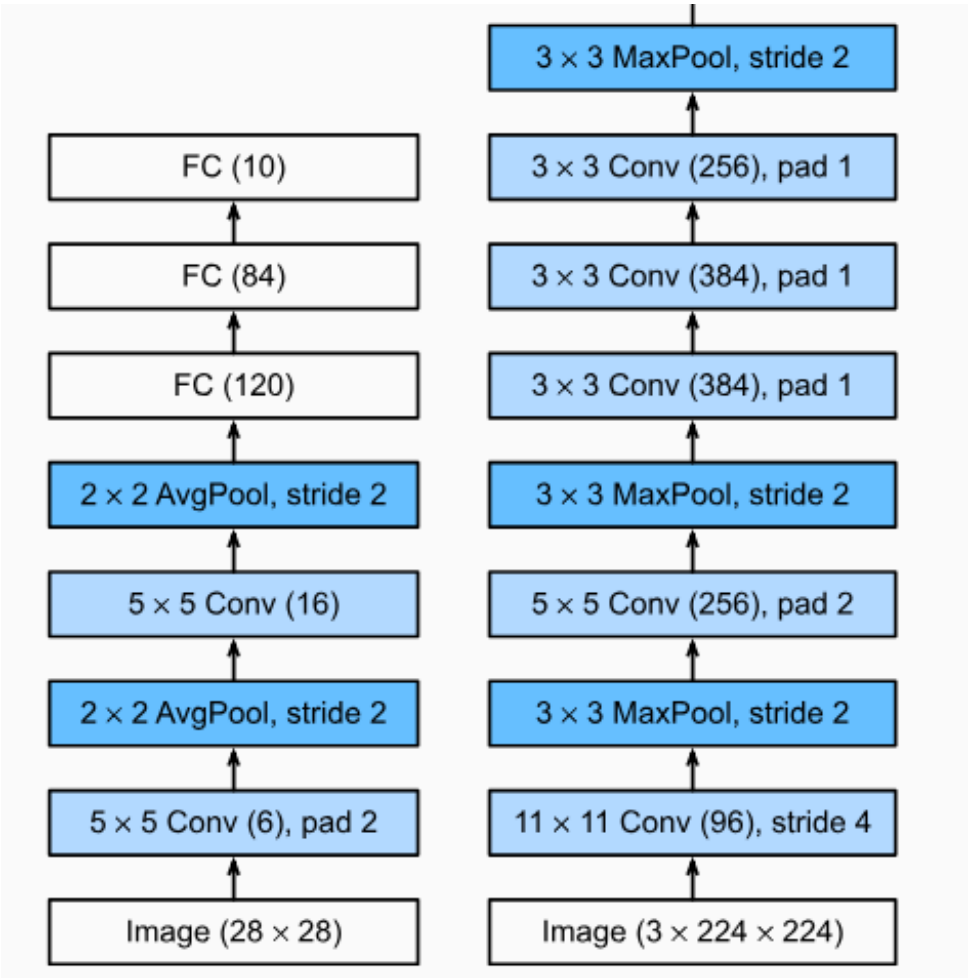
distribution. The kernels of the second, fourth, and fifth convolutional layers are connected only to those kernel maps in the previous layer which reside on the same GPU. The kernels of the third convolutional layer are connected to all kernel maps in the second layer. The neurons in the fully-connected layers are connected to all neurons in the previous layer.

Activation Functions

Besides, AlexNet changed the sigmoid activation function to a simpler ReLU activation function. On one hand, the computation of the ReLU activation function is simpler. For example, it does not have the exponentiation operation found in the sigmoid activation function. On the other hand, the ReLU activation function makes model training easier when using different parameter initialization methods. This is because, when the output of the sigmoid activation function is very close to 0 or 1, the gradient of these regions is almost 0, so that backpropagation cannot continue to update some of the model parameters. In contrast, the gradient of the ReLU activation function in the positive interval is always 1. Therefore, if the model parameters are not properly initialized, the sigmoid function may obtain a gradient of almost 0 in the positive interval, so that the model cannot be effectively trained.

In short, AlexNet contains 5 convolutional layers and 3 fully connected layers. Relu is applied after very convolutional and fully connected layer. Dropout is applied before the first and the second fully connected year. The network has 62.3 million parameters and needs 1.1 billion computation units in a forward pass. We can also see convolution layers,

which accounts for 6% of all the parameters, consumes 95% of the computation.



Training

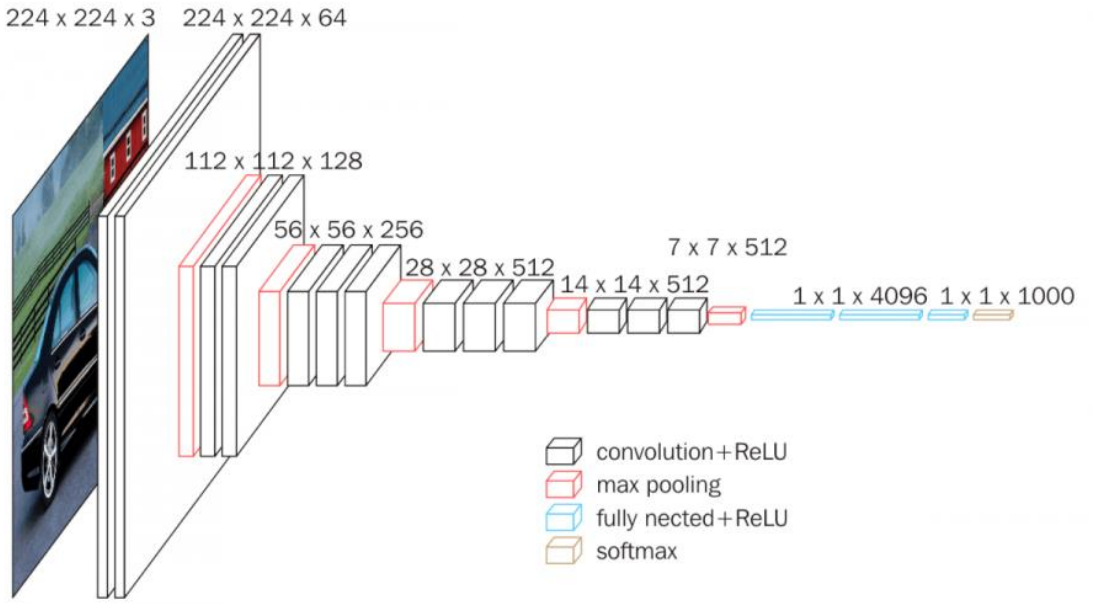
AlexNet takes 90 epochs which were trained for 6 days simultaneously on two Nvidia Geforce GTX 580 GPUs which is the reason for why their network is split into two pipelines. SGD with learning rate 0.01, momentum 0.9 and weight decay 0.0005 is used. Learning rate is divided by 10 once the accuracy plateaus.

VGG Network

What is VGG?

VGG stands for Visual Geometry Group; it is a standard deep Convolutional Neural Network (CNN) architecture with multiple layers. The "deep" refers to the number of layers with VGG-16 or VGG-19 consisting of 16 and 19 convolutional layers.

The VGG architecture is the basis of ground-breaking object recognition models. Developed as a deep neural network, the VGGNet also surpasses baselines on many tasks and datasets beyond ImageNet. Moreover, it is now still one of the most popular image recognition architectures.



The VGG network is constructed with very small convolutional filters. The VGG-16 consists of 13 convolutional layers and three fully connected layers.

Let's take a brief look at the architecture of VGG:

- **Input:** The VGGNet takes in an image input size of 224×224. For the ImageNet competition, the creators of the model cropped out the center 224×224 patch in each image to keep the input size of the image consistent.
- **Convolutional Layers**: VGG's convolutional layers leverage a minimal receptive field, i.e., 3×3, the smallest possible size that still captures up/down and left/right. Moreover, there are also 1×1 convolution filters acting as a linear transformation of the input. This is followed by a ReLU unit, which is a huge innovation from AlexNet that reduces training time. ReLU stands for rectified linear unit activation function; it is a piecewise linear function that will output the input if positive; otherwise, the output is zero. The convolution stride is fixed at 1 pixel to keep the spatial resolution preserved after convolution (stride is the number of pixel shifts over the input matrix).
- **Hidden Layers:** All the hidden layers in the VGG network use ReLU. VGG does not usually leverage Local Response Normalization (LRN) as it increases memory consumption and training time. Moreover, it makes no improvements to overall accuracy.

- **Fully-Connected Layers:** The VGGNet has three fully connected layers. Out of the three layers, the first two have 4096 channels each, and the third has 1000 channels, 1 for each class.

What is VGG16?

The VGG model, or VGGNet, that supports 16 layers is also referred to as VGG16, which is a convolutional neural network model proposed by A. Zisserman and K. Simonyan from the University of Oxford. These researchers published their model in the research paper titled, Very Deep Convolutional Networks for Large-Scale Image Recognition

The VGG16 model achieves almost 92.7% top-5 test accuracy in ImageNet. ImageNet is a dataset consisting of more than 14 million images belonging to nearly 1000 classes. Moreover, it was one of the most popular models submitted to ILSVRC-2014. It replaces the large kernel-sized filters with several 3×3 kernel-sized filters one after the other, thereby making significant improvements over AlexNet. The VGG16 model was trained using Nvidia Titan Black GPUs for multiple weeks.

As mentioned above, the VGGNet-16 supports 16 layers and can classify images into 1000 object categories, including keyboard, animals, pencil, mouse, etc. Additionally, the model has an image input size of 224-by-224.

The number 16 in the name VGG refers to the fact that it is 16 layers deep neural network (VGGnet). This means that VGG16 is a pretty extensive network and has a total of around 138 million parameters. Even

according to modern standards, it is a huge network. However, VGGNet16 architecture's simplicity is what makes the network more appealing. Just by looking at its architecture, it can be said that it is quite uniform.

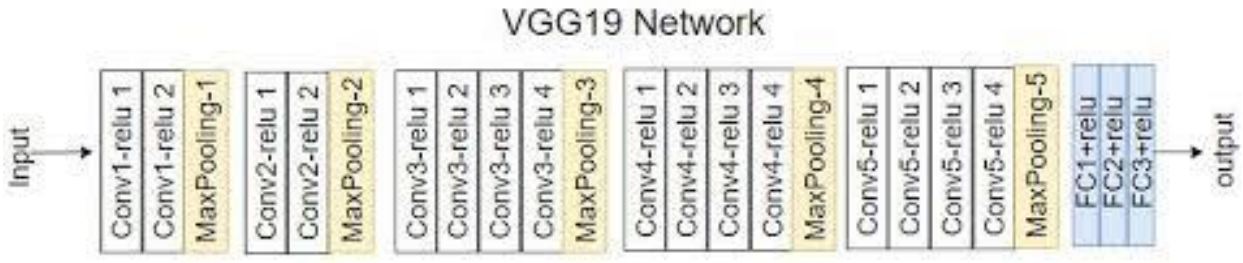
There are a few convolution layers followed by a pooling layer that reduces the height and the width. If we look at the number of filters that we can use, around 64 filters are available that we can double to about 128 and then to 256 filters. In the last layers, we can use 512 filters.

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

What is VGG19?

The concept of the VGG19 model (also VGGNet-19) is the same as the VGG16 except that it supports 19 layers. The "16" and "19" stand for the

number of weight layers in the model (convolutional layers). This means that VGG19 has three more convolutional layers than VGG16.



Complexity and challenges

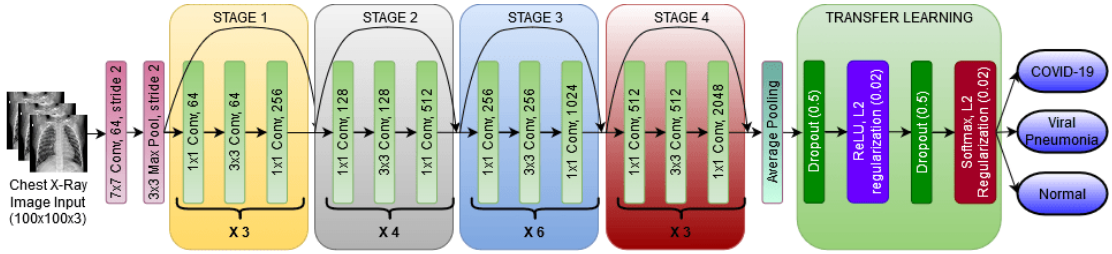
The number of filters that we can use doubles on every step or through every stack of the convolution layer. This is a major principle used to design the architecture of the VGG16 network. One of the crucial downsides of the VGG16 network is that it is a huge network, which means that it takes more time to train its parameters.

Because of its depth and number of fully connected layers, the VGG16 model is more than 533MB. This makes implementing a VGG network a time-consuming task.

The VGG16 model is used in several deep learning image classification problems, but smaller network architectures such as GoogLeNet and SqueezeNet are often preferable. In any case, the VGGNet is a great building block for learning purposes as it is straightforward to implement.

Performance of VGG Models

VGG16 highly surpasses the previous versions of models in the ILSVRC-2012 and ILSVRC-2013 competitions. Moreover, the VGG16 result is competing for the classification task winner (GoogLeNet with 6.7% error) and considerably outperforms the ILSVRC-2013 winning submission Clarifai. It obtained 11.2% with external training data and around 11.7% without it. In terms of the single-net performance, the VGGNet-16 model achieves the best result with about 7.0% test error, thereby surpassing a single GoogLeNet by around 0.9%.



Deep Residual Networks (ResNet, ResNet50)

Deep residual networks like the popular ResNet-50 model is a convolutional neural network (CNN) that is 50 layers deep. A Residual Neural Network (ResNet) is an Artificial Neural Network (ANN) of a kind that stacks residual blocks on top of each other to form a network.

Why is ResNet so popular?

This model was immensely successful, as can be ascertained from the fact that its ensemble won the top position at the ILSVRC 2015 classification competition with an error of only 3.57%. Additionally, it also came first in the ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation in the ILSVRC & COCO competitions of 2015.

What Is ResNet-50?

ResNet has many variants that run on the same concept but have different numbers of layers. Resnet50 is used to denote the variant that can work with 50 neural network layers.

GoogLeNet

The GoogLeNet architecture consists of 22 layers (27 layers including pooling layers), and part of these layers are a total of 9 inception modules

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Characteristics and features of GoogLeNet

The input layer of the GoogLeNet architecture takes in an image of the dimension 224 x 224.

Type: This refers to the name of the current layer of the component within the architecture

Patch Size: Refers to the size of the sweeping window utilized across conv and pooling layers. Sweeping windows have equal height and width.

Stride: Defines the amount of shift the filter/sliding window takes over the input image.

Output Size: The resulting output dimensions(height, width, number of feature maps) of the current architecture component after the input is passed through the layer.

Depth: Refer to the number of levels/layers within an architecture component.

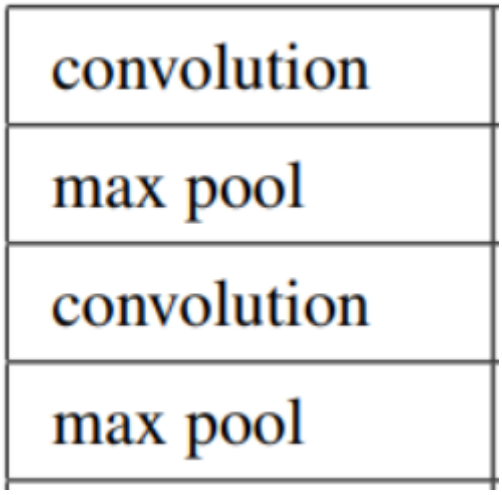
#1x1 #3x3 #5x5: Refers to the various convolutions filters used within the inception module.

#3X3 reduce #5x5 reduce: Refers to the numbers of 1x1 filters used before the convolutions.

Pool Proj: This is the number of 1x1 filters used after pooling within an inception module.

Params: Refers to the number of weights within the current architecture component.

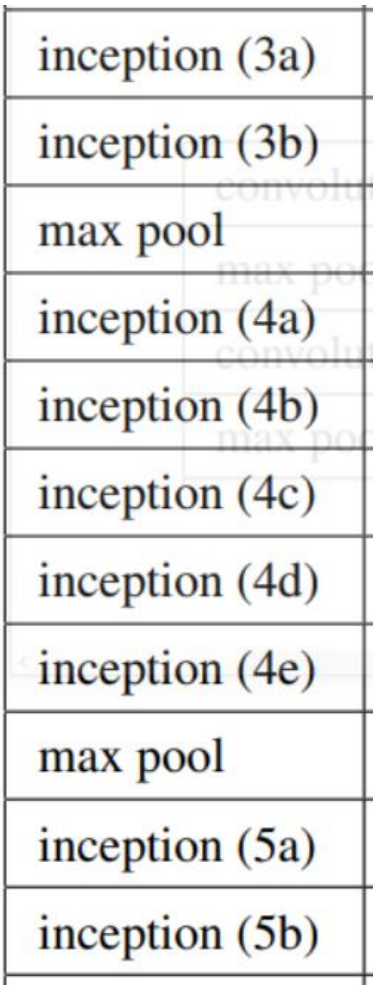
Ops: Refers to the number of mathematical operations carried out within the component.



The first conv layer uses a filter (patch) size of 7x7, which is relatively large compared to other patch sizes within the network. This layer's primary purpose is to immediately reduce the input image, but not lose spatial information by utilizing large filter sizes.

The input image size (height and width) is reduced by a factor of four at the second conv layer and a factor of eight before reaching the first inception module, but a larger number of feature maps are generated.

The second conv layer has a depth of two and leverages the 1x1 conv block, which as the effect of dimensionality reduction. Dimensionality reduction through 1x1 conv block allows the decrease of computational load by lessening the layers' number of operations.



The average pooling layer takes a mean across all the feature maps produced by the last inception module and reduced the input height and width to 1x1.

A dropout layer (40%) is utilized just before the linear layer. The dropout layer is a regularisation technique that is used during training to prevent overfitting of the network.

The linear layer consists of 1000 hidden units, which corresponds to the 1000 classes present within the Imagenet dataset.

The final layer is the softmax layer; this layer uses the softmax function, an activation function utilised to derive the probability distribution of a set of numbers within an input vector.

The output of a softmax activation function is a vector in which its set of values represents the probability of a class or event occurrence. The values within the vector all add up to 1.

Auxilary Classifiers

Before bringing the exploration of the GoogLeNet architecture to a close, there's one more component that was implemented by the creators of the network to regularise and prevent overfitting. This additional component is known as an **Auxilary Classifier**.

One main problem of an extensive network is that they suffer from vanishing gradient descent. Vanishing gradient descent occurs when the update to the weights that arises from backpropagation is negligible within the bottom layers as a result of relatively small gradient value. Simply kept, the network stops learning during training.

Auxilary Classifiers are added to the intermediate layers of the architecture, namely the third(Inception 4[a]) and sixth (Inception4[d]).

Auxilary Classifiers are only utilised during training and removed during inference. The purpose of an auxiliary classifier is to perform a classification based on the inputs within the network's midsection and add the loss calculated during the training back to the total loss of the network.

The loss of auxiliary classifiers is weighted, meaning the calculated loss is multiplied by 0.3.

Feature Extraction Techniques

Feature extraction is a part of the dimensionality reduction process, in which, an initial set of the raw data is divided and reduced to more manageable groups. So when you want to process it will be easier. The most important characteristic of these large data sets is that they have a large number of variables. These variables require a lot of computing resources to process. So Feature extraction helps to get the best feature from those big data sets by selecting and combining variables into features, thus, effectively reducing the amount of data. These features are easy to process, but still able to describe the actual data set with accuracy and originality.

There are a number of feature descriptors out there. Here are a few of the most popular ones:

- HOG: Histogram of Oriented Gradients
- SIFT: Scale Invariant Feature Transform

- SURF: Speeded-Up Robust Feature
- LBP: Local Binary Pattern

Why Feature Extraction is Useful?

The technique of extracting the features is useful when you have a large data set and need to reduce the number of resources without losing any important or relevant information. Feature extraction helps to reduce the amount of redundant data from the data set.

In the end, the reduction of the data helps to build the model with less machine effort and also increases the speed of learning and generalization steps in the machine learning process.

Local Binary Pattern (LBP)

LBP is one of the most commonly used features for texture discrimination tasks like face, facial expressions, gesture, scene and object recognition. LBP was conceived well before SIFT and HOG.

Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. Due to its discriminative power and computational simplicity, LBP texture operator has become a popular approach in various applications. It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps

the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination variations. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings.

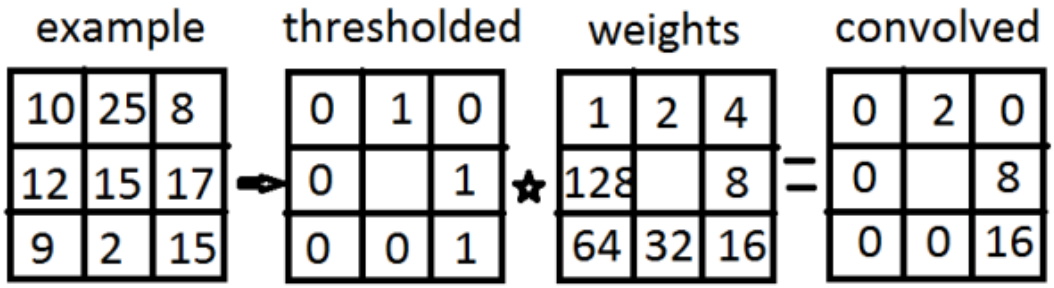
LBP Calculation

For calculating the LBP, the LBP code for each pixel is calculated and the histogram of LBP codes is constructed as the LBP feature.

To calculate the LBP, for each pixel p, the 8 neighbors of the center pixel are compared with the pixel p and the neighbors x are assigned a value 1 if x ≥ p. below Figure shows how LBP code for a 3×3 area is calculated. This process is computed across the whole image.

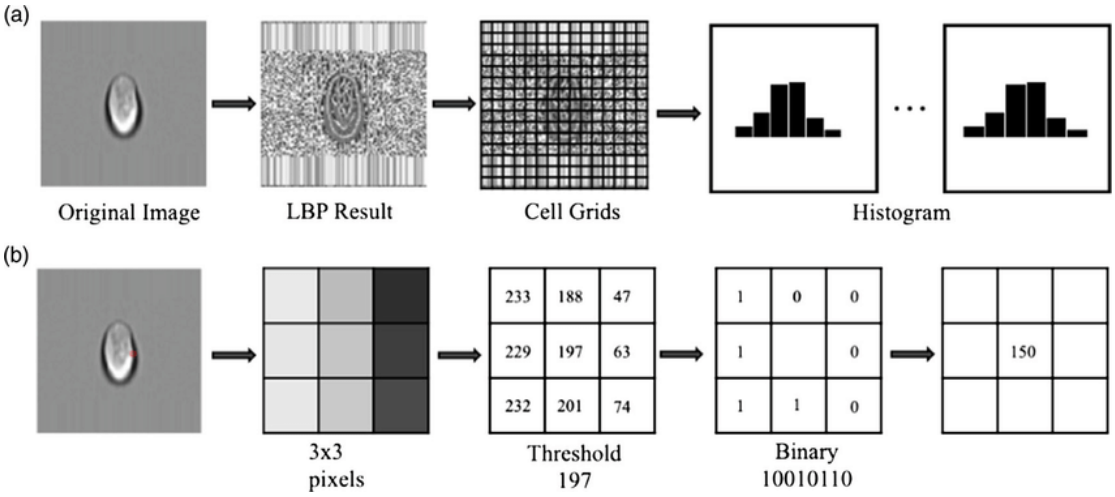
$$\mathrm{LBP}(x_c, y_c) = \sum_{i=0}^{7} s(g_i - g_c) 2^i$$

$$s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$



LBP = 2 + 8 + 16 = 26

C = (25+17+15)/3 - (10+8+12+9+2)/5 = -22



Introduction to the HOG Feature Descriptor

- HOG, or Histogram of Oriented Gradients, is a feature descriptor that is often used to extract features from image data. It is widely used in computer vision tasks for object detection.
- Let's look at some important aspects of HOG that makes it different from other feature descriptors:
- The HOG descriptor focuses on the structure or the shape of an object. In the case of edge features, we only identify if the pixel is an edge or not.

HOG is able to provide the edge direction as well. This is done by extracting the **gradient and orientation** (or you can say magnitude and direction) of the edges

- Additionally, these orientations are calculated in **'localized' portions**. This means that the complete image is broken down into smaller regions and for each region, the gradients and orientation are calculated. We will discuss this in much more detail in the upcoming sections
- Finally the HOG would generate a **Histogram** for each of these regions separately. The histograms are created using the gradients and orientations of the pixel values, hence the name 'Histogram of Oriented Gradients. The HOG feature descriptor counts the occurrences of gradient orientation in localized portions of an image.

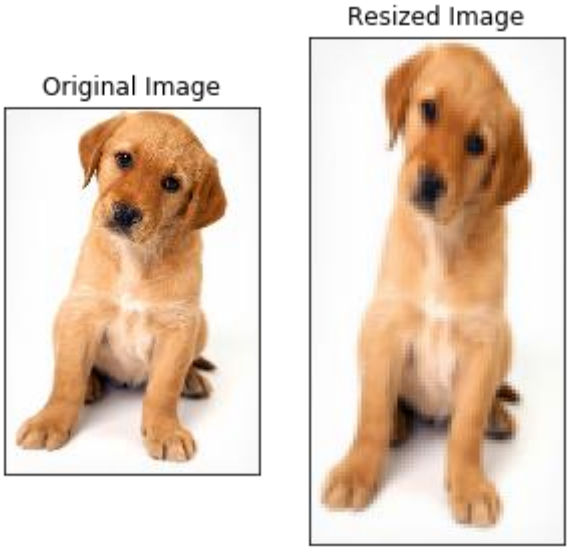
Process of Calculating the Histogram of Oriented Gradients (HOG)

Step 1: Preprocess the Data (64 x 128)

This is a step most of you will be pretty familiar with. Preprocessing data is a crucial step in any machine learning project and that's no different when working with images.

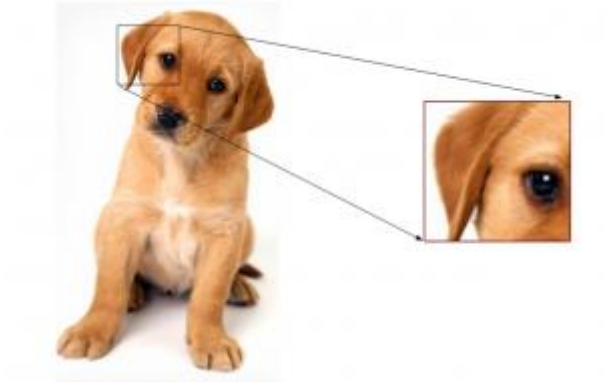
We need to preprocess the image and bring down the width to height ratio to 1:2. The image size should preferably be 64 x 128. This is because we will be dividing the image into 8*8 and 16*16 patches to extract the features. Having the specified size (64 x 128) will make all our calculations pretty simple. In fact, this is the exact value used in the original image

Coming back to the example we have, let us take the size 64 x 128 to be the standard image size for now. Here is the resized image:



Step 2: Calculating Gradients (direction x and y)

The next step is to calculate the gradient for every pixel in the image. **Gradients are the small change in the x and y directions.** Here, I am going to take a small patch from the image and calculate the gradients on that:



We will get the pixel values for this patch. Let's say we generate the below pixel matrix for the given patch (the matrix shown here is merely used as an example and these are not the original pixel values for the given patch):

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

I have highlighted the pixel value 85. Now, to determine the gradient (or change) in the x-direction, we need to subtract the value on the left from the pixel value on the right. Similarly, to calculate the gradient in the y-direction, we will subtract the pixel value below from the pixel value above the selected pixel.

Hence the resultant gradients in the x and y direction for this pixel are:

- Change in X direction(G_x) = 89 – 78 = 11
- Change in Y direction(G_y) = 68 – 56 = 8

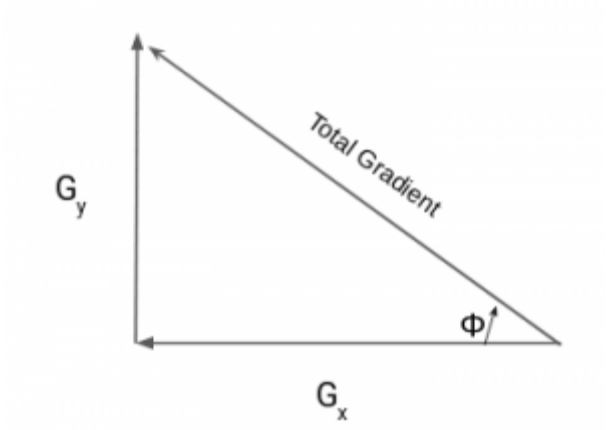
This process will give us two new matrices – one storing gradients in the x-direction and the other storing gradients in the y direction. This is similar to using a Sobel Kernel of size 1. **The magnitude would be**

higher when there is a sharp change in intensity, such as around the edges.

We have calculated the gradients in both x and y direction separately. The same process is repeated for all the pixels in the image. The next step would be to find the magnitude and orientation using these values.

Step 3: Calculate the Magnitude and Orientation

Using the gradients we calculated in the last step, we will now determine the magnitude and direction for each pixel value. For this step, we will be using the Pythagoras theorem .Take a look at the image below:



The gradients are basically the base and perpendicular here. So, for the previous example, we had Gx and Gy as 11 and 8. Let's apply the Pythagoras theorem to calculate the total gradient magnitude:

$$\text{Total Gradient Magnitude} = \sqrt{[(G_x)^2+(G_y)^2]}$$

$$\text{Total Gradient Magnitude} = \sqrt{[(11)^2+(8)^2]} = 13.6$$

Next, calculate the orientation (or direction) for the same pixel. We know that we can write the tan for the angles:

$$\tan(\Phi) = Gy / Gx$$

Hence, the value of the angle would be:

$$\Phi = \text{atan}(Gy / Gx)$$

The orientation comes out to be 36 when we plug in the values. So now, for every pixel value, we have the total gradient (magnitude) and the orientation (direction). We need to generate the histogram using these gradients and orientations.

But hang on – we need to take a small break before we jump into how histograms are created in the HOG feature descriptor. Consider this a small step in the overall process. And we'll start this by discussing some simple methods of creating Histograms using the two values that we have – gradients and orientation.

Different Methods to Create Histograms using Gradients and Orientation

A histogram is a plot that shows the frequency distribution of a set of continuous data. We have the variable (in the form of bins) on the x-axis and the frequency on the y-axis. Here, we are going to take the angle or orientation on the x-axis and the frequency on the y-axis.

Method 1:

Let us start with the simplest way to generate histograms. We will take each pixel value, find the orientation of the pixel and update the frequency table.

Here is the process for the highlighted pixel (85). Since the orientation for this pixel is 36, we will add a number against angle value 36, denoting the frequency:

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Frequency						1									
Angle	1	2	3	4 ...	35	36	37	38	39....	175	176	177	178	179	180

The same process is repeated for all the pixel values, and we end up with a frequency table that denotes angles and the occurrence of these angles in the image. This frequency table can be used to generate a histogram with angle values on the x-axis and the frequency on the y-axis.

That's one way to create a histogram. Note that here the bin value of the histogram is 1. Hence we get about 180 different buckets, each representing an orientation value. Another method is to create the histogram features for higher bin values.

Method 2:

This method is similar to the previous method, except that here we have a bin size of 20. So, the number of buckets we would get here is 9.

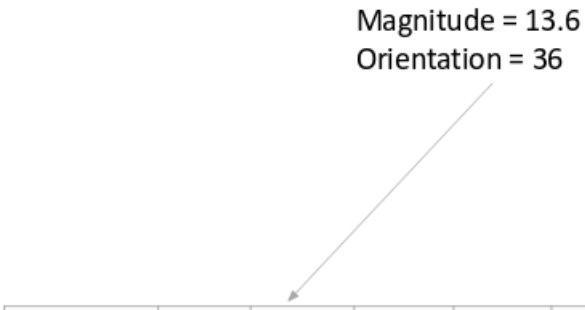
Again, for each pixel, we will check the orientation, and store the frequency of the orientation values in the form of a 9 x 1 matrix. Plotting this would give us the histogram:

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Magnitude		1							
Bin	0	20	40	60	80	100	120	140	160

Method 3:

The above two methods use only the orientation values to generate histograms and do not take the gradient value into account. Here is another way in which we can generate the histogram – instead of using the frequency, we can use the gradient magnitude to fill the values in the matrix. Below is an example of this:

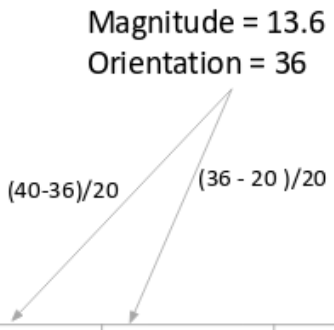


Magnitude		13.6							
Bin	0	20	40	60	80	100	120	140	160

You might have noticed that we are using the orientation value of 30, and updating the bin 20 only. Additionally, we should give some weight to the other bin as well.

Method 4:

Let's make a small modification to the above method. Here, we will add the contribution of a pixel's gradient to the bins on either side of the pixel gradient. Remember, the higher contribution should be to the bin value which is closer to the orientation.



Magnitude		(4/20)*13.6	(16/20)*13.6						
Bin	0	20	40	60	80	100	120	140	160

This is exactly how histograms are created in the HOG feature descriptor.

Step 4: Calculate Histogram of Gradients in 8×8 cells (9×1)

The histograms created in the HOG feature descriptor are not generated for the whole image. Instead, the image is divided into 8×8 cells, and the histogram of oriented gradients is computed for each cell.

By doing so, we get the features (or histogram) for the smaller patches which in turn represent the whole image. We can certainly change this value here from 8 x 8 to 16 x 16 or 32 x 32.

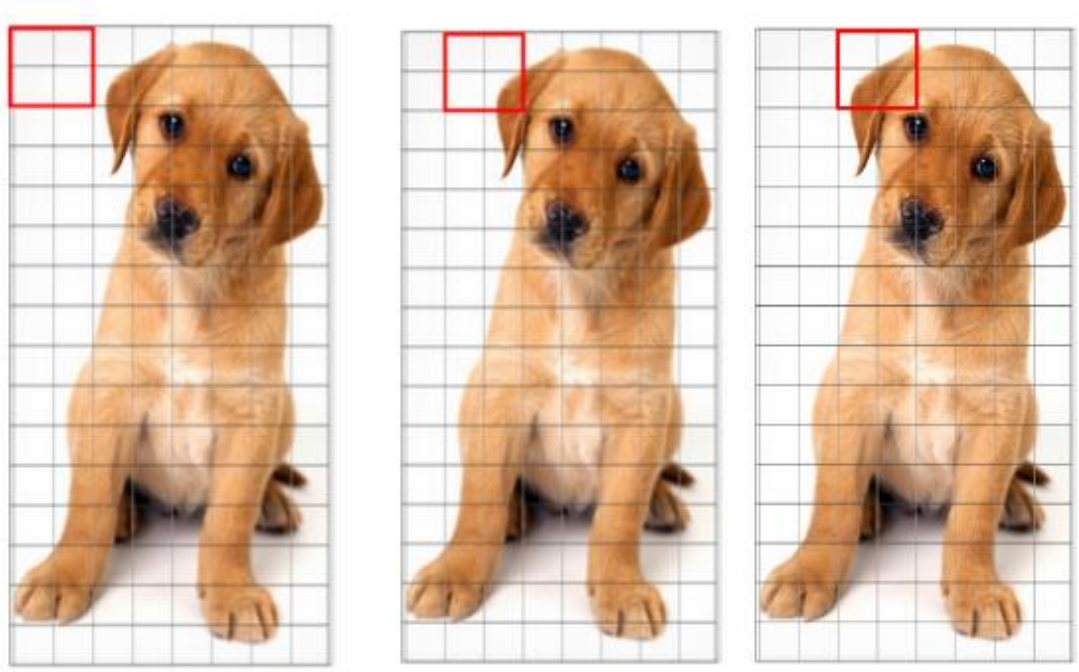
If we divide the image into 8×8 cells and generate the histograms, we will get a 9 x 1 matrix for each cell. This matrix is generated using method 4 that we discussed in the previous section.

Once we have generated the HOG for the 8×8 patches in the image, the next step is to normalize the histogram.

Step 5: Normalize gradients in 16×16 cell (36×1)

Although we already have the HOG features created for the 8×8 cells of the image, the gradients of the image are sensitive to the overall lighting. This means that for a particular picture, some portion of the image would be very bright as compared to the other portions.

We cannot completely eliminate this from the image. But we can reduce this lighting variation by normalizing the gradients by taking 16×16 blocks. Here is an example that can explain how 16×16 blocks are created:



Here, we will be combining four 8×8 cells to create a 16×16 block. And we already know that each 8×8 cell has a 9×1 matrix for a histogram. So, we would have four 9×1 matrices or a single 36×1 matrix. To normalize this matrix, we will divide each of these values by the square root of the sum of squares of the values. Mathematically, for a given vector V:

$$V = [a1, a2, a3, \ldots a36]$$

We calculate the root of the sum of squares:

$$k = \sqrt{(a1)2 + (a2)2 + (a3)2 + \ldots (a36)2}$$

And divide all the values in the vector V with this value k:

$$\text{Normalised Vector} = \left(\frac{a_1}{k}, \frac{a_2}{k}, \frac{a_3}{k}, \ldots \frac{a_{36}}{k} \right)$$

The resultant would be a normalized vector of size 36×1.

Step 6: Features for the complete image

We are now at the final step of generating HOG features for the image. So far, we have created features for 16×16 blocks of the image. Now, we will combine all these to get the features for the final image.

Can you guess what would be the total number of features that we will have for the given image? We would first need to find out how many such 16×16 blocks would we get for a single 64×128 image:

We would have 105 (7×15) blocks of 16×16. Each of these 105 blocks has a vector of 36×1 as features. Hence, the total features for the image would be 105 x 36×1 = 3780 features.

What is SURF?

SURF (Speeded-Up Robust Features) is a feature detection framework introduced by Herbert Bay and his colleagues at ETH Zurich. SURF interest points are in-plane rotation-invariant, robust to noise, and overall, extremely fast to calculate. This procedure can be divided into three steps:

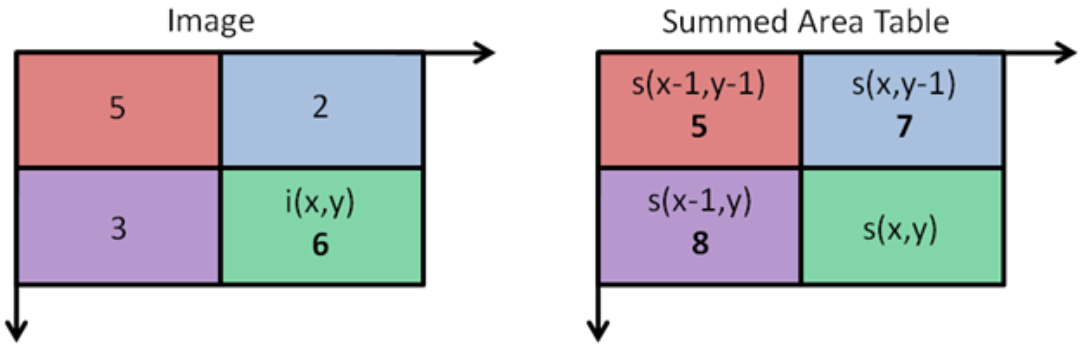
1. Interest Point Detection

2. Interest Point Description

3. Interest Point Matching

Detection: Integral Images

he first step to detect the interest points by SURF algorithm by using Integral Image. In the beginning the image has converted from RGB color space into Grayscale color. After that, integral image has been calculated to speeded up the calculations and decrease the time consuming. Integral image is used for calculating the average the intensity within the image. The following figure, shows how the integral image has been calculated for the given pixels as shown below:



Interest Point Description

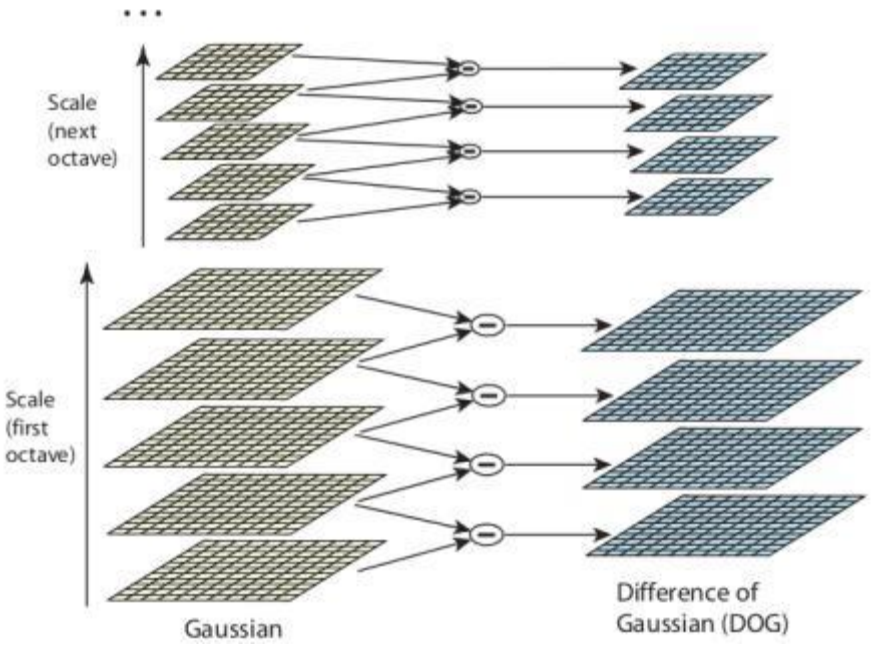
To detect interest points, SURF uses an integer approximation of the determinant of Hessian blob detector, which can be computed with 3 integer operations using a recomputed integral image. Its feature descriptor is based on the sum of the Haar wavelet response around the point of interest.

SIFT algorithm

1. Scale-space Extreme Detection

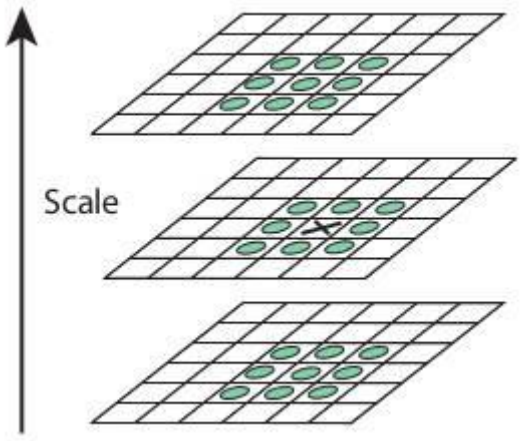
From the image above, it is obvious that we can't use the same window to detect keypoints with different scale. It is OK with small corner. But to detect larger corners we need larger windows. For this, scale-space filtering is used. In it, Laplacian of Gaussian is found for the image with various σ values. LoG acts as a blob detector which detects blobs in various sizes due to change in σ. In short, σ acts as a scaling parameter. For eg, in the above image, gaussian kernel with low σ gives high value for small corner while gaussian kernel with high σ fits well for larger corner. So, we can find the local maxima across the scale and space which gives us a list of (x,y,σ) values which means there is a potential keypoint at (x,y) at σ scale.

But this LoG is a little costly, so SIFT algorithm uses Difference of Gaussians which is an approximation of LoG. Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and kσ. This process is done for different octaves of the image in Gaussian Pyramid. It is represented in below image:



image

Once this DoG are found, images are searched for local extrema over scale and space. For eg, one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale. It is shown in below image:



image

2. Keypoint Localization

Once potential keypoints locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extreme, and if the intensity at this extreme is less than a threshold value (0.03 as per the paper), it is rejected. This threshold is called contrast Threshold in OpenCV

DoG has higher response for edges, so edges also need to be removed. For this, a concept similar to Harris corner detector is used. They used a 2x2 Hessian matrix (H) to compute the principal curvature. We know from Harris corner detector that for edges, one eigen value is larger than the other. So here they used a simple function,

If this ratio is greater than a threshold, called edgeThreshold in OpenCV, that keypoint is discarded. It is given as 10 in paper.

So it eliminates any low-contrast keypoints and edge keypoints and what remains is strong interest points.

3. Orientation Assignment

Now an orientation is assigned to each keypoint to achieve invariance to image rotation. A neighbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created (It is weighted by gradient magnitude and gaussian-weighted circular window with σ equal to 1.5 times the scale of keypoint). The highest peak in the histogram is taken

and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with same location and scale, but different directions. It contribute to stability of matching.

4. Keypoint Descriptor

Now keypoint descriptor is created. A 16x16 neighbourhood around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

5. Keypoint Matching

Keypoints between two images are matched by identifying their nearest neighbours. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected.