CHAPTER: 1
INTRODUCTION TO IMAGE PROCESSING

An image may defined as a 2-dimensional function, f(x,y), where x & y are spatial coordinates, & the amplitude of at any pair of coordinates (x,y) is called the intensity or gray level of the image at that point. When x,y & the amplitude values of are all finite, discrete quantities, we call the image as Digital image. A digital image differs from a photo in that the values are all discrete. A digital image can be considered as a large array of discrete dots, each of which has a brightness associated with it. These dots are called picture elements, or more simply pixels.

1.1 IMAGE PROCESSING

Image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a 2-D signal and applying standard signal-processing techniques to it.
It is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. Usually, **Image Processing** system includes treating images as two dimensional signals while applying already set signal processing methods to them.

Figure 1.1 Removal of Noise from the image

1.2 APPLICATION OF IMAGE PROCESSING

- Automatic visual inspection system
- Biomedical imaging techniques
- Defence surveillance
- Content based image retrieval
- Moving object tracking
- Image and video compression

1.3 PURPOSE OF IMAGE PROCESSING

The purpose of image processing is divided into 5 groups. They are:

1. Visualization - Observe the objects that are not visible.

2. Image sharpening and restoration - To create a better image.

3. Image retrieval - Seek for the image of interest.

4. Measurement of pattern – Measures various objects in an image.

5. Image Recognition – Distinguish the objects in an image.

1.4 IMAGE FORMATS

BMP (Microsoft Windows Bitmap): The Microsoft Windows Bitmap (BMP) file format is one of several graphics file formats supported by the Microsoft Windows operating environment. BMP is the native bitmap format of Windows and is used to store virtually any type of bitmap data.

GIF (Graphics Interchange Files): GIF stands for "Graphics Interchange Format." GIF is an image file format commonly used for images on the web and sprites in software programs. Unlike the JPEG image format, GIFs uses lossless compression that does not degrade the quality

of the image. However, GIFs store image data using indexed colour, meaning a standard GIF image can include a maximum of 256 colours.

HDF (Hierarchical Data Format): Hierarchical Data Format (HDF) is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of data.

JPEG (Joint Photographic Experts Group): JPEG is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography. The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality.

PCX (Paint Brush): The PaintBrush (PCX) file format is a bitmap format originally developed by the ZSoft Corporation for the PC Paintbrush program.

PNG (Portable Network Graphics): Portable Network Graphics is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most widely used lossless image compression format on the Internet.

TIFF (Tagged Image File Format): Tagged Image File Format, abbreviated TIFF or TIF, is a computer file format for storing raster graphics images, popular among graphic artists, the publishing industry, and photographers.

XWD (X Window Dump): XWD stands for X Window dump. This file format is used to store and restore screen window images. Two versions of this file exist, the X10 window dump and the X11 window dump. These versions differ slightly in the format of the data (i.e. headers and bits per pixel). The version X10 XWD file cannot have more than 8 BPP image data.

1.5 IMAGE COORDINATE SYSTEM

Pixel coordinates: In this coordinate system, the image is treated as a grid of discrete elements, ordered from top to bottom and left to right.

Spatial Coordinates: In this spatial coordinate system, locations in an image are positions on a plane, and they are described in terms of x and y (not r and c as in the pixel coordinate system).

1.6 IMAGE REPRESENTATION

A digital image differs from a photo in that the values are all discrete. Usually they take integer values. A digital image can be considered as a large two dimensional array of discrete cells, each of which has a brightness associated with it. These dots are called pixels.

1. Grayscale: It is represented as a M × N matrix, with pixel intensity varying from 0-255. Zero denotes black, 255 denote white, and the rest are used for shades of grey in between.

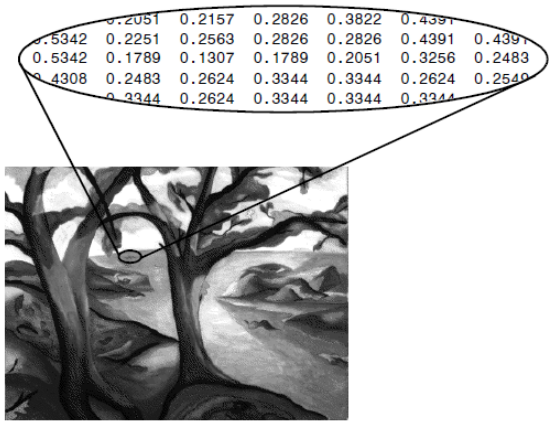


Figure 1.2- Grayscale image

2. Truecolor RGB: A true colour red-green-blue (RGB) image is represented as a three-dimensional *M* × N × 3 double matrix. Each pixel has red, green, blue components along the third dimension with values in [0, 1]. If each of these components has a range 0–255, this gives a total of **2563** different possible colours.

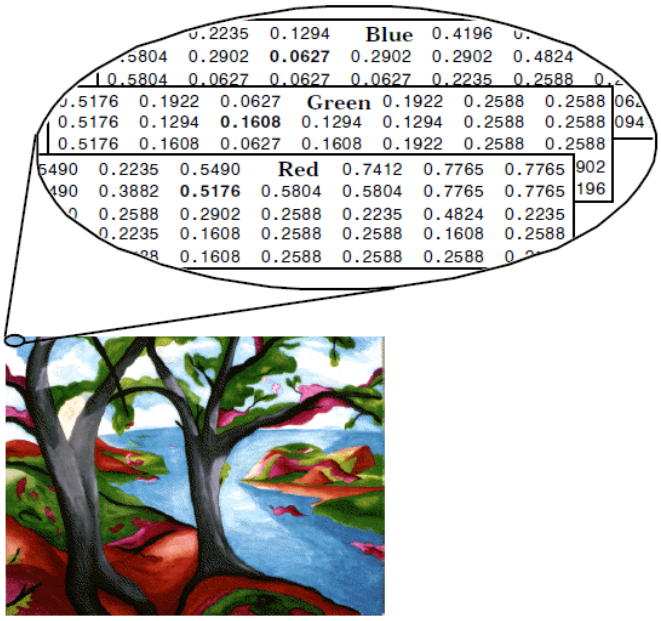


Figure 1.3 – RGB Image

3. Indexed: Indexed (paletted) images are represented with an index matrix of size M×N and a colormap matrix of size K×3. The image has in total K different colours. The colormap holds all colours used in the image and the index matrix represents the pixels by referring to colours in the colormap.

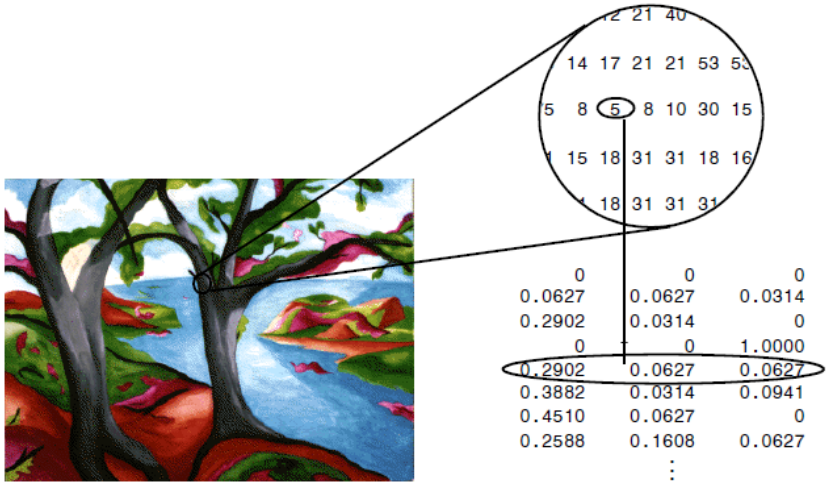


Figure 1.4 – Indexed image

4. Binary: A binary image is represented by an M×N logical matrix where pixel values are 1 (true) or 0 (false).

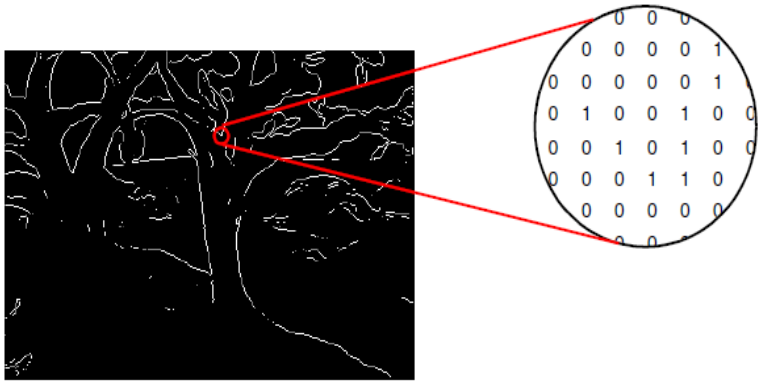


Figure 1.5 – Binary image

NOTE: Grayscale is usually the preferred format for image processing, while for the optical character recognition, Binary images offer the largest ease of processing. In cases requiring colour, an RGB colour image can be decomposed and handled as three separate grayscale images. Indexed images must be converted to grayscale or RGB for most operations.

1.7 DOMAINS

We can deal with images in many domains. Fourier series and frequency domain is purely mathematics, so we will try to minimize that math's part and focus more on its use in DIP.

Spatial domain

In simple spatial domain, we directly deal with the image matrix, whereas in frequency domain, we deal an image like this. The spatial domain is the normal image space, in which a change in position in I directly projects to a change in position in S. Distances in I (in pixels) correspond to real distances (e.g. in meters) in S.

This concept is used most often when discussing the frequency with which image values change, that is, over how many pixels does a cycle of periodically repeating intensity variations occur. One would refer to the number of pixels over which a pattern repeats (its periodicity) in the spatial domain.

In most cases, the Fourier Transform will be used to convert images from the spatial domain into the frequency domain. A related term used in this context is spatial frequency, which refers to the (inverse of the) periodicity with which the image intensity values change. Image features with high spatial frequency (such as edges) are those that change greatly in intensity over short image distances. Another term used in this context is spatial derivative, which refers to how much the image intensity values change per change in image position.

Frequency Domain

We first transform the image to its frequency distribution. Then our black box systems perform whatever processing it has to performed, and the output of the black box in this case is not an image, but a transformation. After performing inverse transformation, it is converted into an image which is then viewed in spatial domain.

Here we have used the word transformation. What does it actually mean?

Transformation

A signal can be converted from time domain into frequency domain using mathematical operators called transforms. There are many kind of transformation that does this. Some of them are given below.

1. Fourier Series: A Fourier series is an expansion of a periodic function in terms of an infinite sum of sines and cosines. Fourier series make use of the orthogonality relationships of the sine and cosine functions. The computation and study of Fourier series is known as harmonic analysis.

2. Fourier Transformation: The Fourier transform decomposes a function of time, i.e., a signal into the frequencies that make it up. The Fourier transform of a function of time itself is a complex-valued function of frequency, whose absolute value represents the magnitude of the frequency component present in the original function, and whose complex argument is the phase offset of the basic sinusoid in that frequency.

3. Laplace Transform: The Laplace transform is very similar to the Fourier transform. While the Fourier transform of a function is a complex function of areal variable (frequency), the Laplace transform of a function is a complex function of a complex variable. Laplace transforms are usually restricted to functions of t with t>0.

4. Z Transform: The Z-transform converts a sequence of real or complex numbers, a discrete-time into a signal, which is complex frequency domain representation. It can be considered as a discrete-time equivalent of the Laplace transform.

Out of all these, we will thoroughly discuss Fourier series and Fourier transformation in our next tutorial.

Time Domain

To explain about the time domain, we would like to compare it with frequency domain. The time domain (or spatial domain for image processing) and the frequency domain are both continuous, infinite domains. There is no explicit or implied periodicity in either domain. This is what we call the Fourier transform. The time domain is continuous and the time-domain functions are periodic. The frequency domain is discrete. We call this the Fourier series. The time domain is discrete and infinite, and the frequency domain is continuous. In the frequency domain the transform is periodic. This is the discrete-time Fourier transform (DTFT). The time domain and the frequency domain are both discrete and finite. Although finite, the time and frequency domains are both implicitly periodic. This form is the discrete Fourier transform (DFT).

CHAPTER: 2
INTRODUCTION TO PYTHON

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community after 30 years.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms including object oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

2.1 HISTORY

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum's long influence on Python is reflected in the title given to him by the Python community: *Benevolent Dictator For Life* (BDFL) – a post from which he gave himself permanent vacation on July 12, 2018.

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python

2.6.x and 2.7.x version series. Releases of Python 3 include the `2to3` utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler to improve performance under concurrent workloads.

2.2 FEATURES

The language's core philosophy is summarized in the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax (such as that of Perl) in favor of a simpler, less-cluttered grammar. As Alex Martelli put it: "To describe something as 'clever' is *not* considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it".

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of CPython that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonists, Pythonistas, and Pythoneers.

2.3 SYNTAX AND SEMANTICS

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule.

Statements and control flow

Python's statements include (among others):

- The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version ofvariables) illuminates many other features of the language. Assignment in C, e.g., x = 2, translates to "typed variable name x receives a copy of numeric

value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, x = 2, translates to name x receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed binding the name to the object. Since the name's storage location doesn't contain the indicated value, it is improper to call it a variable. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., x = 2; y = 2; z = 2 result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it. However at a given time a name will be bound to *some* object, which **will** have a type; thus there is dynamic typing.

- The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).
- The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
- The while statement, which executes a block of code as long as its condition is true.
- The try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.
- The raise statement, used to raise a specified exception or re-raise a caught exception.
- The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
- The def statement, which defines a function or method.
- The with statement, from Python 2.5 released on September 2006, which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behavior and replaces a common try/finally idiom.

- The pass statement, which serves as a NOP. It is syntactically needed to create an empty code block.
- The assert statement, used during debugging to check for conditions that ought to apply.
- The print statement was changed to the print() function in Python 3.

Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will. However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators. Before 2.5, generators were lazy iterators; information was passed unidirectionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.

Methods

Methods on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter to access instance data, in contrast to the implicit `self` (or `this`) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).

Typing

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class `object` is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`). In versions of

Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long term plan is to support gradual typing and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named mypy supports compile-time type checking.

Libraries

Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals, manipulating regular expressions, and unit testing.

As of March 2018, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 130,000 packages with a wide range of functionality, including:

- Graphical user interfaces
- Web frameworks
- Multimedia
- Databases
- Networking
- Test frameworks
- Automation
- Web scraping
- Documentation
- System administration
- Scientific computing
- Text processing
- Image processing

2.4 USES

Since 2003, Python has consistently ranked in the top ten most popular programming languages in the TIOBE Programming Community Index where, as of January 2018, it is the fourth most popular language (behind Java, C, and C++). It was selected Programming Language of the Year in 2007 and 2010.

An empirical study found that scripting languages, such as Python, are more productive than conventional languages, such as C and Java, for programming problems involving string manipulation and search in a dictionary, and determined that memory consumption was often "better than Java and not much worse than C or C++".

Python can serve as a scripting language for web applications, e.g., via mod_wsgi for the Apache web server. With Web Server Gateway Interface, a standard API has evolved to facilitate these applications like Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle and support developers in the design and maintenance of complex applications. Pyjs and IronPython can be used to develop the client-side of Ajax-based applications. SQLAlchemy can be used as data mapper to a relational database. Twisted is a framework to program communications between computers, and is used (for example) by Dropbox.

Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing,[133][134] with specialized libraries such as Biopython and Astropy providing domain-specific functionality.SageMath is a mathematical software with a "notebook" programmable in Python: its library covers many aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus.

Python has been successfully embedded in many software products as a scripting language, including in finite element method software such as Abaqus, 3D parametric modeler like FreeCAD, 3D animation packages such as 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, the visual effects compositor Nuke, 2D imaging programs like GIMP. Inkscape, Scribus and Paint Shop Pro, and musical notation programs like scorewriter and capella. GNU Debugger uses Python as a pretty printer to show complex structures such as C++ containers. Esri promotes Python as the

best choice for writing scripts in ArcGIS. It has also been used in several video games, and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go.Python is also used in algorithmic trading and quantitative finance. Python can also be implemented in APIs of online brokerages that run on other languages by using wrappers.

Python has been used in artificial intelligence projects. As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.

Many operating systems include Python as a standard component. It ships with most Linux distributions, AmigaOS 4, FreeBSD, NetBSD, OpenBSD and macOS, and can be used from the command line (terminal). Many Linux distributions use installers written in Python: Ubuntu uses the Ubiquity installer, while Red Hat Linux and Fedora use the Anaconda installer. Gentoo Linux uses Python in its package management system,Portage.

Python is used extensively in the information security industry, including in exploit development.

Most of the Sugar software for the One Laptop per Child XO, now developed at Sugar Labs, is written in Python.

The Raspberry Pi single-board computer project has adopted Python as its main user-programming language.

LibreOffice includes Python, and intends to replace Java with Python. Its Python Scripting Provider is a core feature since Version 4.0 from 7 February 2013.

CHAPTER: 3

ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI), sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals. In computer science AI research is defined as the study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals. Colloquially, the term "artificial intelligence" is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving".

3.1 NEURAL NETWORKS

Artificial neural networks (**ANN**) or **connectionist systems** are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labelled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, e.g., that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

An ANN is based on a collection of connected units or nodes called artificial neurons which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

In common ANN implementation, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called 'edges'. Artificial neurons

and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. Artificial neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

Neural networks offer a number of advantages, including requiring less formal statistical training, ability to implicitly detect complex nonlinear relationships between dependent and independent variables, ability to detect all possible interactions between predictor variables, and the availability of multiple training algorithms. Disadvantages include its "black box" nature, greater computational burden, proneness to over fitting, and the empirical nature of model development.

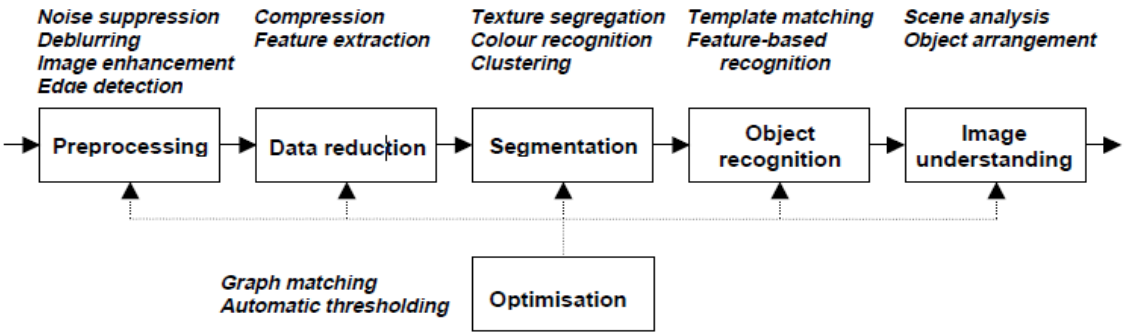


Figure: 3.1 Explaining process stages for image processing using neural network model

3.2 CLASSIFICATION

It can be classified into two categories:-

- Supervised training
- Unsupervised training

3.3 SUPERVISED TRAINING

Supervised learning as the name indicates a presence of supervisor as teacher. Basically supervised learning is a learning in which we teach or train the machine using data which is well labeled that means some data is already tagged with correct answer. After that, machine is provided with new set of examples (data) so that supervised learning algorithm analyses the training data(set of training examples) and produces an correct outcome from labeled data.

For instance, suppose you are given an basket filled with different kinds of fruits. Now the first step is to train the machine with all different fruits one by one like this:

- If shape of object is rounded and depression at top having color Red then it will be labelled as –**Apple**.
- If shape of object is long curving cylinder having color Green-Yellow then it will be labelled as –**Banana**.

Now suppose after training the data, you have given a new separate fruit say Banana from basket and asked to identify it.

Since machine has already learnt the things from previous data and this time have to use it wisely. It will first classify the fruit with its shape and color, and would confirm the fruit name as BANANA and put it in Banana category. Thus machine learns the things from training data(basket containing fruits) and then apply the knowledge to test data(new fruit).

Supervised learning classified into two categories of algorithms:

- **Classification**: A classification problem is when the output variable is a category, such as “Red” or “blue” or “disease” and “no disease”.

- **Regression**: A regression problem is when the output variable is a real value, such as "dollars" or "weight".

3.4 CONVOLUTIONAL NEURAL NETWORK

A convolutional neural network (CNN) is a class of deep, feed-forward networks, composed of one or more convolutional layers with fully connected layers (matching those in typical Artificial neural networks) on top. It uses tied weights and pooling layers. In particular, max-pooling is often structured via Fukushima's convolutional architecture. This architecture allows CNNs to take advantage of the 2D structure of input data.

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.

Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution. This only has significance for the indices in the matrix, and thus which weights are placed at which index.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms werehand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

CNNs are suitable for processing visual and other two-dimensional data. They have shown superior results in both image and speech applications. They can be trained with standard backpropagation. CNNs are easier to train than other regular, deep, feed-forward neural networks and have many fewer parameters to estimate. Examples of applications in computer vision include DeepDream and robot navigation.

CONVOLUTION

A convolution in the general continue case is defined as the integral of the product of two functions (signals) after one is reversed and shifted:

$f(t) * g(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$

As a result, a convolution produces a new function (signal). The convolution is a commutative operation, therefore f(t) * g(t) = g(t) * f(t)

In the 2D discrete space, the convolution operation is defined as:

$$O(i,j)=\sum u(-\infty\infty)\sum v(-\infty\infty)F(u,v)I(i-u,j-v)$$

In the image domain where the signals are finite, this formula becomes:

$$O(i,j)=\sum u(-2k-1,\ 2k+1)\sum v(-2k-1,\ 2k+1)F(u,v)I(i-u,j-v)$$

Where:

- O(i,j) is the output pixel, in position (i,j)
- 2k+1 is the side of a square, odd convolutional filter
- F is the convolutional filter
- I is the input image

This operation (single convolutional step) is done for every location (i,j) of the input image I that completely overlaps with the convolutional filter.

MAXPOOLING

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which *max pooling* is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control over fitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance.

The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples at every depth slice in the input by 2 along both width and height, discarding

75% of the activations. In this case, every max operation is over 4 numbers. The depth dimension remains unchanged.

In addition to max pooling, the pooling units can use other functions, such as average pooling or L2-norm pooling. Average pooling was often used historically but has recently fallen out of favor compared to max pooling, which works better in practice.

Due to the aggressive reduction in the size of the representation, the trend is towards using smaller filters or discarding the pooling layer altogether.

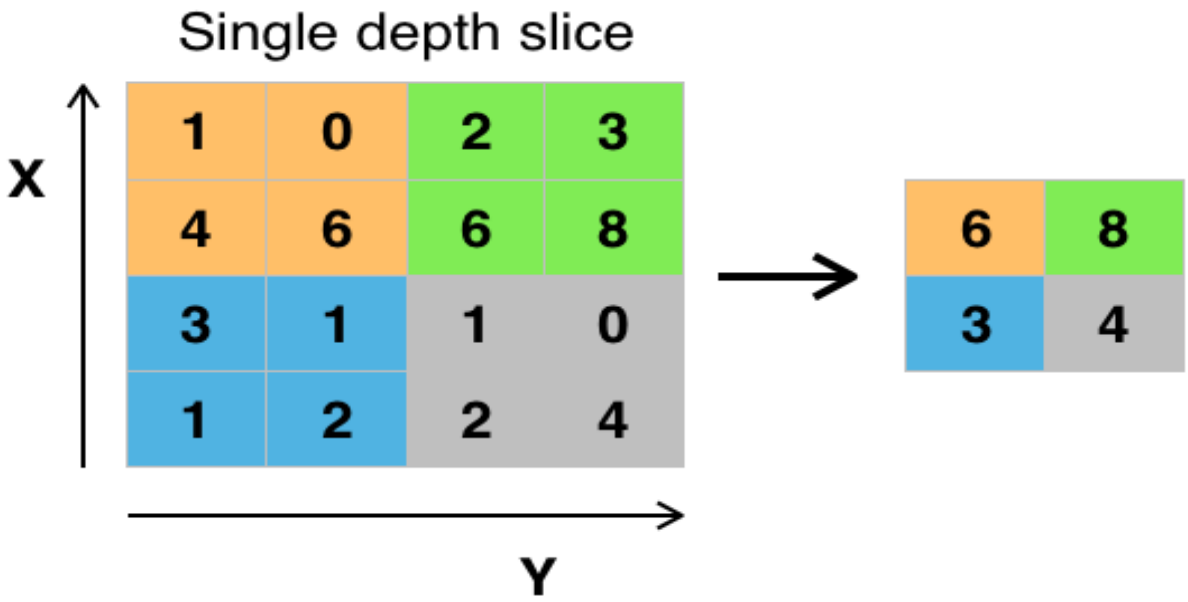


Fig 3.2 Max pooling with a 2x2 filter and stride = 2

Activation function

In artificial neural networks, the **activation function** of a node defines the output of that node, or "neuron," given an input or set of inputs. This output is then used as input for the next node and so on until a desired solution to the original problem is found. A standard computer chip circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, only *nonlinear* activation functions allow such networks to compute nontrivial

problems using only a small number of nodes. In artificial neural networks this function is also called the transfer function.

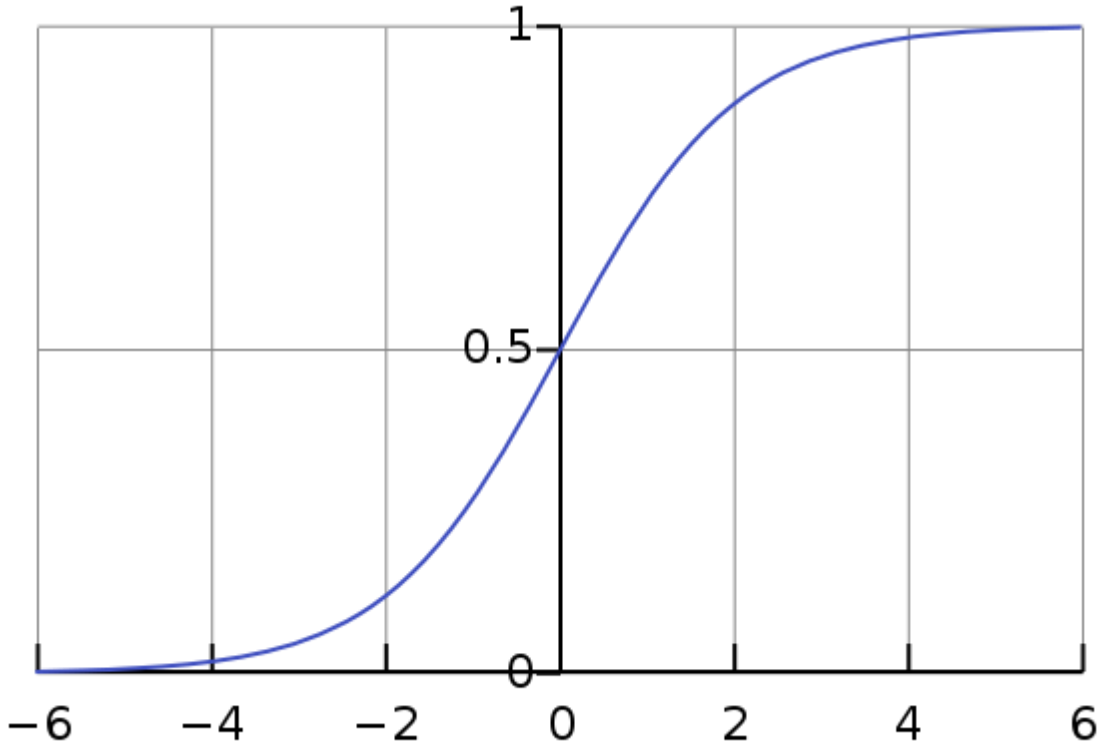


Fig 3.3 Sigmoid function

Activation function having types:

- Identity function
- Binary step function
- Bipolar step function
- Sigmoidal function
  - Binary sigmoidal function
  - Bipolar sigmoidal function
- Ramp function

Comparison of activation functions

Some desirable properties in an activation function include:

- Nonlinear – When the activation function is non-linear, then a two-layer neural network can be proven to be a universal function approximator. The identity activation function does not satisfy this property. When multiple layers use the identity activation function, the entire network is equivalent to a single-layer model.
- Continuously differentiable – This property is desirable (RELU is not continuously differentiable and has some issues with gradient-based optimization, but it is still possible) for enabling gradient-based optimization methods. The binary step activation function is not differentiable at 0, and it differentiates to 0 for all other values, so gradient-based methods can make no progress with it.
- Range – When the range of the activation function is finite, gradient-based training methods tend to be more stable, because pattern presentations significantly affect only limited weights. When the range is infinite, training is generally more efficient because pattern presentations significantly affect most of the weights. In the latter case, smaller learning rates are typically necessary.
- Monotonic – When the activation function is monotonic, the error surface associated with a single-layer model is guaranteed to be convex.
- Smooth Functions with a Monotonic derivative – These have been shown to generalize better in some cases.

ReLU layer

ReLU is the abbreviation of Rectified Linear Units. This layer applies the non-saturating activation function $f(x) = \max(0,x)$. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent $f(x) = \tanh(x)$ and the sigmoid function $f(x) = (1+ e^{-x})^{-1}$. ReLU is often preferred to other functions, because it trains the neural network several times faster without a significant penalty to generalisation accuracy.

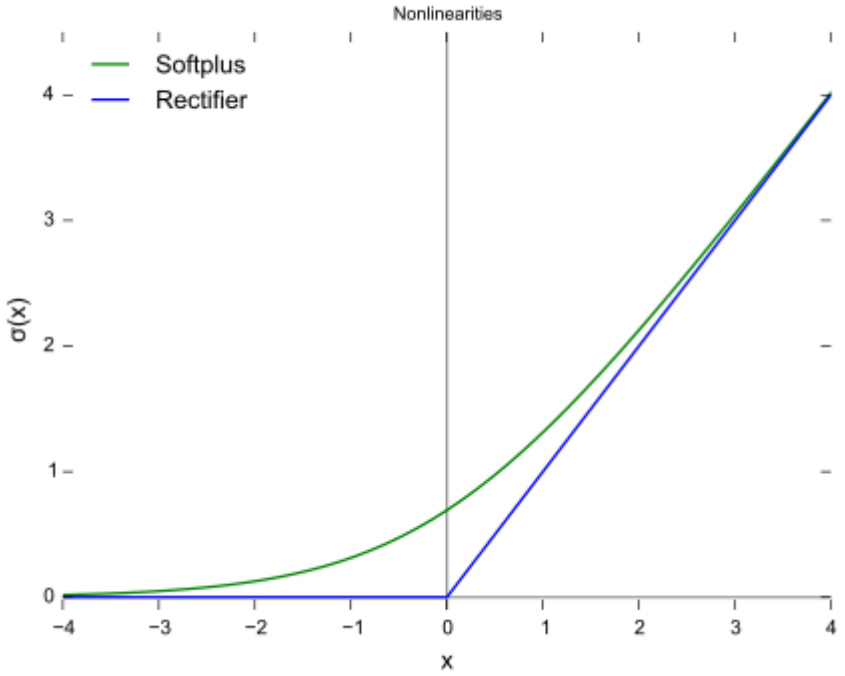


Fig 3.4 ReLu layer curve

3.5 UNSUPERVISED TRAINING

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.

Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore machine is restricted to find the hidden structure in unlabeled data by our-self.

For instance, suppose it is given an image having both dogs and cats which have not seen ever. Thus machine has no any idea about the features of dogs and cat so we can't categorize it in dogs and cats. But it can categorize them according to their similarities, patterns and differences i.e., we can easily categorize the above picture into two parts. First first may contain all pics having **dogs** in it and second part may contain all pics having **cats** in it. Here you didn't learn anything before, means no training data or examples.

Unsupervised learning classified into two categories of algorithms:

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

3.6 AUTOENCODER

An **autoencoder** is a neural network that has three layers: an input layer, a hidden (encoding) layer, and a decoding layer. The network is trained to reconstruct its inputs, which forces the hidden layer to try to learn good representations of the inputs.

An **autoencoder** is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. Recently, the autoencoder concept has become more widely used for learning generative models of data. Some of the most powerful AI in the 2010s has involved sparse autoencoders stacked inside of deep neural networks.

Autoencoders belong to the neural network family, but they are also closely related to PCA (principal components analysis).

Some Key Facts about the autoencoder:

- It is an unsupervised ML algorithm similar to PCA
- It minimizes the same objective function as PCA
- It is a neural network
- The neural network's target output is its input

Types of Autoencoders :

1. Denoising autoencoder

2. Sparse autoencoder

3. Variational autoencoder **(**VAE**)**

4. Contractive autoencoder (CAE)

PURPOSE

An autoencoder learns to compress data from the input layer into a short code, and then uncompress that code into something that closely matches the original data. This forces the autoencoder to engage in dimensionality reduction, for example by learning how to ignore noise. Some architectures use stacked sparse autoencoder layers for image recognition. The first autoencoder might learn to encode easy features like corners, the second to analyze the first layer's output and then encode less local features like the tip of a nose, the third might encode a whole nose, etc., until the final autoencoder encodes the whole image into a code that matches (for example) the concept of "cat". An alternative use is as a generative model: for example, if a system is manually fed the codes it has learned for "cat" and "flying", it may attempt to generate an image of a flying cat, even if it has never seen a flying cat before.

CONVOLUTIONAL AUTOENCODERS

The convolution operator allows filtering an input signal in order to extract some part of its content. Autoencoders in their traditional formulation do not take into account the fact that a signal can be seen as a sum of other signals. Convolutional Autoencoders, instead, use the convolution operator to exploit this observation. They learn to encode the input in a set of simple signals and then try to reconstruct the input from them.

Convolutional AutoEncoders (CAEs) approach the filter definition task from a different perspective: instead of manually engineer convolutional filters we let the model learn the optimal filters that minimize the reconstruction error. These filters can then be used in any other computer vision task.

CAEs are the state-of-art tools for unsupervised learning of convolutional filters. Once these filters have been learned, they can be applied to any input in order to extract features. These features, then, can be used to do any task that requires a compact representation of the input, like classification.

The encoder-decoder models in context of recurrent neural networks (RNNs) are **sequence to sequence mapping models.** An RNN encoder-decoder takes a sequence as input and generates another sequence as output.

For example, a sentence in English can be considered as a sequence (of words) which can be input and a French translation of the sentence is generated as an output which again is a sequence (of words).

As the name suggests, encoder-decoder models consist of two parts: an encoder and a decoder. The **encoder network** is that part of the network that takes the input sequence and maps it to an encoded representation of the sequence. The encoded representation is then used by the **decoder network** to generate an output sequence.

SKIP CONNECTIONS

A skip connection in a neural network is a connection which skips a layer and connects to the next available layer.

Skip architecture as the name suggests skips some layer in the neural network and feeds the output of one layer as the input to the next layer as well as some other layer (instead of only the next layer).

Skip connections help traverse information in deep neural networks. Gradient information can be lost as we pass through many layers, this is called vanishing gradient. Advantages of skip connection are they pass feature information to lower layers so using it to classify minute details becomes easier. As you do maxpooling some amount of spatial information is lost. skip connections can help increasing accuracy of classification because final layer feature will have more information.

CHAPTER: 4

GENERATIVE ADVERSARIAL NETWORK

Generative Adversarial Network (GAN) is a type of construct in neural network technology that offers a lot of potential in the world of artificial intelligence. A generative adversarial network is composed of two neural networks: a generative network and a discriminative network. These work together to provide high-level simulation of conceptual tasks.

In a generative adversarial network, the generative network constructs results from input, and "shows" them to the discriminative network. The discriminative network is supposed to distinguish between authentic and synthetic results given by the generative network.

Experts sometimes describe this as the generative network trying to "fool" the discriminative network, which has to be trained to recognize particular sets of patterns and models. The use of generative adversarial networks is somewhat common in image processing, and in the development of new deep stubborn networks that move toward more high-level simulation of human cognitive tasks. Scientists are looking at the potential that generative adversarial networks have to advance the power of neural networks and their ability to "think" in human ways.

4.1 GENERATIVE VS DISCRIMINATIVE ALGORITHMS

The question a generative algorithm tries to answer is: Assuming this email is spam, how likely are these features? While discriminative models care about the relation between y and x, generative models care about "how you get x." They allow you to capture p(x|y), the probability of x given y, or the probability of features given a class. (That said, generative algorithms can also be used as classifiers. It just so happens that they can do more than categorize input data.)

Another way to think about it is to distinguish discriminative from generative like this:

- Discriminative models learn the boundary between classes
- Generative models model the distribution of individual classes

How GANs Work

One neural network, called the *generator*, generates new data instances, while the other, the *discriminator*, evaluates them for authenticity; i.e. the discriminator decides whether each instance of data it reviews belongs to the actual training dataset or not.

Let's say we're trying to do something more banal than mimic the Mona Lisa. We're going to generate hand-written numerals like those found in the MNIST dataset, which is taken from the real world. The goal of the discriminator, when shown an instance from the true MNIST dataset, is to recognize them as authentic.

Meanwhile, the generator is creating new images that it passes to the discriminator. It does so in the hopes that they, too, will be deemed authentic, even though they are fake. The goal of the generator is to generate passable hand-written digits, to lie without being caught. The goal of the discriminator is to identify images coming from the generator as fake.

Here are the steps a GAN takes:

- The generator takes in random numbers and returns an image.
- This generated image is fed into the discriminator alongside a stream of images taken from the actual dataset.
- The discriminator takes in both real and fake images and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake.

So you have a double feedback loop:

- The discriminator is in a feedback loop with the ground truth of the images, which we know.
- The generator is in a feedback loop with the discriminator.

You can think of a GAN as the combination of a counterfeiter and a cop in a game of cat and mouse, where the counterfeiter is learning to pass false notes, and the cop is learning to detect them. Both are dynamic; i.e. the cop is in training, too (maybe the central bank is flagging bills that slipped through), and each side comes to learn the other's methods in a constant escalation.

- The discriminator network is a standard convolutional network that can categorize the images fed to it, a binomial classifier labeling images as real or fake. The generator is an inverse convolutional network, in a sense: While a standard convolutional classifier takes an image and downsamples it to produce a probability, the generator takes a vector of random noise and upsamples it to an image. The first throws away data through downsampling techniques like maxpooling, and the second generates new data.
- Both nets are trying to optimize a different and opposing objective function, or loss function, in a zero-zum game. This is essentially an actor-critic model. As the discriminator changes its behavior, so does the generator, and vice versa. Their losses push against each other.

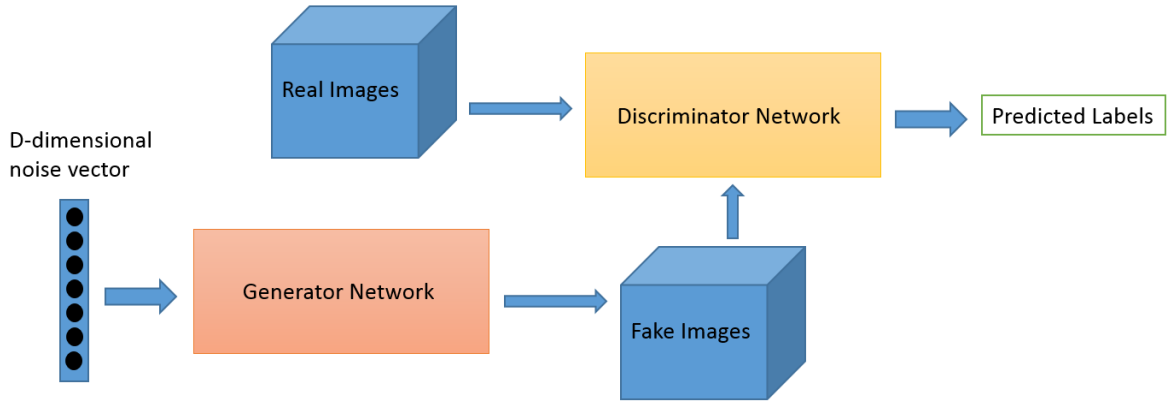


Fig 4.1 Working of GAN

4.2 BACKPROPAGATION

A DNN can be discriminatively trained with the standard backpropagation algorithm. Backpropagation is a method to calculate the gradient of the loss function (produces the cost associated with a given state) with respect to the weights in an ANN.

The weight updates of backpropagation can be done via stochastic gradient descent using the following equation:

$$W_{ij}(t+1) = W_{ij}(t) + \eta(\partial C/\partial W_{ij}) + \xi(t)$$

where,η is the learning rate, C is the cost (loss) function and $\xi(t)$ a stochastic term.

FINAL PROJECT

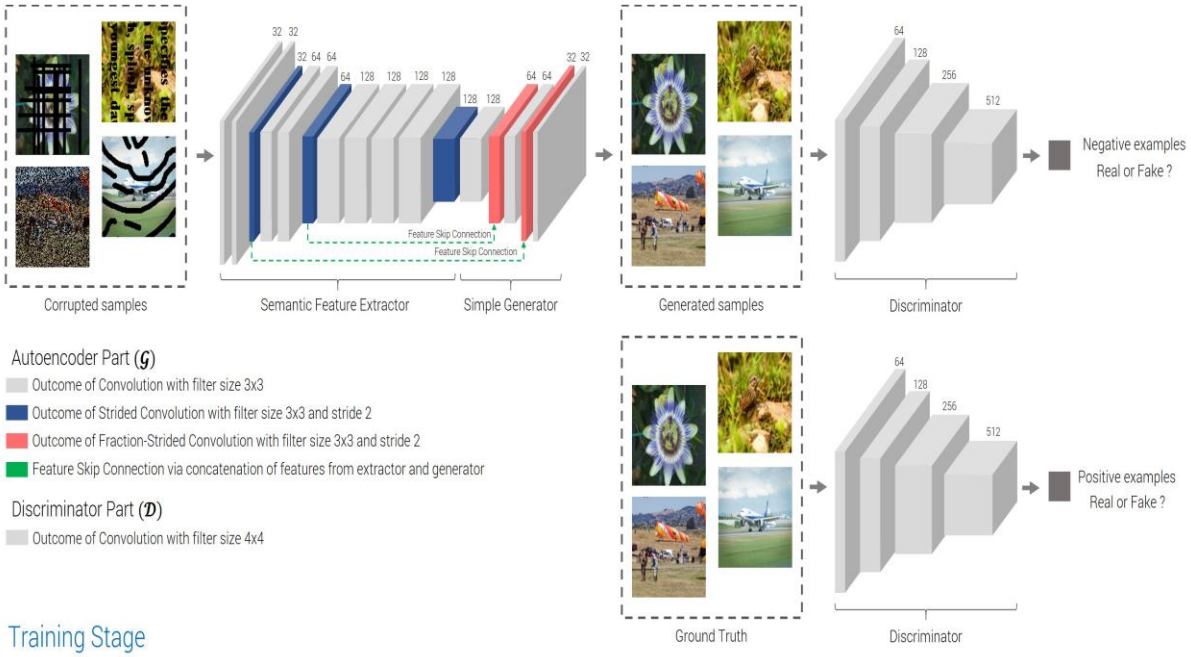
Image impainting and completion are classical problems in computer vision and graphics. The objective is to fill semantic and reasonable contents to the corruptions (missing regions) in an image. Humans can fill the missing regions by the empirical knowledge to the diverse object structures from the real world. Nevertheless, it is not easy for machines to learn a wide variety of structures in natural images and predict what to fill in unknown missing-data regions in images. Thus, it is crucial to know how to learn an image transformation from a corrupted image with missing data to a completed image.

Semantic Feature Extractor: Given a corrupted image I with size H _W as input, we use a part of layers (before conv4-1) from VGG19 architecture as our semantic feature extractor (encoder) to obtain a high-level and semantic feature patch. To preserve image details, we replace all the pooling layer with strided convolutions as the pooling layer has been proved that it tends to lose some image information in the reconstruction based networks.

Simple Generator: Common deep encoder-decoder networks use symmetric structure that extracts features and generates outcome through the same number of layers. However, if the layer goes deeper, it would be difficult to train on GPUs efficiently due to explosion of parameters and memory usage. In addition, the deeper structure in the decoder, the harder to propagate learned feature information from the encoder. Therefore, we construct a simple generator (or decoder) that takes the semantic feature patch as input and then reconstructs a complete image in a short path.

Feature Skip Connections: In image completion problems, corrupted images and output images share a certain amount of low-level features, like prominent information of noncorrupted regions, luminance and resolutions.

Discriminator in GANs related works, it often requires several tricks to train a generator and discriminator jointly. Otherwise, we can easily confront mode collapse and obtain unstable results. So, we follow the same discriminator architecture from that outputs avalue of realism for a given input image.


Corrupted samples
Semantic Feature Extractor
Simple Generator
Feature Skip Connection
Feature Skip Connection
Generated samples
Discriminator
Negative examples
Real or Fake ?
Autoencoder Part ($\mathcal{G}$)
Outcome of Convolution with filter size 3x3
Outcome of Strided Convolution with filter size 3x3 and stride 2
Outcome of Fraction-Strided Convolution with filter size 3x3 and stride 2
Feature Skip Connection via concatenation of features from extractor and generator
Discriminator Part ($\mathcal{D}$)
Outcome of Convolution with filter size 4x4
Ground Truth
Discriminator
Positive examples
Real or Fake ?
Training Stage
Testing Stage
Corrupted Images
Semantic Feature Extractor
Simple Generator
Our results

CODE

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct  1 11:32:35 2018

@author: Kashish
"""

###### Import Libraries ######
import tensorflow as tf
import numpy as np
from glob import glob
import os
import cv2

###### Build a class for model ######
class Model():
    def __init__(self):
        pass

    def new_conv_layer(self, bottom, filter_shape, activation = tf.identity, padding = 'SAME', stride = 1, name = None):
        with tf.variable_scope(name):
            w = tf.get_variable("W", shape = filter_shape, initializer = tf.random_normal_initializer(0., 0.005))
            b = tf.get_variable("b", shape = filter_shape[-1], initializer = tf.constant_initializer(0.))
            conv = tf.nn.conv2d(bottom, w, [1, stride, stride, 1], padding = padding)
```

```
            bias = activation(tf.nn.bias_add(conv, b))
        return bias

    def new_deconv_layer(self, bottom, filter_shape, output_shape, activation = tf.identity, padding = 'SAME', stride = 1, name = None):
        with tf.variable_scope(name):
            W = tf.get_variable("W", shape = filter_shape, initializer = tf.random_normal_initializer(0., 0.005))
            b = tf.get_variable("b", shape = filter_shape[-2], initializer = tf.constant_initializer(0.))
            deconv = tf.nn.conv2d_transpose(bottom, W, output_shape, [1, stride, stride, 1], padding = padding)
            bias = activation(tf.nn.bias_add(deconv, b))
        return bias

    def new_fc_layer(self, bottom, output_size, name):
        shape = bottom.get_shape().as_list()
        dim = np.prod(shape[1:])
        x = tf.reshape(bottom, [-1, dim])
        input_size = dim
        with tf.variable_scope(name):
            w = tf.get_variable("W",shape = [input_size, output_size], initializer = tf.random_normal_initializer(0., 0.005))
            b = tf.get_variable("b", shape = [output_size], initializer = tf.constant_initializer(0.))
            fc = tf.nn.bias_add(tf.matmul(x, w), b)
        return fc

    def leaky_relu(self, bottom, leak=0.1):
        return tf.maximum(leak*bottom, bottom)

    def build_reconstruction( self, images, is_train ):
```

```
    with tf.variable_scope('GEN'):
        # Convolution Layer
        # VGG Model
        conv1_1 = self.new_conv_layer(images, [3,3,3,32], stride=1, name="conv1_1" )
        conv1_1 = tf.nn.elu(conv1_1)
        conv1_2 = self.new_conv_layer(conv1_1, [3,3,32,32], stride=1, name="conv1_2" )
        conv1_2 = tf.nn.elu(conv1_2)
        # MaxPooling-1
        conv1_stride = self.new_conv_layer(conv1_2, [3,3,32,32], stride=2, name="conv1_stride")

        conv2_1 = self.new_conv_layer(conv1_stride, [3,3,32,64], stride=1, name="conv2_1" )
        conv2_1 = tf.nn.elu(conv2_1)
        conv2_2 = self.new_conv_layer(conv2_1, [3,3,64, 64], stride=1, name="conv2_2" )
        conv2_2 = tf.nn.elu(conv2_2)
        # MaxPooling-2
        conv2_stride = self.new_conv_layer(conv2_2, [3,3,64,64], stride=2, name="conv2_stride")

        conv3_1 = self.new_conv_layer(conv2_stride, [3,3,64,128], stride=1, name="conv3_1" )
        conv3_1 = tf.nn.elu(conv3_1)
        conv3_2 = self.new_conv_layer(conv3_1, [3,3,128, 128], stride=1, name="conv3_2" )
        conv3_2 = tf.nn.elu(conv3_2)
        conv3_3 = self.new_conv_layer(conv3_2, [3,3,128,128], stride=1, name="conv3_3" )
        conv3_3 = tf.nn.elu(conv3_3)
        conv3_4 = self.new_conv_layer(conv3_3, [3,3,128, 128], stride=1, name="conv3_4" )
        conv3_4 = tf.nn.elu(conv3_4)
        # MaxPoolin-3
```

```
        conv3_stride = self.new_conv_layer(conv3_4, [3,3,128,128], stride=2, name="conv3_stride") # Final feature map (temporary)

        conv4_stride = self.new_conv_layer(conv3_stride, [3,3,128,128], stride=2, name="conv4_stride") # 16 -> 8
        conv4_stride = tf.nn.elu(conv4_stride)

        conv5_stride = self.new_conv_layer(conv4_stride, [3,3,128,128], stride=2, name="conv5_stride") # 8 -> 4
        conv5_stride = tf.nn.elu(conv5_stride)

        conv6_stride = self.new_conv_layer(conv5_stride, [3,3,128,128], stride=2, name="conv6_stride") # 4 -> 1
        conv6_stride = tf.nn.elu(conv6_stride)

        # Deconvolution Layer with Skip Connections
        deconv5_fs = self.new_deconv_layer( conv6_stride, [3,3,128,128], conv5_stride.get_shape().as_list(), stride=2, name="deconv5_fs")
        debn5_fs = tf.nn.elu(deconv5_fs)

        skip5 = tf.concat([debn5_fs, conv5_stride], 3)
        channels5 = skip5.get_shape().as_list()[3]

        deconv4_fs = self.new_deconv_layer( skip5, [3,3,128,channels5], conv4_stride.get_shape().as_list(), stride=2, name="deconv4_fs")
        debn4_fs = tf.nn.elu(deconv4_fs)

        skip4 = tf.concat([debn4_fs, conv4_stride], 3)
        channels4 = skip4.get_shape().as_list()[3]
```

```
        deconv3_fs = self.new_deconv_layer( skip4, [3,3,128,channels4],
conv3_stride.get_shape().as_list(), stride=2, name="deconv3_fs")
        debn3_fs = tf.nn.elu(deconv3_fs)

        skip3 = tf.concat([debn3_fs, conv3_stride], 3)
        channels3 = skip3.get_shape().as_list()[3]

        deconv2_fs = self.new_deconv_layer( skip3, [3,3,64,channels3],
conv2_stride.get_shape().as_list(), stride=2, name="deconv2_fs")
        debn2_fs = tf.nn.elu(deconv2_fs)

        skip2 = tf.concat([debn2_fs, conv2_stride], 3)
        channels2 = skip2.get_shape().as_list()[3]

        deconv1_fs = self.new_deconv_layer( skip2, [3,3,32,channels2],
conv1_stride.get_shape().as_list(), stride=2, name="deconv1_fs")
        debn1_fs = tf.nn.elu(deconv1_fs)

        skip1 = tf.concat([debn1_fs, conv1_stride], 3)
        channels1 = skip1.get_shape().as_list()[3]

        recon = self.new_deconv_layer( skip1, [3,3,3,channels1],
images.get_shape().as_list(), stride=2, name="recon")

    return recon

###### Global Variables ######

pen_size = 1
img_idx = 0
drawing = False
```

```
ix, iy = -1, -1
vis_size = 320
blank_size = 20

###### Utilities ######

def nothing(x):
    pass

def draw(event, x, y, flags, param):
    global ix, iy, drawing

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix, iy = x, y
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            cv2.line(img, (ix, iy), (x, y), (255, 255, 255), pen_size)
            ix, iy = x, y
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        cv2.line(img, (ix, iy), (x, y), (255, 255, 255), pen_size)

def masking(img):
    mask = (np.array(img[:,:,0]) == 255) & (np.array(img[:,:,1]) == 255) &
(np.array(img[:,:,2]) == 255)
    mask = np.dstack([mask,mask,mask]);
    return (True ^ mask) * np.array(img)

###### Image Pre-Processing ######
```

```
img_paths = []
img_paths.extend( sorted(glob(os.path.join('testimages/', '*.bmp'))) )
img_ori = cv2.imread( img_paths[img_idx]) / 255.
img = img_ori
empty = np.zeros((vis_size, vis_size, 3))
blank = np.zeros((vis_size, blank_size, 3)) + 1
text_region = np.zeros((blank_size, 2*vis_size + blank_size, 3)) + 1.
recon_img = empty

cv2.namedWindow("Window", cv2.WINDOW_NORMAL)
cv2.setMouseCallback('Window', draw)

###### Tensorflow Session ######

sess = tf.Session()

pretrained_model_path = 'model_mscoco'

is_train = tf.placeholder( tf.bool )
images_tf = tf.placeholder( tf.float32, shape=[1, vis_size, vis_size, 3], name="images")

model = Model()
reconstruction_ori = model.build_reconstruction(images_tf, is_train)

saver = tf.train.Saver(max_to_keep=100)
saver.restore( sess, pretrained_model_path )

###### Main Function Loop ######
while (1):
    view = np.hstack((img, blank,recon_img[:,:,[2,1,0]]))
    window = np.vstack( (view, text_region) )
```

```
    cv2.imshow('Window', window)
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
    elif k == 99:
        masked_input = masking(img)
        masked_input = masked_input[:,:,[2,1,0]]
        shape3d = np.array( masked_input ).shape
        model_input = np.array( masked_input ).reshape(1, shape3d[0], shape3d[1], shape3d[2])
        model_output = sess.run(reconstruction_ori,feed_dict={images_tf: model_input, is_train: False})
        recon_img = np.array(model_output)[0,:,:,:].astype(float)
        cv2.imwrite( os.path.join('results', img_paths[img_idx][21:35]), ((recon_img[:,:,[2,1,0]]) * 255) )
        cv2.imwrite( os.path.join('inputs', img_paths[img_idx][21:35]), ((img) * 255) )
cv2.destroyAllWindows()
```

RESULT

IMAGE 1:

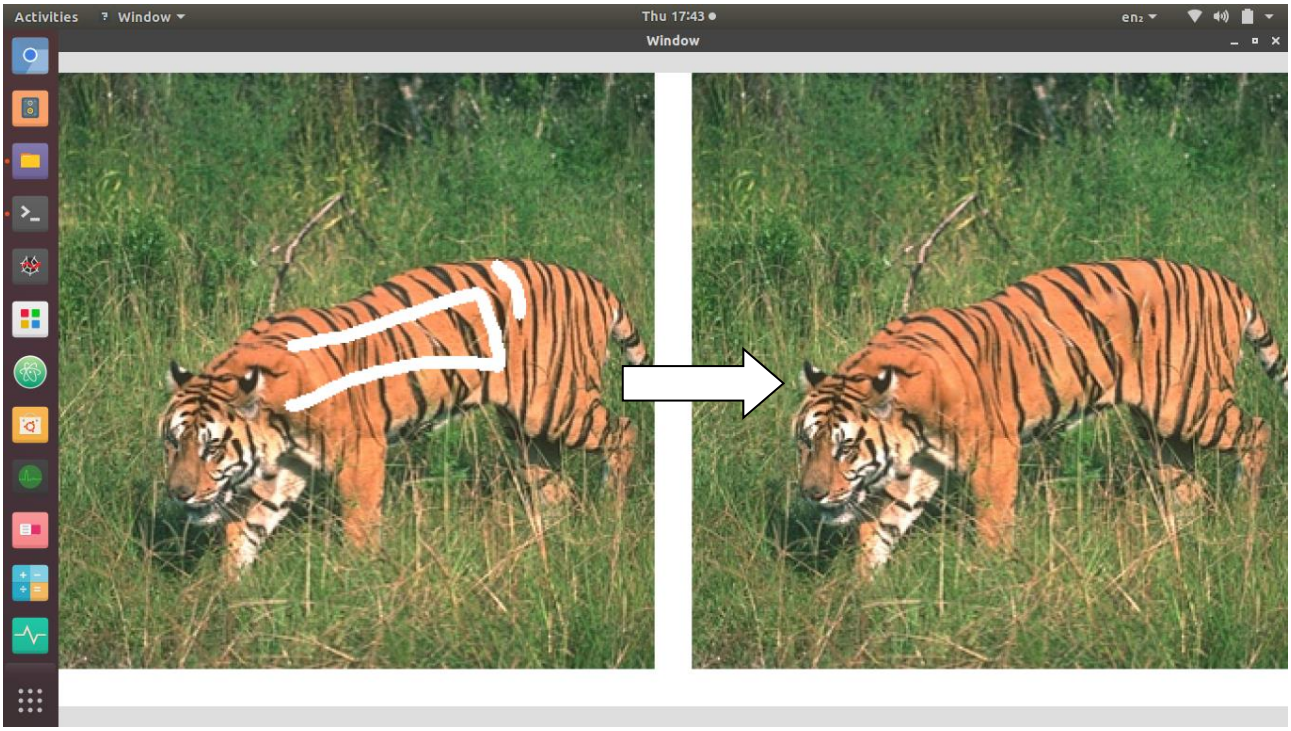


IMAGE 2:

CONCLUSION

With recent advances in deep learning AI algorithms are developed to such extents that not only they can compete but they can even defeat humans. As a new framework Generative adversarial network GAN also advances its usage for reconstructing and producing better synthetic images than previous models.

In this project, a noisy or damaged image is reconstructed with the use of generative adversarial network. The use of sigmoid function is also studied in our work i.e., how it can be used to reconstruct an image. Further, looking towards more advancement in this area.

Image processing is an important area to work upon nowadays, as everything is going digitalised, which increases its importance to study such possible technologies.

Various other uses of GAN can be defined as following:

- Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network
- Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks
- Generative Adversarial Text to Image Synthesis

REFERENCES

1. https://www.techopedia.com/definition/32515/generative-adversarial-network-gan
2. https://towardsdatascience.com/generative-adversarial-networks-explained-34472718707a
3. https://ieeexplore.ieee.org/document/8168181
4. https://www.tutorialspoint.com/dip/image_processing_introduction.htm
5. https://stats.stackexchange.com/questions/238381/what-is-the-origin-of-the-autoencoder-neural-networks
6. https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi