PRATHYUSHA

ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

REGULATION 2021 – V SEMESTER

CCS349- IMAGE AND VIDEO ANALYTICS

CCS349 IMAGE AND VIDEO ANALYTICS

COURSE OBJECTIVES:

- To understand the basics of image processing techniques for computer vision.
- To learn the techniques used for image pre-processing.
- To discuss the various object detection techniques.
- To understand the various Object recognition mechanisms.
- To elaborate on the video analytics techniques.

UNIT I INTRODUCTION 6

Computer Vision – Image representation and image analysis tasks - Image representations – digitization – properties – color images – Data structures for Image Analysis - Levels of image data representation - Traditional and Hierarchical image data structures.

UNIT II IMAGE PRE-PROCESSING 6

Local pre-processing - Image smoothing - Edge detectors - Zero-crossings of the second derivative - Scale in image processing - Canny edge detection - Parametric edge models - Edges in multispectral images - Local pre-processing in the frequency domain - Line detection by local preprocessing operators - Image restoration.

UNIT III OBJECT DETECTION USING MACHINE LEARNING 6

Object detection– Object detection methods – Deep Learning framework for Object detection– bounding box approach-Intersection over Union (IoU) –Deep Learning Architectures-R-CNN-Faster R-CNN-You Only Look Once(YOLO)-Salient features-Loss Functions-YOLO architectures

UNIT IV FACE RECOGNITION AND GESTURE RECOGNITION 6

Face Recognition-Introduction-Applications of Face Recognition-Process of Face Recognition- DeepFace solution by Facebook-FaceNet for Face Recognition- Implementation using FaceNetGesture Recognition.

UNIT V VIDEO ANALYTICS 6

Video Processing – use cases of video analytics-Vanishing Gradient and exploding gradient problem - RestNet architecture-RestNet and skip connections-Inception Network-GoogleNet architecture Improvement in Inception v2-Video analytics-RestNet and Inception v3.

30 PERIODS

30 PERIODS

LIST OF EXERCISES

1. Write a program that computes the T-pyramid of an image.
2. Write a program that derives the quad tree representation of an image using the homogeneity criterion of equal intensity
3. Develop programs for the following geometric transforms: (a) Rotation (b) Change of scale (c) Skewing (d) Affine transform calculated from three pairs of corresponding points (e) Bilinear transform calculated from four pairs of corresponding points.
4. Develop a program to implement Object Detection and Recognition
5. Develop a program for motion analysis using moving edges, and apply it to your image sequences.
6. Develop a program for Facial Detection and Recognition
7. Write a program for event detection in video surveillance system

TOTAL: 60 PERIODS

COURSE OUTCOMES:

At the end of this course, the students will be able to:

CO1: Understand the basics of image processing techniques for computer vision and video analysis.

CO2: Explain the techniques used for image pre-processing.

CO3: Develop various object detection techniques.

CO4: Understand the various face recognition mechanisms.

CO5: Elaborate on deep learning-based video analytics.

TEXT BOOK:

1. Milan Sonka, Vaclav Hlavac, Roger Boyle, “Image Processing, Analysis, and Machine Vision”, 4nd edition, Thomson Learning, 2013.

2. Vaibhav Verdhan, 2021, Computer Vision Using Deep Learning Neural Network Architectures with Python and Keras, Apress 2021(UNIT-III,IV and V)

REFERENCES

1. Richard Szeliski, “Computer Vision: Algorithms and Applications”, Springer Verlag London Limited,2011.
2. Caifeng Shan, FatihPorikli, Tao Xiang, Shaogang Gong, “Video Analytics for Business Intelligence”, Springer, 2012.
3. D. A. Forsyth, J. Ponce, “Computer Vision: A Modern Approach”, Pearson Education, 2003.
4. E. R. Davies, (2012), “Computer & Machine Vision”, Fourth Edition, Academic Press.

CCS349 IMAGE AND VIDEO ANALYTICS

UNIT I : INTRODUCTION

Computer Vision – Image representation and image analysis tasks - Image representations – digitization – properties – color images – Data structures for Image Analysis - Levels of image data representation - Traditional and Hierarchical image data structures.

1.1. INTRODUCTION:

Vision allows humans to perceive and understand the world surrounding them, while computer vision aims to duplicate the effect of human vision by electronically perceiving and understanding an image.

Giving computers the ability to see is not an easy task—we live in a three-dimensional (3D) world, and when computers try to analyze objects in 3D space, the visual sensors available (e.g., TV cameras) usually give two-dimensional (2D) images, and this projection to a lower number of dimensions incurs an enormous loss of information. Sometimes, equipment will deliver images that are 3D but this may be of questionable value: analyzing such datasets is clearly more complicated than 2D, and sometimes the
'three-dimensionality' is less than intuitive to us . . . terahertz scans are an example of this. Dynamic scenes such as those to which we are accustomed, with moving objects or a moving camera, are increasingly common and represent another way of making computer vision more complicated.

Figure 1.1 could be witnessed in thousands of farmyards in many countries, and serves to illustrate just some of the problems that we will face.

Figure1.1 : A frame from a video of a typical farm- yard scene: the cow is one of a number walking naturally from right to left.

There are many reasons why we might wish to study scenes such as this, which are attractively simple *to us*. The beast is moving slowly, it is clearly black and white, its movement is rhythmic, etc.; however, automated analysis is very troubled—in fact, the animal's boundary is often very difficult to distinguish clearly

from the background, the motion of the legs is self-occluding and (subtly) the concept of ‘cow-shaped’ is not something easily encoded. The application from which this picture was taken made use of many of the algorithms: starting at a low level, moving features were identified and grouped. A ‘training phase’ taught the system what a cow might look like in various poses (see Figure 1.2), from which a model of a ‘moving’ cow could be derived (see Figure 1.3).

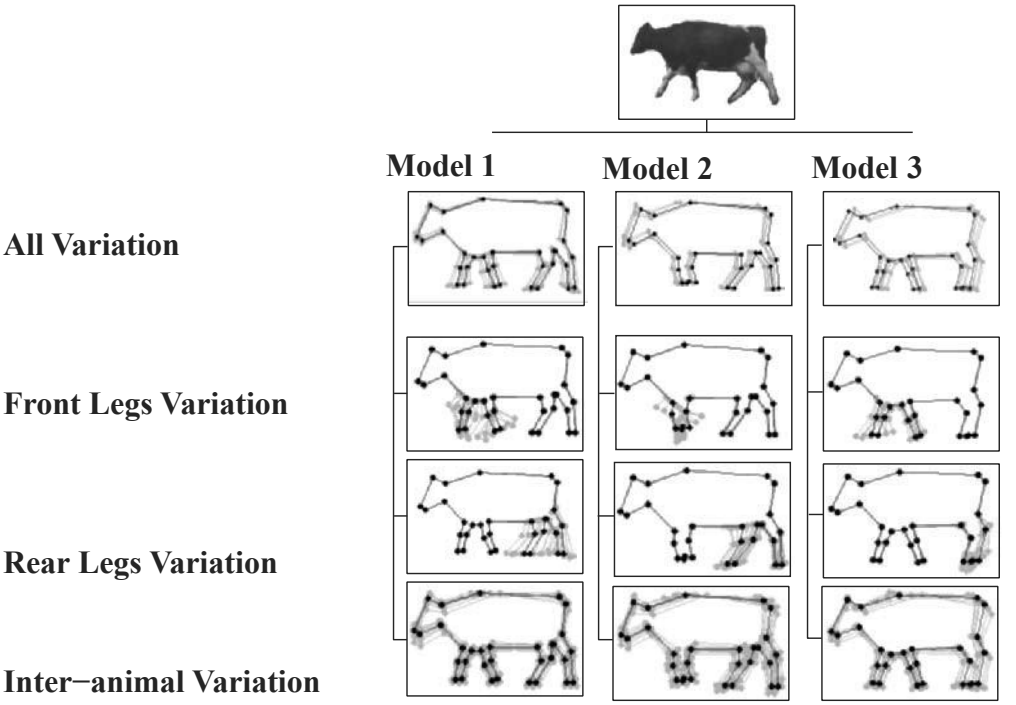


Figure 1.2: Various models for a cow Shadow: a straight-line boundary approximation has been learned from training data and is able to adapt to different animals and different forms of occlusion.

These models could then be fitted to new (‘unseen’) video sequences. Crudely, at this stage anomalous behavior such as lameness could be detected by the model failing to fit properly, or well.

Thus we see a sequence of operations—image capture, early processing, segmentation, model fitting, motion prediction, qualitative/quantitative conclusion—that is characteristic of image understanding and computer vision problems. Each of these phases (which may not occur sequentially!) may be addressed by a number of algorithms which we

---

shall cover in due course.

The application was serious; there is a growing need in modern agriculture for automatic monitoring of animal health, for example to spot lameness. A limping cow is trivial for a human to identify, but it is very challenging to do this automatically.

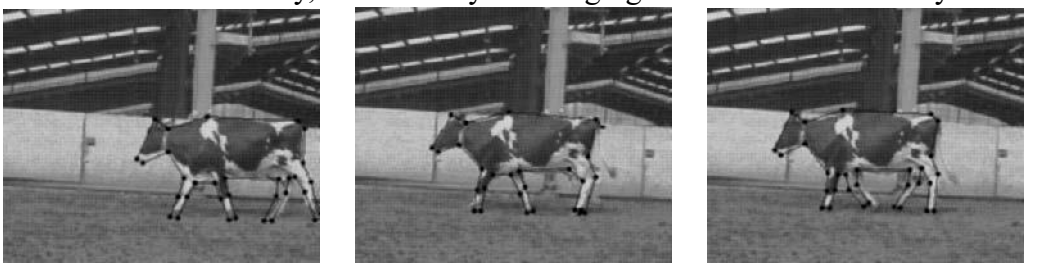


Figure 1.3: Three frames from a cow sequence: notice the model can cope with partial occlusion as the animal enters the scene, and the different poses exhibited.

This example is relatively simple to explain, but serves to illustrate that many computer vision techniques use the results and methods of mathematics, pattern recognition, artificial intelligence (AI), psycho-physiology, computer science, electronics, and other scientific disciplines.

1.2. COMPUTER VISION:

Computer vision is a **field of artificial intelligence (AI) enabling computers to derive information** from images, videos and other inputs.

Why is computer vision difficult?

Six complex landscape of computer vision are.

1) **Loss of information in 3D→ ❼ 2D** is a phenomenon which occurs in typical image capture devices such as a camera or an eye. Their geometric properties have been approximated by a pinhole model for centuries (a box with a small hole in it—a

'camera obscura' in Latin). This physical model corresponds to a mathematical model of perspective projection; Figure 1.4 summarizes the principle. The projective transformation maps points along rays but does not preserve angles and collinearity.

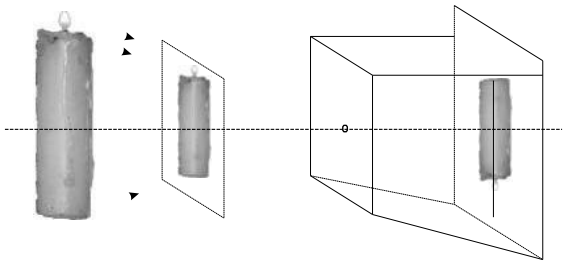


Figure 1.4: The pinhole model of imaging geometry does not distinguish size of objects.

The main trouble with the pinhole model and a single available view is that the projective transformation sees a small object close to the camera in the same way as a big object remote from the camera. In this case, a human needs a 'yardstick' to guess the actual size of the object which the computer does not have. 2)

The interpretation of images, which is unconsciously solved by humans, serves as the fundamental basis for computer vision. When a human tries to understand an image then previous knowledge and experience is brought to the current observation. Human ability to reason allows representation of long-gathered knowledge, and its use to solve new problems. Artificial intelligence has worked for decades to endow computers with the capability to understand observations; while progress has been tremendous, the practical ability of a machine to understand observations remains very limited.

From the mathematical logic and/or linguistics point of view, image interpretation can be seen as a mapping

$$\textit{interpretation: image data} \longrightarrow \textit{model}\ .$$

The (logical) model means some specific world in which the observed objects make sense. Examples might be nuclei of cells in a biological sample, rivers in a satellite image, or parts in an industrial process being checked for quality. There may be several interpretations of the same image(s). Introducing interpretation to computer vision allows us to use concepts from mathematical logic, linguistics as syntax (rules describing correctly formed expressions), and semantics (study of meaning). Considering observations (images) as an instance of formal expressions, semantics studies relations between expressions and their meanings. The interpretation of image(s) in computer vision can be understood as an instance of semantics.

Practically, if the image understanding algorithms know into which particular domain the observed world is constrained, then automatic analysis can be used for complicated problems.

3) **Noise** is inherently present in each measurement in the real world. Its existence calls for mathematical tools which are able to cope with uncertainty; an example is probability theory. Of course, more complex tools make the image analysis much more complicated compared to standard (deterministic) methods.
4) **Too much data.** Images are big, and video—increasingly the subject of vision applications–correspondingly bigger. Technical advances make processor and memory requirements much less of a problem than they once were, and much can be achieved with consumer level products. Nevertheless, efficiency in problem solutions is still important and many applications remain short of real-time performance.
5) **Brightness measured** in images is given by complicated image formation physics.

   The radiance (brightness, image intensity) depends on the irradiance≈ **(light source type, intensity and position), the observer's position, the surface local geometry, and the surface reflectance properties.** The inverse tasks are ill-posed—for example, to reconstruct local surface orientation from intensity variations. For this reason, imagecapture physics is usually avoided in practical attempts at image understanding. Instead, a direct link between the appearance of objects in scenes and their interpretation is sought.
6) **Local window vs. need for global view.** Commonly, image analysis algorithms analyze a particular storage bin in an operational memory (e.g., a pixel in the image) and its local neighborhood; the computer sees the image through a keyhole; this makes it very difficult to understand more global context. This problem has a long tradition in artificial intelligence: in the 1980s McCarthy argued that formalizing context was a crucial step toward the solution of the problem of generality. It is often very difficult to interpret an image if it is seen only locally or if only a few local keyholes are available. Figure 1.5 illustrates this pictorially. How context is taken into account is an important facet of image analysis.

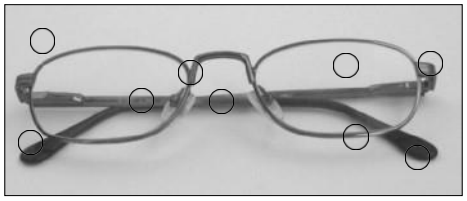


Figure 1.5: Illustration of the world seen through several keyholes providing only a local context.

It is very difficult to guess what object is depicted; the complete image is shown in Figure 1.6.

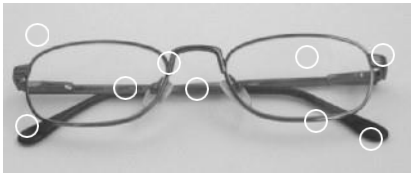


Figure 1.6: It is easy for humans to interpret an image if it is seen globally: compare to Figure 1.5.

1.3 IMAGE REPRESENTATION AND IMAGE ANALYSIS TASKS:

Image understanding by a machine can be seen as an attempt to find a relation between input image(s) and previously established models of the observed world. Transition from the input image(s) to the model reduces the information contained in the image to relevant information for the application domain. This process is usually divided into several steps and several levels representing the image are used.

The bottom layer contains raw image data and the higher levels interpret the data. Computer vision designs these intermediate representations and algorithms serving to establish and maintain relations between entities within and between layers.

Image representation can be roughly divided according to data organization into four levels.

The boundaries between individual levels are inexact, and more detailed divisions are also proposed in the literature. Figure 1.7 suggests a bottom up approach, from signals with almost no abstraction, to the highly abstract description needed for image understanding. The flow of information does not need to be unidirectional; often feedback loops are introduced which allow the modification of algorithms according to intermediate results.

This hierarchy of image representation and related algorithms is frequently categorized in an even simpler way—*low-level* image processing and *high-level* image understanding.

Low-level processing methods usually use very little knowledge about the content of images. In the case of the computer knowing image content, it is usually provided by high-level algorithms or directly by a human who understands the problem domain.

Low- level methods may include image compression, pre-processing methods for noise filtering, edge extraction, and image sharpening. Low- level image processing uses data which resemble the input image; for example, an input image captured by a TV camera is 2D in nature, being described by an image function $f(x, y)$ whose value is usually brightness depending on the co-ordinates x, y of the location in the image.

If the image is to be processed using a computer it will be digitized first, after which it may be represented by a rectangular matrix with elements corresponding to the brightness at appropriate image locations.

More probably, it will be presented in color, usually three channels: red, green and blue. Very often, such a data set will be part of a video stream with an associated frame rate. Nevertheless, the raw material will be a set or sequence of matrices which represent the inputs and outputs of low-level image processing.

High-level processing is based on knowledge, goals, and plans of how to achieve those goals, and artificial intelligence methods are widely applicable. High-level computer vision tries to imitate human cognition and the ability to make decisions according to the information contained in the image.

In the example described, high-level knowledge would be related to the 'shape' of a cow and the subtle interrelationships between the different parts of that shape, and their (inter-) dynamics.

High-level vision begins with some form of formal model of the world, and then the 'reality' perceived in the form of digitized images is compared to the model. A match is attempted, and when differences emerge, partial matches (or subgoals) are sought that overcome them; the computer switches to low-level image processing to find information needed to update the model. This process is then repeated iteratively, and 'understanding' an image thereby becomes a co-operation between top-down and bottom-up processes. A feedback loop is introduced in which high-level partial results create tasks for low-level image processing, and the iterative image understanding process should eventually converge to the global goal.

Computer vision is expected to solve very complex tasks, the goal being to obtain similar results to those provided by biological systems. To illustrate the complexity of these tasks, consider Figure 1.8 in which a particular image representation is presented— the value on the vertical axis gives the brightness of its corresponding location in the [gray-scale] image. Consider what this image might be before looking at Figure 1.9, which is a rather more common representation of the same image.

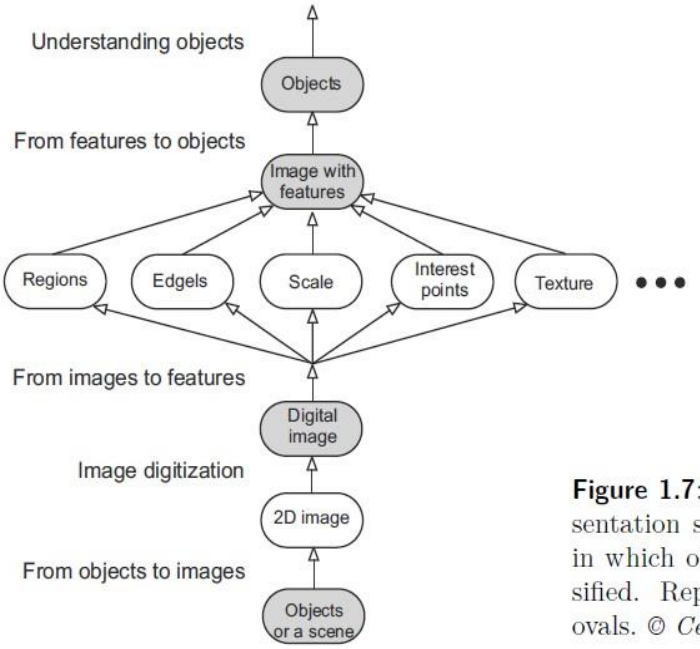


Figure 1.7: Four possible levels of image representation suitable for image analysis problems in which objects have to be detected and classified. Representations are depicted as shaded ovals. © *Cengage Learning 2015.*

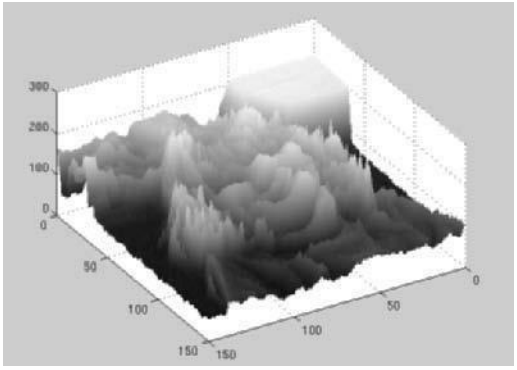


Figure 1.8: An unusual image representation.

Figure 1.9: Another representation of Figure 1.8.

Both representations contain exactly the same information, but for a human observer it is very difficult to find a correspondence between them, and without the second, it is unlikely that one would recognize the face of a child. The point is that a lot of a priori knowledge is used by humans to interpret the images; the machine only begins with an array of numbers and so will be attempting to make identifications and draw conclusions from data that to us are more like Figure 1.8 than Figure 1.9. Increasingly, data capture equipment is providing very large data sets that do *not* lend themselves to straightforward interpretation by humans—we have already mentioned terahertz imaging as an example.

Internal image representations are not directly understandable—while the computer is able to process local parts of the image, it is difficult for it to locate global knowledge. General knowledge, domain-specific knowledge, and information extracted from the image will be essential in attempting to 'understand' these arrays of numbers.

Low-level computer vision techniques overlap almost completely with digital image processing, which has been practiced for decades. The following sequence of processing steps is commonly seen:

An image is captured by a sensor (such as a camera) and digitized; then the computer suppresses noise (image pre-processing) and may be enhances some object features which are relevant to understanding the image. Edge extraction is an example of processing carried out at this stage.

Image segmentation is the next step, in which the computer tries to separate objects from the image background and from each other. Total and partial segmentation may be distinguished; total segmentation is possible only for very simple tasks, an example being the recognition of dark non-touching objects from a light background. For example, in analyzing images of printed text (an early step in optical character recognition, OCR) even this superficially simple problem is very hard to solve without error. In more complicated problems (the general case), low-level image processing techniques handle the partial segmentation tasks, in which only the cues which will aid further high-level

processing are extracted. Often, finding parts of object boundaries is an example of lowlevel partial segmentation.

Object description and classification in a totally segmented image are also understood as

part of low-level image processing. Other low-level operations are image compression, and techniques to extract information from (but not *understand*) moving scenes.

Low-level image processing and high-level computer vision differ in the data used.

Low-level data are comprised of original images represented by matrices composed of brightness (or similar) values, while high-level data originate in images as well, but only those data which are relevant to high-level goals are extracted, reducing the data quantity considerably.

High-level data represent knowledge about the image content—for example, object size, shape, and mutual relations between objects in the image. High-level data are usually expressed in symbolic form.

Many low-level image processing methods were proposed in the 1970s or earlier: research is trying to find more efficient and more general algorithms and is implementing them on more technologically sophisticated equipment, in particular, parallel machines (including GPU's) are being used to ease the computational load. The requirement for better and faster algorithms is fuelled by technology delivering larger images (better spatial or temporal resolution), and color.

A complicated and so far unsolved problem is how to order low-level steps to solve a specific task, and the aim of automating this problem has not yet been achieved. It is usually still a human operator who finds a sequence of relevant operations, and domain-specific knowledge and uncertainty cause much to depend on this operator's intuition and previous experience.

High-level vision tries to extract and order image processing steps using all available knowledge—image understanding is the heart of the method, in which feedback from high-level to low-level is used. Unsurprisingly this task is very complicated and computationally intensive.

Consider *3D vision problems* for a moment. We adopt the user's view, i.e., what tasks performed routinely by humans would be good to accomplish by machines. What is the relation of these 3D vision tasks to low-level (image processing) and high-level (image analysis) algorithmic methods? There is no widely accepted view in the academic community. Links between (algorithmic) components and representation levels are tailored to the specific application solved, e.g., navigation of an autonomous vehicle. These applications have to employ specific knowledge about the problem solved to be

competitive with tasks which humans solve. Many researchers in different fields work on related problems and research in 'cognitive systems' could be the key which may disentangle the complicated world of perception which includes also computer vision.

Figure 1.10 depicts several 3D vision tasks and algorithmic components expressed on different abstraction levels. In most cases, the bottom-up and top-down approach is adopted to fulfill the task.

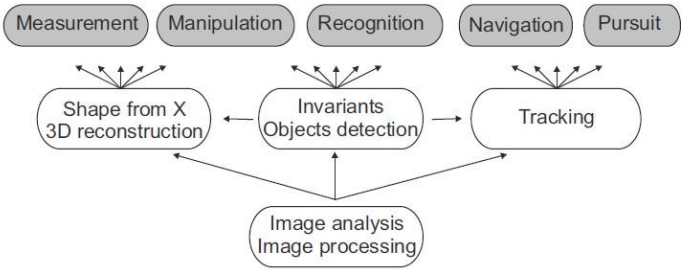


Figure 1.10: Several 3D computer vision tasks from the user's point of view are on the upper line (filled). Algorithmic components on different hierarchical levels support it in a bottom-up fashion.

1.4. IMAGE, ITS REPRESENTATIONS AND PROPERTIES

Image representations:

Mathematical models are often used to describe images.

A monochrome or monochromatic image, object or palette is composed of one color (or values of one color). Images using only shades of grey are called grayscale (typically digital) or black-and-white (typically analog).

A scalar function might be sufficient to describe a monochromatic image, while vector functions may be used to represent color images consisting of three component colors.

Functions are categorized as

- **Continuous**
- **Discrete**
- D**ig ital**.

A continuous function has continuous domain and range; if the domain set is discrete, then we have a discrete function; if the range set is also discrete, then we have a digital function. Many of these functions will be linear, and correspondingly simple to deal with.

Image can be modeled by a continuous function of two variables $f(x, y)$ where (x, y) are co-ordinates in a plane, or perhaps three variables $f(x, y, t)$, where t is time. This model is reasonable in the great majority of applications. Nevertheless, it is worth realizing that an 'image' may be acquired in many ways. We shall note often that color is the norm, even when algorithms are presented for monochromatic images, but we do not need to constrain ourselves to the visible spectrum. Infra-red cameras are now very common (for example, for night-time surveillance). Other parts of the electro-magnetic [EM] spectrum may also be used; microwave imaging, for example, is becoming widely available. Further, image acquisition outside the EM spectrum is also common: in the medical domain, datasets are generated via magnetic resonance (MR), X-ray computed tomography (CT), ultrasound etc. All of these approaches generate large arrays of data requiring analysis and understanding and with increasing frequency these arrays are of 3 or more dimensions.

The continuous image function

The (gray-scale) image function values correspond to brightness at image points. The function value can express other physical quantities as well (temperature, pressure distribution, distance from the observer, etc.). **Brightness** integrates different optical quantities—using brightness as a basic quantity allows us to avoid the complicated process of image formation.

The image on the retina or on a camera sensor is intrinsically two-dimensional (2D). We shall call such an image bearing information about brightness points an **intensity image**. The 2D image on the imaging sensor is commonly the result of projection of a three-dimensional (3D) scene. The simplest mathematical model for this is a pin-hole camera.

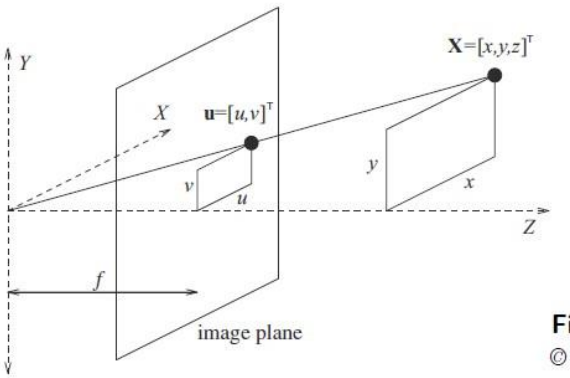


Figure 2.1: Perspective projection geometry.

The image plane is the plane that contains the object's projected image.
The 2D intensity image is the result of a **perspective projection** of the 3D scene, which is modeled by the image captured by a pin-hole camera illustrated in Figure 2.1. In this figure, the image plane has been reflected with respect to the *XY* plane in order not to get a mirrored image with negative co-ordinates. The quantities x, y, and z are coordinates of the point **X** in a 3D scene, and ***f* is the distance from the pinhole to the image plane. *f* is commonly called the focal length** because in lenses it has a similar meaning. The projected point **u** has co-ordinates (u, v) in the 2D image plane, which can easily be derived from similar triangles.

$$u = \frac{x f}{z}, \qquad v = \frac{y f}{z}. \tag{2.1}$$

A non-linear perspective projection is often approximated by a linear **parallel** (or **orthographic) projection**, where f. Implicitly, x says that the orthographic projection is a limiting case$^{\rightarrow \infty}$ of the perspective$^{\rightarrow \infty}$ projection for faraway objects.

When 3D objects are mapped into the camera plane by perspective projection, a lot of information disappears because such a transform is not one-to-one.

Recovering information lost by perspective projection is only one, problem of computer vision—another is understanding image brightness.

The only information available in an intensity image is the brightness of the appropriate pixel, which is dependent on a number of independent factors such as object surface reflectance properties (given by the surface material, microstructure, and marking), illumination properties, and object surface orientation with respect to viewer and light source.

It is a non-trivial and again ill-posed problem to separate these components when trying to recover the 3D geometry of an object from the intensity image.

Some applications work with 2D images directly—for example, an image of a flat specimen viewed by a microscope with transparent illumination, a character drawn on a sheet of paper, the image of a fingerprint, etc. Many basic and useful methods used in digital image analysis do not therefore depend on whether the object was originally 2D or 3D.

Image processing often deals with **static** images, in which time is constant. A monochromatic static image is represented by a continuous image function $f(x, y)$ whose arguments are coordinates in the plane.

Computerized image processing uses digital image functions which are usually rep resented by matrices, so co-ordinates are natural numbers. The domain of the image function is a region R in the plane

$$R = (x, y),\ 1 \leq x \leq x_m,\ 1 \leq y \leq y_n \ , \qquad (2.2)$$

where x_m, y_n represent the maximal co-ordinates. The function has a limited domain—infinite summation or integration limits can be used, as the image function value is zero outside the domain R. The customary orientation of co-ordinates in an image is in the normal Cartesian fashion (horizontal x-axis, vertical y-axis, origin bottom-left), although the (*row, column*, origin top-left) orientation used in matrices is also often used.

The range of image function values is also limited; by convention, in monochromatic images the lowest value corresponds to black and the highest to white. Brightness values bounded by these limits are **gray-levels.** The quality of a digital image grows in proportion to the spatial, spectral, radiometric, and time resolutions. The **spatial resolution** is given by the proximity of image samples in the image plane; **spectral resolution** is given by the bandwidth of the light frequencies captured by the sensor; **radiometric resolution** corresponds to the number of distinguishable gray-levels; and **time resolution** is given by the interval between time samples at which images are captured. The question of time resolution is important in dynamic image analysis, where time sequences of images are processed.

Images $f(x, y)$ can be treated as deterministic functions or as realizations of stochastic processes. Mathematical tools used in image description have roots in linear system theory, integral transforms, discrete mathematics, and the theory of stochastic processes.
Mathematical transforms usually assume that the image function $f(x, y)$ is 'well- behaved', meaning that the function is integrable, has an invertible Fourier transform, etc.

Image digitization

An image to be processed by computer must be represented using an appropriate discrete data structure, for example, a matrix.
An image captured by a sensor is expressed as a continuous function $f(x, y)$ of two coordinates in the plane.
Image digitization means that the function $f(x, y)$ is **sampled** into a matrix with M rows and N columns. Image **quantization** assigns to each continuous sample an integer value—the continuous range of the image function $f(x, y)$ is split into K intervals. The finer the sampling (i.e., the larger M and N) and quantization (the larger K), the better the approximation of the continuous image function $f(x, y)$ achieved.

Image function sampling poses two questions. First, the sampling period should be determined—this is the distance between two neighboring sampling points in the image. Second, the geometric arrangement of sampling points (sampling grid) should be set.

Sampling

A continuous image is digitized at **sampling points**. These sampling points are ordered in the plane, and their geometric relation is called the **grid**. The digital image is then a data structure, usually a matrix. Grids used in practice are usually square (Figure 2.2a) or hexagonal (Figure 2.2b). It is important to distinguish the grid from the raster; the **raster** is the grid on which a neighborhood relation between points is defined.

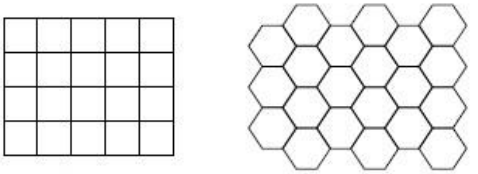


Figure 2.2: (a) Square grid. (b) Hexagonal grid.

One infinitely small sampling point in the grid corresponds to one picture element also called a **pixel** or **image element** in the digital image; in a three-dimensional image, an image element is called a **voxel** (volume element). The set of pixels together covers the entire image; however, the pixel captured by a real digitization device has finite size (since the sampling function is not a collection of ideal Dirac impulses but a collection of limited impulses). The pixel is a unit which is not further divisible from the image analysis point of view. We shall often refer to a pixel as a 'point'.

Quantization

A value of the sampled image $f_s(j\ \Delta x,\ k\ \Delta y)$ is expressed as a digital value in image processing.

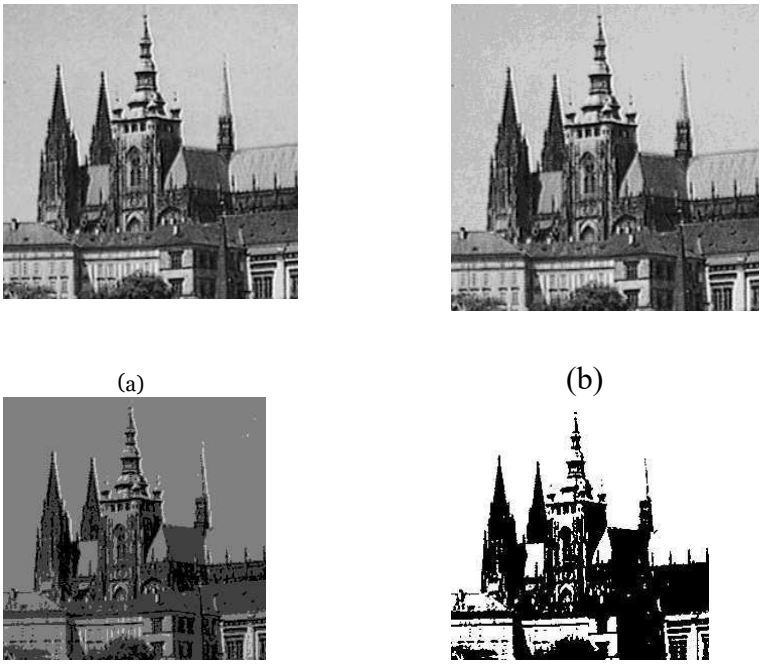
The transition between continuous values of the image function (brightness) and its digital equivalent is called **quantization**.

The number of quantization levels should be high enough to permit human perception of fine shading details in the image.

Most digital image processing devices use quantization into k equal intervals. If b bits are used to express the values of the pixel brightness then the number of brightness levels is $k = 2b$.

Eight bits per pixel per channel (one each for red, green, blue) are commonly used although systems using other numbers (e.g., 16) can be found. An efficient computer representation of brightness values in digital images requires that eight bits, four bits, or one bit are used per pixel, meaning that one, two, or eight pixel brightness can be stored in one byte.

The main problem in images quantized with insufficient brightness levels is the occurrence of false contours which effect arises when the number of brightness levels is lower than that which humans can easily distinguish. This number is dependent on many factors—for example, the average local brightness—but displays which avoid this effect will normally provide a range of at least 100 intensity levels. This problem can be reduced when quantization into intervals of unequal length is used; the size of intervals corresponding to less probable brightness in the image is enlarged.



(a) (b)

(c) (d)

Figure 2.3: Brightness levels. (a) 64. (b) 16. (c) 4. (d) 2. © *Cengage Learning 2015.*

Figures 3.11a and 2.3a-d demonstrate the effect of reducing the number of brightness levels in an image. An original image with 256 brightness levels has its number of brightness

---

levels reduced to 64 (Figure 2.3a), and no degradation is perceived. Figure 2.3b uses 16 brightness levels and false contours begin to emerge, and this becomes clearer in Figure 2.3c with four brightnesses and in Figure 2.3d with only two.

1.5.DIGITAL IMAGE PROPERTIES

A digital image has several properties, both metric and topological, which are some- what different from those of continuous two-dimensional functions. Human perception of digital images is a frequent aspect, since judgment of image quality is also important.

Metric and topological properties of digital images

A digital image consists of picture elements with finite size—these pixels carry informa tion about the brightness of a particular location in the image. Usually (and we assume this hereafter) pixels are arranged into a rectangular sampling grid. Such a digital image is represented by a two-dimensional matrix whose elements are natural numbers corresponding to the quantization levels in the brightness scale.

Some intuitively clear properties of continuous images have no straightforward anal- ogy in the domain of digital images. **Distance** is an important example. Any function D holding the following three condition is a 'distance' (or a *metric*)

$D(\mathbf{p}, \mathbf{q}) \geq 0$; $D(\mathbf{p}, \mathbf{q}) = 0$ if and only if $\mathbf{p} = \mathbf{q}$, *identity,*

$D(\mathbf{p}, \mathbf{q}) = D(\mathbf{q}, \mathbf{p})$, *symmetry,*

$D(\mathbf{p}, \mathbf{r}) \leq D(\mathbf{p}, \mathbf{q}) + D(\mathbf{q}, \mathbf{r})$, *triangular inequality.*

The distance between points with co-ordinates (i, j) and (h, k) may be defined in several different ways.

The **Euclidean distance** D_E known from classical geometry and everyday experi ence is defined by

$$D_E\big((i,j),(h,k)\big) = \sqrt{(i-h)^2 + (j-k)^2}\,. \quad (2.3)$$

The advantage of Euclidean distance is that it is intuitively obvious. The disadvantages are costly calculation due to the square root, and its non-integral value.

The distance between two points can also be expressed as the minimum number of elementary steps in the digital grid which are needed to move from the starting point to the end point. If only horizontal and vertical moves are allowed, the **'city block' distance** distance D_4 is obtained (also called the L_1 metric or Manhattan distance, because of the analogy with the distance between two locations in a city with a rectangular grid of streets):

$D_4(i, j)$, $(h, k) = |\, i - h \,| + |\, j - k \,|$. (2.4)

If moves in diagonal directions are allowed in the digitization grid, we obtain the distance D_8, or **'chessboard' distance**. D_8 is equal to the minimal number of king- moves on the chessboard from one part to another:

$$D_8\big((i,j),(h,k)\big) = \max\{\ |\ i-h\ |,\ |\ j-k\ |\ \}$$ (2.5) These distance definitions are illustrated in Figure 2.4.

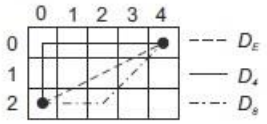


Figure 2.4: Distance metrics *De*, *D*4, and *D*8.

Pixel **adjacency** is another important concept in digital images. Two pixels (**p**, **q**) are called **4-neighbors** if they have distance D_4(**p**, **q**) = 1. Analogously, **8-neighbors** have D_8(**p**, **q**) = 1—see Figure 2.5.

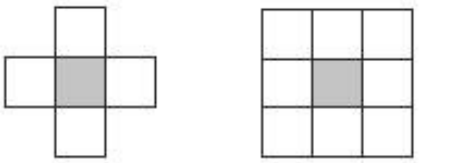


Figure 2.5: Neighborhood of the representative pixel (gray filled pixel in the middle). © *Cengage Learning 2015.*

It will become necessary to consider important sets consisting of several adjacent pixels—**regions** (in set theory, a region is a connected set). More descriptively, we can define a **path** from pixel *P* to pixel *Q* as a sequence of points $A_1, A_2, \ldots, A_n$, where $A_1 = P$, $A_n = Q$, and A_{i+1} is a neighbor of A_i, $i = 1, \ldots, n$ 1; then a region is a set of pixels in which there is a path between any pair of its pixels, all of whose pixels also belong to the set.

If there is a path between two pixels in the set of pixels in the image, these pixels are called **contiguous**

The relation 'to be contiguous' is reflexive, symmetric, and transitive and therefore defines a decomposition of the set (the image in our case) into equivalence classes (regions). Figure 2.6 illustrates a binary image decomposed by the relation 'contiguous' into three regions.

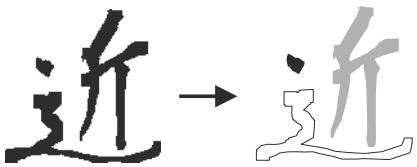


Figure 2.6: The relation ‘to be contiguous’ decomposes an image into individual regions. The Japanese Kanji character meaning ‘near from here’ decomposes into 3 regions.

Assume that R_i are disjoint regions in the image which were created by the relation ‘to be contiguous’, and further assume (to avoid special cases) that these regions do not touch the image bounds. Let R be the union of all regions R_i; R^C be the set complement of R with respect to the image. The subset of R^C which is contiguous with the image bounds is called the **background**, and the remainder of the complement R^C is called **holes**. A region is called **simple contiguous** if it has no holes. Equivalently, the complement of a simply contiguous region is contiguous. A region with holes is called **multiple contiguous**.

Note that the concept of region uses only the property ‘to be contiguous’. Secondary properties can be attached to regions which originate in image data interpretation. It is common to call some regions in the image **objects**; a process which determines which regions in an image correspond to objects in the world is a part of image **segmentation** and is discussed in Chapters 6 and 7.

The brightness of a pixel is a very simple property which can be used to find objects in some images; if, for example, a pixel is darker than some predefined value (threshold), then it belongs to the object. All such points which are also contiguous constitute one object. A hole consists of points which do not belong to the object and are surrounded by the object, and all other points constitute the background. An example is the black printed text on this white sheet of paper, in which individual letters are objects. White areas surrounded by the letter are holes, for example, the area inside the letter ‘o’. Other white parts of the paper are the background.

These neighborhood and contiguity definitions on the square grid create interesting paradoxes. Figure 2.7a shows two digital line segments with 45◦ slope. If 4-connectivity is used, the lines are not contiguous at each of their points. An even worse conflict with intuitive understanding of line properties is also illustrated; two perpendicular lines do intersect in one case (upper right intersection) and do not intersect in another case (lower left), as they do not have any common point (i.e., their set intersection is empty).

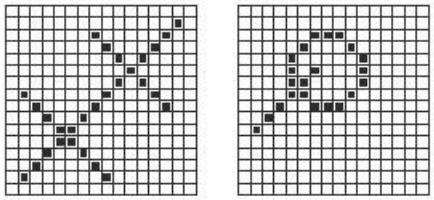


Figure 2.7: Paradoxes of crossing lines.

It is known from Euclidean geometry that each closed curve (e.g., a circle) divides the plane into two non-contiguous regions. If images are digitized in a square grid using 8connectivity, we can draw a line from the inner part of a closed curve into the outer part which does not intersect the curve (Figure 2.7b). This implies that the inner and outer parts of the curve constitute only one region because all pixels of the line belong to only one region, giving another paradox. One possible solution to contiguity paradoxes is to treat objects using 4-neighborhoods and background using 8-neighborhoods (or vice versa).

These problems are typical on square grids—a hexagonal grid (see Figure 2.2), however, solves many of them. Any point in the hexagonal raster has the same distance to all its six neighbors. There are some problems peculiar to the hexagonal raster as well (for example, it is difficult to express a Fourier transform on it). For reasons of simplicity and ease of processing, most digitizing devices use a square grid despite the known drawbacks.

An alternative approach to the connectivity problems is to use discrete topology based on cellular complexes. This approach develops a complete strand of image encoding and segmentation that deals with many issues we shall come to later, such as the representation of boundaries and regions. The idea, first proposed by the German mathematician Riemann in the nineteenth century, considers families of sets of different dimensions; points, which are 0-dimensional sets, may then be assigned to sets containing higher-dimensional structures (such as pixel arrays). This approach permits the removal of the paradoxes we have seen.

The **distance transform**—also called the **distance function** or **chamfering algorithm** or simply **chamfering**—is a simple application of the concept of distance. The idea is important as it provides the basis of several fast algorithms that will be seen multiple times in this book. The distance transform provides the distance of pixels from some image subset (perhaps describing objects or some features). The resulting ‘image’ has pixel values of 0 for elements of the relevant subset, low values for close pixels, and then high values for pixels remote from it—the appearance of this array gives the name to the technique.

For illustration, consider a binary image, in which ones represent the objects and zeros the background. The input image is shown in Figure 2.8 and the result of the D_4 distance transform is illustrated in Figure 2.9.

To calculate the transform, a two-pass algorithm has been suggested for distances D_4 and D_8. The idea is to traverse the image by a small local mask. The first pass starts from the top-left of the image and moves horizontally

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

Figure 2.8: Input image: gray pixels correspond to objects and white to background.

5	4	4	3	2	1	0	1
4	3	3	2	1	0	1	2
3	2	2	2	1	0	1	2
2	1	1	2	1	0	1	2
1	0	0	1	2	1	0	1
1	0	1	2	3	2	1	0
1	0	1	2	3	3	2	1
1	0	1	2	3	4	3	2

Figure 2.9: Result of the $[D_4]$ distance transform. © *Cengage Learning 2015.*

left to right until it reaches the bounds of the image and then returns to the beginning of the next row. The second pass goes from the bottom-right corner in the opposite bottom-up, right to left direction using a different local mask. The effectiveness of the algorithm comes from propagating the values of the previous image investigation in a ‘wave-like’ manner. The masks used in calculations are shown in Figure 2.10.

Algorithm 2.1: Distance transform

1. Choose a distance N_{max} that exceeds the image dimension with respect to the chosen distance metric (D_4 or D_8). Initialize an image F of the same dimension as the input: set pixels corresponding to the subset(s) to be chamfered to 0, and all others to N_{max}.
2. Pass through image pixels from top to bottom and left to right. For a given pixel, consider neighbors above and to the left and set

$$F(\mathbf{p}) = \min_{\mathbf{q} \in AL} F(\mathbf{p}), D(\mathbf{p}, \mathbf{q}) + F(\mathbf{q}) \quad .$$

3. Pass through image pixels from bottom to top and rightto left. For a given pixel, consider neighbors above and to the left and set

$$F(\mathbf{p}) = \min_{\mathbf{q} \in BR} F(\mathbf{p}), D(\mathbf{p}, \mathbf{q}) + F(\mathbf{q}) \quad .$$

4. If any pixels remain still set at N_{max}, go to (2).
5. The array F now holds a chamfer of the chosen subset(s).

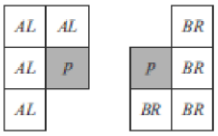
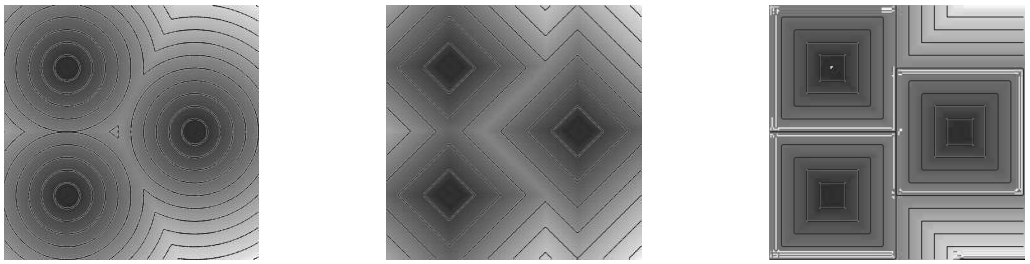


Figure 2.10: Pixel neighborhoods used in distance transform calculations—**p** is the central one. The neighborhood on the left is used in the first pass (top-down, left to right); that on the right is used in the second (bottom-up, right to left). © *Cengage Learning 2015.*



(a) (b) (c)

Figure 2.11: Three distances used often in distance transform calculations—the input consists of three isolated 'ones'. Output distance is visualized as intensity; lighter values denote higher distances. Contour plots are superimposed for better visualization. (a) Euclidean distance *DE* . (b) City block distance *D*4. (c) Chessboard distance *D*8.

This algorithm needs obvious adjustments at image boundaries, where the sets AL and BR are truncated. It is open to various improvements by using different distance calculations. Distance transform performance for D_E, D_4, D_8 on an input consisting only of three distinct 'ones' is shown in Figure 2.11.

The distance transform has many applications, e.g., in discrete geometry, path planning and obstacle avoidance in mobile robotics, finding the closest feature in the image, and skeletonization.

An **edge** is a further important concept used in image analysis. This is a local property of a pixel and its immediate neighborhood—it is a vector given by a magnitude and direction which tells us how fast the image intensity varies in a small neighborhood of a pixel. Images with many brightness levels are used for edge computation, and the gradient of the image function is used to compute edges. The edge direction is perpendicular to the gradient direction which points in the direction of the fastest image function growth.

The related concept of the **crack edge** creates a structure between pixels in a similar manner to that of cellular complexes. However, it is more pragmatic and less mathematically rigorous. Four crack edges are attached to each pixel, which are defined by its relation to its 4-neighbors. The direction of the crack edge is that of increasing bright- ness, and is a multiple of 90◦, while its magnitude is the absolute difference between the brightness of the relevant pair of pixels. Crack edges are illustrated in Figure 2.12 and may be be used in considering image segmentation.

The **border** (boundary) of a region is another important concept in image analysis. The border of a region R is the set of pixels within the region that have one or more neighbors outside R. The definition corresponds to an intuitive understanding of the border as a set of points at the bound of the region. This definition is sometimes referred to as the **inner border** to distinguish it from the **outer border**, that is, the border of the background (i.e., its complement) of the region. Inner and outer borders are illustrated in Figure 2.13. Due to the discrete nature of the image, some inner border elements which would be distinct in the continuous case coincide in the discrete case, as can be seen with the one-pixel-wide line at the right of Figure 2.13.

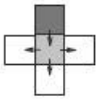


Figure 2.12: Crack edges. © *Cengage Learning 2015.*

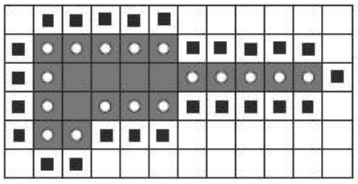


Figure 2.13: Region inner borders shown as white circles and outer borders shown as black squares (using 4-neighborhoods). © *Cengage Learning 2015.*

While edges and borders are related, they are not the same thing. ‘Border’ is a global concept related to a region, while ‘edge’ expresses local properties of an image function. One possibility for finding boundaries is chaining the significant edges (points with high gradient of the image function). Methods of this kind are described in Section 6.2.

A region is described as **convex** if any two points within it are connected by a straight line segment, and the whole line lies within the region—see Figure 2.14. The property of convexity decomposes all regions into two equivalence classes: convex and non-convex.

Figure 2.14: A convex region (left) and non-convex region (right). © *Cengage Learning 2015.*

A **convex hull** of a region is the smallest convex region containing the input (possibly non-convex) region. Consider an object whose shape resembles the letter ‘R’ (see Figure 2.15). Imagine a thin rubber band pulled around the object; the shape of the rubber band provides the convex hull of the object.

Figure 2.15: Description using topological components: An ‘R’ object, its convex hull, and the associated lakes and bays. © *Cengage Learning 2015.*

Topological properties are not based on the distance concept. Instead, they are invariant to *homeomorphic* transforms which can be illustrated for images as **rubber sheet transforms**. Imagine a small rubber balloon with an object painted on it; topo- logical properties of the object are those which are invariant to arbitrary stretching of the rubber sheet. Stretching does not change contiguity of the object parts and does not change the number of holes in regions. We use the term ‘topological properties’ of the region to describe its qualitative properties invariant to small changes (e.g., the property of being convex), even though an arbitrary homeomorphic transformation can change a convex region to a nonconvex one and vice versa. Considering the rubber sheet analogy, we mean that the stretching of the sheet is only gentle.

An object with non-regular shape can be represented by a collection of its topological components, Figure 2.15. The set inside the convex hull which does not belong to an object is called the **deficit of convexity**. This can be split into two subsets: **lakes** (dark gray) are

fully surrounded by the object; and **bays** (light gray) are contiguous with the border of the convex hull of the object.

The convex hull, lakes, and bays are sometimes used for object description; these features are used in Chapter 8 (object description) and in Chapter 13 (mathematical morphology).

Histograms

The **brightness histogram** $h_f(x)$ of an image provides the frequency of the brightness value x in the image—the histogram of an image with L gray-levels is represented by a onedimensional array with L elements.

Algorithm 2.2: Computing the brightness histogram

1. Assign zero values to all elements of the array h_f.
2. For all pixels (x, y) of the image f, increment $h_f\ f(x, y)$ by 1.

The histogram provides a natural bridge between images and a probabilistic description. We might want to find a first-order probability function $p_1(x; x, y)$ to indicate the probability that pixel (x, y) has brightness x. Dependence on the position of the pixel is not of interest in the histogram; the function $p_1(x)$ is of interest and the brightness histogram is its estimate. The histogram is often displayed as a bar graph, see Figure 2.16.

The histogram is usually the only global information about the image which is available. It is used when finding optimal illumination conditions for capturing an image, gray-scale transformations, and image segmentation to objects and background. Note

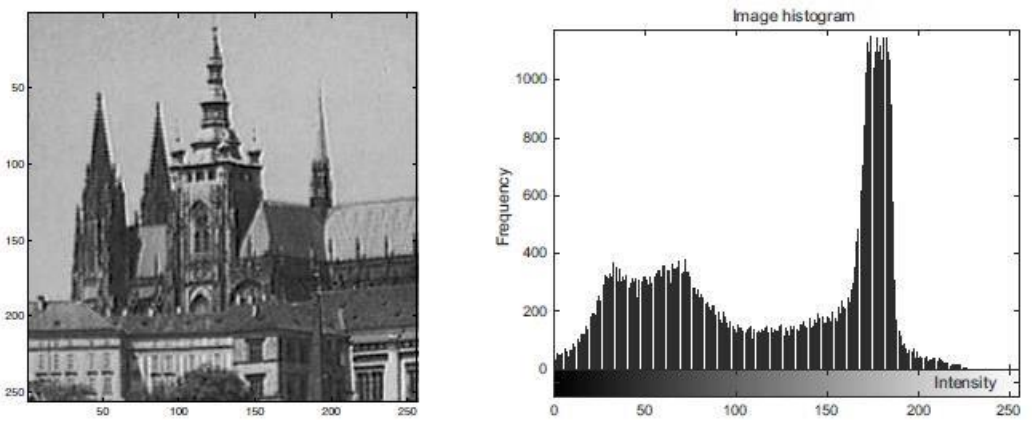


Figure 2.16: Original image (a) and its brightness histogram (b).

that one histogram may correspond to several images; for instance, a change of the object position on a constant background does not affect it.

The histogram of a digital image typically has many local minima and maxima, which may complicate its further processing. This problem can be avoided by local smoothing of the histogram; this may be done, for example, using local averaging of neighboring histogram elements as the base, so that a new histogram $h'f(x)$ is calculated according to

$$h'_f(z) = \frac{1}{2K+1} \sum_{j=-K}^{K} h_f(z+j)\,, \qquad (2.6)$$

where K is a constant representing the size of the neighborhood used for smoothing. This algorithm would need some boundary adjustment, and carries no guarantee of removing all local minima.

Entropy

If a probability density p is known then image information content can be estimated regardless of its interpretation using **entropy** H. The concept of entropy has roots in thermodynamics and statistical mechanics but it took many years before it was related to information. The information-theoretic formulation comes from Shannon and is often called **information entropy**.

An intuitive understanding of information entropy relates to the amount of uncertainty about an event associated with a given probability distribution. The entropy can serve as an measure of 'disorder'. As the level of disorder rises, entropy increases and events are less predictable.

Entropy is formally defined assuming a discrete random variable X with possible outcomes (called also states) $x_1, \ldots, x_n$. Let $p(x_k)$ be the probability of the outcome x_k, $k = 1, \ldots n$. Then the entropy is defined as

$$H(X) \equiv \sum_{k=1}^{n} p(x_k) \log_2 \left(\frac{1}{p(x_k)} \right) = -\sum_{k=1}^{n} p(x_k)\, \log_2 p(x_k)\,. \qquad (2.7)$$

The entropy of the random variable X is the sum, over all possible outcomes k of X, of the product of the probability of outcome x_k with the logarithm of the inverse of the probability of x_k.

The base of the logarithm in this formula determines the unit in which entropy is measured. If this base is two then the entropy is given in bits. Recall that the probability density $p(x_k)$ needed to calculate the entropy is often estimated using a gray- level histogram in image analysis.

Entropy measures the uncertainty about the realization of a random variable. For Shannon, it served as a proxy capturing the concept of information contained in a message as opposed to the portion of the message that is strictly determined and predictable by inherent structures. For example, we shall explore entropy to assess redundancy in an image for image compression.

Visual perception of the image

Anyone who creates or uses algorithms or devices for digital image processing should take into account the principles of human image perception. If an image is to be analyzed by a human the information should be expressed using variables which are easy to perceive; these are psycho-physical parameters such as contrast, border, shape, texture, color, etc. Humans will find objects in images only if they may be distinguished effortlessly from the background. A detailed description of the principles of human visual perception can be found in. Human perception of images provokes many illusions, the understanding of which provides valuable clues about visual mecha nisms.

The situation would be relatively easy if the human visual system had a linear response to composite input stimuli—i.e., a simple sum of individual stimuli. A decrease of some stimulus, e.g., area of the object in the image, could be compensated by its intensity, contrast, or duration. In fact, the sensitivity of human senses is roughly logarithmically proportional to the intensity of an input signal. In this case, after an initial logarithmic transformation, response to composite stimuli can be treated as linear.

Contrast

Contrast is the local change in brightness and is defined as the ratio between average brightness of an object and the background. Strictly speaking, we should talk about luminance4 instead of brightness if our aim is to be physically precise. The human eye is logarithmically sensitive to brightness, implying that for the same perception, higher brightness requires higher contrast.

Apparent brightness depends very much on the brightness of the local surroundings; this effect is called conditional contrast. Figure 2.17 illustrates this with five circles of the same size surrounded by squares of different brightness. Humans perceive the brightness of the small circles as different.

Figure 2.17: Conditional contrast effect. Circles inside squares have the same brightness and are perceived as having different brightness values. © *Cengage Learning 2015.*

Acuity

Acuity is the ability to detect details in an image. The human eye is less sensitive to slow and fast changes in brightness in the image plane but is more sensitive to intermediate changes. Acuity also decreases with increasing distance from the optical axis.

Resolution in an image is firmly bounded by the resolution ability of the human eye; there is no sense in representing visual information with higher resolution than that of

---

the viewer. Resolution in optics is defined as the inverse value of a maximum viewing angle between the viewer and two proximate points which humans cannot distinguish, and so fuse together.

Human vision has the best resolution for objects which are at a distance of about 250 mm from an eye under illumination of about 500 lux; this illumination is provided by a 60 W bulb from a distance of 400 mm. Under these conditions the distance between two distinguishable points is approximately 0.16 mm.

Some visual illusions

Human perception of images is prone to many illusions.

Object borders carry a lot of information for humans. Boundaries of objects and simple patterns such as blobs or lines enable adaptation effects similar to conditional contrast, mentioned above. The Ebbinghaus illusion is a well-known example—two circles of the same diameter in the center of images appear to have different diameters (Figure 2.18).

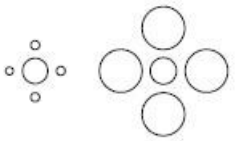


Figure 2.18: The Ebbinghaus illusion. © *Cengage Learning 2015.*

Perception of one dominant shape can be fooled by nearby shapes. Figure 2.19 shows parallel diagonal line segments which are not perceived as parallel. Figure 2.20 contains rows of black and white squares which are all parallel. However, the vertical zigzag squares disrupt our horizontal perception.

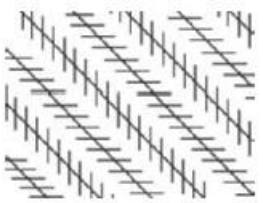


Figure 2.19: Disrupted parallel diagonal lines. © *Cengage Learning 2015.*

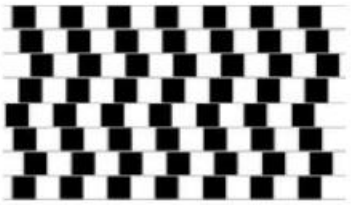


Figure 2.20: Horizontal lines are parallel, although not perceived as such. © *Cengage Learning 2015.*

Perceptual grouping

Perceptual grouping [Palmer, 1999] is a principle used in computer vision to aggregate elements provided by low-level operations such as edges, which are small blobs to bigger chunks having some meaning. Its roots are in Gestalt psychology first postulated by Wertheimer in 1912 [Brett King and Wertheimer, 2005]. Gestalt psychology proposes that the operational principle of the mind and brain is holistic, parallel, and with self- organizing tendencies.

Gestalt theory was meant to have general applicability; its main tenets, however, were induced almost exclusively from observations on visual perception. The overriding theme of the theory is that stimulation is perceived in organized or configuration terms. Gestalt in German means configuration, structure or pattern of physical, biological, or psychological phenomena so integrated to constitute a functional unit with properties not derivable by summation of its parts. Patterns take precedence over elements and have properties that are not inherent in the elements themselves.

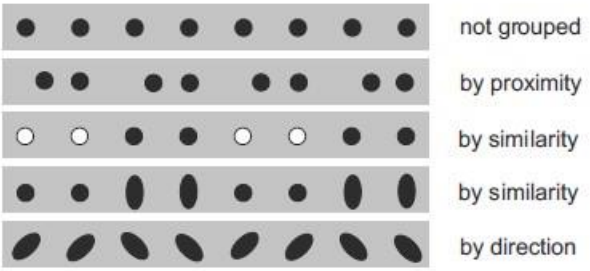


Figure 2.21: Grouping according to properties of elements.

The human ability to group items according to various properties is illustrated in Figure 2.21. Perceived properties help people to connect elements together based on strongly perceived properties as parallelism, symmetry, continuity and closure taken in a loose sense as illustrated in Figure 2.22.

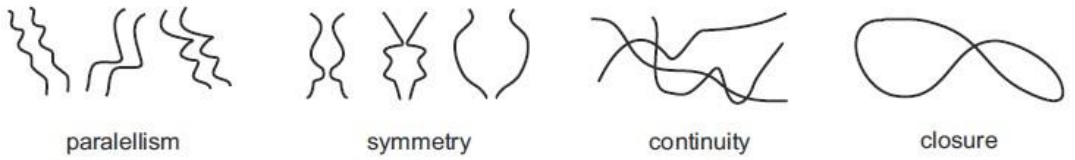


Figure 2.22: Illustration of properties perceived in images which allow humans to group together elements in cluttered scenes.

It has been demonstrated that mimicking perceptual grouping in machine vision system is a plausible technique. It permits the creation of more meaningful chunks of information from meaningless outcomes of low-level operations such as edge detection. Such grouping is useful in image understanding. This principle will be used in this book mainly for image segmentation.

Image quality

An image might be degraded during capture, transmission, or processing, and measures of image quality can be used to assess the degree of degradation. The quality required naturally depends on the purpose for which an image is used.

Methods for assessing **image quality** can be divided into two categories: subjective and objective. Subjective methods are often used in television technology, where the ultimate criterion is the perception of a selected group of professional and lay viewers. They appraise

an image according to a list of criteria and give appropriate marks. Details about subjective methods may be found in [Pratt, 1978].

Objective quantitative methods measuring image quality are more interesting for our purposes. Ideally such a method also provides a good subjective test, and is easy to apply; we might then use it as a criterion in parameter optimization. The quality of an image $f(x, y)$ is usually estimated by comparison with a known reference image $g(x, y)$ [Rosenfeld and Kak, 1982], and a synthesized image is often used for this purpose. One class of methods uses simple measures such as the mean quadratic difference $\Sigma - (g - f)2$, but this does not distinguish a few big differences from many small differences. Instead of the mean quadratic difference, the mean absolute difference or simply maximal absolute difference may be used. Correlation between images f and g is another alternative.

Another class measures the resolution of small or proximate objects in the image. An image consisting of parallel black and white stripes is used for this purpose; then the number of black and white pairs per millimeter gives the resolution.

2.2.1 Noise in images

Real images are often degraded by some random errors—this degradation is usually called **noise**. Noise can occur during image capture, transmission, or processing, and may be dependent on, or independent of, the image content.

1) white noise

white noise is **a random signal having equal intensity at different frequencies, giving it a constant** power spectral density.

2) Gaussian noise

Gaussian noise, named after Carl Friedrich Gauss, is a kind of signal noise that has a probability density function equal to that of the normal distribution (which is also known as the Gaussian distribution)

3) Additive noise

Additive noise is the undesired abrupt signal that gets added into some genuine signal.

4) Multiplicative noise

Multiplicative noise is the undesired abrupt signal that gets multiplied into some genuine signal.

5) Quantization noise

Quantization noise results when a continuous random variable is converted to a discrete one or when a discrete random variable is converted to one with fewer levels. In images, quantization noise often occurs in the acquisition process.

6) Impulse noise

Impulse noise is a category of noise that includes unwanted, almost instantaneous sharp sounds —typically caused by electromagnetic interference, scratches on disks, gunfire, explosions, and synchronization issues in digital audio.

7) Salt and pepper noise

salt--and--pepper noise describes a situation where random pixels get replaced by extremely dark or bright values.

1.6 COLOR IMAGES

Human color perception adds a subjective layer on top of underlying objective physical properties—the wavelength of electromagnetic radiation.
color may be considered a psycho-physical phenomenon.

Color has long been used in painting, photography and films to display the surrounding world to humans in a similar way to that in which it is perceived in reality. There is considerable literature on the variants in the naming of colors across languages, which is a very subtle affair. The human visual system is not very precise in perceiving color in absolute terms; if we wish to express our notion of color precisely we would describe it relative to some widely used color which is used as a standard: e.g.,
the red of a British pillar box. There are whole industries which present images to humans— the press, films, displays, and hence a desire for color constancy. In computer vision, we have the advantage of using a camera as a measuring device, which yields measurements in absolute quantities.

Newton reported in the 17th century that white light from the sun is a spectral mixture, and used the optical prism to perform decomposition. This was a radical idea to propose at

time; over 100 years later influential scientists and philosophers such as Goethe refused to believe it.

2.3.1 Physics of color

The electromagnetic spectrum is illustrated in Figure 2.23.

Only a narrow section of the electromagnetic spectrum is visible to a human, with wavelength from approximately 380 nm to 740 nm. Visible colors with the wavelengths shown in Figure 2.24 are called **spectral colors** and are those which humans see when white light is decomposed using a Newtonian prism, or which are observed in a rainbow on the sky. Colors can be represented as combinations of the **primary colors**, e.g., red,

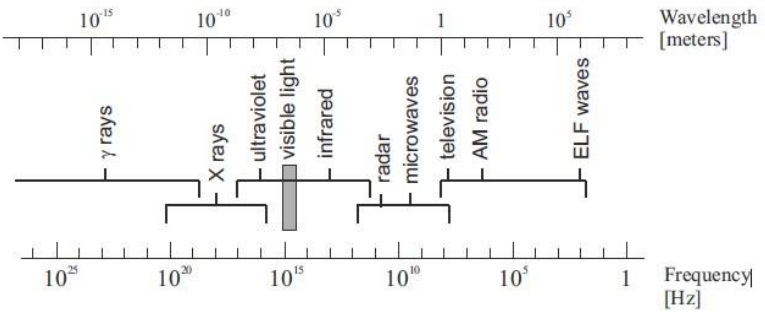
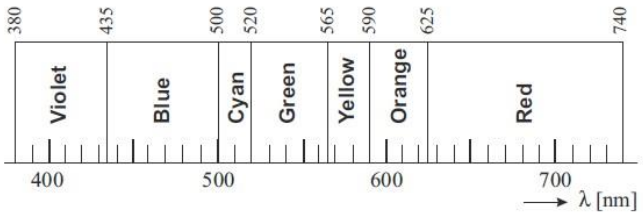


Figure 2.23: Division of the electromagnetic spectrum (ELF is Extremely Low Frequencies).



.

Figure 2.24: Wavelength λ of the spectrum visible to humans.

green, and blue, which for the purposes of standardization have been defined as 700 nm, 546.1 nm, and 435.8 nm, respectively [Pratt, 1978], although this standardization does not imply that all colors can be synthesized as combinations of these three.

The intensity of irradiation for different wavelengths λ usually changes. This varia tion is expressed by a **power spectrum** (called also power spectrum distribution) $S(\lambda)$. Why do we see the world in color? There are two predominant physical mechanisms describing what happens when a surface is irradiated. First, the **surface reflection** rebounds incoming energy in a similar way to a mirror. The spectrum of the reflected light remains the same as that of the illuminant and it is independent of the surface— recall that shiny metals 'do not have a color'. Second, the energy diffuses into the material and reflects randomly from the internal pigment in the matter. This mechanism is called **body reflection** and is predominant in dielectrics such as plastic or paints. Figure 2.25 illustrates both surface reflection (mirroring along surface normal **n**) and body reflection.

Colors are caused by the properties of pigment particles which absorb certain wavelengths from the incoming illuminant wavelength spectrum.

Most sensors used for color capture, e.g., in cameras, do not have direct access to color; the exception is a **spectrophotometer** which in principle resembles Newton's prism. Incoming irradiation is decomposed into spectral colors and intensity along the spectrum with changing wavelength λ is measured in a narrow wavelength band, for in- stance, by a mechanically moved point sensor. Actual spectrophotometers use diffraction gratings instead of a glass prism.

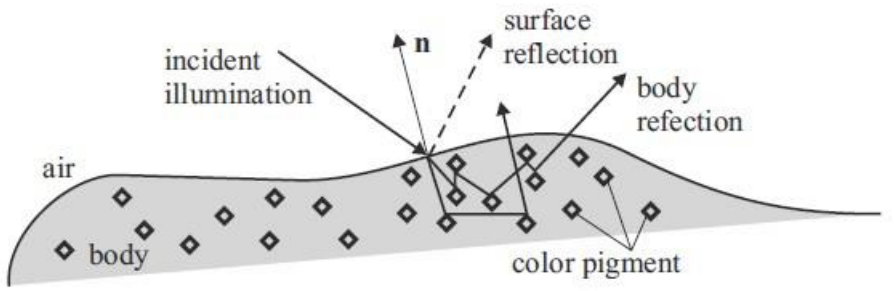


Figure 2.25: Observed color of objects is caused by certain wavelength absorptions by pigment particles in dielectrics.

Sometimes, intensities measured in several narrow bands of wavelengths are collected in a vector describing each pixel. Each spectral band is digitized independently and is represented by an individual digital image function as if it were a monochromatic image. In this way, **multispectral images** are created. Multispectral images are commonly used in remote sensing from satellites, airborne sensors and in industry. Wavelength usually span from ultraviolet through the visible section to infrared. For instance, the LAND- SAT 7 satellite transmits digitized images in eight spectral bands from near-ultraviolet to infrared.

Color perceived by humans

Evolution has developed a mechanism of indirect color sensing in humans and some animals. Three types of sensors receptive to the wavelength of incoming irradiation have been established in humans, thus the term **trichromacy**. Color sensitive receptors on the human retina are the **cones**. The other light sensitive receptors on the retina are the **rods** which are dedicated to sensing monochromatically in low ambient light conditions. Cones are categorized into three types based on the sensed wavelength range: S (short) $\approx$ with maximum sensitivity$\approx$ at 430 nm, M (medium) at 560 nm, and L (long) at 610 nm. Cones S, M, L are occasionally called cones B, G and R, respectively, but that is slightly misleading. We do not see red solely because an L cone is activated. Light with equally distributed wavelength spectrum looks white to a human, and an unbalanced spectrum appears as some shade of color.

The reaction of a photoreceptor or output from a sensor in a camera can be modeled mathematically. Let i be the specific type of sensor, $i = 1, 2, 3,$ (the retinal cone type S, M, L in the human case). Let $R_i(\lambda)$ be the spectral sensitivity of the sensor, $I(\lambda)$ be the spectral density of the illumination, and $S(\lambda)$ describe how the surface patch reflects each wavelength of the illuminating light. The spectral response q_i of the i-th sensor, can be modeled by integration over a certain range of wavelengths

$$q_i = \int_{\lambda_1}^{\lambda_2} I(\lambda)\, R_i(\lambda)\, S(\lambda)\, \mathrm{d}\lambda\,. \qquad (2.16)$$

Consider the cone types S, M, L. How does the vector (q_S, q_M, q_L) represent the color of the surface patch? It does not according to equation (2.16) since the output from the photosensors depends on the three factors $I(\lambda)$, $S(\lambda)$ and $R(\lambda)$. Only the factor $S(\lambda)$ is related to the surface patch. Only in the ideal case, when the illumination is perfectly white, i.e., $I(\lambda) = 1$, can we consider (q_S, q_M, q_L) as an estimate of the color of the surface.

Figure 2.26 illustrates qualitatively the relative sensitivities of S, M, L cones. Measurements were taken with the white light source at the cornea so that absorption of wavelength in cornea, lens and inner pigments of the eye is taken into account [Wandell, 1995].
A phenomenon called **color metamer** is relevant. A metamer, in general, means two things that are physically different but perceived as the same. Red and green adding to produce yellow is a color metamer, because yellow could have also been produced by a spectral color. The human visual system is fooled into perceiving that red and green is the same as yellow.

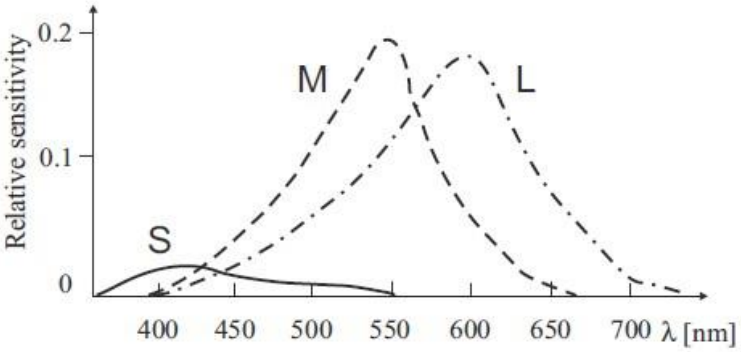


Figure 2.26: Relative sensitivity of S, M, L cones of the human eye to wave- length.

Consider a color matching experiment in which someone is shown a pattern consisting of two adjacent color patches. The first patch displays a test light—a spectral color of certain wavelength. The second patch is created as an additive combination of three selected primary lights, e.g., colors red=645.2 nm, green=525.3 nm and blue=444.4 nm. The observer is asked to control the red, green and blue intensities until both patches look identical. This color matching experiment is possible because of the color metamer. The result of measurements (redrawn from [Wandell, 1995]) is in Figure 2.27. Negative lobes can be seen on the curves for red and green in this figure. This would seem to be impossible. For wavelengths exhibiting negative values the three additive lights do not perceptually match the spectral color because it is darker. If the perceptual match has to be obtained then the observer has to add the intensity to the patch corresponding to the spectral color. This increase of this intensity is depicted as a decrease in the color matching function. Hence the negative values.

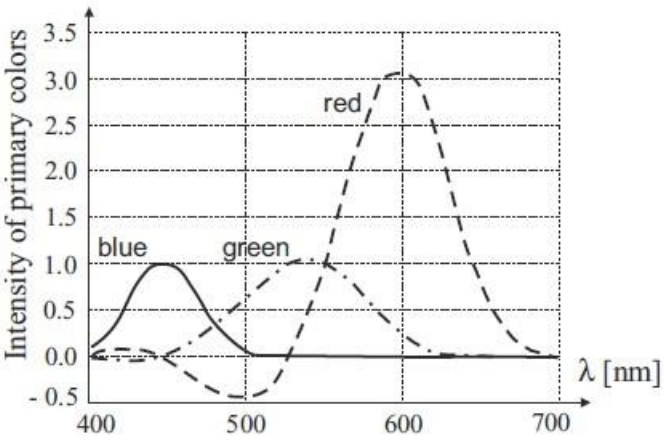


Figure 2.27: Color matching functions ob- tained in the color matching experiment. Intensities of the selected primary colors which perceptually match spectral color of given wavelength λ.

Human vision is prone to various illusions. Perceived color is influenced, besides the spectrum of the illuminant, by the colors and scene interpretation surrounding the observed color. In addition, eye adaptation to changing light conditions is not very fast and perception is influenced by adaptation. Nevertheless, we assume for simplicity that the spectrum of light coming to a point on the retina fully determines the color.

Since color can be defined by almost any set of primaries, the world community agreed on primaries and color matching functions which are widely used. The **color model** was introduced as a mathematical abstraction allowing us to express colors as tuples of numbers, typically as three or four values of color components. Being motivated by the press and the development of color film, in 1931, CIE (International Commission on Illumination, still acting in Lausanne, Switzerland) issued a technical standard called **XYZ color space**.

The standard is given by the three imaginary lights X=700.0 nm, Y =546.1 nm, Z=435.8 nm and by the color matching functions $X(\lambda)$, $Y(\lambda)$ and $Z(\lambda)$ corresponding to the perceptual ability of an average human viewing a screen through an aperture providing a 2◦ field of view. The standard is artificial because there is no set of physically realizable primary lights that would yield the color matching functions in the experiment. Nevertheless, if we wanted to characterize the imaginary lights then, roughly speaking, X red, Y green and Z blue. The CIE standard is an example of an absolute standard, i.e., defining unambiguous representation of color which does not depend on other external factors. There are more recent and more precise absolute standards: CIELAB 1976 (ISO 13665) and Hunter Lab

(http://www.hunterlab.com). Later, we will also dis- cuss relative color standards such as RGB color space: there are several RGB color spaces used—two computer devices may display the same RGB image differently.

The XYZ color standard fulfills three requirements:

- Unlike the color matching experiment yielding negative lobes of color matching functions, the matching functions of XYZ space are required to be non-negative.
- The value of $Y(\lambda)$ should coincide with the brightness (luminance).
- Normalization is performed to assure that the power corresponding to the three color matching functions is equal (i.e., the area under all three curves is equal). The resulting color matching functions are shown in Figure 2.28. The actual color is a mixture (more precisely a convex combination) of

$$c_X X + c_Y Y + c_Z Z, \tag{2.17}$$

where $0 \; c_X, c_Y, c_Z \; 1$ are weights (intensities) in the mixture. The subspace of colors perceivable by humans$^{\leq}$ is called$^{\leq}$ the color gamut and is demonstrated in Figure 2.29.

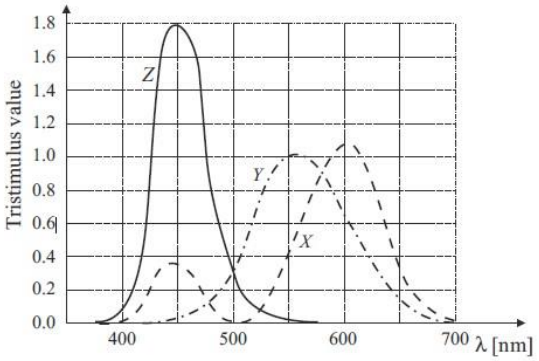


Figure 2.28: Color matching functions for the CIE standard from 1931. $X(\lambda)$, $Y(\lambda)$, $Z(\lambda)$ are color matching functions. *Based on [Wandell, 1995].*

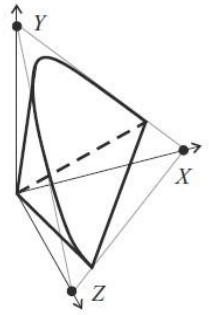


Figure 2.29: Color gamut - a subspace of the X, Y, Z color space showing all colors perceivable by humans. © *Cen-*

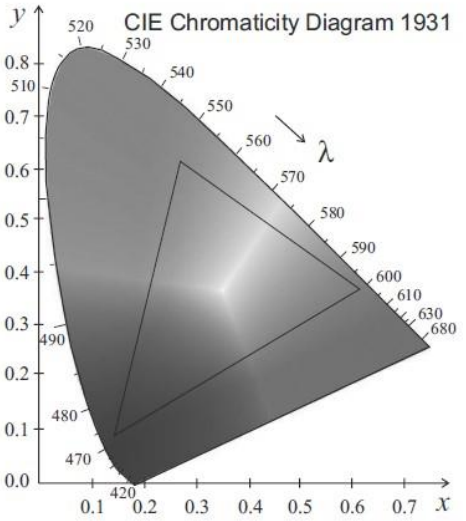


Figure 2.30: CIE chromaticity diagram is a projection of XYZ color space into a plane. The triangle depicts a subset of colors spanned by red, green, and blue. These are TV colors, i.e., all possible colors, which can be seen on a CRT display. © *Cengage Learning 2015. A color version of this figure may be seen in the color inset—Plate 1.*

3D figures are difficult to represent, and so a planar view of a 3D color space is used. The projection plane is given by the plane passing through extremal points on all three axes, i.e., points *X, Y, Z*. The new 2D coordinates *x, y* are obtained as

$$x = \frac{X}{X+Y+Z}\,, \qquad y = \frac{Y}{X+Y+Z}\,, \qquad z = 1 - x - y\,.$$

The result of this plane projection is the CIE chromaticity diagram, see Figure 2.30. The horseshoe like subspace contains colors which people are able to see. All mono- chromatic spectra visible to humans map into the curved part of the horseshoe—their wavelengths are shown in Figure 2.30.

Display and printing devices use three selected real primary colors (as opposed to three synthetic primary colors of XYZ color space). All possible mixtures of these primary colors fail to cover the whole interior of the horseshoe in CIE chromaticity diagram. This situation is demonstrated qualitatively for three particular devices in Figure 2.31.

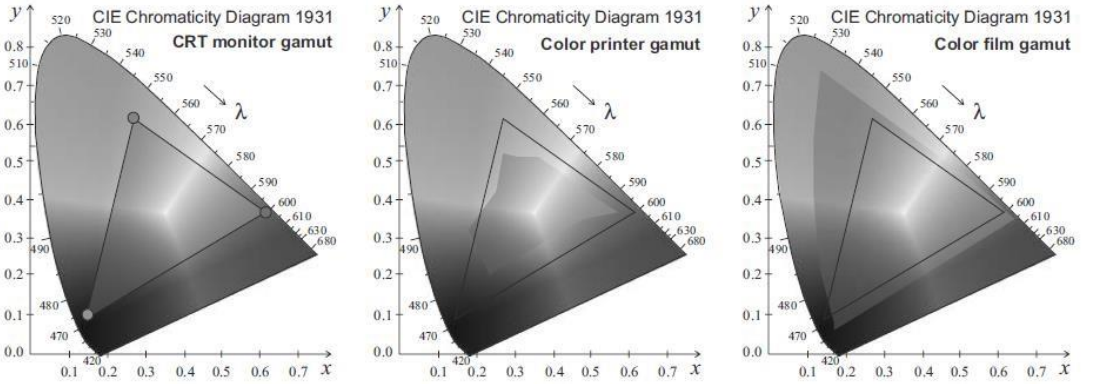


Figure 2.31: Gamuts which can be displayed using three typical display devices. © *Cengage Learning 2015. A color version of this figure may be seen in the color inset—Plate 2.*

Color spaces

Several different primary colors and corresponding color spaces are used in practice, and these spaces can be transformed into each other. If the absolute color space is used then the transformation is the one-to-one mapping and does not lose information (except for rounding errors). Because color spaces have their own gamuts, information is lost if the transformed value appears out of the gamut. See [Burger and Burge, 2008] for a full explanation and for algorithms; here, we list several frequently used color spaces.

The **RGB** color space has its origin in color television where Cathode Ray Tubes (CRT) were used. RGB color space is an example of a relative color standard (as opposed to the absolute one, e.g., CIE 1931). The primary colors (R–red, G–green and B–blue) mimicked phosphor in CRT luminophore. The model uses additive color mixing to inform what kind of light needs to be emitted to produce a given color. The value of a particular color is expressed as a vector of three elements—intensities of three primary colors,× and a transformation to a different color space is expressed by a 3 3 matrix. Assume that values for each primary are quantized to $m = 2^n$ values; let the highest – intensity value be $k = m - 1$; then (0, 0, 0) is black,

(k, k, k) is (television) white, (k, 0, 0) is 'pure' red, and so on. The value $k = 255 = 28 \quad 1$ is common, i.e., 8 bits per color channel. There are $2563 = 224 = 16,777,216$ possible colors in such a discretized space.

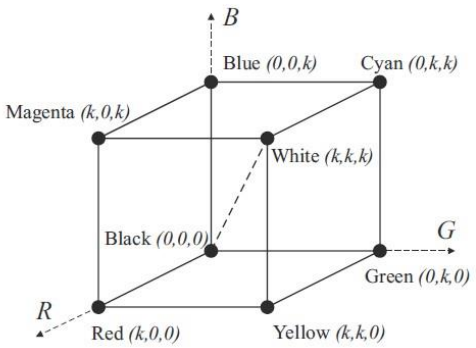


Figure 2.32: RGB color space with primary colors red, green, blue and secondary colors yellow, cyan, magenta. Gray-scale images with all intensities lie along the dashed line connecting black and white in RGB color space.

The RGB model may be thought of as a 3D co-ordinatization of color space (see Figure 2.32); note the secondary colors which are combinations of two pure primaries. There are specific instances of the RGB color model such as sRGB, Adobe RGB and Adobe Wide Gamut RGB, which differ slightly in transformation matrices and the gamut. One of the transformations between RGB and XYZ color spaces is

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.24 & -1.54 & -0.50 \\ -0.98 & 1.88 & 0.04 \\ 0.06 & -0.20 & 1.06 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix},$$
$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.41 & 0.36 & 0.18 \\ 0.21 & 0.72 & 0.07 \\ 0.02 & 0.12 & 0.95 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \qquad (2.18)$$

The US and Japanese color television formerly used **YIQ** color space. The Y component describes intensity and I, Q represent color. YIQ is another example of additive

color mixing. This system stores a luminance value with two chrominance values, corresponding approximately to the amounts of blue and red in the color. This color space corresponds closely to the YUV color model in the PAL television norm (Australia, Eu-

rope, except France, which uses SECAM). YIQ color space is rotated 33◦ with respect to the YUV color space. The YIQ color model is useful since the Y component provides all that is necessary for a monochrome display; further, it exploits advantageous properties of the human visual system, in particular our sensitivity to **luminance**, the perceived energy of a light source.

The **CMY**—for Cyan, Magenta, Yellow—color model uses subtractive color mixing which is used in printing processes. It describes what kind of inks need to be applied so the light reflected from the white substrate (paper, painter's canvas) and passing through the inks produces a given color. CMYK stores ink values for black in addition. Black can be generated from C, M and Y components but as it is abundant in printed documents, it is of advantage to have a special black ink. Many CMYK colors spaces are used for different sets of inks, substrates, and press characteristics (which change the color transfer function for each ink and thus change the appearance).

HSV – Hue, Saturation, and Value (also known as HSB, hue, saturation, brightness) is often used by painters because it is closer to their thinking and technique. Artists commonly use three to four dozen colors (characterized by the hue; technically, the dominant wavelength). If another color is to be obtained then it is mixed from the given ones, for example, 'purple' or 'orange'. The painter also wants colors of different saturation, e.g., to change 'fire brigade red' to pink. She will mix the 'fire brigade red' with white (and/or black) to obtain the desired lower saturation. The HSV color model is illustrated in Figure 2.33.

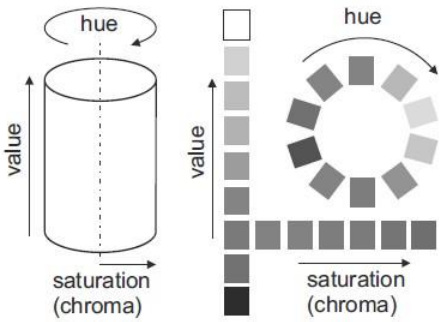


Figure 2.33: HSV color model illustrated as a cylinder and unfolded cylinder. © *Cengage Learning 2015. A color version of this figure may be seen in the color inset—Plate 3.*

HSV decouples intensity information from color, while hue and saturation corre- spond to human perception, thus making this representation very useful for developing image processing algorithms. This will become clearer as we proceed to describe image enhancement algorithms (for example, equalization Algorithm 5.1), which if applied to

each component of an RGB model would corrupt the human sense of color, but which would work more or less as expected if applied to the intensity component of HSV (leaving the color information unaffected). HSL (hue, saturation, lightness/luminance), also known as HLS or HSI (hue, saturation, intensity) is similar to HSV. 'Lightness' replaces 'brightness'. The difference is that the brightness of a pure color is equal to the brightness of white, while the lightness of a pure color is equal to the lightness of a medium gray.

Color Models

	spaces	
Colorimetric	XYZ	Colorimetric calculations
Device oriented, nonuniform spaces	RGB, UIQ	Storage, processing, coding, color TV
Device oriented, Uniform spaces	LAB, LUV	Color difference, analysis
User oriented	HSL, HIS	Color perception, computer graphics

Applications

Palette images:

Palette images (also called **indexed images**) provide a simple way to reduce the amount of data needed to represent an image. The pixel values constitute a link to a **lookup table** (also called a color table, color map, **palette**). The table contains as many entries as the range of possible values in the pixel, which is typically 8 bits 256 values. Each entry of the table ≡ maps the pixel value to the color, so there are three values, one for each of three color components. In the typical case of the RGB color model, values for red, green and blue are provided. It is easy to see that this approach would reduce data consumption to one-third if each of the RGB channels had originally been using 8 bits (plus size of the look up table).

Many widely used image formats for raster images such as TIFF, PNG and GIF can store palette images.

If the number of colors in the input image is less than or equal to the number of entries in the lookup table then all colors can be selected and no loss of information occurs. Such images may be cartoon movies, or program outputs. In the more common case, the number of colors in the image exceeds the number of entries in the lookup table, a subset of colors has to be chosen, and a loss of information occurs.

This color selection may be done many ways. The simplest is to quantize color space regularly into cubes of the same size. In the 8 bit example, there would be $8 \times 8 \times 8 = 512$ such cubes. If there is, e.g., a green frog in green grass in the picture then there will not be enough shades of green available in the lookup table to display the image well. In such a case, it is better to check which colors appear in the image by creating histograms for all three color components and quantize them to provide more shades for colors which occur in the image frequently. If an image is converted to a palette representation then the nearest color (in some metric sense) in the lookup table is used to represent the original color. This is an instance of **vector quantization** which is widely used in analyzing large multi-dimensional datasets. It is also possible to view the occupation by the pixels of RGB space as a **cluster analysis** problem, susceptible to algorithms such as k-means.

The term **pseudo-color** is usually used when an original image is gray-level and is displayed in color; this is often done to exploit the color discriminatory power of human vision. The same palette machinery as described above is used for this purpose; a palette is loaded into the lookup table which visualizes the particular gray-scale image the best. It could either enhance local changes, or might provide various views of the image. Which palette to choose depends on the semantics of the image and cannot be derived from image statistics alone. This selection is an interactive process.

Almost all computer graphics cards work with palette images directly in hardware. The content of the lookup table will be filled by the programmer.

Color constancy

Consider the situation in which the same surface is seen under different illumination, e.g., for a Rubik's cube in Figure 2.34. The same surface colors are shown fully illuminated and in shadow. The human vision system is able to abstract to a certain degree from the illumination changes and perceive several instances of a particular color as the same. This phenomenon is called color constancy. Of course, it would be desirable to equip artificial perception systems based on photosensors with this ability too, but this is very challenging.

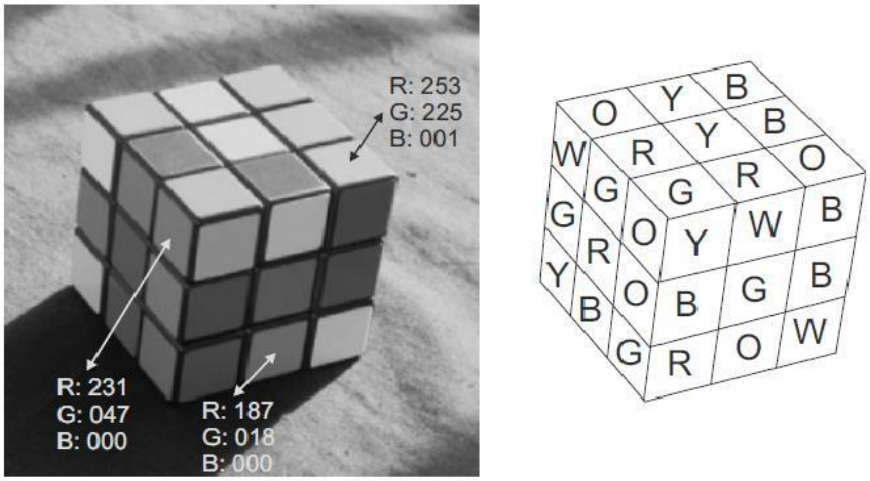


Figure 2.34: Color constancy: The Rubik cube is captured in sunlight, and two of three visible sides of the cube are in shadow. The white balance was set in the shadow area. There are six colors on the cube: R-red, G-green, B-blue, O-orange, W-white, and Y-yellow. The assignment of the six available colors to 3 9 visible color patches is shown on the right. Notice how different the same color patch can be: see *RGB* values for the three instances of orange. © *Cengage Learning 2015.*[×] *A color version of this figure may be seen in the color inset—Plate 4.*

Recall equation (2.16) which models the spectral response q_i of the i-th sensor by integration over a range of wavelengths as a multiplication of three factors: spectral sensitivity $R_i(\lambda)$ of the sensor $i = 1, 2, 3$, spectral density of the illumination $I(\lambda)$, and surface reflectance $S(\lambda)$. A color vision system has to calculate the vector q_i for each pixel as if $I(\lambda) = 1$. Unfortunately, the spectrum of the illuminant $I(\lambda)$ is usually unknown.

Assume for a while the ideal case in which the spectrum $I(\lambda)$ of the illuminant is known. Color constancy could be obtained by dividing the output of each sensor with its sensitivity to the illumination. Let qi' be the spectral response after compensation for the illuminant (called von Kries coefficients), $qi' = \rho_i \, q_i$, where

$$\rho_i = 1 \Big/ \int_{\lambda_1}^{\lambda_2} I(\lambda)\, R_i(\lambda)\, \mathrm{d}\lambda \,. \qquad (2.19)$$

Partial color constancy can be obtained by multiplying color responses of the three photosensors with von Kries coefficients ρ_i.

In practice, there are several obstacles that make this procedure intractable. First, the illuminant spectrum $I(\lambda)$ is not known; it can only be guessed indirectly from reflec tions in surfaces. Second, only the approximate spectrum is expressed by the spectral response q_i of the i-th sensor. Clearly the color constancy problem is ill-posed and cannot be solved without making additional assumptions about the scene.

Several such assumptions have been suggested in the literature. It can be assumed that the average color of the image is gray. In such a case, it is possible to scale the sensitivity of each sensor type until the assumption becomes true. This will result in an insensitivity to the color of the illumination. This type of color compensation is often used in automatic white balancing in video cameras. Another common assumption is that the brightest point in the image has the color of the illumination. This is true when the scene contains specular reflections which have the property that the illuminant is reflected without being transformed by the surface patch.

The problem of color constancy is further complicated by the perceptual abilities of the human visual system. Humans have quite poor quantitative color memory, and also perform color adaptation. The same color is sensed differently in different local contexts.

1.7. DATA STRUCTURES FOR IMAGE ANALYSIS

Data and an algorithm are the two essentials of any program. Data organization often considerably affects the simplicity of the selection and the implementation of an algorithm, and the choice of data structures is therefore a fundamental question when writing a program.

1.8: LEVELS OF IMAGE DATA REPRESENTATION

The aim of computer visual perception is to find a relation between an input image and models of the real world. During the transition from the raw input image to the model, image information becomes denser and semantic knowledge about the interpretation of image data is used more. Several levels of visual information representation are defined on the way between the input image and the model; computer vision then comprises a design of the:

- **Intermediate representations** (data structures).
- **Algorithms** used for the creation of representations and introduction of relations between them.

The representations can be stratified in four levels [Ballard and Brown, 1982]—however, there are no strict borders between them and a more detailed classification of the representational levels is used in some applications. These four representational levels are ordered from signals at a low level of abstraction to the description that a human can perceive. The information flow between the levels may be bi-directional, and for some specific uses, some representations can be omitted.

1) The lowest representational level—**iconic images**—consists of images containing original data: integer matrices with data about pixel brightness. Images of this kind are also outputs of pre-processing operations used for highlighting some aspects of the image important for further treatment.
2) The second level is **segmented images**. Parts of the image are joined into groups that probably belong to the same objects. For instance, the output of the segmentation of a scene with polyhedra is either line segments coinciding with borders or two-dimensional regions corresponding to faces of bodies. It

is useful to know something about the application domain while doing image segmentation; it is then easier to deal with noise and other problems associated with erroneous image data.

3) The third level is **geometric representations** holding knowledge about 2D and 3D shapes. Quantification of a shape is very difficult but also very important. Geometric representations are useful while doing general and complex simulations of the influence of illumination and motion in real objects. We also need them for the transition between natural raster images acquired by a camera) and data used in computer graphics (CAD— computer-aided design, DTP— desktop publishing).
4) The fourth representational level is **relational models**. They give us the ability to treat data more efficiently and at a higher level of abstraction. A priori knowledge about the case being solved is usually used in processing of this kind. Artificial intelligence (AI) techniques are often explored; the information gained from the image may be represented by semantic nets or frames.

An example will illustrate a priori knowledge. Imagine a satellite image of a piece of land, and the task of counting planes standing at an airport; the a priori knowledge is the position of the airport, which can be deduced, for instance, from a map. Relations to other objects in the image may help as well, e.g., to roads, lakes, or urban areas. Additional a priori knowledge is given by geometric models of planes for which we are searching. Segmentation will attempt to identify meaningful regions such as runways, planes and other vehicles, while third-level reasoning will try to make these identifications more definite. Fourth-level reasoning may, for example, determine whether the plane is arriving, departing or undergoing maintenance, etc.

1.9.TRADITIONAL IMAGE DATA STRUCTURES

Traditional image data structures such as matrices, chains, graphs, lists of object properties, and relational databases are important not only for the direct representation of image information, but also as a basis for more complex hierarchical methods of image representation.

1.9.1. Matrices

A **matrix** is the most common data structure for low-level representation of an image. Elements of the matrix are integer numbers corresponding to brightness, or to another property of the corresponding pixel of the sampling grid. Image data of this kind are usually the direct output of the image-capturing device. Pixels of both rectangular and hexagonal sampling grids can be represented by a matrix. The correspondence between data and matrix elements is obvious for a rectangular grid; with a hexagonal grid every even row in the image is shifted half a pixel to the right.

Image information in the matrix is accessible through the co-ordinates of a pixel that correspond with row and column indices. The matrix is a full representation of the image, independent of the contents of image data—it implicitly contains **spatial relations** among semantically important parts of the image. The space is two-dimensional in the case of an image. One very natural spatial relation is the **neighborhood relation**. Some special images that are represented by matrices are:

- A **binary image** (an image with two brightness levels only) is represented by a matrix containing only zeros and ones.

- Several matrices can contain information about one **multispectral image**. Each single matrix contains one image corresponding to one spectral band.

- Matrices of different resolution are used to obtain **hierarchical image data structures**. Such hierarchical representations can be very convenient for parallel computers with the 'processor array' architecture.

Most programming languages use a standard array data structure to represent a matrix. Historically, memory limitations were a significant obstacle to image applications, but this is no longer the case.

There is much image data in the matrix. Algorithms can be sped up if global information is derived from the original image matrix first—global information is more concise and occupies less memory. We have already mentioned the most popular example of global information—the histogram. Looking at the image from a probabilistic point of view, the normalized histogram is an estimate of the probability density of a phenomenon: that an image pixel has a certain brightness.

Another example of global information is the **co-occurrence matrix** [Pavlidis, 1982], which represents an estimate of the probability of two pixels appearing in a spatial relationship in which a pixel (i_1, j_1) has intensity x and a pixel (i_2, j_2) has intensity y. Suppose that the probability depends only on a certain spatial relation r between a pixel of brightness x and a pixel of brightness y; then information about the relation r is recorded in the square co-occurrence matrix C_r, whose dimensions correspond to the number of brightness levels of the image. To reduce the number of matrices C_r, introduce some simplifying assumptions; first consider only direct neighbors, and then treat relations as symmetrical (without orientation). The following algorithm calculates the co-occurrence matrix C_r from the image $f(i, j)$.

Algorithm 4.1: Co-occurrence matrix $C_r(x, y)$ for the relation r

1. Set $C_r(x, y) = 0$ for all $x, y \in [0, L]$, where L is the maximum brightness.
2. For all pixels (i_1, j_1) in the image, determine all (i_2, j_2) which have the relation r with the pixel (i_1, j_1), and perform

$$C_r f(i_1, j_1), f(i_2, j_2) = C_r f(i_1, j_1), f(i_2, j_2) + 1 .$$

If the relation r is *to be a southern or eastern 4-neighbor of the pixel* (i_1, j_1)*, or identity*, elements of the co-occurrence matrix have some interesting properties. Diagonal elements of the matrix $C_r(k, k)$ are equal to the area of the regions in the image with brightness k, and so correspond to the histogram. Off-diagonal elements $C_r(k, j)$ are equal to the length of the border dividing regions with brightnesses k and j, $k\,j$. For instance, in an image with low contrast, the elements of the co-occurrence matrix that are far from the diagonal are equal to zero or are very small. For high-contrast images the opposite is true.

The main reason for considering co-occurrence matrices is their ability to describe texture: this approach is introduced in Chapter 15.

The **integral image** is another matrix representation that holds global image information [Viola and Jones, 2001]. It is constructed so that its values $ii(i, j)$ in the location (i, j) represent the sums of all the original image pixel-values left of and above (i, j):

$$ii(i,j) = \sum_{k \le i, l \le j} f(k,l)\,, \qquad (4.1)$$

where f is the original image. The integral image can be efficiently computed in a single image pass using recurrences:

Algorithm 4.2: Integral image construction

1. Let $s(i, j)$ denote a cumulative row sum, and set $s(i, -1) = 0$.
2. Let $ii(i, j)$ be an integral image, and set $ii(-1, j) = 0$.
3. Make a single row-by-row pass through the image. For each pixel (i, j) calculate the cumulative row sums $s(i, j)$ and the integral image value $ii(i, j)$ using

$$s(i, j) = s(i, j - 1) + f(i, j)\,, \quad (4.2)$$

$$ii(i, j) = ii(i - 1, j) + s(i, j)\,. \quad (4.3)$$

4. After completing a single pass through the image, the integral image ii is constructed.

The main use of integral image data structures is in rapid calculation of simple rectangle image features at multiple scales. This kind of features is used for rapid object identification (Section 10.7) and for object tracking (Section 16.5).

Figure 4.1 illustrates that any rectangular sum can be computed using four array references, and so a feature reflecting a difference between two rectangles requires eight references. Considering the rectangle features shown in Figure 4.2a,b, the two-rectangle features require only six array references since the rectangles are adjacent. Similarly, the three- and four-rectangle features of Figure 4.2c,d can be calculated using eight and nine references to the integral image values, respectively. Such features can be computed extremely efficiently and in constant time once the integral image is formed.

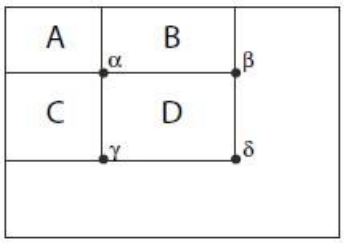


Figure 4.1: Calculation of rectangle features from an integral image. The sum of pixels within rectangle *D* can be obtained using four array references. $Dsum^{-} = ii(\delta) + ii(\alpha)$ $(ii(\beta) + ii(\gamma))$, where $ii(\alpha)$ is the value of the integral image at point α (and similarly for β, γ, δ). .

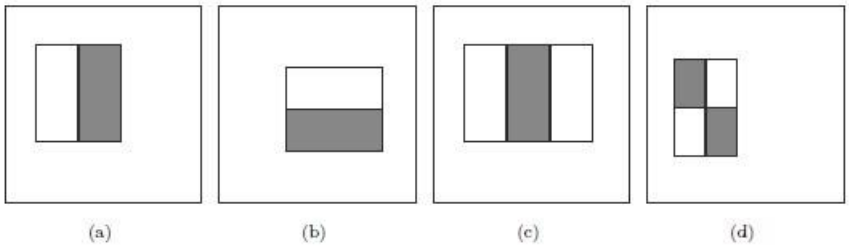


Figure 4.2: Rectangle-based features may be calculated from an integral image by subtraction of the sum of the shaded rectangle(s) from the non-shaded rectangle(s). The figure shows (a,b) two- rectangle, (c) three-rectangle, and (d) four-rectangle features. Sizes of the individual rectangles can be varied to yield different features as well as features at different scales. Contributions from the regions may be normalized to account for possibly unequal region sizes. © *Cengage Learning 2015.*

Chains

Chains are used for the description of object borders in computer vision. One element of the chain is a basic symbol; this approach permits the application of formal language theory for computer vision tasks. Chains are appropriate for data that can be arranged as a sequence of symbols, and the neighboring symbols in a chain usually correspond to the neighborhood of primitives in the image. The primitive is the basic descriptive element that is used in syntactic pattern recognition.

This rule of proximity (neighborhood) of symbols and primitives has exceptions—for example, the first and the last symbol of the chain describing a closed border are not neighbors, but the corresponding primitives in the image are. Similar inconsistencies are typical of image description languages, too. Chains are linear structures, which is why they cannot describe spatial relations in the image on the basis of neighborhood or proximity.

Chain codes are often used for the description of object borders, or other one-pixel-wide lines in images. The border is defined by the co-ordinates of its reference pixel and the sequence of symbols corresponding to the line of the unit length in several pre-defined orientations. Notice that a chain code is of a relative nature; data are expressed with respect to some reference point. Figure 4.3 shows an example of a chain code in which where 8neighborhoods are used—4-neighborhoods can be used as well.

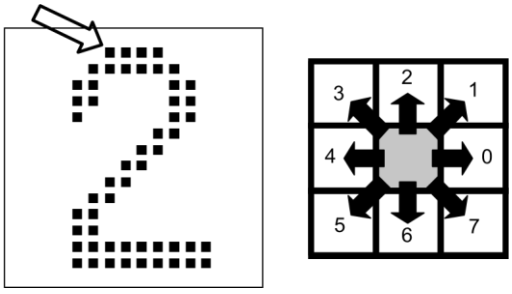


Figure 4.3: An example chain code; the reference pixel starting the chain is marked by an arrow: 00077665555556600000006444444442221111112234445652211.

If local information is needed from the chain code, then it is necessary to search through the whole chain systematically. For instance, if we want to know whether the border turns somewhere to the left by 90◦, we must just find a sample pair of symbols in the chain—it is simple. On the other hand, a question about the shape of the border near the pixel (i_0, j_0) is not trivial. It is necessary to investigate all chain elements until the pixel (i_0, j_0) is found and only then we can start to analyze a short part of the border that is close to the pixel (i_0, j_0).

The description of an image by chains is appropriate for syntactic pattern recognition based on formal language theory methods. When working with real images, the problem of how to deal with uncertainty caused by noise arises, which is why several syntactic analysis techniques with deformation correction have arisen [Lu and Fu, 1978]. Another way to deal with noise is to smooth the border or to approximate it by another curve. This new border curve is then described by chain codes [Pavlidis, 1977].

Run length coding has been used for some time to represent strings of symbols in an image matrix. For simplicity, consider a binary image first. Run length coding records only areas that belong to objects in the image; the area is then represented as a list of lists. Various schemes exist which differ in detail—a representative one describes each row of the image by a sublist, the first element of which is the row number. Subsequent terms are coordinate pairs; the first element of a pair is the beginning of a run and the second is the end (the beginning and the end are described by column coordinates). There can be several such sequences in the row. Run length coding is illustrated in Figure 4.4. The main advantage of run length coding is the existence of simple algorithms for intersections and unions of regions in the image.

Run length coding can be used for an image with multiple brightness levels as well—in this case sequences of neighboring pixels in a row that has constant brightness are considered. In the sublist we must record not only the beginning and the end of the sequence, but its brightness, too.

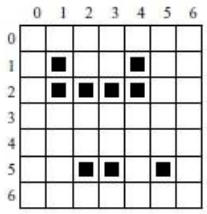


Figure 4.4: Run length coding; the code is ((1114)(214)(52355)).

Topological data structures

Topological data structures describe the image as a set of elements and their relations; these relations are often represented using graphs. A **graph** $G = (V, E)$ is an algebraic structure which consists of a set of nodes $V = \{v_1, v_2, \ldots, v_n\}$ and a set of arcs $E = \{ e_1, e_2,$

. . . , e_m } . Each arc e_k is incident to an unordered (or ordered) pair of nodes {v_i, v_j } which are not necessarily distinct. The *degree* of a node is equal to the number of incident arcs of the node.

A **weighted graph** is a graph in which values are assigned to arcs, to nodes, or to both—these values may, for example, represent weights, or costs.

The **region adjacency graph** is typical of this class of data structures, in which nodes correspond to regions and neighboring regions are connected by an arc. The segmented image consists of regions with similar properties (brightness, texture, color, . . .) that correspond to some entities in the scene, and the neighborhood relation is fulfilled when the regions have some common border. An example of an image with areas labeled by numbers and the corresponding region adjacency graph is shown in Figure 4.5; the label 0 denotes pixels out of the image. This label is used to indicate regions that touch borders of the image in the region adjacency graph.

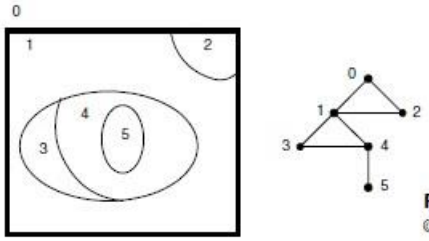


Figure 4.5: An example region adjacency graph.

The region adjacency graph has several attractive features. If a region encloses other regions, then the part of the graph corresponding with the areas inside can be separated by a cut in the graph. Nodes of degree 1 represent simple holes.

Arcs of the graph can include a description of relations between neighboring regions— the relations *to be to the left* or *to be inside* are common. It can be used for matching with a stored pattern for recognition purposes.

The region adjacency graph is usually created from the **region map**, which is a matrix of the same dimensions as the original image matrix whose elements are identification labels of the regions. To create the region adjacency graph, borders of all regions in the image are traced, and labels of all neighboring regions are stored. The region adjacency graph can also easily be created from an image represented by a quadtree (Section 4.3.2).

The region adjacency graph stores information about the neighbors of all regions in the image explicitly. The region map contains this information as well, but it is much more difficult to recall from there. If we want to relate the region adjacency graph to the region map quickly, it is sufficient for a node in the region adjacency graph to be marked by the identification label of the region and some representative pixel (e.g., the top left pixel of the region).

Construction of the boundary data structures that represent regions is not trivial, and is considered in Section 6.2.3. Region adjacency graphs can be used to approach region merging (where, for instance, neighboring regions thought to have the same image interpretation are merged into one region)—this topic is considered in Section 10.10. In particular, note that merging representations of regions that may border each other more than once can be intricate, for example, with the creation of 'holes' not present before the merge—see Figure 4.6.

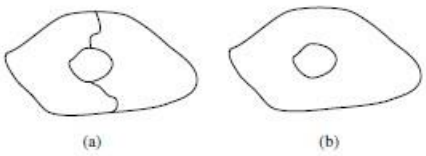


Figure 4.6: Region merging may create holes: (a) Before a merge. (b) After.

Relational structures

Relational databases [Kunii et al., 1974] can also be used for representation of information from an image; all the information is then concentrated in relations between semantically important parts of the image—objects—that are the result of segmentation. Relations are recorded in the form of tables. An example of such a representation is shown in Figure 4.7 and Table 4.1, where individual objects are associated with their names and other features, e.g., the top-left pixel of the corresponding region in the image. Relations between objects are expressed in the relational table. Here, such a relation is *to be inside*; for example, the object 7 (pond) is situated inside the object 6 (hill).

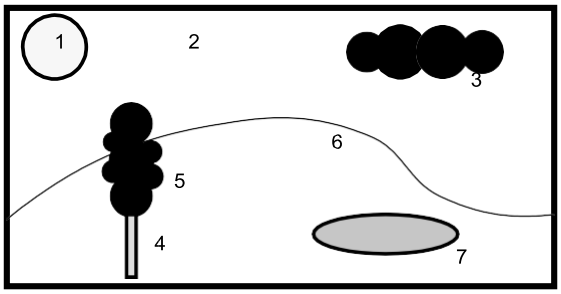


Figure 4.7: Description of objects using relational structure.

No.	Object name	Color	Min. row	Min. col.	Inside
1	sun	white	5	40	2
2	sky	blue	0	0	–
3	cloud	gray	20	180	2
4	tree trunk	brown	95	75	6
5	tree crown	green	53	63	–
6	hill	light green	97	0	–
7	pond	blue	100	160	6

Table 4.1: Relational table. © *Cengage Learning 2015.*

Description by means of relational structures is appropriate for higher levels of image understanding. In this case searches using keys, similar to database searches, can be used to speed up the whole process.

1.10 HIERARCHICAL DATA STRUCTURES

Computer vision is by its nature very computationally expensive, if for no other reason than the large amount of data to be processed. Usually a very quick response is expected because video real-time or interactive systems are desirable. One of the solutions is to use parallel computers (in other words brute force). Unfortunately there are many computer vision problems that are very difficult to divide among processors, or decompose in any way. Hierarchical data structures make it possible to use algorithms which decide a strategy for processing on the basis of relatively small quantities of data. They work at the finest resolution only with those parts of the image for which it is essential, using knowledge instead of brute force to ease and speed up the processing. We are going to introduce two typical hierarchical structures, pyramids and quadtrees.

Pyramids

Pyramids are among the simplest hierarchical data structures. We distinguish between **M-pyramids** (matrix-pyramids) and **T-pyramids** (tree-pyramids).

A **Matrix-pyramid** (M-pyramid) is a sequence $\{ M_L, M_{L-1}, \ldots, M_0 \}$ of images, where M_L has the same dimensions and elements as the original image, and M_{i-1} is derived from the M_i by reducing the resolution by one-half. When creating pyramids, it is customary to work with square matrices having dimensions equal to powers of 2—then M_0 corresponds to one pixel only.

M-pyramids are used when it is necessary to work with an image at different resolutions simultaneously. An image having one degree smaller resolution in a pyramid contains four times less data, so it is processed approximately four times as quickly.

Often it is advantageous to use several resolutions simultaneously rather than choose just one image from the M-pyramid. For such algorithms we prefer to use **tree-pyramids**, a tree structure.

Let $2L$ be the size of an original image (the highest resolution). A tree- pyramid (T-pyramid) is defined by:

1. A set of nodes $P = \{ p = (k, i, j)$ such that level $k \in [0, L]$; $i, j \in [0, 2k - 1] \}$
2. A mapping F between subsequent nodes P_{k-1}, P_k of the pyramid
$$F(k, i, j) = (k - 1, \text{floor}(i/2), \text{floor}(j/2)) .$$
3. A function V that maps a node of the pyramid P to Z, where Z is the subset of the whole numbers corresponding to the number of brightness levels, for example, $Z = \{0, 1, 2, \ldots, 255\}$.

Nodes of a T-pyramid correspond for a given k with image points of an M-pyramid; elements of the set of nodes $P = \{ (k, i, j) \}$ correspond with individual matrices in the Mpyramid—k is called the level of the pyramid. An image $P = \{ (k, i, j) \}$ for a specific k constitutes an image at the k^{th} level of the pyramid. F is the so-called parent mapping, which is defined for all nodes P_k of the T-pyramid except its root (0, 0, 0). Every node of

the T-pyramid has four child nodes except leaf nodes, which are nodes of level L that correspond to the individual pixels in the image.

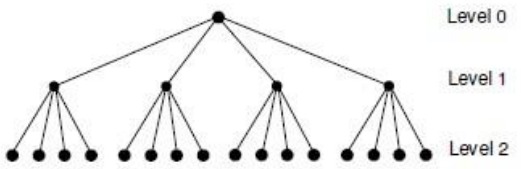


Figure 4.8: T-pyramid.

Values of individual nodes of the T-pyramid are defined by the function V. Values of leaf nodes are the same as values of the image function (brightness) in the original image at the finest resolution; the image size is $2L$. Values of nodes

in other levels of the tree are either an arithmetic mean of four child nodes or they are defined by coarser sampling, meaning that the value of one child (e.g., top left) is used. Figure 4.8 shows the structure of a simple T-pyramid.

The number of image pixels used by an M-pyramid for storing all matrices is given by

$$N^2\left(1+\frac{1}{4}+\frac{1}{16}+\ldots\right)\approx 1.33\,N^2, \qquad (4.4)$$

where N is the dimension of the original matrix (the image of finest resolution)—usually a power of two, $2L$.

The T-pyramid is represented in memory similarly. Arcs of the tree need not be recorded because addresses of the both child and parent nodes are easy to compute due to the regularity of the structure. An algorithm for the effective creation and storing of a Tpyramid is given in [Pavlidis, 1982].

Quadtrees

Quadtrees are modifications of T-pyramids. Every node of the tree except the leaves has four children (NW, north-western; NE, north-eastern; SW, south-western; SE, south-eastern). Similarly to T-pyramids, the image is divided into four quadrants at each hierarchical level; however, it is not necessary to keep nodes at all levels. If a parent node has four children of the same value (e.g., brightness), it is not necessary to record

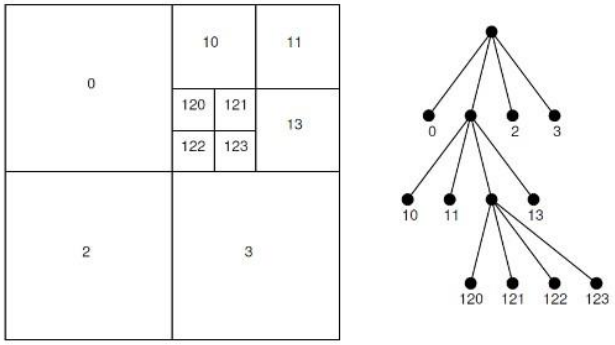


Figure 4.9: Quadtree..

them. This representation is less expensive for an image with large homogeneous regions; Figure 4.9 is an example of a simple quadtree.

An advantage of image representation by means of quadtrees is the existence of simple algorithms for addition of images, computing object areas, and statistical moments. The main disadvantage of quadtrees and pyramid hierarchical representations is their dependence on the position, orientation, and relative size of objects. Two similar images with just very small differences can have very different pyramid or quadtree representations. Even two images depicting the same, slightly shifted scene, can have entirely different representations.

These disadvantages can be overcome using a normalized shape of quadtree in which we do not create the quadtree for the whole image, but for its individual objects. Geometric features of objects such as the center of gravity and principal axis are used; the center of gravity and principal axis of every object are derived first and then the smallest enclosing square centered at the center of gravity having sides parallel with the principal axes is located. The square is then represented by a quadtree. An object described by a normalized quadtree and several additional items of data (co- ordinates of the center of gravity, angle of main axes) is invariant to shifting, rotation, and scale.

Quadtrees are usually represented by recording the whole tree as a list of its individual nodes, every node being a record with several items characterizing it. An example is given in Figure 4.10. In the item *Node type* there is information about whether the node is a leaf or inside the tree. Other data can be the level of the node in the tree, position in the picture, code of the node, etc. This kind of representation is expensive in memory. Its advantage is easy access to any node because of pointers between parents and children.

Node type
Pointer to the NW son
Pointer to the NE son
Pointer to the SW son
Pointer to the SE son
Pointer to the father
Other data

Figure 4.10: Record describing a quadtree node.

.

It is possible to represent a quadtree with less demand on memory by means of a **leaf code**. Any point of the picture is coded by a sequence of digits reflecting successive divisions of the quadtree; zero means the NW (north-west) quadrant, and likewise for other quadrants: 1-NE, 2-SW, 3-SE. The most important digit of the code (on the left) corresponds to the division at the highest level, the least important one (on the right) with the last division. The number of digits in the code is the same as the number of levels of

the quadtree. The whole tree is then described by a sequence of pairs—the leaf code and the brightness of the region. Programs creating quadtrees can use recursive procedures to advantage.

T-pyramids are very similar to quadtrees, but differ in two basic respects. A T- pyramid is a balanced structure, meaning that the corresponding tree divides the image regardless of the contents, which is why it is regular and symmetric. A quadtree is not balanced. The other difference is in the interpretation of values of the individual nodes. Quadtrees have seen widespread application, particularly in the area of Geographic Information Systems (GIS) where, along with their three-dimensional generalization *octrees*, they have proved very useful in hierarchical representation of layered data.

Other pyramidal structures

The pyramidal structure is widely used, and has seen several extensions and modi- fications. Recalling that a (simple) M-pyramid was defined as a sequence of images $\{ M_L, M_{L-1}, \ldots, M_0 \}$ in which M_i is a 2 x 2 reduction of M_{i+1}, we can define the notion of a **reduction window**; for every cell c of M_i, the reduction window is its set of children in M_{i+1}, $w(c)$. Here, a *cell* is any single element of the image M_i at the corresponding level of pyramidal resolution. If the images are constructed such that all interior cells have the same number of neighbors (e.g., a square grid, as is customary), and they all have the same number of children, the pyramid is called **regular**.

A taxonomy of regular pyramids may be constructed by considering the reduction window together with the **reduction factor** λ, which defines the rate at which the image area decreases between levels;

$$\lambda \leq \frac{|M_{i+1}|}{|M_i|}, i = 0, 1, \ldots, L-1 \, . \qquad -$$

In the simple case, in which reduction windows do not overlap and are× 2 2, we have $\lambda = 4$; if we choose to let the reduction windows overlap, the factor will reduce. The notation used to describe this characterization of regular pyramids is (*reduction window*)/(*reduction factor*). Figure 4.11 illustrates some simple examples.

The reduction window of a given cell at level i may be propagated down to higher resolution than level $i + 1$. For a cell c_i at level i, we can write $w^0(c_i) = w(c_i)$, and then recursively define

$$w^{k+1}(c_i) = \bigcup_{q \in w(c_i)} w^k(q) \, , \qquad (4.5)$$

$^k(c_i)$ is the **equivalent window** that covers all cells at level $i+k+1$ that link to the cell c_i. Note that the shape of this window is going to depend on the type of pyramid—for example, an $n \times n/2$ pyramid will generate octagonal equivalent windows, while for an

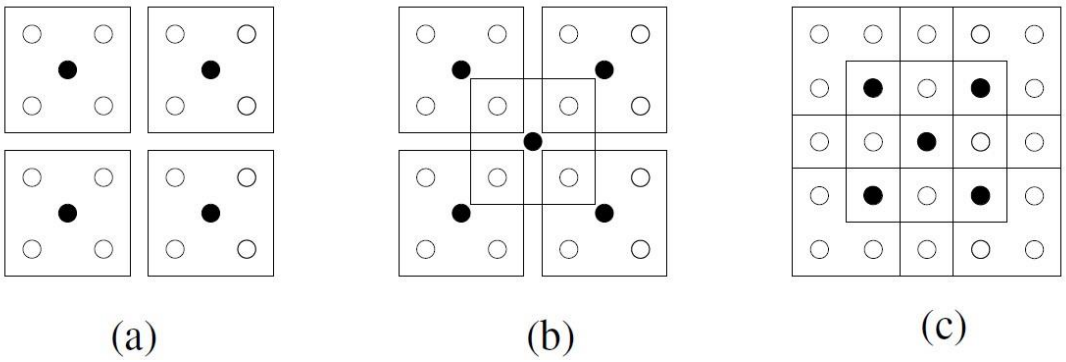


Figure 4.11: Several regular pyramid definitions.
(a) 2 x 2/4. (b) 2 x 2/2. (c) 3 x 3/2. (Solid dots are at the higher level, i.e., the lowerresolution level.)

nx n/4 pyramid they will be square. Use of non-square windows prevents domination of square features, as is the case for simple 2x 2/4 pyramids.

The 2 × 2/4 pyramid is widely used and is what is usually called an 'image pyramid'; the 2 × 2/2 structure is often referred to as an 'overlap pyramid'. 5 × 5/2 pyramids have been used in compact image coding, where the image pyramid is augmented by a **Laplacian** pyramid of differences. Here, the Laplacian at a given level is computed as the per-pixel difference between the image at that level, and the image derived by 'expanding' the image at the next lower resolution. The Laplacian may be expected to have zero (or close) values in areas of low contrast, and therefore be amenable to compression.

Irregular pyramids are derived from contractions of graphical representations of images (for example, region adjacency graphs). Here, a graph may be reduced to a smaller one by selective removal of arcs and nodes. Depending on how these selections are made, important structures in the parent graph may be retained while reducing its overall complexity. The pyramid approach is quite general and lends itself to many developments—for example, the reduction algorithms need not be deterministic.

UNIT II IMAGE PRE-PROCESSING

Local pre-processing - Image smoothing - Edge detectors - Zero-crossings of the second derivative - Scale in image processing - Canny edge detection - Parametric edge models - Edges in multispectral images - Local pre-processing in the frequency domain - Line detection by local preprocessing operators - Image restoration.

2.1. Local pre-processing:

Local pre-processing methods are divided into two groups according to the goal of the processing.

1) **Smoothing** aims to suppress noise or other small fluctuations in the image; it is equivalent to the suppression of high frequencies in the Fourier transform domain. Unfortunately, smoothing also blurs all sharp edges that bear important information about the image.

2) **Gradient operators** are based on local derivatives of the image function. Derivatives are bigger at locations of the image where the image function undergoes rapid changes, and the aim of gradient operators is to indicate such locations in the image.

   Gradient operators have a similar effect to suppressing low frequencies in the Fourier transform domain.

Noise is often high frequency in nature; if a gradient operator is applied to an image, the noise level increases simultaneously. Clearly, smoothing and gradient operators have conflicting aims. Some pre-processing algorithms solve this problem and permit smoothing and edge enhancement simultaneously.

Another classification of local pre-processing methods is according to the transformation properties; **linear** and **non-linear** transformations can be distinguished. Linear operations calculate the resulting value in the output image pixel f (i, j) as a linear combination of brightnesses in a local neighborhood O of the pixel g(i, j) in the input image. The contribution of the pixels in the neighborhood O is weighted by coefficients h:

$$f(i,j) = \sum_{(m,n)\in\mathcal{O}} \sum h(i-m, j-n)\, g(m,n)\,. \tag{5.23}$$

Equation (5.23) is equivalent to discrete convolution with the kernel h, which is called a **convolution mask**. Rectangular neighborhoods are often used with an odd number of pixels in rows and columns, enabling specification of the central pixel of the neighbor- hood.

The choice of the local transformation, size, and shape of the neighborhood depends strongly on the size of objects in the processed image. If objects are rather large, an image can be enhanced by smoothing of small degradations.

2.2 Image smoothing:

Image smoothing uses redundancy in image data to suppress noise, usually by some form of averaging of brightness values in some neighborhood . Smoothing poses the problem of blurring sharp edges, and so we shall consider smoothing methods which are **edge preserving** here, the average is computed only from points in the neighborhood which have similar properties to the point being processed.

Local image smoothing can effectively eliminate impulse noise or degradations appearing as thin stripes, but does not work if degradations are large blobs or thick stripes. Such problems may be addressed by image restoration techniques.

Averaging, statistical principles of noise suppression

Assume that the noise value ν at each pixel is an independent random variable with zero mean and standard deviation σ. We might capture the same static scene under the same conditions n times. From each captured image a particular pixel value g_i, i = 1, . . . , n is selected. An estimate of the correct value can be obtained as an average of these values, with corresponding noise values $\nu_1, \ldots, \nu_n$

$$\frac{g_1 + \ldots + g_n}{n} + \frac{\nu_1 + \ldots + \nu_n}{n} .$$

The second term here describes the noise, which is again a random value with zero mean and standard deviation $\sigma/\sqrt{n}$. Thus, if n images of the same scene are available, smoothing can be accomplished without blurring the image by

$$f(i,j) = \frac{1}{n} \sum_{k=1}^{n} g_k(i,j) .$$

This reasoning is a well-known statistical result: a random sample is taken from a population and the corresponding sample mean value is calculated. If random samples are repeatedly selected and their sample mean values calculated, we would obtain a distribution of sample mean values. This distribution of sample means has some useful properties:

- The mean value of the distribution of sample mean values is equal to the population mean.
- The distribution of sample mean values has variance $\sigma/\sqrt{n}$, which is clearly smaller than that of than the original population.
- If the original distribution is normal (Gaussian) then the distribution of sample mean values is also normal. Better, the distribution of sample means

converges to normal whatever the original distribution. This is the **central limit theorem**.

- From the practical point of view, it is important that not too many random se- lections have to be made. The central limit theorem tell us the distribution of sample mean values without the need to create them. In statistics, usually about

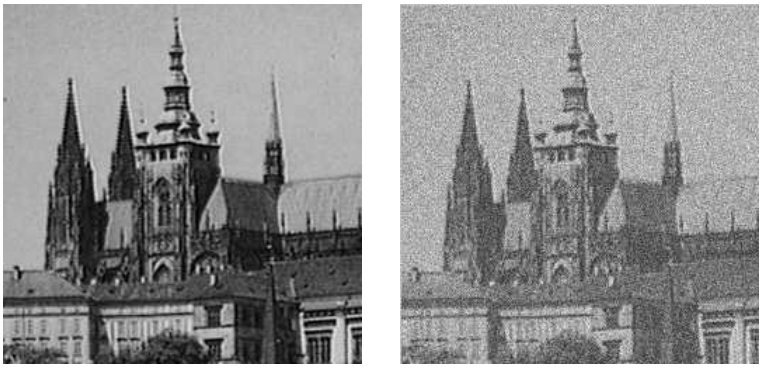
30 samples are considered the lowest limit of the necessary number of observations. Usually, only one noise corrupted is available, and averaging is then performed in a local neighborhood. Results are acceptable if the noise is smaller in size than the smallest objects of interest in the image, but blurring of edges is a serious disadvantage. Averaging is a special case of discrete convolution [equation (5.23)]. × For a 3x 3 neighborhood, the convolution mask h is

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} .$$

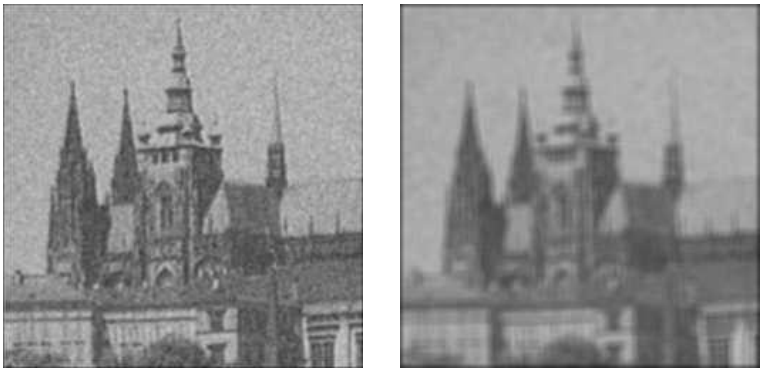
The significance of the pixel in the center of the convolution mask h or its 4neighbors is sometimes increased, as it better approximates the properties of noise with a Gaussian probability distribution.

$$h = \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} , \qquad h = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

There are two commonly used smoothing filters whose coefficients gradually decrease to have near-zero values at the window edges. This is the best way to minimize spurious oscillations in the frequency spectrum. These are the Gaussian and the Butterworth filters. Larger convolution masks for averaging by Gaussian filter are created according to the Gaussian distribution formula (equation 5.47) and the mask coefficients are normalized to have a unit sum.



(a) (b)



(c) (d)

Figure 5.9: Noise with Gaussian distribution and averaging filters. (a) Original image. (b) Superimposed noise (random Gaussian noise characterized by zero mean and standard deviation equal to one-half of the gray-level standard deviation of the original image). (c) 3 × 3 averaging. (d) 7 × 7 averaging.

An example will illustrate the effect of this noise suppression (low resolution images, 256 256, were chosen deliberately to show the discrete nature of the process). Figure 5.9a shows an original image of Prague castle; Figure 5.9b shows the same image with superimposed additive noise with Gaussian distribution; Figure 5.9c shows the result of averaging with a 3x 3 convolution mask (equation 5.27)— noise is significantly reduced and the image is slightly blurred. Averaging with a larger mask (7 x 7) is demonstrated in Figure 5.9d, where the blurring is much more serious.

Such filters can be very computationally costly, but this is considerably reduced in the important special case of **separable filters**. Separability in 2D means that the convolution kernel can be factorized as a product of two one-dimensional vectors, and theory provides a clue as to which convolution masks are separable.

As an example, consider a binomic filter. Its elements are binomic numbers which are created as a sum of the corresponding two numbers in Pascal's triangle. Consider such a filter of size 5x 5—it can be decomposed into a product of two

1D vectors, h_1, h_2.

$$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \begin{bmatrix} \\ h_1 \\ \\ \end{bmatrix} \begin{bmatrix} \quad h_2 \quad \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Suppose a convolution kernel is of size 2N + 1. Equation (5.23) allows the convolution to be Rewritten taking account of the special properties of separability

$$g(x,y) = \sum_{m=-N}^{N} \sum_{n=-N}^{N} h(m,n)\, f(x+m, y+n) = \sum_{m=-N}^{N} h_1(m) \sum_{n=-N}^{N} h_2(n)\, f(x+m, y+n).$$

The direct calculation of the convolution according to equation (5.23) would need, in our case of 5 5 convolution kernel, 25 multiplications and 24 additions for each pixel. If the separable filter is used then only 10 multiplications and 8 additions suffice.

Averaging with limited data validity

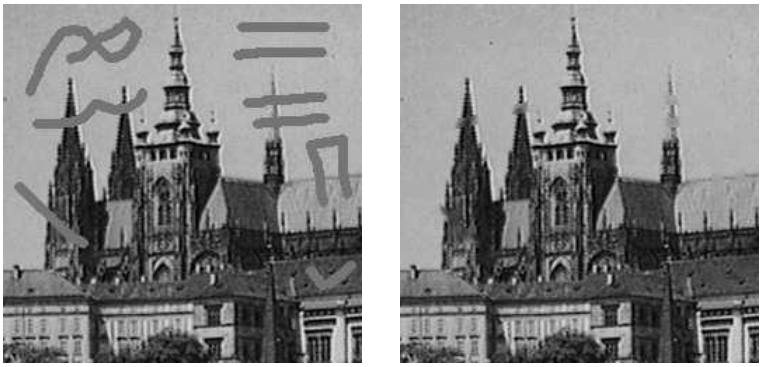
Methods that average with limited data validity try to avoid blurring by averaging only those pixels which satisfy some criterion, the aim being to prevent involving pixels that are part of a separate feature.

A very simple criterion is to define a brightness interval of invalid data [min, max] (typically corresponding to noise of known image faults), and apply image averaging only to pixels in that interval. For a point (m, n), the convolution mask is calculated in the neighborhood O by the non-linear formula

$$h(i,j) = \begin{cases} 1 & \text{for } g(m+i, n+j) \notin [\text{min,max}] \\ 0 & \text{otherwise} \end{cases},$$

where (i, j) specify the mask element. Therefore, only values of pixels with invalid gray-levels are replaced with an average of their neighborhoods, and only valid data contribute to the averages.

A second method performs averaging only if the computed brightness change of a pixel is in some pre-defined interval; this permits repair to large-area errors resulting from slowly changing brightness of the background without affecting the rest of the image. A third method uses edge strength (i.e., gradient magnitude) as a criterion. The magnitude of some gradient operator is first computed for the entire image, and only pixels with a small gradient are used in averaging. This method effectively rejects averaging at edges and therefore suppresses blurring, but setting of the threshold is laborious.



(a) (b)

Figure 5.10: Averaging with limited data validity. (a) Original corrupted image. (b) Result of corruption removal.

Averaging according to inverse gradient

Within a convolution mask of odd size, the inverse gradient δ of a point (i, j) with respect to the central pixel (m, n) is defined as

$$\delta(i,j) = \frac{1}{|g(m,n) - g(i,j)|} \,. \tag{5.29}$$

If g(m, n) = g(i, j), then we define δ(i, j) = 2, so δ is in the interval (0, 2], and is smaller at the edge than in the interior of a homogeneous region. Weight coefficients in the convolution mask h are normalized by the inverse gradient, and the whole term is multiplied by 0.5 to keep brightness values in the original range: the mask coefficient corresponding to the central pixel is defined as h(i, j) = 0.5. The constant 0.5 has the effect of assigning half the weight to the central pixel (m, n), and the other half to its neighborhood

$$h(i,j) = 0.5 \frac{\delta(i,j)}{\sum_{(m,n)\in\mathcal{O}} \delta(i,j)} . \qquad (5.30)$$

This method assumes sharp edges. When the convolution mask is close to an edge, pixels from the region have larger coefficients than pixels near the edge, and it is not blurred. Isolated noise points within homogeneous regions have small values of the inverse gradient; points from the neighborhood take part in averaging and the noise is removed.

Averaging using a rotating mask

The smoothing discussed thus far was linear, which has the disadvantage that edges in the image are inevitably blurred. Alternative non-linear methods exist which reduce this. The neighborhood of the current pixel is inspected and divided into two subsets by a homogeneity criterion of the user's choice. One set consists of all pixels neighboring the current pixel or any pixel already included in this set, which satisfy the homogeneity criterion. The second set is the complement. This selection operation is non-linear and causes the whole filter to be non-linear. Having selected the homogeneous subset containing the current pixel, the most probable value is sought in it by a linear or non- linear technique.

Averaging using a rotating mask is such a non-linear method that avoids edge blur- ring, and the resulting image is in fact sharpened. The brightness average is calculated only within this region; a brightness dispersion σ^2 is used as the region homogeneity measure. Let n be the number of pixels in a region R and g be the input image. Dispersion σ^2 is calculated as

$$\sigma^2 = \frac{1}{n} \sum_{(i,j)\in R} \left(g(i,j) - \frac{1}{n} \sum_{(i,j)\in R} g(i,j) \right)^2 . \qquad (5.31)$$

Having computed region homogeneity, we consider its shape and size. The eight possible 3x3 masks that cover a 5 x 5 neighborhood of a current pixel (marked by the small cross) are shown in Figure 5.11. The ninth mask is the 3 x 3 neighborhood of the current pixel itself. Other mask shapes—larger or smaller—can also be used.

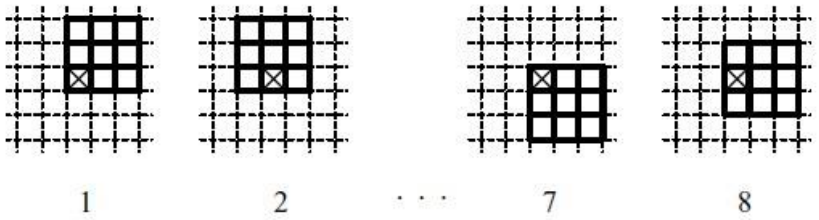


Figure 5.11: Eight possible rotated 3×3 masks.

Algorithm 5.2: Smoothing using a rotating mask

1. Consider each image pixel (i, j).
2. Calculate dispersion for all possible mask rotations about pixel (i, j) according to equation (5.31).
3. Choose the mask with minimum dispersion.
4. Assign to the pixel f (i, j) in the output image f the average brightness in the chosen mask.

Algorithm 5.2 can be used iteratively and the process converges quite quickly to a stable state. The size and shape of masks influence the convergence—the smaller the mask, the smaller are the changes and more iterations are needed. A larger mask suppresses noise faster and the sharpening effect is stronger. On the other hand, information about details smaller than the mask may be lost. The number of iterations is also influenced by the shape of regions in the image and noise properties.

Median filtering

In probability theory, the **median** divides the higher half of a probability distribution from the lower half. For a random variable x, the median M is the value for which the probability of the outcome $x < M$ is 0.5. The median of a finite list of real numbers is simply found by ordering the list and selecting the middle member. Lists are often constructed to be odd in length to secure uniqueness.

Median filtering is a non-linear smoothing method that reduces the blurring of edges, in which the idea is to replace the current point in the image by the median of the brightnesses in its neighborhood. The median in the neighborhood is not affected by individual noise spikes and so median smoothing eliminates impulse noise quite well. Further, as median filtering does not blur edges much, it can be applied iteratively. Clearly, performing a sort on pixels within a (possibly large) rectangular window at every pixel position may become very expensive. A more efficient approach [Huang et al., 1979; Pitas and Venetsanopoulos, 1990] is to notice that as the window moves across a row by one column, the only change to its

– contents is to lose the leftmost column and replace it with a new right column—for a median window of m rows and n columns, mn 2m pixels are unchanged and do not need re-sorting. The algorithm is as follows:

Algorithm 5.3: Efficient median filtering

1. Set
$$t = \frac{mn}{2} .$$
(We would always avoid unnecessary floating point operations: if m and n are both odd, round t.)

2. Position the window at the beginning of a new row, and sort its contents. Construct a histogram H of the window pixels, determine the median m, and record n_m, the number of pixels with intensity less than or equal to m.

3. For each pixel p in the leftmost column of intensity p_g, perform
$$H[p_g] = H[p_g] - 1 .$$
Further, if $p_g \leq m$, set
$$n_m = n_m - 1 .$$

4. Move the window one column right. For each pixel p in the rightmost column of intensity p_g, perform
$$H[p_g] = H[p_g] + 1 .$$
If $p_g \leq m$, set
$$n_m = n_m + 1 .$$

5. If $n_m = t$ then go to (8).

6. If $n_m > t$ then go to (7).
Repeat
$$m = m + 1 ,$$
$$n_m = n_m + H[m] ,$$
until $n_m \geq t$. Go to (8).

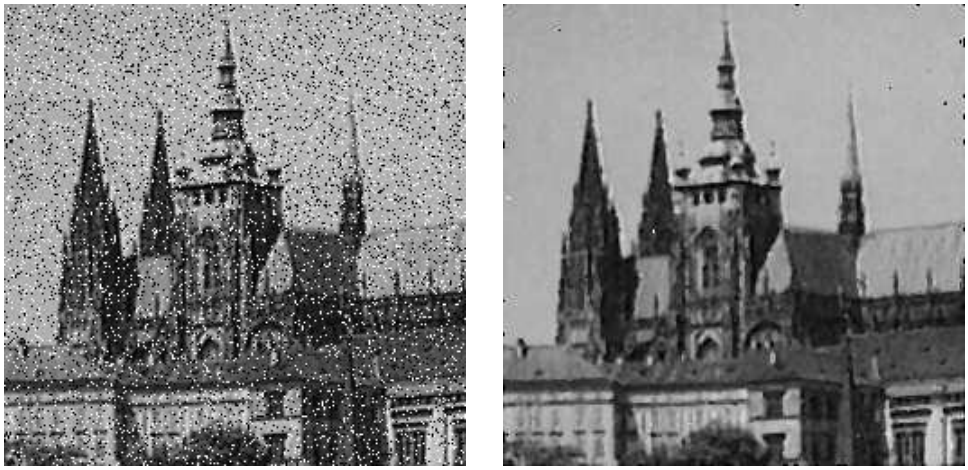
7. (We have $n_m > t$, if here). Repeat
$$n_m = n_m - H[m] , \; m = m - 1 ,$$
until $n_m \leq t$.

8. If the right-hand column of the window is not at the right-hand edge of the image, go to (3).

9. If the bottom row of the window is not at the bottom of the image, go to (2).

Median filtering is illustrated in Figure 5.12. The main disadvantage of median filtering in a rectangular neighborhood is its damaging of thin lines and sharp corners—this can be avoided if another shape of neighborhood is used. For instance, if horizontal/vertical lines need preserving, a neighborhood such as that in Figure 5.13 can be used.

Median smoothing is a special instance of more general **rank filtering** techniques, the idea of which is to order pixels in some neighborhood into a sequence. The results of pre-processing are some statistics over this sequence, of which the median is one possibility. Another variant is the maximum or the minimum values of the sequence. This defines generalizations of dilation and erosion operators in images with more brightness values.



(a) (b)

Figure 5.12: Median filtering. (a) Image corrupted with impulse noise (14% of image area covered with bright and dark dots). (b) Result of 3x 3 median filtering.

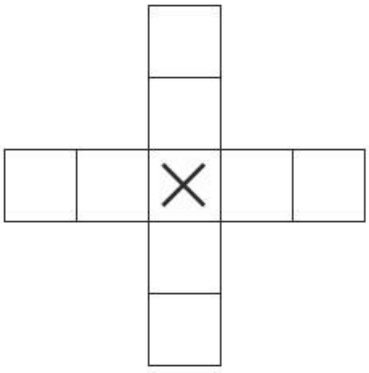


Figure 5.13: Horizontal/vertical line preserving neighbor- hood for median filtering

Non-linear mean filter

The non-linear mean filter is another generalization of averaging techniques; it is defined by

$$f(m,n) = u^{-1}\left(\frac{\sum_{(i,j)\in\mathcal{O}} a(i,j)\, u(g(i,j))}{\sum_{(i,j)\in\mathcal{O}} a(i,j)}\right), \qquad (5.32)$$

where f (m, n) is the result of the filtering, g(i, j) is the pixel in the input image,O and is a local neighborhood of the current pixel (m, n). The function u of one variable has an inverse function u^{-1}; the a(i, j) are weight coefficients.

If the weights a(i, j) are constant, the filter is called **homomorphic**. Some homomorphic filters used in image processing are:

- Arithmetic mean, u(g) = g .
- Harmonic mean, u(g) = 1/g .
- Geometric mean, u(g) = log g .

2.3. Edge detectors:

Edge detectors are a collection of very important local image pre-processing methods used to locate changes in the intensity function; edges are pixels where brightness changes abruptly.

- Edges are those places in an image that correspond to **object boundaries.**
- Edges are pixels where image brightness changes abruptly.

Neurological and psychophysical research suggests that locations in the image in which the function value changes abruptly are important for image perception. Edges are to a certain degree invariant to changes of illumination and viewpoint. If only edge elements with strong magnitude (edges) are considered, such information often suffices for image understanding. The positive effect of such a process is that it leads to significant reduction of image data. Nevertheless such data reduction does not undermine understanding the content of the image (interpretation) in many cases. Edge detection provides appropriate generalization of the image data; for instance, line drawings perform such a generalization.

We shall consider which physical phenomena in the image formation process lead to abrupt changes in image values—see Figure 5.15. Calculus describes changes of continuous functions using derivatives; an image function depends on two variables—co-ordinates in the image plane—and so operators describing edges are expressed using partial derivatives. A change of the image function can be described by a gradient that points in the direction of the largest growth of the image function.

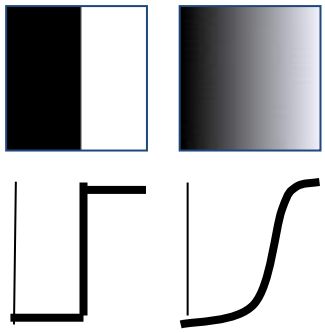


Figure 5.15: Origin of edges, i.e., physical phenomena in image formation process which lead to edges in images. At right, a Canny edge detection.

An edge is a property attached to an individual pixel and is calculated from the image function behavior in a neighborhood of that pixel. It is a **vector variable** with two components, **magnitude** and **direction**. The edge magnitude is the magnitude of the gradient, and the edge direction φ is rotated with respect to –

the gradient direction ψ by 90◦. The gradient direction gives the direction of maximum growth of the function, e.g., from black f (i, j) = 0 to white f (i, j) = 255. This is illustrated in Figure 5.16, in which closed lines are lines of equal brightness.

The orientation 0◦ points east.

Edges are often used in image analysis for finding region boundaries. Provided that the region has homogeneous brightness, its boundary is at the pixels where the image function varies and so in the ideal case without noise consists of pixels with high edge

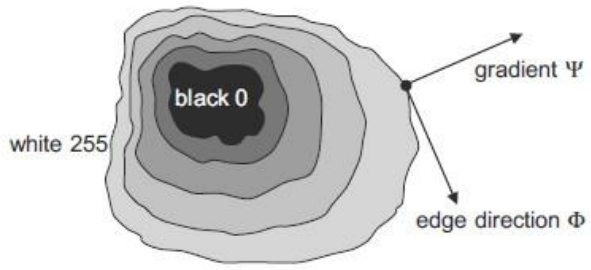


Figure 5.16: Gradient direction and edge direction.

magnitude. It can be seen that the boundary and its parts (edges) are perpendicular to the direction of the gradient.

Figure 5.17 shows examples of several standard edge profiles. Edge detectors are usually tuned for some type of edge profile.

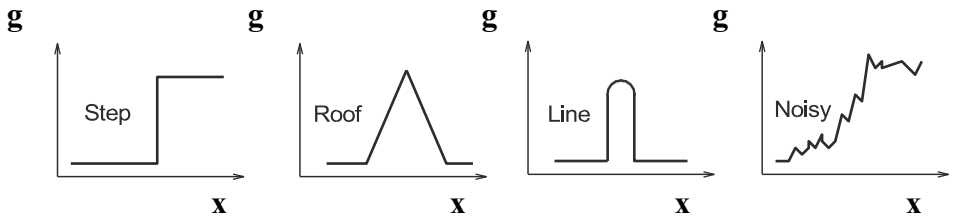


Figure 5.17: Typical edge profiles.

The gradient magnitude | grad g(x, y)| and gradient direction ψ are continuous image functions calculated as

$$|\operatorname{grad} g(x,y)| = \sqrt{\left(\frac{\partial g}{\partial x}\right)^2 + \left(\frac{\partial g}{\partial y}\right)^2},$$

$$\psi = \arg\left(\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}\right),$$

where arg(x, y) is the angle (in radians) from the x axis to (x, y). Sometimes we are interested only in edge magnitudes without regard to their orientations— a linear differential operator called the **Laplacian** may then be used. The Laplacian has the same properties in all directions and is therefore invariant to rotation. It is defined as

$$\nabla^2 g(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}.$$

Image **sharpening** [Rosenfeld and Kak, 1982] has the objective of making edges steeper— the sharpened image is intended to be observed by a human. The sharpened output image f is obtained from the input image g as

$$f(i, j) = g(i, j) - C\, S(i, j), \qquad (5.36)$$

where C is a positive coefficient which gives the strength of sharpening and S(i, j) is a measure of the image function sheerness, calculated using a gradient operator. The Laplacian is very often used for this purpose. Figure 5.18 gives an example of image sharpening using a Laplacian.

Image sharpening can be interpreted in the frequency domain as well. We know that the result of the Fourier transform is a combination of harmonic functions.

The derivative of the harmonic function sin(nx) is n cos(nx); thus the higher the frequency, the higher the magnitude of its derivative.

A similar image sharpening technique to that of equation (5.36), called **unsharp masking**, is often used in printing industry applications. A signal proportional to an unsharp (e.g., heavily blurred by a smoothing operator) image is subtracted from the original image. A digital image is discrete in nature and so equations (5.33) and (5.34), containing derivatives, must be approximated by **differences**. The first differences of the image g in the vertical direction (for fixed i) and in the horizontal direction (for fixed j) are given by

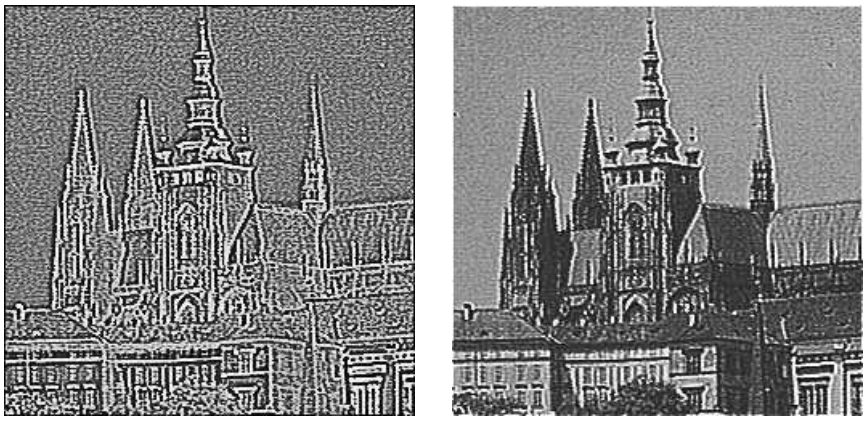
$$\Delta_i\, g(i, j) = g(i, j) - g(i - n, j) ,$$
$$\Delta_j\, g(i, j) = g(i, j) - g(i, j - n) ,$$

(5.37)

where n is a small integer, usually 1. The value n should be chosen small enough to provide a good approximation to the derivative, but large enough to neglect unimportant changes in the image function. Symmetric expressions for the differences,

$$\Delta_i\, g(i, j) = g(i + n, j) - g(i - n, j) ,$$
$$\Delta_j\, g(i, j) = g(i, j + n) - g(i, j - n) ,$$

are not usually used because they neglect the impact of the pixel (i, j) itself.



(a) (b)

Figure 5.18: Laplace gradient operator. (a) Laplace edge image using the 8-connectivity mask.
(b) Sharpening using the Laplace operator equation 5.36, C = 0.7 . Compare the sharpening effect with the original image in Figure 5.9a.

Gradient operators as a measure of edge sheerness can be divided into three categories:

1. Operators approximating derivatives of the image function using differences. Some are rotationally invariant (e.g., Laplacian) and thus are computed from one convolution mask only. Others, which approximate first derivatives, use several masks. The orientation is estimated on the basis of the best matching of several simple patterns.
2. Operators based on zero-crossings of the image function second derivative (e.g., Marr-Hildreth or Canny edge detectors).
3. Operators which attempt to match an image function to a parametric model of edges.

Edge detection is an extremely important step facilitating higher-level image analysis and remains an area of active research. Examples of the variety of

approaches found in current literature are fuzzy logic, neural networks, or wavelets. It may be difficult to select the most appropriate edge detection strategy.

Individual gradient operators that examine small local neighborhoods are in fact convolutions (cf. equation 5.23), and can be expressed by convolution masks. Operators which are able to detect edge direction are represented by a collection of masks, each corresponding to a certain direction.

Roberts operator

The Roberts operator is one of the oldest [Roberts, 1965], and is very easy to compute as it uses only a 2 × 2 neighborhood of the current pixel. Its masks are

$$h_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \qquad h_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

so the magnitude of the edge is computed as g(i,¯ j) − g(i + 1, j + 1)¯ +¯ g(i, j + 1) − g(i + 1¯ , j) . (5.40)

The primary disadvantage of the Roberts operator is its high sensitivity to noise, because very few pixels are used to approximate the gradient.

Laplace operator

The Laplace operator∇ 2 is a very popular operator approximating the second derivative which gives the edge magnitude only. The Laplacian, equation (5.35), is

× approximated in digital images by a convolution sum. A 3x3 mask h is often used; for 4-neighborhoods and 8-neighborhoods it is defined as

$$h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad h = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

A Laplacian operator with stressed significance of the central pixel or its neighborhood is sometimes used. In this approximation it loses invariance to rotation

$$h = \begin{bmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{bmatrix}, \qquad h = \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

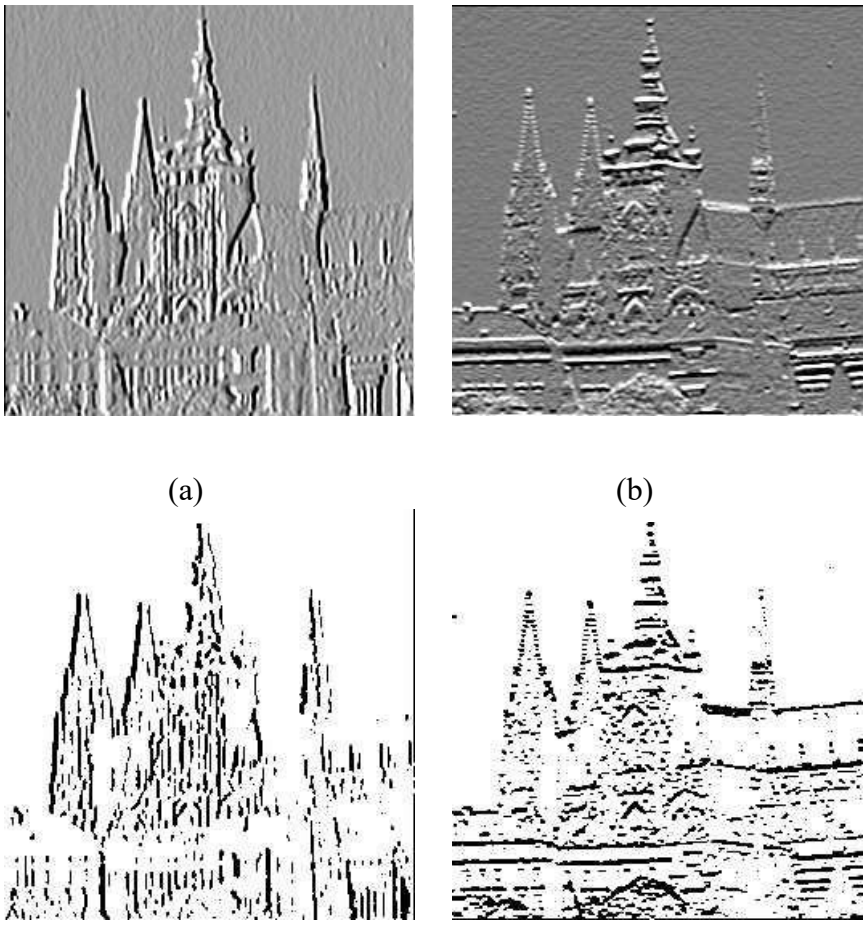
The Laplacian operator has a disadvantage—it responds doubly to some edges in the image.

Prewitt operator

The Prewitt operator, similarly to the Sobel, Kirsch, and some other operators, approximates the first derivative. The gradient is estimated in eight (for a 3x 3 convolution mask) possible directions, and the convolution result of greatest magnitude indicates the gradient direction. Larger masks are possible. We present only the first three 3 x 3 masks for each operator; the others can be created by simple rotation.

$$h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \quad h_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, \quad h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

The direction of the gradient is given by the mask giving maximal response. This is also the case for all the following operators approximating the first derivative.



(a) (b)

(c) (d)

Figure 5.19: First-derivative edge detection using Prewitt operators. (a) North direction (the brighter the pixel value, the stronger the edge). (b) East direction. (c) Strong edges from (a).
(d) Strong edges from (b).

Sobel operator

The Sobel operator is often used as a simple detector of horizontality and verticality of edges, in which case only masks h_1 and h_3 are used. If the h_1 response is y and the h_3 response x, we might then derive edge strength (magnitude) as

$$\sqrt{x^2 + y^2} \qquad \text{or} \qquad |x| + |y|$$

And direction as arctan(y/x).

Kirsch operator

$$h_1 = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix}, \quad h_2 = \begin{bmatrix} 3 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & -5 & 3 \end{bmatrix}, \quad h_3 = \begin{bmatrix} -5 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & 3 & 3 \end{bmatrix}$$

To illustrate the application of gradient operators on real images, consider again the image given in Figure 5.9a. The Laplace edge image calculated is shown in Figure 5.18a; the value of the operator has been histogram equalized to enhance its visibility.

The properties of an operator approximating the first derivative are demonstrated using the Prewitt operator—results of others are similar. The original image is again given in Figure 5.9a; Prewitt approximations to the directional gradients are in Figures 5.19a,b, in which north and east directions are shown. Significant edges (those with above-threshold magnitude) in the two directions are given in Figures 5.19c,d.

2.4. Zero-crossings of the second derivative:

In the 1970s, Marr's theory concluded from neurophysiological experiments that

object boundaries are the most important cues that link an intensity image with its interpretation. Edge detection techniques existing at that time (e.g., the Kirsch, Sobel, and Pratt operators) were based on convolution in very small neighborhoods and worked well only for specific images. The main disadvantage of these edge detectors is their dependence on the size of the object and sensitivity to noise.

An edge detection technique based on the **zero-crossings** of the second derivative (**Marr-Hildreth** edge detector) explores the fact that a step edge corresponds to an abrupt change in the image function. The first derivative of the image function should have an extremum at the position corresponding to the edge in the image, and so the second derivative should be zero at the same position; however, it is much easier and more precise to find a zero-crossing position than an extremum. In Figure 5.20 this principle is illustrated in 1D for the sake of simplicity. Figure 5.20a shows step edge profiles of the original image function with two different slopes, Figure 5.20b depicts the first derivative of the image function, and Figure 5.20c illustrates the second derivative; notice that this crosses the zero level at the same position as the edge.

Considering a step-like edge in 2D, the 1D profile of Figure 5.20a corresponds to a cross section through the 2D step. The steepness of the profile will change if the

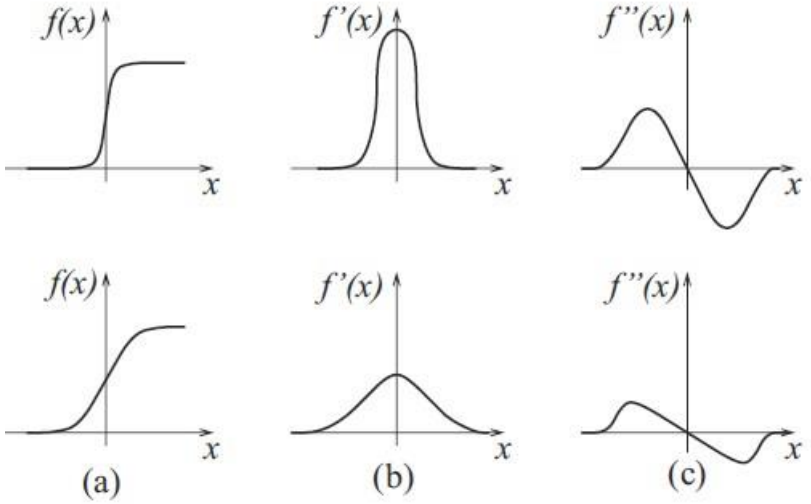


Figure 5.20: 1D edge profile of the zero-crossing.

orientation of the cutting plane changes—the maximum steepness is observed when the plane is perpendicular to the edge direction.

The crucial question is how to compute the second derivative robustly. One possibility is to smooth an image first (to reduce noise) and then compute second derivatives. When choosing a smoothing filter, there are two criteria that should be fulfilled. First, the filter should be smooth and roughly band limited in the frequency domain to reduce the possible number of frequencies at which function changes can take place. Second, the constraint of spatial localization requires the response of a filter to be from nearby points in the image. These two criteria are conflicting, but they can be optimized simultaneously using a Gaussian distribution. In practice, one has to be more precise about what is meant by the localization performance of an operator, and the Gaussian may turn out to be suboptimal. We shall consider this in the next section.

The 2D Gaussian smoothing operator G(x, y) (also called a Gaussian filter, or simply a Gaussian) is given by

$$G(x,y) = e^{-(x^2+y^2)/2\sigma^2} ,$$

where x, y are the image co-ordinates and σ is a standard deviation of the associated probability distribution. Sometimes this is presented with a normalizing factor

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \qquad \text{or} \qquad G(x,y) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-(x^2+y^2)/2\sigma^2}$$

The standard deviation σ is the only parameter of the Gaussian filter—it is proportional to the size of the neighborhood on which the filter operates. Pixels more

distant from the center of the operator have smaller influence, and pixels farther than 3σ from the center have negligible influence.

Our goal is to obtain a second derivative of a smoothed 2D function f (x, y). We have already seen that the Laplace operator gives the second derivative, and is non- directional (isotropic). Consider then the Laplacian of an image f (x, y) smoothed by a Gaussian (expressed using a convolution). The operation is often abbreviated as **LoG**, from **Laplacian of Gaussian**

$$\nabla^2 \left[G(x, y, \sigma) * f(x, y) \right] .$$

The order of performing differentiation and convolution can be interchanged because of the linearity of the operators involved

$$\left[\nabla^2 G(x, y, \sigma) \right] * f(x, y) .$$

The derivative of the Gaussian filter $\nabla^2 G,$ can be pre-computed analytically, since it is independent of the image under consideration, and so the complexity of the composite operation is reduced. From equation (5.47), we see

$$\frac{\partial G}{\partial x} = -\left(\frac{x}{\sigma^2} \right) e^{-(x^2+y^2)/2\sigma^2}$$

and similarly for y. Hence

$$\frac{\partial^2 G}{\partial x^2} = \frac{1}{\sigma^2} \left(\frac{x^2}{\sigma^2} - 1 \right) e^{-(x^2+y^2)/2\sigma^2} , \quad \frac{\partial^2 G}{\partial y^2} = \frac{1}{\sigma^2} \left(\frac{y^2}{\sigma^2} - 1 \right) e^{-(x^2+y^2)/2\sigma^2}$$

After introducing a normalizing multiplicative coefficient c, we get a convolution mask of a LoG operator:

$$h(x, y) = c \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right) e^{-(x^2+y^2)/2\sigma^2} ,$$

where c normalizes the sum of mask elements to zero. Because of its shape, the inverted LoG operator is commonly called a **Mexican hat**. An example of a 5 × 5 discrete approximation (wherein a 17 × 17 mask is also given) is

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}.$$

Of course, these masks represent truncated and discrete representations of infinite continuous functions, and care should be taken in avoiding errors in moving to this representation.

Finding second derivatives in this way is very robust. Gaussian smoothing effectively suppresses the influence of the pixels that are more than a distance 3σ from the current pixel; then the Laplace operator is an efficient and stable measure of changes in the image.

After image convolution $\nabla^2 G$, with the locations in the convolved image where the zero level is crossed correspond to the positions of edges. The advantage of this approach compared to classical edge operators of small size is that a larger area surrounding the current pixel is taken into account; the influence of more distant points decreases according to the σ of the Gaussian. In the ideal case of an isolated step edge, the σ variation does not affect the location of the zero-crossing. Convolution masks become large for larger σ; for

example, σ = 4 needs a mask about 40 pixels wide. Fortunately, there is a separable decomposition of the $\nabla^2 G$, operator that can speed up computation considerably.

The practical implication of Gaussian smoothing is that edges are found reliably. If only globally significant edges are required, the standard deviation σ of the Gaussian smoothing filter may be increased, having the effect of suppressing less significant evidence.

The $\nabla^2 G$, operator can be very effectively approximated by convolution with a mask that is the difference of two Gaussian averaging masks with substantially different σ—this method is called the **difference of Gaussians**, abbreviated as **DoG**.

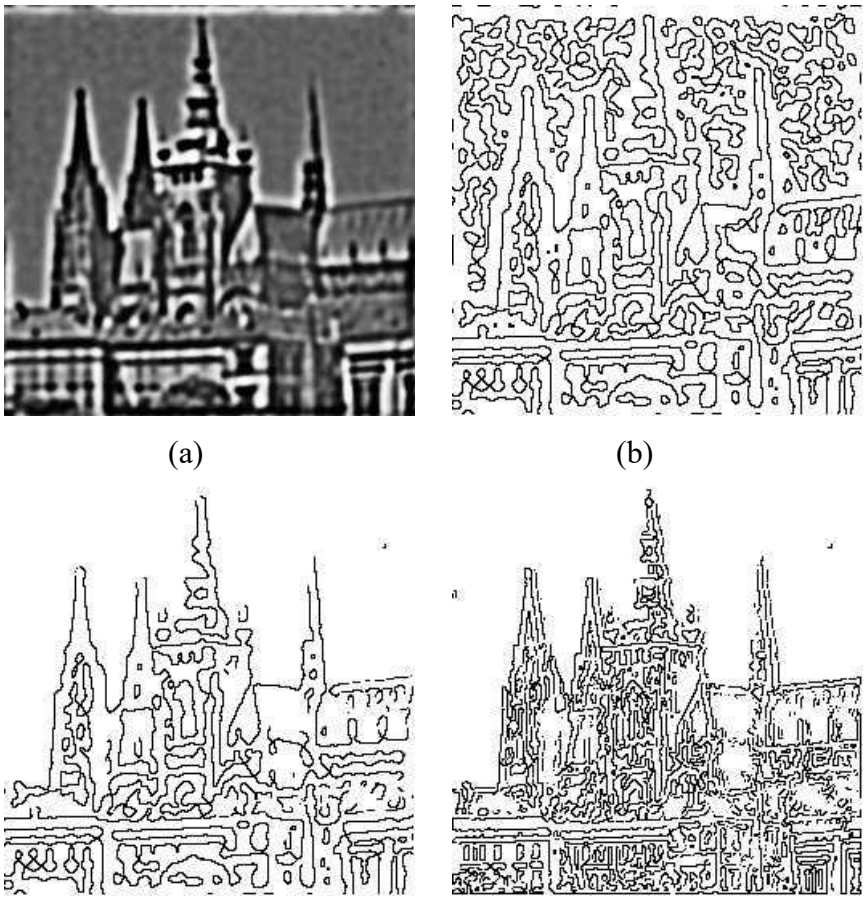
When implementing a zero-crossing edge detector, trying to detect zeros in the LoG or DoG image will inevitably fail, while naive approaches of thresholding the

×

LoG/DoG image and defining the zero-crossings in some interval of values close to zero give piece- wise disconnected edges at best. To create a well-functioning second-derivative edge detector, it is necessary to implement a true zero-crossing detector. A simple detector may identify a zero-crossing in a moving 2 × 2 window, assigning an edge label to any one corner pixel, say the upper left, if LoG/DoG image values of both polarities occur in the 2 x 2 window;

no edge label would be given if values within the window are either all positive or all negative. Another post-processing step to avoid detection of zero- crossings corresponding to non-significant edges in regions of almost constant gray-level would admit only those zero-crossings for which there is sufficient edge evidence from a first-derivative edge detector. Figure 5.21 provides several examples of edge detection using zero crossings of the second derivative.

Many other approaches improving zero-crossing performance can be found in the literature; some of them are used in pre-processing or post-processing steps. The traditional second-derivative zero-crossing technique has disadvantages as



(a) (b)

(c) (d)

Figure 5.21: Zero-crossings of the second derivative, see Figure 5.9a for the original image.

(a) DoG image ($\sigma 1 = 0.10$, $\sigma 2 = 0.09$), dark pixels correspond to negative values, bright pixels to positive. (b) Zero-crossings of the DoG image. (c) DoG zerocrossing edges after removing edges lacking first-derivative support. (d) LoG zero-crossing edges ($\sigma = 0.20$) after removing edges lacking first-derivative support—note different scale of edges due to different Gaussian smoothing parameters.

well. First, it smooths the shape too much; for example, sharp corners are lost. Second, it tends to create closed loops of edges (nicknamed the 'plate of spaghetti' effect).

Neurophysiological experiments provide evidence that the human eye retina in the form of the **ganglion cells** performs operations very similar to the $\nabla^2 G$, operations. Each such cell responds to light stimuli in a local neighborhood called the **receptive field**, which has a center-surround organization of two complementary types, off-center and on-center. When a light stimulus occurs, activity of on-center cells increases and that of off-center cells is inhibited. The retinal operation on the image can be described analytically as the convolution of the image with the ∇2G operator.

2.5. Scale in image processing:

- Many image processing techniques work locally, theoretically at the level of individual pixels—edge detection methods are an example. The essential problem in such computation is **scale**.
- Edges correspond to the gradient of the image function, which is computed as a difference between pixels in some neighborhood.
- There is seldom a sound reason for choosing a particular size of neighborhood, since the 'right' size depends on the size of the objects under investigation.
- To know what the objects are assumes that it is clear how to interpret an image, and this is not in general known at the pre-processing stage.
- The solution to the problem formulated above is a special case of a general paradigm called the **system approach**. This methodology is common in cybernetics or general system theory to study complex phenomena.
- The phenomenon under investigation is expressed at different resolutions of the de- scription, and a formal model is created at each resolution. Then the qualitative behavior of the model is studied under changing resolution of the description. Such a methodology enables the deduction of meta-knowledge about the phenomenon that is not seen at the individual description levels.
- Different description levels are easily interpreted as different scales in the domain of digital images. The idea of scale is fundamental to Marr's edge detection technique, where different scales are provided by different sizes of Gaussian filter masks. The aim was not only to eliminate fine scale noise but also to separate events at different scales arising from distinct physical processes.
- Assume that a signal has been smoothed with several masks of variable sizes. Every setting of the scale parameters implies a different description, but it is not known which one is correct; for many tasks, no one scale is categorically correct. If the ambiguity introduced by the scale is inescapable, the goal of scale-independent description is to reduce this ambiguity as much as possible.
- Here we shall consider just three examples of the application of multiple scale description to image analysis.

1. The first approach aims to process planar noisy curves at a range of scales— the segment of curve that represents the underlying structure of

the scene needs to be found. The problem is illustrated by an example of two noisy curves.

- One of these may be interpreted as a closed curve, while the other could be described as two intersecting straight lines.
- Local tangent direction and curvature of the curve are significant only with some idea of scale after **the curve is smoothed by a Gaussian filter with varying standard deviations.**

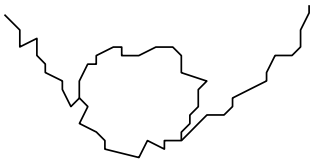


Figure 5.22: Curves that may be analyzed at multiple scales.

2. A second approach, called **scale-space filtering**, tries to describe signals qualitatively with respect to scale. The problem was formulated for 1D signals f (x), but it can easily be generalized for 2D functions as images.

   The original 1D signal f (x) is smoothed by convolution with a 1D Gaussian

$$G(x,\sigma) = e^{-x^2/2\sigma^2} \,. \qquad (5.51)$$

If the standard deviation σ is slowly changed, the function

$$F(x,\sigma) = f(x) * G(x,\sigma) \qquad (5.52)$$

represents a surface on the (x, σ) plane that is called the **scale-space image**.

Inflection points of the curve F (x, σ_0) for a distinct value σ_0

$$\frac{\partial^2 F(x,\sigma_0)}{\partial x^2} = 0 \qquad \text{and} \qquad \frac{\partial^3 F(x,\sigma_0)}{\partial x^3} \neq 0 \qquad (5.53)$$

describe the curve f (x) qualitatively. The positions of inflexion points can be drawn as a set of curves in (x, σ) co-ordinates. Coarse to fine analysis of the curves corresponding to inflexion points, i.e., in the direction of decreasing value of the σ, localizes large-scale events.

The qualitative information contained in the scale-space image can be transformed into a simple **interval tree** that expresses the structure of the signal f (x) over all observed scales. The interval tree is built from the root that corresponds to the largest scale (σ_{max}), and then the scale-space image is searched in the direction of decreasing σ. The interval tree branches at those points where new curves corresponding to inflexion points appear

3. The third example of the application of scale is that used by the popular **Canny edge detector**. Since the Canny detector is a significant and widely used contribution to edge detection techniques, its principles will be explained in detail.

2.6. Canny edge detection:

Canny proposed an approach to edge detection that is optimal for step edges corrupted by white noise. The optimality of the detector is related to three criteria.

1) The **detection** criterion expresses the fact that important edges should not be missed and that there should be no spurious responses.
2) The **localization** criterion says that the distance between the actual and located position of the edge should be minimal.
3) The **one response** criterion minimizes multiple responses to a single edge. This is partly covered by the first criterion, since when there are two responses to a single edge, one of them should be considered as false. This third criterion solves the problem of an edge corrupted by noise and works against non-smooth edge operators.

Canny's derivation is based on several ideas.

1. The edge detector was expressed for a 1D signal and the first two optimality criteria. A closed-form solution was found using the calculus of variations.
2. If the third criterion (multiple responses) is added, the best solution may be found by numerical optimization. The resulting filter can be approximated effectively with error less than 20% by the first derivative of a Gaussian smoothing filter with standard deviation σ [Canny, 1986]; the reason for doing this is the existence of an effective implementation. There is a strong similarity here to the LoG based Marr-Hildreth edge detector.
3. The detector is then generalized to two dimensions. A step edge is given by its position, orientation, and possibly magnitude (strength). It can be shown that convolving an image with a symmetric 2D Gaussian and then differentiating in the direction of the gradient (perpendicular to the edge direction) forms a simple and effective directional operator (recall that the Marr-Hildreth zerocrossing operator does not give information about edge direction, as it uses a Laplacian filter).

   Suppose G is a 2D Gaussian [equation (5.47)] and assume we wish to convolve the image with an operator G_n which is a first derivative of G in some direction **n**

$$G_n = \frac{\partial G}{\partial \mathbf{n}} = \mathbf{n}\nabla G\,. \qquad (5.54)$$

   We would like **n** to be perpendicular to the edge: this direction is not known in advance, but a robust estimate of it based on the smoothed gradient direction is available. If f is the image, the normal to the edge **n** is estimated as

$$\mathbf{n} = \frac{\nabla(G * f)}{\left|\nabla(G * f)\right|}\,. \qquad (5.55)$$

The edge location is then at the local maximum of the image f convolved with the operator G_n in the direction **n**

$$\frac{\partial}{\partial \mathbf{n}} G_n * f = 0 \,. \qquad (5.56)$$

Substituting in equation (5.56) for G_n from equation (5.54), we get

$$\frac{\partial^2}{\partial \mathbf{n}^2} G * f = 0 \,. \qquad (5.57)$$

This equation (5.57) illustrates how to find local maxima in the direction perpendicular to the edge; this operation is often referred to as **non-maximal suppression** (see also Algorithm 6.4).

As the convolution and derivative are associative operations in equation (5.57), we can first convolve an image f with a symmetric Gaussian G and then compute the directional second-derivative using an estimate of the direction **n** computed according to equation (5.55). The strength of the edge (magnitude of the gradient of the image intensity function f) is measured as

$$\left|G_n * f\right| = \left|\nabla (G * f)\right| \,. \qquad (5.58)$$

4. Spurious responses to a single edge caused by noise usually create a 'streaking' problem that is very common in edge detection in general. The output of an edge detector is usually thresholded to decide which edges are significant, and streaking may break up edge contours as the operator fluctuates above and below the threshold. Streaking can be eliminated by **thresholding with**

Algorithm 5.4: Canny edge detector

1. Convolve an image f with a Gaussian of scale σ.
2. Estimate local edge normal directions $\mathbf{n}$ using equation (5.55) for each pixel in the image.

$$\mathbf{n} = \frac{\nabla(G * f)}{|\nabla(G * f)|} . \qquad (5.55)$$

3. Find the location of the edges using equation (5.57) (non-maximal suppression).

$$\frac{\partial^2}{\partial \mathbf{n}^2} G * f = 0 . \qquad (5.57)$$

4. Compute the magnitude of the edge using equation (5.58).

$$|G_n * f| = |\nabla(G * f)| . \qquad (5.58)$$

5. Threshold edges in the image with hysteresis to eliminate spurious responses.
6. Repeat steps (1) through (5) for ascending values of the standard deviation σ.
7. Aggregate the final information about edges at multiple scale using the 'feature synthesis' approach.

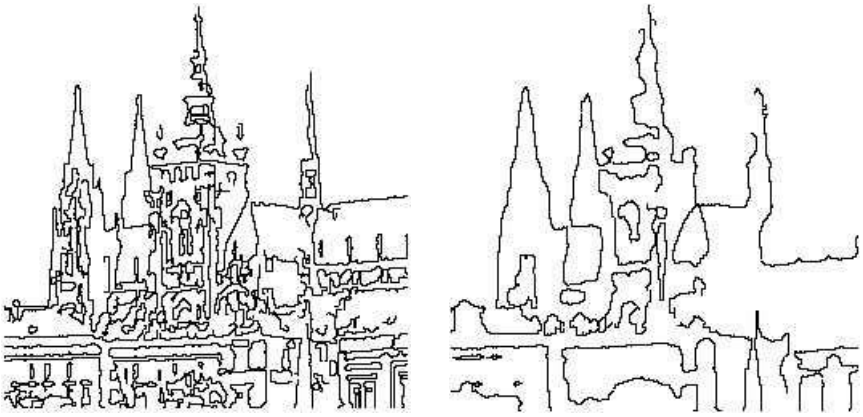
hysteresis, employing a hard (high) threshold and a soft (lower) threshold— see Algorithm 6.5. The low and high thresholds are set according to an estimated signal-to-noise ratio.

5. The correct scale for the operator depends on the objects contained in the image. The solution to this unknown is to use multiple scales and aggregate information from them. Different scales for the Canny detector are represented by different standard deviations σ of the Gaussians. There may be several scales of operators that give significant responses to edges (i.e., signal-to-noise ratio above the thresh- old); in this case the operator with the smallest scale is chosen, as it gives the best localization of the edge.

   Canny proposed a **feature synthesis** approach. All significant edges from the operator with the smallest scale are marked first, and the edges of a hypothetical operator with larger σ are synthesized from them (i.e., a prediction is made of how the large σ should perform on the evidence gleaned from the smaller σ. Then the synthesized edge response is compared with the actual edge response for larger σ. Additional edges are marked only if they have a significantly stronger response than that predicted from synthetic output.

   This procedure may be repeated for a sequence of scales, a cumulative edge map being built by adding those edges that were not identified at smaller scales.

Figure 5.23a shows the edges of Figure 5.9a detected by a Canny operator with σ = 1.0. Figure 5.23b shows the edge detector response for σ = 2.8 (feature synthesis has not been applied here).



(a) (b)

Figure 5.23: Canny edge detection at two different scales. © Cengage Learning 2015.

Canny's detector represents a complicated but major contribution to edge detection. Its full implementation is unusual, it being common to find implementations that omit feature synthesis—that is, just steps 1–5 of Algorithm 5.4.

2.7. Parametric edge models:

Parametric models are based on the idea that the discrete image intensity function can be considered a sampled and noisy approximation of an underlying continuous or piecewise continuous image intensity function.

While this function is not known, it can be estimated from the available discrete image intensity function and image properties can be determined from this continuous estimate, possibly with subpixel precision.

It is usually impossible to represent image intensities using a single continuous function since a single function leads to high-order intensity functions in x and y. Instead, piecewise continuous function estimates called **facets** are used to represent (a neighborhood of) each image pixel. Such an image representation is called a **facet model.**

The intensity function in a neighborhood can be estimated using models of different complexity.

The simplest one is the flat facet model that uses piecewise constants and each pixel neighborhood is represented by a flat function of constant intensity. The sloped model uses piecewise linear functions forming a sloped plane fitted to local image intensities.

Quadratic and bi-cubic facet models employ more complex functions.

Once the facet model parameters are available for each image pixel, edges can be detected as extrema of the first directional derivative and/or zero-crossings of the second directional derivative of the local continuous facet model functions.

An example will illustrate: consider a bi-cubic facet model $g(i, j) = c_1 + c_2 x + c_3 y + c_4 x^2 + c_5 x y + c_6 y^2 + c_7 x^3 + c_8 x^2 y + c_9 x y^2 + c_{10} y^3$, (5.59) whose parameters are estimated from a pixel neighborhood (the co-ordinates of the central pixel are (0,0)). This may be performed by, e.g., a least-squares method with SVD; alternatively, coefficients c_i can be computed directly using a set of ten 5x5 kernels. Once parameters are available at each pixel, edges may be located as extrema of the first directional derivative, or zero crossings of the second derivative, of the local facet model functions.

Benefits:

1) Edge detectors based on parametric models describe edges more precisely than convolution-based edge detectors.
2) They carry the potential for subpixel edge localization.

Limitations:

- Their computational requirements are much higher.
- Promising extensions combine facet models with Canny's edge detection criteria and relaxation labeling.

2.8. Edges in multi-spectral images:

One pixel in a multi-spectral image is described by an n-dimensional vector, and brightness values in n spectral bands are the vector components. There are several possibilities for the detection of edges in multi-spectral images.

Trivially, we might detect edges separately in individual image spectral components using the ordinary local gradient operators. Individual images of edges can be combined to get the resulting image, with the value corresponding to edge magnitude and direction being a selection or combination of the individual edge spectral components.

Alternatively, we may create a multi-spectral edge detector which uses brightness information from all n spectral bands; this approach is also applicable to multi-dimensional images forming three- or higher-dimensional data volumes. The neighborhood used has size 2x n pixels, where the 2x 2 neighborhood is similar to that of the Roberts gradient, equation (5.39). The coefficients weighting the influence of the component pixels are similar to the correlation coefficients. Let f(i, j) denote the arithmetic mean of the brightnesses corresponding to the pixels with the same co-ordinates (i, j) in all n spectral component images, and f_r be the brightness of the r^{th} spectral component. The edge detector result in pixel (i, j) is given as the minimum of the following expression:

$$\frac{\sum_{r=1}^{n}\left[d(i,j)\right]\left[d(i+1,j+1)\right]}{\sqrt{\sum_{r=1}^{n}\left[d(i,j)\right]^2\sum_{r=1}^{n}\left[d(i+1,j+1)\right]^2}}\;\frac{\sum_{r=1}^{n}\left[d(i+1,j)\right]\left[d(i,j+1)\right]}{\sqrt{\sum_{r=1}^{n}\left[d(i+1,j)\right]^2\sum_{r=1}^{n}\left[d(i,j+1)\right]^2}},$$

where $d(k,l)=f_r(k,l)-\overline{f}(k,l)$. (5.60)

2.9. Local pre-processing in the frequency domain:

The Fourier transform makes convolution of two images in the frequency domain very easy, and it is natural to consider applying many of the filters in the frequency domain. Such operations are usually called **spatial frequency filtering**.

Assume that f is an input image and F is its Fourier transform. A convolution filter h can be represented by its Fourier transform H; h may be called the unit pulse response of the filter and H the frequency transfer function, and either of the representations h or H can be used to describe the filter. The Fourier transform of the filter output after an image f has been convolved with the filter h can be computed in the frequency domain

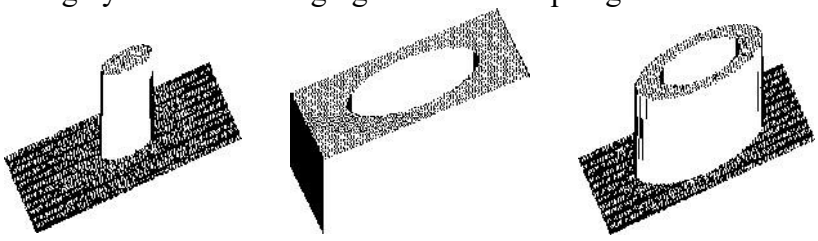
$$G = F.*, \tag{5.61}$$

where . represents an element$*$ -by-element multiplication of matrices F and H (not matrix multiplication). The filtered image g can be obtained by applying the inverse

Fourier transform to **G**—equation (3.28).

Some basic examples of spatial filtering are linear **low-pass**, **high-pass**, and **band- pass** frequency filters.

1) A low-pass filter is defined by a frequency transfer function H(u, v) with small values at points located far from the co-ordinate origin in the frequency domain (that is, small transfer values for high spatial frequencies) and large values at points close to the origin (large transfer values for low spatial frequencies)—see Figure 5.24a. It preserves low spatial frequencies and suppresses high spatial frequencies, and has behavior similar to smoothing by standard averaging—it blurs sharp edges.



(a) **(b)** **(c)**

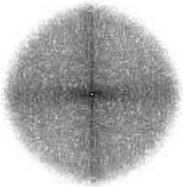
Figure 5.24: Frequency filters displayed in 3D. (a) Low-pass filter. (b) High-pass filter. (c) Band- pass filter.

2) A high-pass filter is defined by small transfer function values located around the frequency co-ordinate system origin, and larger values outside this area— larger transfer coefficients for higher frequencies (Figure 5.24b).

- Band-pass filters, which select frequencies in a certain range for enhancement, are constructed in a similar way, and also filters with directional response, etc. (Fig- ure 5.24c).

The most common image enhancement problems include noise suppression, edge enhancement, and removal of noise which is structured in the frequency spectrum. Noise represents a high-frequency image component, and it may be suppressed applying a low-pass filter as shown in Figure 5.25, which demonstrates the principles of frequency filtering on Fourier image spectra; the original image spectrum is multiplied by the filter spectrum and a low-frequency image spectrum results. Unfortunately, all high-frequency phenomena are suppressed, including high frequencies that are not related to noise (sharp edges, lines, etc.). Low-pass filtering results in a blurred image.

(a) (b)



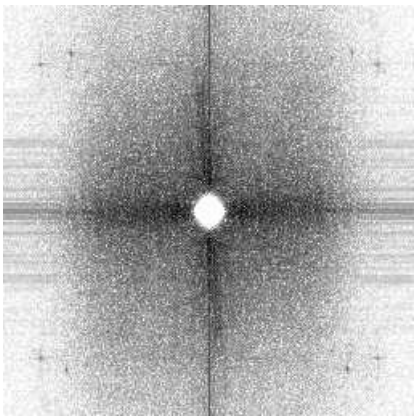
(c) (d)

Figure 5.25: Low-pass frequency-domain filtering—for the original image and its spectrum see Figure 3.7. (a) Spectrum of a low-pass filtered image, all higher frequencies filtered out. (b) Image resulting from the inverse Fourier transform applied to spectrum (a). (c) Spectrum of a low-pass filtered image, only very high frequencies filtered out. (d) Inverse Fourier transform applied to spectrum (c).

Again, edges represent a high-frequency image phenomenon. Therefore, to enhance them, low-frequency components of the image spectrum must be suppressed—to achieve this, a high-frequency filter must be applied.

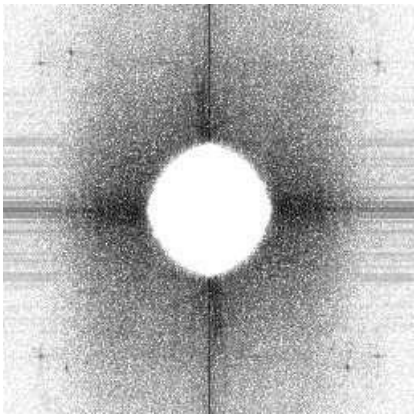
To remove noise which is structured in the frequency domain, the filter design must include a priori knowledge about the noise properties. This knowledge may be acquired either from the image data or from the corrupted image Fourier spectrum, where the structured noise usually causes notable peaks.

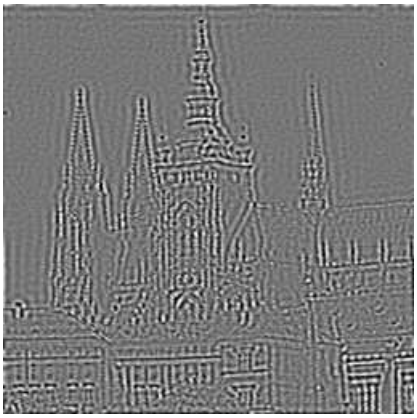
Some examples of frequency domain image filtering are shown in Figures 5.25–5.28. The original image was shown in Figure 3.8 and its frequency spectrum in Figure 3.7. Figure 5.26 shows results after application of a high-pass filter followed by an inverse Fourier transform. It can be seen that edges represent high-frequency phenomena in the image. Results of band-pass filtering can be seen in Figure 5.27. Figure 5.28 gives an even more powerful example of frequency filtering—removal of periodic noise. The vertical noise lines in the original image are transformed into frequency spectrum peaks after the transform. To remove these frequencies, a filter was designed which suppresses the periodic noise in the image, which is visible as white circular areas.



(a)

(b)





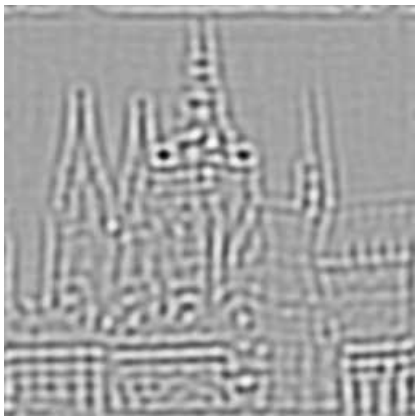
(c) (d)

Figure 5.26: High-pass frequency domain filtering. (a) Spectrum of a high-pass filtered image, only very low frequencies filtered out. (b) Image resulting from the inverse Fourier transform applied to spectrum (a). (c) Spectrum of a high-pass filtered image, all lower frequencies filtered out. (d) Inverse Fourier transform applied to spectrum (c).

There are several filters which prove useful for filtering in the frequency domain: two important representatives of them are the Gaussian and Butterworth filters. Choose an isotropic filter for simplicity,

$D(u,v) = D(r) = \sqrt{u^2+v^2}$, and let D_0 be a parameter of the filter called the cut-off frequency. For the Gaussian, D_0 coincides with the dispersion σ. The Fourier spectrum of a low-pass Gaussian filter G_{low} is

$$G_{\text{low}}(u,v) = \exp\left(-\frac{1}{2}\left(\frac{D(u,v)}{D_0}\right)^2\right) . \qquad (5.62)$$



(a) (b)

Figure 5.27: Band-pass frequency domain filtering. (a) Spectrum of a band-passfiltered image, low and high frequencies filtered out. (b) Image resulting from the inverse Fourier transform applied to spectrum (a).

The Butterworth filter is specified to have maximally flat frequency response over a spectrum band, and is also called a 'maximally flat magnitude filter'. The frequency response of the 2D low-pass Butterworth filter B_{low} of degree n is

$$B_{\text{low}} = \frac{1}{1 + \left(\frac{D(u,v)}{D_0}\right)^n} . \qquad (5.63)$$

The usual Butterworth filter degree is n = 2, which will be used here. Figure 5.29 illustrates the shape of the Gaussian and Butterworth filters for $D_0 = 3$ in 1D plots.

The high-pass filter is created easily from the low-pass filter. If the Fourier frequency spectrum of a low-pass filter is H_{low}, the high-pass filter can be created by just flipping it vertically, $H_{high} = 1-H_{low}$.

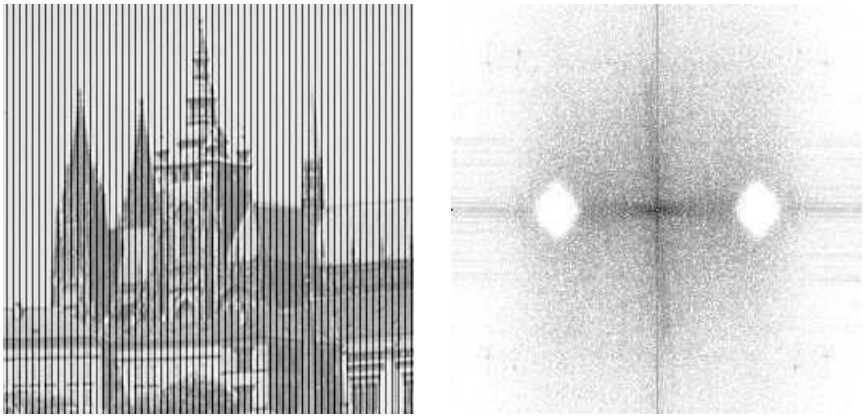
Another useful pre-processing technique operating in the frequency domain is an instance of **homomorphic filtering**. Homomorphic filtering is used to remove multiplicative noise. The aim of the particular homomorphic filtering is to simultaneously increase contrast and normalize image intensity across the image.

The assumption is that the image function f (x, y) can be factorized as a product of two independent multiplicative components in each pixel: illumination i(x, y) and the reflectance r(x, y) at the point in the observed scene, f (x, y) = i(x, y) r(x, y). These two components can be separated in some images because the illumination component tends to vary slowly and the reflectance component varies more quickly. The idea of the separation is to apply a logarithmic transform to the input image

$$z(x, y) = \log f (x, y) = \log i(x, y) + \log r(x, y) . \qquad (5.64)$$

If the image z(x, y) is converted to Fourier space (denoted by capital letters) then its additive components remain additive due to the linearity of the Fourier transform

$$Z(u, v) = I(u, v) + R(u, v) . \qquad (5.65)$$



(a) (b)

(c)

Figure 5.28: Periodic noise removal. (a) Noisy image. (b) Image spectrum used for image reconstruction—note that the areas of frequencies corresponding with periodic vertical lines are filtered out. (c) Filtered image. © Cengage Learning 2015.

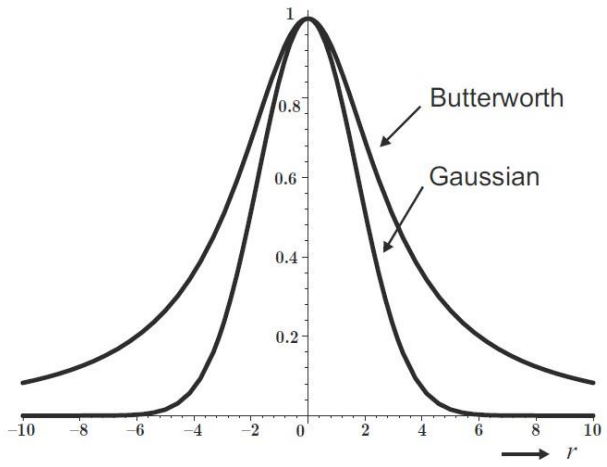


Figure 5.29: Gaussian and Butterworth low-pass filters. © *Cengage Learning 2015.*

Assume that the Fourier spectrum Z(u, v) is filtered by the filter H(u, v) and the spectrum S(u, v) is the result

$$S = H .* Z = H .* I + H .* R . \quad (5.66)$$

Usually a high-pass filter is used for this purpose; assuming a high-pass Butterworth filter, it has to be damped in order not to suppress low frequencies entirely as they bear needed information too. The Butterworth filter modified by damping coefficient 0.5 is shown in Figure 5.30

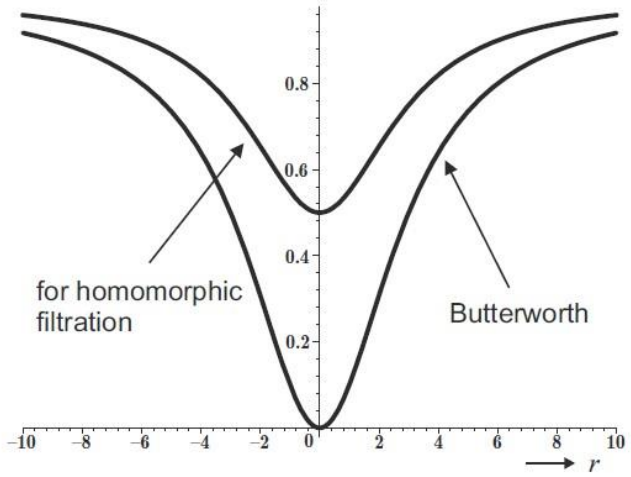


Figure 5.30: High-pass filter used in homomorphic filtering. It is a Butterworth filter damped by a 0.5 coefficients to retain some low frequencies. © *Cengage Learning 2015.*

Having the filtered spectrum S(u, v), we can return to spatial coordinates using the inverse Fourier transform, s(x, y) = −1S(u, v). Recall that the logarithm was first applied to the input image f (x, y) in equation (5.64). Now the image has to be transformed by the logarithm inverse function; this inverse function is the exponential. The result—the image g(x, y) filtered by the homomorphic filter—is given by g(x, y) = exp s(x, y) .

An illustration of the effect of homomorphic filtering is in Figure 5.31, an image of a person in a dark tunnel with strong illumination at the entrance. Detail of the tunnel surface on the top and right side are not visible because the surface is too dark. The result of homomorphic filtering is in Figure 5.31b. More details can be seen in this image.

(a) (b)

Figure 5.31: Illustration of homomorphic filtering. (a) Original image. (b) Homomorphic filtering.

2.10. Line detection by local pre-processing operators:

Several other local operations exist which do not belong to the taxonomy given in Section 5.3, as they are used for different purposes such as line finding, line thinning, and line filling operators. Another group of operators finds '**interest points**' or '**locations of interest**' in the image.

It is interesting to seek features richer than edges which can be reliably detected in the image and which can outperform simple edge detectors in some classes of applications. Line detectors and corner detectors are some such. Line detectors are used to detect linear objects such as dimension lines in engineering drawings or railways or roads in satellite images. Corner detectors and other interest point-like detectors are used mainly to register two or more images one to the other (e.g, in stereo vision, motion analysis, panorama stitching, object recognition from images) or to index the image or dominant objects in it to an image database.

Line finding operators aim to find very thin curves in the image; it is assumed that curves do not bend sharply. Such curves and straight lines are called **lines** for the purpose of describing this technique. If a cross section perpendicular in direction to the tangent of a line is examined, we get a roof profile (see Figure 5.17) when examining edges. We assume that the width of the lines is approximately one or two pixels.

The presence of a line may be detected by local convolution of the image with con- volution kernels which serve as line patterns. The simplest collection of four such patterns of size 3 x 3 is able to detect lines rotated modulo the angle 45o. Three of four such convolution kernels are

$$h_1 = \begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}, \quad h_2 = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}, \quad h_3 = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}, \quad \ldots \quad (5.67)$$

A similar principle can be applied to bigger masks. The case of 5x5 masks is common. Such line detectors sometimes produce more lines than needed, and other non-linear constraints may be added to reduce this number. More sophisticated approaches deter- mine lines in images as ridges and ravines using the facet model. Line detection is frequently used in remote sensing and in document processing;

Local information about edges is the basis of a class of image segmentation techniques. Edges which are likely to belong to object boundaries are usually found by simple thresholding of the edge magnitude—such edge thresholding does not provide ideal contiguous boundaries that are one pixel wide. Sophisticated segmentation techniques that are dealt with in the next chapter serve this purpose. Here, much simpler edge thinning and filling methods are described. These techniques are based on knowledge of small local neighborhoods and are very similar to other local pre-processing techniques.

Thresholded edges are usually wider than one pixel, and **line thinning** techniques may give a better result. One line thinning method uses knowledge about edge orientation and in this case edges are thinned before thresholding. Edge magnitudes and directions provided by some gradient operator are used as input, and

the edge magnitudes of two neighboring pixels perpendicular to the edge direction are examined for each pixel in the image. If at least one of these pixels has edge magnitude higher than the edge magnitude of the examined pixel, then the edge magnitude of the examined pixel is assigned a zero value.

There are many other line thinning methods. In most cases the best results are achieved using mathematical morphology methods.

2.11. Image restoration:

Pre-processing methods that aim to suppress degradation using knowledge about its nature are called **image restoration**. Most image restoration methods are based on convolution applied globally to the whole image. There is a wide literature on restoration and only the basic principles and some simple degradations are considered here.

Image degradation can have many causes: defects of optical lenses, nonlinearity of the electro-optical sensor, graininess of the film material, relative motion between an object and camera, wrong focus, atmospheric turbulence in remote sensing or astronomy, scanning of photographs, etc. The objective of image restoration is to reconstruct the original image from its degraded version.

Image restoration techniques can be classified as **deterministic** or **stochastic**. Deterministic methods are applicable to images with little noise and a known degradation function. The original image is obtained by applying the function inverse to the degraded one. Stochastic techniques try to find the best restoration according to a particular statistical criterion, e.g., a least-squares method. There are three typical degradations with a simple function: relative constant speed movement of the object with respect to the camera, wrong lens focus, and atmospheric turbulence.

In most practical cases, there is insufficient knowledge about the degradation, and it must be estimated and modeled. This may be done on an a priori or a posteriori basis:

2.11.1. A priori knowledge about degradation is either known in advance or can be obtained before restoration. For example, if it is known that the image was degraded

by relative motion of an object with respect to the sensor, then the modeling determines only the speed and direction of the motion. Alternatively, we may seek to to estimate parameters of a device such as a TV camera or digitizer, whose degradation remains unchanged over a period of time and can be modeled by studying a known sample image and its degraded version.

2.11.2. A posteriori knowledge is that obtained by analyzing the degraded image. A typical example is to find some interest points in the image (e.g., corners, straight lines) and guess how they looked before degradation. Another possibility is to use spectral characteristics of the regions in the image that are relatively homogeneous.

A degraded image g can arise from the original image f by a process which can be expressed as

$$g(i,j) = s\left(\int\int_{(a,b)\in\mathcal{O}} f(a,b)\,h(a,b,i,j)\,\mathrm{d}a\,\mathrm{d}b\right) + \nu(i,j)\,, \qquad (5.74)$$

where s is some non-linear function and ν describes the noise. This is often simplified by neglecting the non-linearity and assuming that the function h is invariant with respect to position in the image, giving

$$g(i, j) = (f * h)(i, j) + \nu(i, j)\,. \qquad (5.75)$$

If the noise is not significant in this equation, then restoration equates to inverse convolution (also called deconvolution). If noise is not negligible, then the inverse convolution is solved as an overdetermined system of linear equations. Methods based on minimization of least square error such as Wiener filtering (off-line) or Kalman filtering (recursive, on-line; see Section 16.6.1) are examples [Bates and McDonnell, 1986].

2.11.1. Degradations that are easy to restore

In the Fourier domain, we can express equation (5.75) as

$$G = H\,F\,. \qquad (5.76)$$

Therefore, overlooking image noise ν, knowledge of the degradation function fully facilitates image restoration by inverse convolution (Section 5.4.2).

Relative motion of camera and object

Relative motion of a camera with a mechanical shutter and the photographed object during the shutter open time T causes smoothing of the object in the image. Suppose V is the constant speed in the direction of the x axis; the Fourier transform H(u, v) of the degradation caused in time T.

$$H(u,v) = \frac{\sin(\pi\,V\,T\,u)}{\pi\,V\,u}\,.$$

Wrong lens focus

Image smoothing caused by imperfect focus of a thin lens can be described by the function

$$H(u,v) = \frac{J_1(a\,r)}{a\,r}\,,$$

where J_1 is the Bessel function of the first order, $r^2 = u^2 + v^2$, and a is the displacement— the model is not space invariant.

Atmospheric turbulence

Atmospheric turbulence is degradation that needs to be restored in remote sensing and astronomy. It is caused by temperature non-homogeneity in the

atmosphere that deviates passing light rays. One mathematical model [Hufnagel and Stanley, 1964]
is

$$H(u, v) = e^{-c(u2 + v2)5/6}, \quad (5.79)$$

where c is a constant that depends on the type of turbulence which is usually found experimentally. The exponent 5/6 is sometimes replaced by 1.

2.11.2. Inverse filtering

Inverse filtering assumes that degradation was caused by a linear function h(i, j) (cf. equation 5.75) and considers the additive noise ν as another source of degradation. It is further assumed that ν is independent of the signal. After applying the Fourier transform to equation (5.75), we get

$$G(u, v) = F(u, v)\, H(u, v) + N(u, v). \quad (5.80)$$

The degradation can be eliminated using the restoration filter with a transfer function that is inverse to the degradation h. We derive the original image F (its Fourier transform to be exact) from its degraded version G (equation 5.80), as

$$F(u, v) = G(u, v)\, H^{-1}(u, v) - N(u, v)\, H^{-1}(u, v). \quad (5.81)$$

This shows that inverse filtering works well for images that are not corrupted by noise [not considering possible computational problems if H(u, v) gets close to zero at some location of the u, v space—fortunately, such locations can be neglected without perceivable effect on the restoration result]. However, if noise is present, two problems arise. First, the noise influence may become significant for frequencies where H(u, v) has small magnitude. This situation usually corresponds to high frequencies u, v. In reality, H(u, v) usually decreases in magnitude much more rapidly than N (u, v) and thus the noise effect may dominate the entire restoration result. Limiting the restoration to a small neighborhood of the u, v origin in which H(u, v) is sufficiently large overcomes this problem, and the results are usually quite acceptable. Secondly, we usually do not have enough information about the noise to determine N (u, v) sufficiently.

2.11.3. Wiener filtering

Wiener (least mean square) filtering [Wiener, 1942; Gonzalez and Woods, 1992; Castle- man, 1996] attempts to take account of noise properties by incorporating a priori know- ledge in the image restoration formula. Restoration by the **Wiener filter** gives an estimate f^ of the original uncorrupted image f with minimal mean square error

$$e^2 = \mathcal{E}\left\{\left(f(i,j) - \hat{f}(i,j)\right)^2\right\}, \quad (5.82)$$

where denotes the mean operator.5 If no constraints are applied to the solution of equation (5.82), then an optimal estimate f^ is the conditional mean value of the ideal image f under the condition g. This approach is complicated from the computational point of view. Moreover, the conditional probability density between the optimal image f and the corrupted image g is not usually known. The optimal estimate is in general a non-linear function of the image g.

Minimization of equation (5.82) is easy if the estimate f is^ a linear combination of the values in image g; the estimate f^ is then close (but not necessarily equal) to the theoretical optimum. The estimate is equal to the theoretical optimum only if the stochastic processes describing images f , g, and the noise ν are homogeneous, and their probability density is Gaussian. These conditions are not usually fulfilled for typical images.

Denote the Fourier transform of the Wiener filter by H_W . Then, the estimate F^ of the Fourier transform F of the original image f can be obtained as

$$\hat{F}(u,v) = H_W(u,v)\, G(u,v)\ . \tag{5.83}$$

H_W is not derived here, but may be found elsewhere [Gonzalez and Woods, 1992] as

$$H_W(u,v) = \frac{H^*(u,v)}{\left|H(u,v)\right|^2 + \left[S_{\nu\nu}(u,v)/S_{ff}(u,v)\right]}\ , \tag{5.84}$$

where H is the transform function of the degradation, denotes complex conjugate, S_{vv} is the spectral density of the noise, and S_{ff} is the spectral density of the undegraded image.

If Wiener filtering is used, the nature of degradation H and statistical parameters of the noise need to be known. Wiener filtering theory solves the problem of optimal a posteriori linear mean square estimates—all statistics (for example, power spectrum)

should be available in advance. Note the term S_{ff} (u, v) in equation (5.84), which rep- resents the spectrum of the undegraded image, which may be difficult to obtain with no foreknowledge of the undegraded image.

Restoration is illustrated in Figure 5.36 where an image that was degraded by 5 pixels motion in the direction of the x axis: Figure 5.36b shows the result of restoration by Wiener filtering.

(a) (b)

Figure 5.36: Restoration of motion blur using Wiener filtering. Courtesy of P. Kohout, Criminalistic Institute, Prague.

Despite its unquestionable power, Wiener filtering suffers several substantial limitations. First, the criterion of optimality is based on minimum mean square error and weights all errors equally, a mathematically fully acceptable criterion that unfortunately does not perform well if an image is restored for human viewing. The reason is that humans perceive the restoration errors more seriously in constant-graylevel areas and in bright regions, while they are much less sensitive to errors located in dark regions and in high-gradient areas. Second, spatially variant degradations cannot be restored using the standard Wiener filtering approach, and these degradations are common. Third, most images are highly non-stationary, containing large homogeneous areas separated by high-contrast edges. Wiener filtering cannot handle non-stationary signals and noise. To deal with real-life image degradations, more sophisticated approaches may be needed. Examples include **power spectrum equalization** and **geometric mean filtering**. These and other specialized restoration techniques can be found in higher-level texts devoted to this topic; is well suited for such a purpose.

UNIT III OBJECT DETECTION USING MACHINE LEARNING

Object detection– Object detection methods – Deep Learning framework for Object detection– bounding box approach-Intersection over Union (IoU) –Deep Learning Architectures-R-CNNFaster R-CNN-You Only Look Once(YOLO)-Salient features-Loss Functions-YOLO architectures

3.1 Object Detection

Object Detection is a computer vision technique to locate objects in an image or in a video. Organizations and researchers are spending huge time and resources to uncover this capability. When we humans look at a picture, we can quickly identify the objects and their respective position in an image. We can quickly categorize if it is an apple or a car or a human being. We can also determine from any angle. The reason is that our minds have been trained in such a way that it can identify various objects. Even if the size of an object gets smaller or bigger, we are able to locate them and detect them. The goal is to replicate this decision-making intelligence using Machine Learning and Deep Learning.

3.1.1 Object classification vs. object localization vs. object detection

Look at the images in Figure 5-1 of a vacuum cleaner. The image classification solutions to classify such images into “a Vacuum Cleaner” or “not.” So we could have easily labeled the first image as a vacuum cleaner.

On the other hand, localization refers to finding the position of the object in an image. So when we do Image Localization, it means that the algorithm is having a dual responsibility of classifying an image as well as drawing a bounding box around it, which is depicted in the second image. In the first image of Figure 5-1, we have a vacuum cleaner, and in the second image, we have localized it.

Figure 5-1 Object detection means identifying and localization of the object. In the first image, we can classify if it is a vacuum cleaner, while in the second image, we are drawing a box around it, which is the localization of the image

To scale the solution, we can have multiple objects in the same image and even multiple objects of different categories in the same image, and we have to identify all of them. And draw the bounding boxes around them. An example can be of a solution trained to detect cars. On a busy road, there will be many cars, and hence the solution should be able to detect each of them and draw bounding boxes around them.

Object detection is surely a fantastic solution. We will now discuss the major object detection use cases in the next section.

3.1.2 Use cases of Object Detection

Deep Learning has expanded many capabilities across domains and organizations. Object detection is a key one and is a very powerful solution which is making huge ripples in our business and personal world. The major use cases of object detection are

1. Object Detection is the key intelligence behind **autonomous driving technology**. It allows the users to detect the cars, pedestrians, the background, motorbikes, and so on to improve road safety.
2. We can **detect objects in the hands of people,** and the solution can be used for security and monitoring purposes. Surveillance systems can be made much more intelligent and accurate. Crowd control systems can be made more sophisticated, and the reaction time will be reduced.
3. A solution might be used for **detecting objects in a shopping basket, and** it can be used by the retailers for the automated transactions. This will speed up the overall process with less manual intervention.
4. Object Detection is also used in testing of mechanical systems and on manufacturing lines. We can **detect objects present on the products which might be contaminating the product quality.**
5. In the medical world, the **identification of diseases by analyzing the images** of a body part will help in faster treatment of the diseases.

There are very less areas where the usage is not envisioned. It is one of the areas which are highly researched, and every day new progress is being made in this domain. Organizations and researchers across the globe are making huge ripples in this area and creating path-breaking solutions.

3.2 Object Detection methods

We can perform object detection using both Machine Learning and Deep Learning. Here are a few Machine Learning solutions:

1. Image segmentation using simple attributes like shape, size, and color of an object.
2. We can use an aggregated channel feature (ACF), which is a variation of channel features. ACF does not calculate the rectangular sums at various locations or scales. Instead, it extracts features directly as pixel values.
3. Viola-Jones algorithm can be used for face detection.

There are other solutions like RANSAC (random sample consensus), Haar feature–based cascade classifier, SVM classification using HOG features, and so on which can be used for object detection.

Deep Learning methods :

The following Deep Learning architectures are commonly being used for Object Detection:

1. R-CNN: Regions with CNN features. It combines Regional Proposals with CNN.
2. Fast R-CNN: A Fast Region–based Convolutional Neural Network.
3. Faster R-CNN: Object detection networks on Region Proposal algorithms to hypothesize object locations.
4. Mask R-CNN: This network extends Faster R-CNN by adding the prediction of segmentation masks on each region of interest.
5. YOLO: You Only Look Once architecture. It proposes a single Neural Network to predict bounding boxes and class probabilities from an image in a single evaluation.
6. SSD: Single Shot MultiBox Detector. It presents a model to predict objects in images using a single deep Neural Network.

3.3 Deep Learning frameworks for Object Detection

Few important components of Object Detection are

- Sliding window approach for Object Detection Bounding box approach
- Intersection over Union (IoU) Non-max suppression Anchor boxes concept

3.3.1 Sliding window approach for Object Detection

When we want to detect objects, a very simple approach can be: why not divide the image into regions or specific areas and then classify each one of them. This approach for object detection is *sliding window*. As the name suggests, it is a rectangular box which slides through the entire image. The box is of fixed length and width with a stride to move over the entire image.

Look at the image of the vacuum cleaner in Figure 5-2. We are using a sliding window at each part of the image. The red box is sliding over the entire image of the vacuum cleaner. From left to right and then vertically, we can observe that different parts of the image are becoming the point of observation. Since the window is sliding, it is referred to as the sliding window approach.

Figure 5-2 The sliding window approach to detect an object and identify it. Notice how the sliding box is moving across the entire image; the process is able to detect but is really a time-consuming process and computationally expensive too

Then for each of these regions cropped, we can classify whether this region contains an object that interests us or not. And then we increase the size of the sliding window and continue the process.

Sliding window has proven to work, but it is a computationally very expensive technique and will be slow to implement as we are classifying all the regions in an image.

Also, to localize the objects, we need a small window size and small stride. But still it is a simple approach to understand.

3.4 Bounding box approach

The sliding window approach outputs less accurate bounding boxes as it is dependent on the size of the window. And hence we have another approach wherein we divide the entire image into grids (x by x), and then for each grid, we define our target label. We can show a bounding box in Figure 5-3.

Figure 5-3 Bounding box can generate the x coordinate, y coordinate, height, and width of the bounding box and the class probability score

A bounding box can give us the following details:

Pc: Probability of having an object in the grid cell (0: no object, 1: an object). Bx:

If Pc is 1, it is the x coordinate of the bounding box.
By: If Pc is 1, it is the y coordinate of the bounding box. Bh:
If Pc is 1, it is the height of the bounding box.
Bw: If Pc is 1, it is the width of the bounding box.
C1: It is the class probability that the object belongs to Class 1. C2:
It is the class probability that the object belongs to Class 2.

If an object lies over multiple grids, then the grid that contains the midpoint of that object is responsible for detecting that object.

3.5 Intersection over Union (IoU):

Intersection over Union is a test to ascertain how close is our prediction to the actual truth.

It is represented by Equation 5-1 and is shown in Figure 5-4.

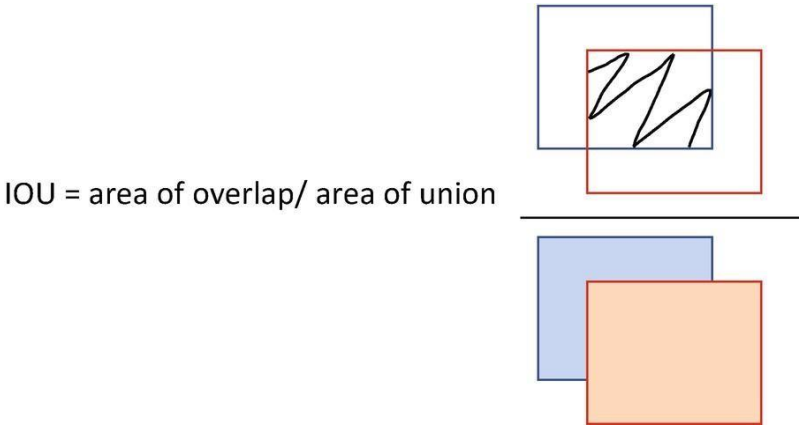


Figure 5-4 Intersection over Union is used to measure the performance of detection. The numerator is the common area, while the denominator is the complete union of the two areas. The higher the value of IoU, the better it is

IoU = Overlapping region/Combined entire region (Equation 5-1)

So, if we get a higher value of Intersection over Union, it means the overlap is better. Hence, the prediction is more accurate and better. It is depicted in the example in Figure 55 to visualize.

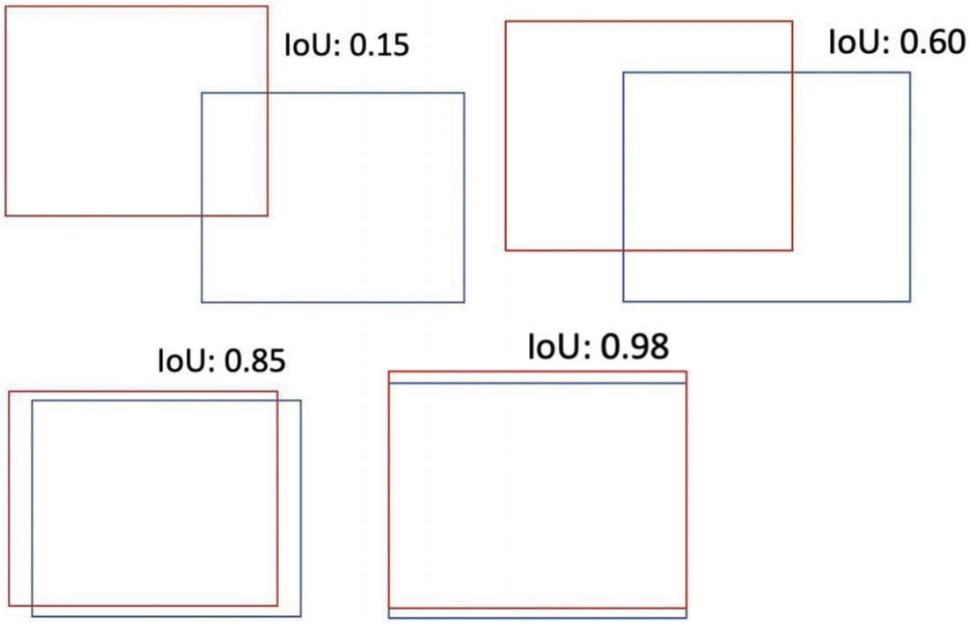


Figure 5-5 IoU values for different positions of the overlapping blocks. If the value is closer to 1.0, it means that the detection is more accurate as compared to the value of 0.15

As we can see in Figure 5-5, for IoU of 0.15, there is very less overlap between the two boxes as compared to 0.85 or 0.90. It means that the one with 0.85 IoU is a better solution to the one with 0.15 IoU. The detection solution can hence be compared directly.

Intersection over Union allows us to measure and compare the performance of various solutions. It also makes it easier for us to distinguish between useful bounding boxes and not-so-important ones. Intersection over Union is an important concept with wide usages. Using it, we can compare and contrast the acceptability of all the possible solutions and choose the best one from them.

3.6 Deep Learning architectures

Deep Learning helps in object detection. We can detect objects of interest in an image or in a video or even in the live video stream. We are going to create a live video stream solution later in the chapter.

We have seen earlier that there are some problems with the sliding window approach. Objects can have different locations in an image and can be of different aspect ratio or size. An object might be covering the entire region; on the other hand, somewhere it will be covering a small percentage only. There might be more than one object present in the image. The objects can be at various angles or dimensions. Or one object can lie in multiple grids. And moreover, some use cases require real-time predictions. It results in having a very large number of regions and hence huge computation power. It will take a considerable amount of time too. The traditional approaches of image analysis and detection will not be of much help in such situations. Hence, we require Deep Learning– based solutions to resolve and develop robust solutions for object detection.

Deep Learning–based solutions allow us to train better and hence get better results.

3.6.1 Region-based CNN (R-CNN)

We understand that having a very large number of regions is a challenge. Ross Girshick et al. proposed R-CNN to address the problem of selecting a large number of regions. RCNN is Region-based CNN architecture. Instead of classifying a huge number of regions, the solution suggests to use selective search and extract only 2000 regions from the image. They are called “Region Proposals.”

The architecture for R-CNN is shown in Figure 5-8.

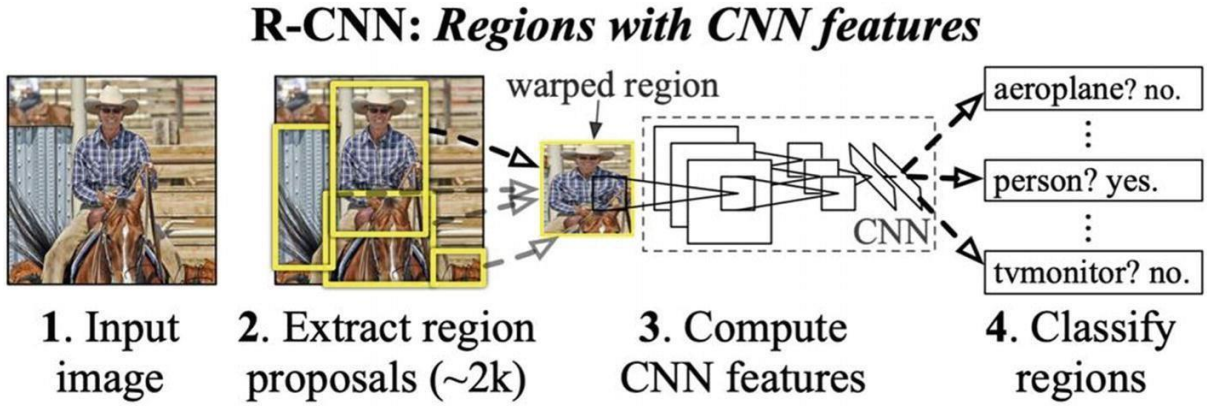


Figure 5-8 The process in R-CNN. Here, we extract region proposals from the input image, compute the CNN features, and then classify the regions. Image source: https://arxiv.org/pdf/1311.2524.pdf and published here with the permission of the researchers

With reference to Figure 5-8 where we have shown the process, let us understand the entire process in detail now:

1.

The first step is to input an image, represented by step 1 in Figure 5-8.

2. Then get the regions we are interested in, which is shown in step 2 in Figure 5-8. These are the 2000 proposed regions. They are detected using the following steps:

a) We create the initial segmentation for the image. b) Then we generate the various candidate regions for the image. c) We combine similar regions into larger ones iteratively. A greedy search approach is used for it.

d) Finally, we use the generated regions to output the final region proposals.

3. Then in the next step, we reshape all the 2000 regions as per the implementation in the CNN.

4. We then pass through each region through CNN to get features for each region.

5. The extracted features are now passed through a support vector machine to classify the presence of objects in the region proposed.

6. And then, we predict the bounding boxes for the objects using bounding box regression. This means that we are making the final prediction about the image. As shown in the last step, we are making a prediction if the image is an airplane or a person or a TV monitor.

The preceding process is used by R-CNN to detect the objects in an image. It is surely an innovative architecture, and it proposes a region of interest as an impactful concept to detect objects.

But there are a few challenges with R-CNN, which are

1. R-CNN implements three algorithms (CNN for extracting the features, SVM for the classification of objects, and bounding box regression for getting the bounding boxes). It makes R-CNN solutions quite slow to be trained.
2. It extracts features using CNN for each image region. And the number of regions is 2000. It means if we have 1000 images, the number of features to be extracted is 1000 times 2000 which again makes it slower.
3. Because of these reasons, it takes 40–50 seconds to make a prediction for an image, and hence it becomes a problem for huge datasets.
4. Also, the selective search algorithm is fixed, and not much improvements can be made.

As R-CNN is not very fast and quite difficult to implement for huge datasets, the same authors proposed Fast R-CNN to overcome the issues. w

3.7 Faster R-CNN

To overcome the slowness in R-CNN and Fast R-CNN, Shaoqing Ran et al. proposed Faster R-CNN. The intuition behind the Faster R-CNN is to replace the selective search which is slow and time-consuming. Faster R-CNN uses the Regional Proposal Network or RPN.

The architecture of Faster R-CNN is shown in Figure 5-10.

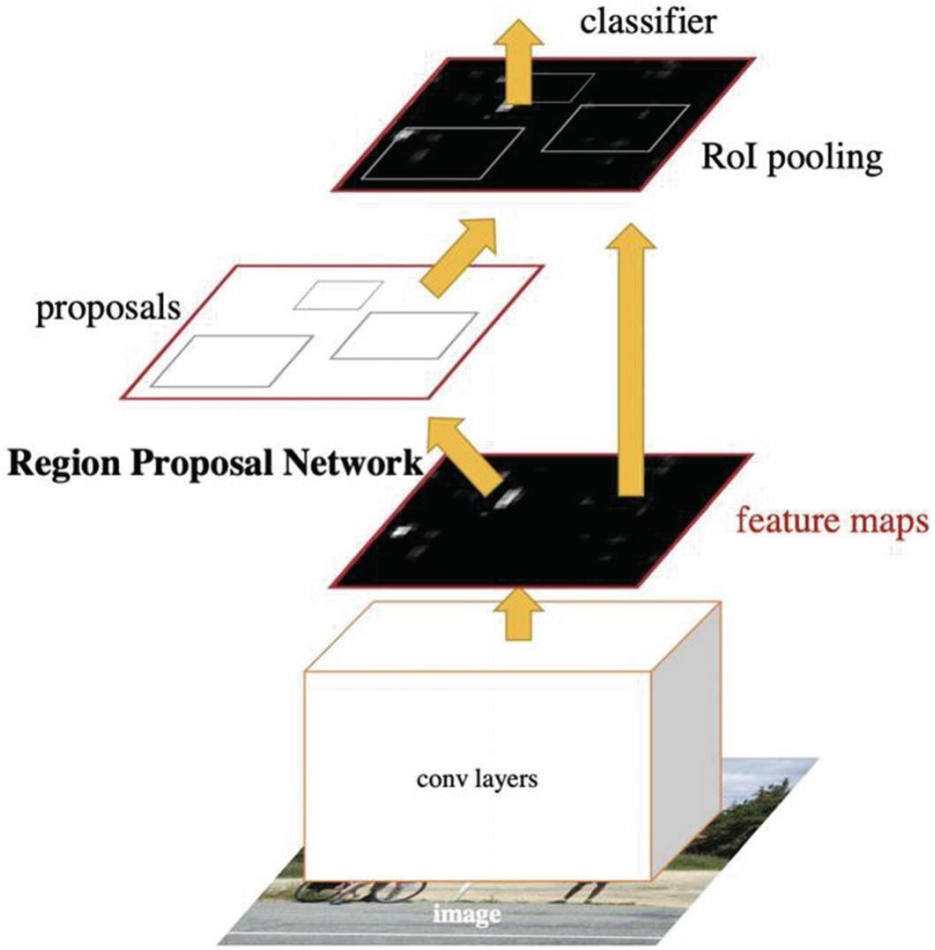


Figure 5-10 Faster R-CNN is an improvement over the previous versions. It consists of two modules – one is a deep convolutional network, and the other is the Fast R-CNN detector

Faster R-CNN, is composed of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector that uses the proposed regions. The entire system is a single, unified network for object detection.

The way a Faster R-CNN works is as follows:

1. We take an input image and make it pass through CNN as shown in Figure 5- 10.
2. From the feature maps received, we apply Region Proposal Networks (RPNs). The way an RPN works can be understood by referring to Figure 5-11.

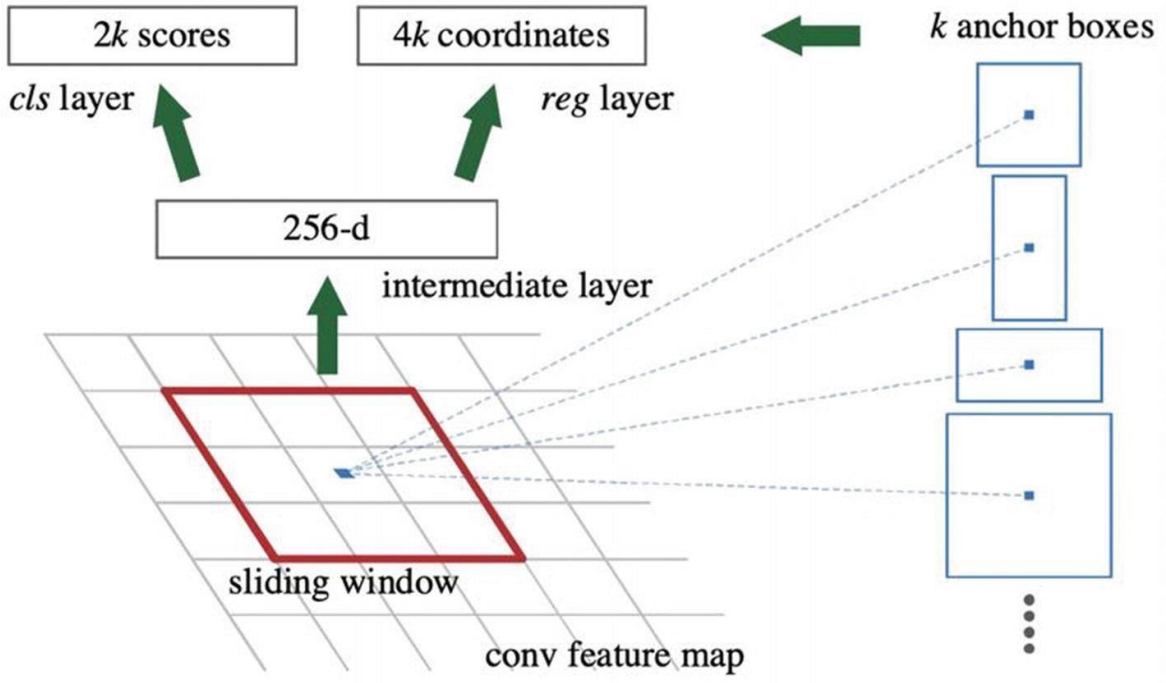


Figure 5-11 Region proposal networks are used in Faster R-CNN. The image has been taken from the original paper

The sub steps followed are

a) RPN takes the feature maps generated from the last step.
b) RPN applies a sliding window and generates k anchor boxes. We have discussed anchor boxes in the last section.
c)The anchor boxes generated are of different shapes and sizes.
d) RPN will also predict that an anchor is an object or not.
e) It will also give the bounding box regressor to adjust the anchors.
f) To be noted is RPN has not suggested the class of the object.
g) We will get object proposals and the respective objectness scores.

3. Apply ROI pooling to make the size of all the proposals the same.
4. And then, finally, we feed them to the fully connected layers with softmax and linear regression.
5. We will receive the predicted Object Classification and respective bounding boxes.

Faster R-CNN is able to combine the intelligence and use deep convolution fully connected layers and Fast R-CNN using proposed regions. The entire solution is a single and unified solution for object detection.

Though Faster R-CNN is surely an improvement in terms of performance over RCNN and Fast R-CNN, still the algorithm does not analyze all the parts of the image simultaneously. Instead, each and every part of the image is analyzed in a sequence. Hence, it requires a large number of passes over a single image to recognize all the objects. Moreover, since a lot of systems are working in a sequence, the performance of one depends on the performance of the preceding steps.

3.8 You Only Look Once (YOLO)

You Only Look Once or YOLO is targeted for real-time object detection. The previous algorithms we discussed use regions to localize the objects in the image. Those algorithms look at a part of the image and not the complete image, whereas in YOLO a single CNN predicts both the bounding boxes and the respective class probabilities. YOLO was proposed in 2016 by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. The actual paper can be accessed at https://arxiv.org/pdf/1506.02640v5.pdf.

To quote from the actual paper, "We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities."

As shown in Figure 5-12, YOLO divides an image into a grid of cells (represented by S). Each of the cells predicts bounding boxes (represented by B). Then YOLO works on each bounding box and generates a confidence score about the goodness of the shape of the box. The class probability for the object is also predicted. Finally, the bounding box having class probability scores above are selected, and they are used to locate the object within that image.

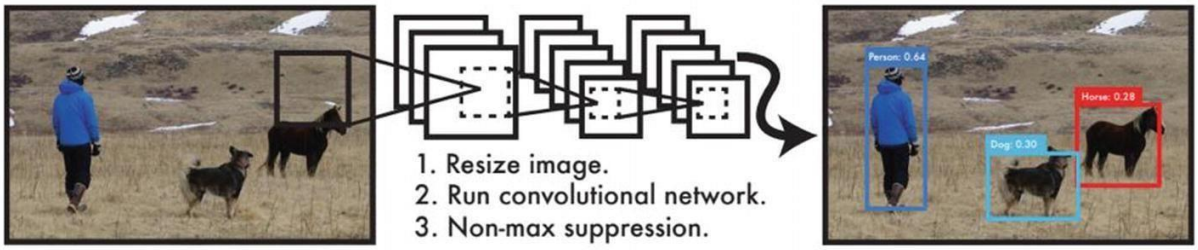


Figure 5-12 The YOLO process is simple; the image has been taken from the original paper https://arxiv.org/pdf/1506.02640v5.pdf

3.8.1 Salient features of YOLO

1. YOLO divides the input image into an SxS grid. To be noted is that each grid is responsible for predicting only one object. If the center of an object falls in a grid cell, that grid cell is responsible for detecting that object.
2. For each of the grid cells, it predicts boundary boxes (B). Each of the boundary boxes has five attributes – the x coordinate, y coordinate, width, height, and a confidence score. In other words, it has (x, y, w, h) and a score. This confidence score is the confidence of having an object inside the box. It also reflects the accuracy of the boundary box.
3. The width w and height h are normalized to the images' width and height. The x and y coordinates represent the center relative to the bound of the grid cells.
4. The confidence is defined as Probability(Object) times IoU. If there is no object, the confidence is zero. Else, the confidence is equal to the IoU between the predicted box and ground truth.

5. Each grid cell predicts C conditional class probabilities – Pr(Classi | Object). These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B.
6. At the test time, we multiply the conditional class probabilities and the individual class predictions. It gives us the class-specific confidence scores for each box. It can be represented in Equation 5-2:

$$|\psi(t_1)\rangle = U(t_1, t_0) \mid \psi(t_0)\rangle$$

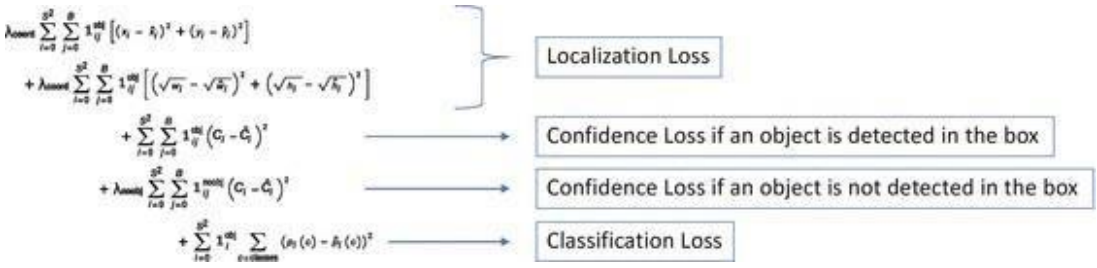
We will now examine how we calculate the loss function in YOLO. It is important to get the loss function calculation function before we can study the entire architecture in detail.

3.8.2 Loss function in YOLO

We have seen in the last section that YOLO predicts multiple bounding boxes for each cell. And we choose the bounding box which has the maximum IoU with the ground truth. To calculate the loss, YOLO optimizes for sum-squared error in the output in the model as sum-squared error is easy to optimize.

The loss function is shown in Equation 5-3 and comprises localization loss, confidence loss, and classification loss. We are first representing the complete

loss function and then describing the terms in detail.



(Equation 5-3)

In Equation 5-3, we have localization loss, confidence loss, and classification loss, where 1^{obj}_{i} denotes if the object appears in cell i and 1^{obj}_{ij} denotes that the j^{th} bounding box predictor in cell i is "responsible" for that prediction.

Let's describe the terms in the preceding equation. Here, we have

A. Localization loss is to measure the errors for the predicted boundary boxes. It measures their location and size errors. In the preceding equation, the first two terms represent the localization loss. 1^{obj}_{i} is 1 if the j^{th} boundary box in cell i is responsible for detecting the object, else the value is 0. λ_{coord} is responsible for the increase in the weight for the loss in the coordinates of the boundary boxes. The default value of λ_{coord} is 5.
B. Confidence loss is the loss if an object is detected in the box. It is the second loss term in the equation shown. In the term earlier, we have

$\hat{C}_i$ is the box confidence score of the box j in cell i.

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

C. The next term is a confidence loss if the object is not detected. In the term earlier, we have

$\mathbb{1}_{ij}^{noobj}$ is the complement of $\mathbb{1}_{ij}^{obj}$.

$\hat{C}_i$ is the box confidence score of the box j in cell i.

λ_{noobj} weights down the loss when detecting background.

D. The final term is the classification loss. If an object is indeed detected, then for each cell it is the squared error of the class probabilities for each class.

$\mathbb{1}_{i}^{obj} = 1$ if an object appears in cell i, otherwise 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i.

The final loss is the sum total of all these components. As the objective of any Deep Learning solution, the objective will be to minimize this loss value.

3.8.3 YOLO architecture

The network design is shown in Figure 5-13 and is taken from the actual paper at https://arxiv.org/pdf/1506.02640v5.pdf.

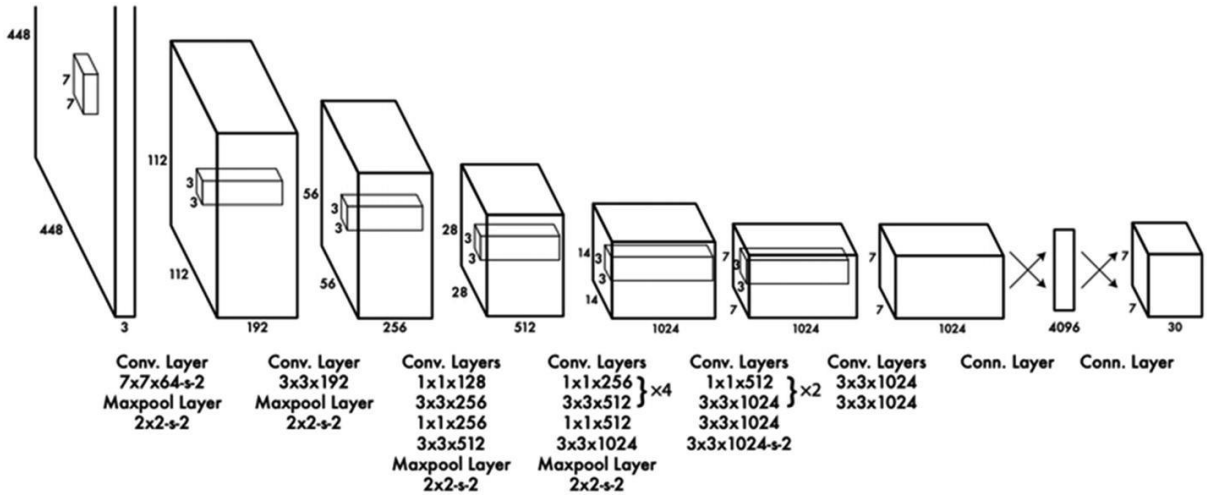


Figure 5-13 The complete YOLO architecture; the image has been taken from the original paper at https://arxiv.org/pdf/1506.02640v5.pdf

In the paper, the authors have mentioned that the network has been an inspiration from GoogLeNet. The network has 24 convolutional layers followed by 2 fully connected layers. Instead of Inception modules used by GoogLeNet, YOLO uses 1x1 reduction layers followed by 3x3 convolutional layers. YOLO might detect the duplicates of the same object. For this, non-maximal suppression has been implemented.

This removes the duplicate lower confidence score.

In Figure 5-14, we have a figure having 13x13 grids. In total, 169 grids are there wherein each grid predicts 5 bounding boxes. Hence, there are a total of 169*5 = 845 bounding boxes. When we apply a threshold of 30% or more, we get 3 bounding boxes as shown in Figure 5-14.

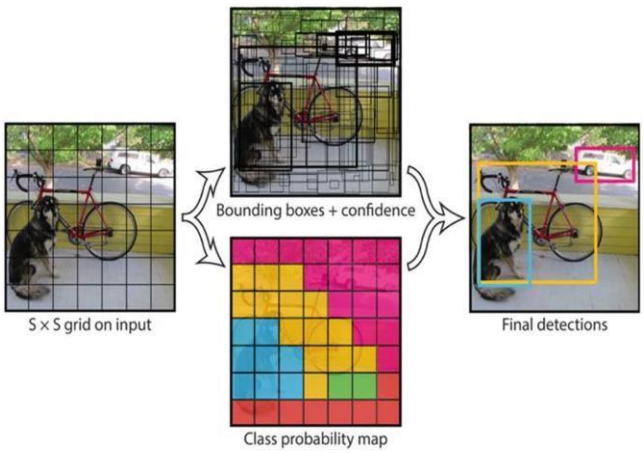


Figure 5-14 The YOLO process divides the region into SxS grids. Each grid predicts five bounding boxes, and based on the threshold setting which is 30% here, we get the final three bounding boxes; the image has been taken from the original paper

So, YOLO looks at the image only once but in a clever manner. It is a very fast algorithm for real-time processing. To quote from the original paper:

1. YOLO is refreshingly simple.
2. YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our Neural Network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems.
3. YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.

4. YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.

There are a few challenges with YOLO too. It suffers from high localization error. Moreover, since each of the grid cells predicts only two boxes and can have only one class as the output, YOLO can predict only a limited number of nearby objects. It suffers from a problem of low recall too. And hence in the next version of YOLOv2 and YOLOv3, these issues were addressed.

YOLO is one of the most widely used object detection solutions. Its uniqueness lies in its simplicity and speed.

Questions:

Part-A

1. What is the concept of anchor boxes and non-max suppression?

To generate the final object detections, tiled anchor boxes that belong to the background class are removed, and the remaining ones are filtered by their confidence score. Anchor boxes with the greatest confidence score are selected using nonmaximum suppression (NMS).

2. How are bounding boxes important for object detection?

In the context of digital image processing, the bounding box denotes the border's coordinates on the X and Y axes that enclose an image. They are used to identify a target and serve as a reference for object detection and generate a collision box for the object.

3. How are R-CNN , Fast R-CNN and Faster R-CNN different and what are the improvements?

	R-CNN	Fast R-CNN	Faster R-CNN
region proposals method	Selective search	Selective search	Region proposal network
Prediction timing	40-50 sec	2 seconds	0.2 seconds
computation	High computation time	High computation time	Low computation time

The mAP on Pascal VOC 2007 test dataset(%)	58.5	66.9 (when trained with VOC 2007 only) 70.0 (when trained with VOC 2007 and 2012 both)	69.9(when trained with VOC 2007 only)
The mAP on Pascal VOC 2012 test dataset (%)	53.3	65.7 (when trained with VOC 2012 only) 68.4 (when trained with VOC 2007 and 2012 both)	67.0(when trained with VOC 2012 only) 70.4 (when trained with VOC 2007 and 2012 both) 75.9(when trained with VOC 2007 and 2012 and COCO)

4. ***What is IoU?***

IoU calculates intersection over the union of the two bounding boxes, the bounding box of the ground truth and the predicted bounding box.

5. ***What are the metrics used for object detection?***

mAP (mean Average precision) is a popular metric in measuring the accuracy of object detectors. Average precision calculates the average precision value for recall value over 0 to 1.

6. ***What is NMS?***

Non-Max Suppression (NMS) is a technique used in many computer vision object detection algorithms. It is a class of algorithms to select one bounding box out of many overlapping bounding boxes for a single class. NMS implementation:

a) Sort the prediction confidence scores in decreasing order.

b) Start from the top scores, ignore any current prediction if we find any previous predictions that have the same class and IoU > Threshold(generally we use 0.5) with the current prediction.
c) Repeat the above step until all predictions are checked.

7. ***What is the loss function in YOLO?***

YOLO uses a sum of squared error between the predictions and the ground truth to calculate the loss. The loss function composes of:

- The **Classification loss**.
- The **Localization loss** (errors between the predicted boundary box and the ground truth).
- The **Confidence loss** (the objectness of the box).

8. ***What is the advantage of two-stage methods?***

In two-stage methods like R-CNN, they first predict *a few* candidate object locations and then use a convolutional neural network to classify each of these candidate object locations as one of the classes or as background.

9. ***What is FPN?***

Feature Pyramid Network (**FPN**) is a feature extractor designed with a feature pyramid concept to improve accuracy and speed. Images are first to pass through the CNN pathway, yielding semantically rich final layers. Then to regain better resolution, it creates a top-down pathway by upsampling this feature map. While the top-down pathway helps detect objects of varying sizes, spatial positions may be skewed. Lateral connections are added between the original feature maps and the corresponding reconstructed layers to improve object localization. It currently provides one of the leading ways to detect objects at multiple scales, and YOLOv3, Faster R-CNN were build up with this technique.

10. ***Why do we use data augmentation?***

Data augmentation is a technique for synthesizing new data by modifying existing data in such a way that the target is not changed, or it is changed in a known way. Data augmentation is important in improving accuracy. Augment data techniques like flipping, cropping, add noise, and color distortion.

11. ***What is the advantage of SDD over Faster R-CNN?***

SSD speeds up the process by removing the need for the region proposal network(RPN) used in Faster R-CNN.

Part – B

1. Explain about Object detection and various Object detection methods.

2. Elaborate about Deep Learning framework for Object detection.
3. Explain about bounding box approach.
4. Discuss about Intersection over Union (IoU).
5. Elaborate about Deep Learning Architectures of R-CNN.
6. Discuss about Faster R-CNN.
7. Discuss about You Only Look Once (YOLO), Salient features, Loss Functions.
8. Illustrate and explain about YOLO architectures.

UNIT IV FACE RECOGNITION AND GESTURE RECOGNITION

Face Recognition-Introduction-Applications of Face Recognition-Process of Face Recognition- DeepFace solution by Facebook-FaceNet for Face Recognition-Implementation using FaceNet, Gesture Recognition.

4.1. Face Recognition and Gesture Recognition

We humans are absorbed by our faces and faces of others, our smiles, our emotions, the different poses we make, and different expressions we have. Our mobile phones and cameras capture all of this. When we recognize a friend, we recognize the face – its shape, eyes, facial characteristics. And quite interestingly, even if we look at the same face from a side pose, we will be able to recognize it.

Surprisingly, we humans are able to detect the face even if we look at it after a long duration. We create that mental position of the attributes of a face, and we are able to recall it easily. At the same time, the gestures which we make using our hands are easily recognizable. Deep Learning is able to help recreate this capability. The usage of face recognition is quite innovative – it can be used across domains of security, surveillance, automation, and customer experience – the use cases are many. There is a lot of research going on in this field.

Face recognition

Face recognition is nothing new. We are born with a natural capability to differentiate and recognize faces. It is a trivial task for us. We can recognize the people we know in any kind of background, different lights, hair color, with cap or sunglasses, and so on. Even if a person has aged or has a beard, we can recognize them. Amazing!

Now we attempt to train the Deep Learning algorithms to achieve the same feat. A task so trivial and effortless for us is not an easy one for the machine. In Figure 6-1, we are having a face, then we are detecting a face, and then recognizing a face.

Figure 6-1 We have a face initially. In the second picture, a face is detected, and finally we are able to recognize a face with a specific name

We can consider face recognition as a special case of object detection. Instead of discovering cars and cats, we are identifying people. But the problem is simpler; we have only the class of object to be detected

– "face." But face detection is not the end state. We have to put a name to that face too, which is not trivial. Moreover, the face can be at any angle; a face can have different backgrounds. So, it is not an easy task.

Also, we might be discovering faces in photographs or in videos. Deep Learning algorithms can help us in developing such capabilities. Deep Learning–based algorithms can handle the computation power, advanced mathematical foundation, and the millions of data points or faces to train better models for face recognition.

4.2. Applications of face recognition

Face recognition is a pretty exciting technology having applications across domains and processes. Some of the key uses are

1. Security management: The face recognition solutions are applicable for both online and offline security systems. Security services, police departments, and secret services utilize the power of machine learning– based face recognition techniques to trace the antisocial elements. Passport verification can happen quicker and in a much more robust fashion. Many countries do maintain a database of criminals' photographs which acts as a starting point to trace the culprits. The technology saves really a great deal of time and energy and allows the investigators to focus their energies on other areas.
2. Identity verification is another big area employing face recognition techniques. One of the most famous examples of ID verification is smartphones. Face ID is used in iPhones and the phone unlocks. Face recognition is being used by online channels and social media to check the identity of the person trying to access the account.
3. It is used by retailers to know when individuals with not-so-good history have entered the premises. When shoplifters, criminals, or fraudsters enter the stores, they act as a threat. Retailers can identify them and take immediate actions to prevent any crime.
4. Marketing becomes much sharper if the business knows the age, gender, and facial expressions of the customer. Giant screens can be installed (in fact have been done) to identify the target audience.
5. Consumer experience is improved when the consumer-product interaction is analyzed. Expressions of people when they touch the products or try them capture the real-world interactions. The data acts as a gold mine for the product teams to make necessary amendments to the product features. At the same time, the operations and in-store team can make the overall shopping experience more enjoyable and interesting.
6. Access to offices, airports, buildings, warehouses, and parking lots can be automated with no human intervention. A security camera takes a picture and compares it with the database to ensure authenticity.

The use cases discussed earlier are only a few of the many applications of face recognition capabilities. The solutions are hence broadly for face *authentication* or

face *verification* or face *identification*. Many organizations and countries are creating huge databases of employees/individuals and investing to sharpen the skills further.

4.3. Process of face recognition

We click pictures from our phones and cameras. The photos are taken of various occasions – marriage, graduation, trips, holidays, conferences, and so on. When we upload the pictures on social media, it automatically detects the face and recognizes who the person is. An algorithm works in the background and does the magic. The algorithm is not only able to detect the face but will put a name to it from all the other faces in the background. We are studying the similar process in this section.

Broadly, we can have these four steps around face recognition as shown in Figure 6-2.

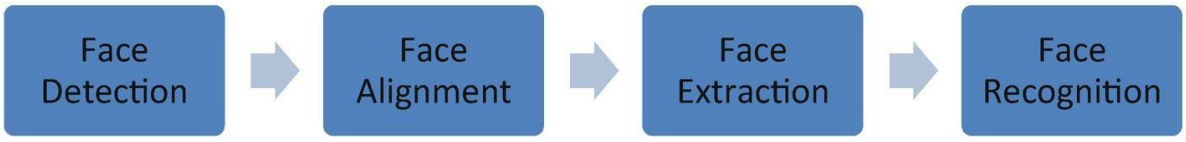


Figure 6-2 Process followed in face detection – right from detection to recognition. We detect a face, perform alignment, extract the features, and finally recognize the face

1. Face *detection* simply means to locate if there is a face or multiple faces in a photograph. And we will create a bounding box around it. OpenCV can be used to detect the presence of a face in the photograph.
2. Once we have detected the face, we *normalize* the attributes of the face like the size and geometry. It is done so that it matches the facial database we have. We also reduce the effect of illumination, head movement, and so on.
3. Next, we *extract* the features from the face. Some of the distinguishing features are eyes, eyebrows, the nostrils, corners of the mouth, and so on.
4. And then we perform the face *recognition*. It means we match the face with the existing ones in the database. We might perform one of the two: a.

   Verify the given face with a known identity. In simple terms, we want to know "Is this Mr. X?". It is a case of one-to-one relationship.

   b.

   Or we might want to know "Who is this guy?," and in such a case we will have a one-to-many relationship.

The problem hence looks like a supervised learning classification problem. we can create a face detection solution using OpenCV. There, we simply identified if there is a face present or not. Face recognition is giving a name to that face. It is imperative to note that without a concrete face detection, face recognition attempts will be futile. After all, first we should know if a face exists, then only we can give a name to that face. In other words, detection is to be done first followed by assigning a name. If there are more than one person in the photograph, we will be assigning names to all the faces detected in the photograph.

This is the entire process of face recognition.

4.4. Deep Learning modes for face recognition

Deep Learning is making its presence felt for face recognition too. Face recognition is similar to any other image classification solution. But faces and attributes of features make face recognition and detection quite a special one.

We can use the standard Convolutional Neural Network for face recognition too. The layers of the network will behave and process the data similar to any other image analysis problem.

There are a overabundance of solutions available, but the most famous are DeepFace, VGGFace, DeepID, and FaceNet as Deep Learning algorithms.

4.5. DeepFace solution by Facebook

DeepFace was proposed by researchers of Facebook AI Research (FAIR) in 2014. The actual paper can be accessed at www.cs.toronto.edu/~ranzato/publications/taigman_cvpr14.pdf.

Figure 6-3 shows the actual architecture of DeepFace, which is taken from the same paper mentioned previously.

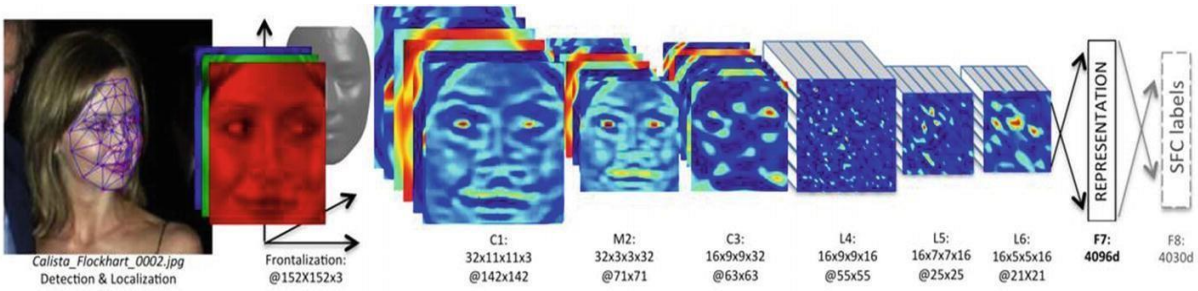


Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate feature maps produced at each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

Figure 6-3 DeepFace architecture is shown here. The figure has been taken from the original paper at https://www.cs.toronto.edu/~ranzato/publications/taigman_cvpr14.pdf

In the architecture shown earlier, we can analyze the various layers and the processes of the network. DeepFace expects an input image as a *3D-aligned* RGB image of 152x152. We will now explore this concept of 3D alignment in detail.

The objective of alignment is to generate a front face from the input image. The complete process is shown in Figure 6-4 which is taken from the same paper.

In the first step, we detect a face using six fiducial points. These six fiducial points are two eyes, tip of the nose, and three points on the lips. In Figure 6-4, it is depicted in step (a). This step detects the face in the image.

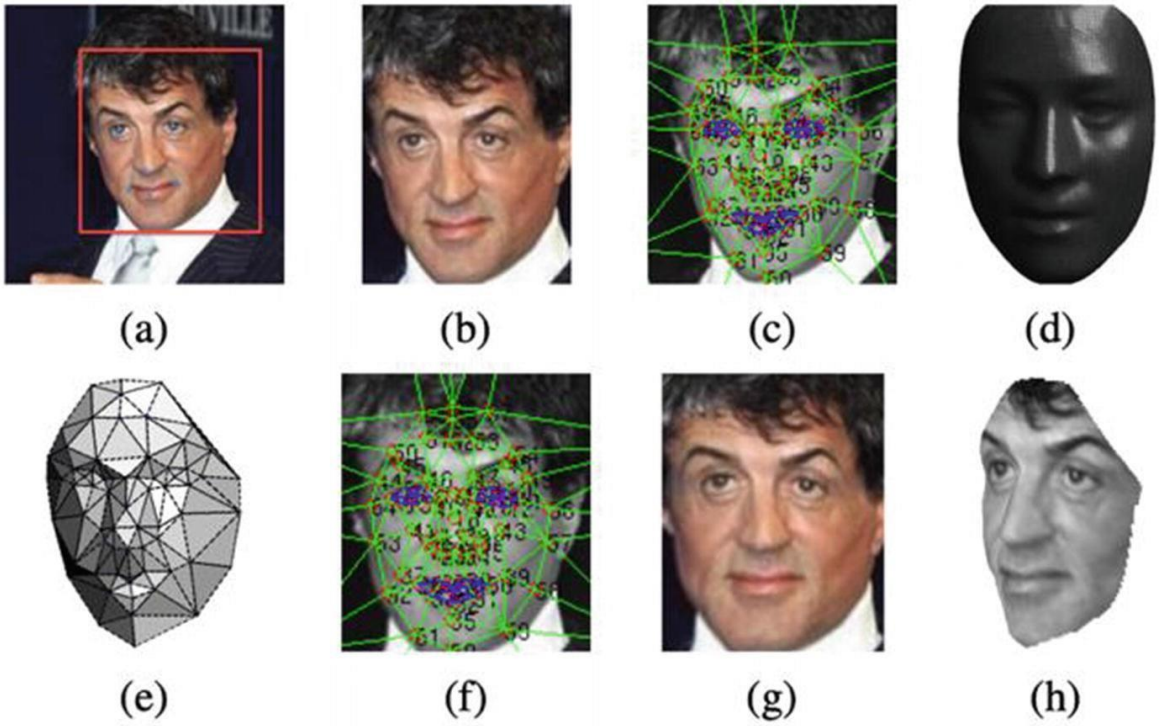


Figure 1. **Alignment pipeline.** (a) The detected face, with 6 initial fiducial points. (b) The induced 2D-aligned crop. (c) 67 fiducial points on the 2D-aligned crop with their corresponding Delaunay triangulation, we added triangles on the contour to avoid discontinuities. (d) The reference 3D shape transformed to the 2D-aligned crop image-plane. (e) Triangle visibility w.r.t. to the fitted 3D-2D camera; darker triangles are less visible. (f) The 67 fiducial points induced by the 3D model that are used to direct the piece-wise affine warpping. (g) The final frontalized crop. (h) A new view generated by the 3D model (not used in this paper).

Figure 6-4 Face alignment process used in DeepFace. The image has been taken from the original paper. We should note how a face is progressively analyzed in steps

In the second step, as shown in step (b), we crop and generate the 2D face from the original image. We should note how the face has been cropped from the original image in this step.

In the next steps, triangles are added on the contours to avoid discontinuities.

We apply 67 fiducial points on the 2D-aligned crop with their corresponding Delaunay Triangulation. A 3D model is generated using a 2D to 3D generator, and the 67 points are plotted. It also allows us to align the out-of-plane rotation. Step (e) shows the visibility with respect to the fitted 2D-3D camera, and in step (f) we can observe the 67 fiducial points induced by the 3D model which are used to direct the piecewise affine wrapping. We will discuss the Delaunay Triangulation briefly now. For a given set “P” of discrete points in a plane, in triangulation DT no point is inside the circumcircle of any triangle in Delaunay Triangulation. Hence, it maximizes the

minimum angles of all the triangles in the triangulation. We are showing the phenomenon in Figure 6-5.

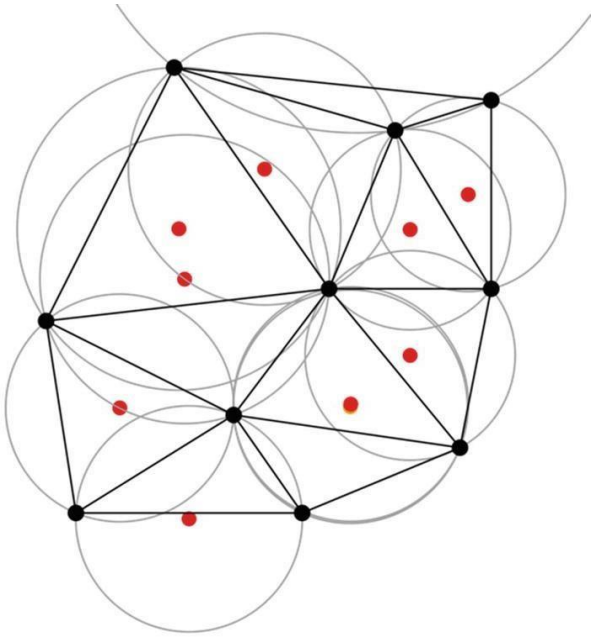


Figure 6-5 Delaunay Triangulation. Image source: https://commons.wikimedia.org/w/index.php?curid=18929097

Finally, we do a final frontalized crop. It is the final step which achieves the objective of 3D frontalization.

Once the step of 3D frontalization is done, then the image is ready to be fed to the next steps in the network. An image of 152x152 size is the input image which is fed to the next layer.

The next layer is the convolutional layer (C1) with 32 filters of size 11x11x3 followed by a 3x3 max pooling layer with a stride of 2. Then the next layer is another convolution with 16 filters and size 9x9x16.

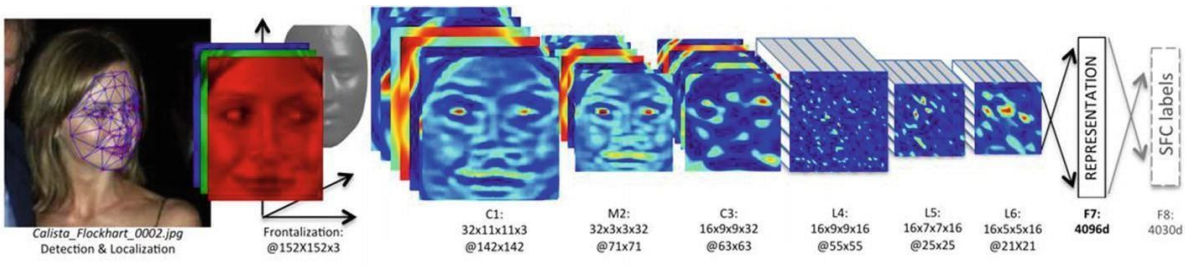


Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate feature maps produced at each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

Figure 6-6 Complete DeepFace architecture is shown here. The image is from the original paper. We can observe that after frontalization has been done, the convolutional process is the next one

And then we have three locally connected layers. We will discuss locally connected layers briefly as they are a bit different from fully connected layers.

Locally connected behave differently from fully connected layers. For a fully connected layer, each neuron of the first layer is connected to the next layer. For locally connected layers, we have different types of filter in a different feature map. For example, when we are classifying if the image is of a face, we can search for the mouth only at the bottom of the image. So locally connected layers are handy if we know that a feature should be restricted within a small space and there is no need to search for that feature across the entire image.

In DeepFace, we have locally connected layers, and hence we can improve the model as we can differentiate between facial regions based on different types of feature maps.

The next to last layer is a fully connected layer which is used for face representation. The final layer is a softmax fully connected layer to do the classification.

The total number of parameters is 120 million. Dropout is being used as a regularization technique but is done only for the final fully connected layers. We also normalize the features between 0 and 1 and do an L2 normalization. The network generates quite sparse feature maps during the training, primarily since ReLU has been used as the activation function.

The validation was done on the LFW (Labeled Faces in the Wild) dataset and SFC dataset. LFW contains more than 13,000 web images of more than 5700 celebrities. SFC is the dataset by Facebook itself having ~4.4 million images of 4030 people each having 800 to 1200 facial images. The ROC curves for both the datasets are shown in Figure 6-7.

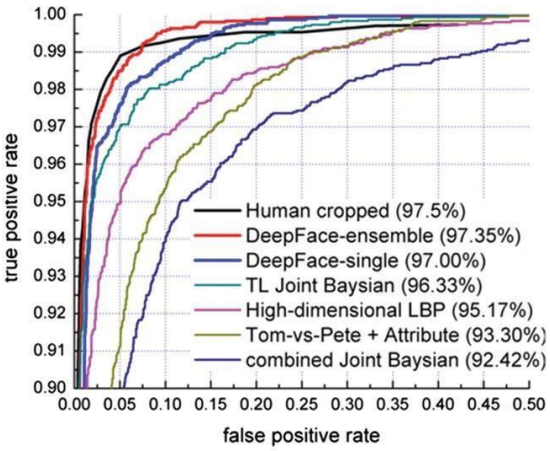


Figure 3. The ROC curves on the *LFW* dataset. Best viewed in color.

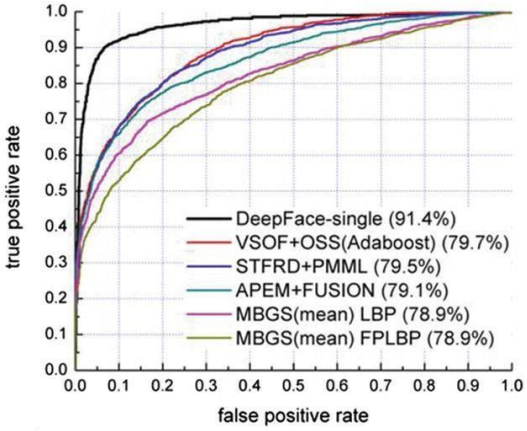


Figure 4. The ROC curves on the *YTF* dataset. Best viewed in color.

Figure 6-7 ROC curve of LFW dataset and YTF dataset taken from the original paper at https://www.cs.toronto.edu/~ranzato/publications/taigman_cvpr14.pdf

DeepFace is one of the novel face recognition modes. It has more than 99.5% accuracy on the LFW dataset. It is able to resolve issues with pose, expression, or light intensity in the background. 3D alignment is quite a unique methodology which further enhances accuracy. The architecture has performed really well on LFW and YouTube Faces dataset (YTF).

4.6. FaceNet for face recognition

FaceNet was proposed by Google researchers Florian Schroff, Dmitry Kalenichenko, and James Philbin in 2015. The original paper is "FaceNet: A Unified Embedding for Face Recognition and Clustering" and can be accessed at https://arxiv.org/abs/1503.03832.

FaceNet does not recommend a completely new set of algorithms or complex mathematical calculations to perform the face recognition tasks. The concept is rather simple.

All the images of faces are first represented in a Euclidean space. And then we calculate the similarity between faces by calculating the respective distances.

Consider this, if we have an image, $Image_1$ of Mr. X, then all the images or faces of Mr. X will be closer to $Image_1$ rather than $Image_2$ of Mr. Y. The concept is shown in Figure 6-8.

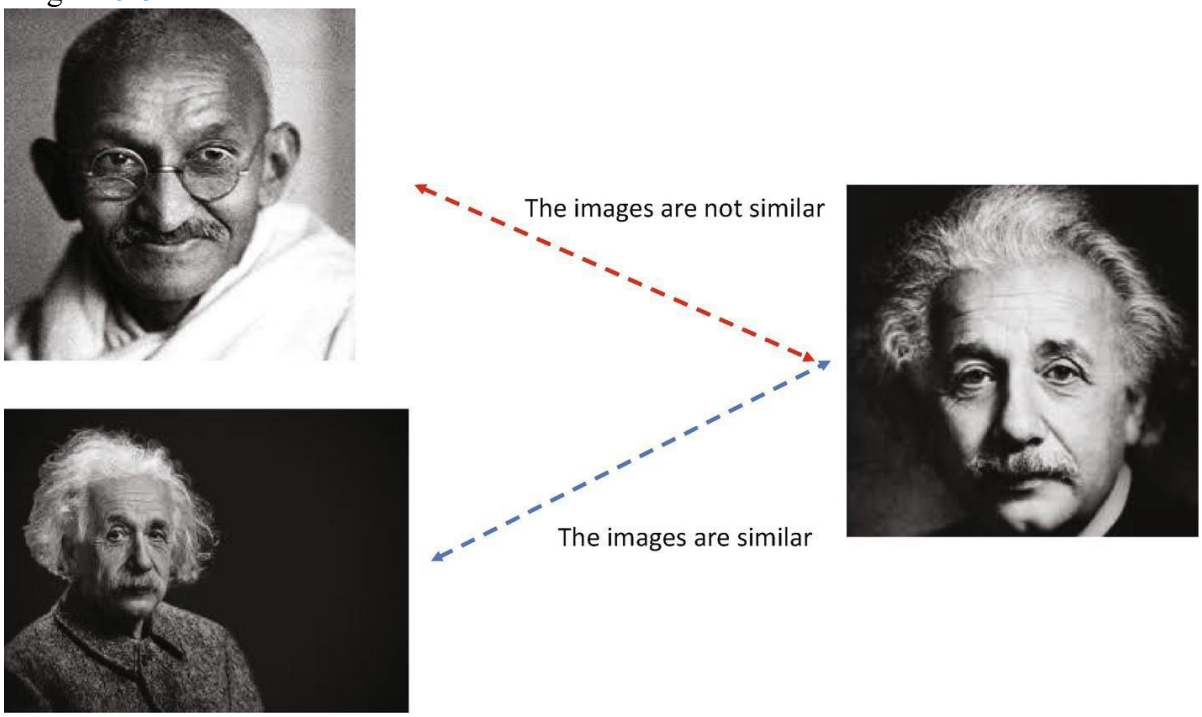


Figure 6-8 The images of Einstein will be similar to each other, and hence the distance between them will be less, while the image of Gandhi will be at a distance

The preceding concept is simpler to understand. We will understand the architecture in detail now. As shown in Figure 6-9, we can examine the complete architecture. The image has been taken from the original paper itself.

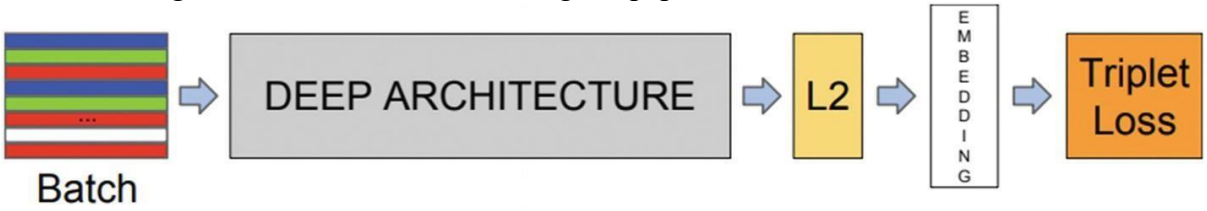


Figure 6-9 FaceNet architecture. The image has been taken from the original paper at https://arxiv.org/abs/1503.03832

The network starts with a batch input layer of the images. And then it is followed by a deep CNN architecture. The network utilizes an architecture like ZFNet or Inception network.

FaceNet implements 1x1 convolutions to decrease the number of parameters. The output of these Deep Learning models is an embedding of images. L2 normalization is performed on the output. These embeddings are quite a useful addition. FaceNet understands the respective mappings from the facial images and then creates embeddings.

Once the embeddings are successfully done, we can simply go ahead, and with the help of newly created embeddings as the feature vector, we can use any standard machine learning technique. The usage of embeddings is the prime difference between FaceNet and other methodologies as other solutions generally implement customized layers for facial verifications.

The created embeddings are then fed to calculate the loss. The images are represented in a Euclidean space. The loss function aims to make the squared distance between two image embeddings of similar images small, whereas the squared distance between different images is large. In other words, the squared distance between the respective embeddings will decide the similarity between the faces.

There is one vital concept implemented in FaceNet – *triplet loss* function.

Triplet loss is shown in Figure 6-10; the image has been taken from the original paper itself.

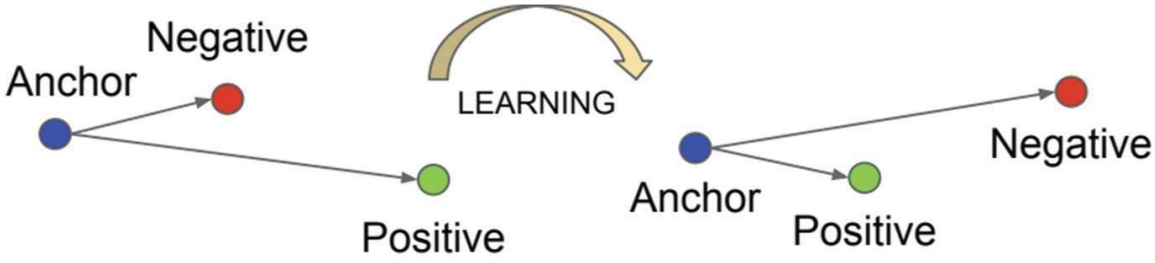


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

Figure 6-10 Triplet loss used in FaceNet. The image has been taken from the original paper at https://arxiv.org/abs/1503.03832

Triplet loss works on the concept we discussed in Figure 6-8 at the start of the FaceNet discussion. The intuition is that we want the images of the same person Mr. X closer to each other. Let us call that $Image_1$ as the anchor image. All the other

images of Mr. X are called the positive images. The images of Mr. Y are referred to as negative images.

Hence, as per triplet loss, we want the distance between the embeddings of the anchor image and positive image to be less as compared to the distance between embeddings of the anchor image and negative image. We would want to achieve Equation 6-1:

$$\left\|x_i^a - x_i^p\right\|_2^2 + \alpha < \left\|x_i^a - x_i^n\right\|_2^2, \forall \left(x_i^a, x_i^p, x_i^n\right) \in \tau.(1) \qquad \text{(Equation 6-1)}$$

where

Anchor image is x_i^a.

Positive image is x_i^p.

Negative image is x_i^n, so basically xi is an image.

α is a margin that is enforced between positive and negative pairs. It is the threshold we set, and it signifies the difference between the respective pairs of images.

T is the set of all the possible triplets in the training set and has cardinality N.

Mathematically, triplet loss can be represented as in Equation 6-2. It is the loss which we wish to minimize.

$$\sum_i^N \left[\left\|f\left(x_i^a\right) - f\left(x_i^p\right)\right\|_2^2 - \left\|f\left(x_i^a\right) - f\left(x_i^n\right)\right\|_2^2 + \alpha\right]_+ .(2)$$

(Equation 6-2)

In the preceding equations, the embedding of an image is represented by f(x) such that x ∈ℝ. It embeds an image x into a d-dimensional Euclidean space. f(xi) is the embedding of an image which is in the form of a vector of size 128.

The solution is dependent on the selection of the image pairs. There can be image pairs which the network will be able to pass. In other words, they will satisfy the condition of the loss. These image pairs might not add much to the learning and might also result in slow convergence.

For better results and faster convergence, we should select the triplets which do violate the condition in Equation 6-1.

The solution is dependent on the selection of the image pairs. There can be image pairs which the network will be able to pass. In other words, they will satisfy the condition of the loss. These image pairs might not add much to the learning and might also result in slow convergence.

For better results and faster convergence, we should select the triplets which do violate the condition in Equation 6-1.

Mathematically, for an anchor image x_i^a, we would want to select a positive image x_i^p such that the similarity is maximum and select a negative image x_i^n such that the similarity is minimum. In other words, we wish to have $\text{argmax}_{xip} \| f(x_i^a) - f(x_i^p) \|_2^2$ which means given an anchor image x_i^a we wish to have a positive image x_i^p such that the distance is maximum.

Similarly, for a given anchor image x_i^a, we wish to get a negative image x_i^n such that the distance is minimum which is represented as $\text{argmin}_{xin} \| f(x_i^a) - f(x_i^n) \|_2^2$.

Now, making this choice is not an easy task. During training, it is ensured that positive and negative are chosen as per the maximum and minimum functions given earlier on the mini-batch. SGD (Stochastic Gradient Descent) with Adagrad is used for training. The two networks which have been used are shown in the following (ZF-Net and Inception). There are 140 million parameters in ZF-Net and 7.5 million parameters for Inception.

layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

Table 1. **NN1.** This table show the structure of our Zeiler&Fergus [22] based model with 1×1 convolutions inspired by [9]. The input and output sizes are described in $rows \times cols \times \#filters$. The kernel is specified as $rows \times cols, stride$ and the maxout [6] pooling size as $p = 2$.

type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

Table 2. **NN2.** Details of the NN2 Inception incarnation. This model is almost identical to the one described in [16]. The two major differences are the use of L_2 pooling instead of max pooling (m), where specified. The pooling is always 3×3 (aside from the final average pooling) and in parallel to the convolutional modules inside each Inception module. If there is a dimensionality reduction after the pooling it is denoted with p. 1×1, 3×3, and 5×5 pooling are then concatenated to get the final output.

The model performed very well with 95.12% accuracy with standard error of 0.39 using the first 100 frames.

On the LFW dataset, quote from the paper:

Our model is evaluated in two modes: 1. Fixed center crop of the LFW provided thumbnail. 2. A proprietary face detector (similar to Picasa [3]) is run on the provided LFW thumbnails. If it fails to align the face (this happens for two images), the LFW alignment is used.

We achieve a classification accuracy of 98.87%±0.15 when using the fixed center crop described in (1) and the record breaking 99.63%±0.09 standard error of the mean when using the extra face alignment (2). FaceNet is a novel solution as it directly learns an embedding into the Euclidean space for face verification. The model is robust enough to be not affected by the pose, lighting, occlusion, or age of the faces.

4.7. Python implementation using FaceNet

The code in this section is quite self-explanatory. We are using a pre-trained FaceNet model with its weights and calculating the Euclidean distance to measure the similarity between two faces. We are using the publicly available facenet_weights from https://drive.google.com/file/d/1971Xk5RwedbudGgTIrGAL4F7Aifu7id1/view by Sefik Ilkin Serengil. The model was converted from Tensorflow to Keras. The base model can be found at https://github.com/davidsandberg/facenet. Step 1: Load the libraries.

```
from keras.models import model_from_json
from inception_resnet_v1 import * import numpy as np

from keras.models import Sequential from keras.models import load_model
from keras.models import model_from_json from keras.layers.core import Dense, Activation from keras.utils import np_utils
```

```
from keras.preprocessing.image import load_img, save_img, img_to_array from keras.applications.imagenet_utils import preprocess_input

import matplotlib.pyplot as plt from keras.preprocessing import image
```

Step 2: Load the model now.

```
face_model = InceptionResNetV1() face_model.load_weights('facenet_weights.h5')
```

Step 3: We will now define three functions – to normalize the dataset, to calculate the Euclidean distance, and to preprocess the dataset.

```
def normalize(x):
    return x / np.sqrt(np.sum(np.multiply(x, x)))

def getEuclideanDistance(source, validate): euclidean_dist = source - validate euclidean_dist = np.sum(np.multiply(euclidean_dist, euclidean_dist)) euclidean_dist = np.sqrt(euclidean_dist) return euclidean_dist

def preprocess_data(image_path):
    image = load_img(image_path, target_size=(160, 160)) image = img_to_array(image) image = np.expand_dims(image, axis=0) image = preprocess_input(image) return image
```

Step 4: Now we will calculate the similarity between the two images.

Here, we have taken these two images of a famous cricket celebrity – Sachin Tendulkar. These two images have been taken from the Internet.

```
img1_representation = normalize(face_model.predict(preprocess_data('image_1.jpeg'))[0,:]) img2_representation = normalize(face_model.predict(preprocess_data('image_2.jpeg'))[0,:])

euclidean_distance = getEuclideanDistance(img1_representation, img2_representation)
```

The Euclidean distance similarity is 0.70. We can also implement a cosine similarity to test the similarity between two images.

4.8. Python solution for gesture recognition

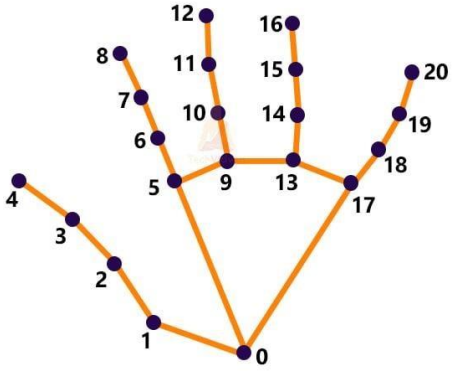
Gesture recognition is one of the most innovative solutions which is helping humans talk to the system. Gesture recognition means that hand or face gestures can be captured by the system and a corresponding action can be taken by the system. It consists of detection, tracking, and recognition as the key components.

1. In detection, the visual part like the hand or a finger or a body part is extracted. The visual part should be within the view of the camera.
2. Then we track the visual part. It ensures that data is captured and analyzed frame by frame.
3. And finally, we recognize the gesture or a group of gestures. Based on the algorithm settings we have done, the training data used, the system will be able to identify the type of gesture that has been made.

Gesture recognition is quite a path-breaking solution and can be used in automation, medical devices, augmented reality, virtual reality, gaming, and so on. The use cases are many, and a lot of research is currently being done in this field.

MediaPipe is a customizable machine learning solutions framework developed by Google. It is an open-source and cross-platform framework, and it is very lightweight. MediaPipe comes with some pre-trained ML solutions such as face detection, pose estimation, hand recognition, object detection, etc.

MediaPipe is used to recognize the hand and the hand key points. MediaPipe returns a total of 21 key points for each detected hand.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Steps to solve the project:

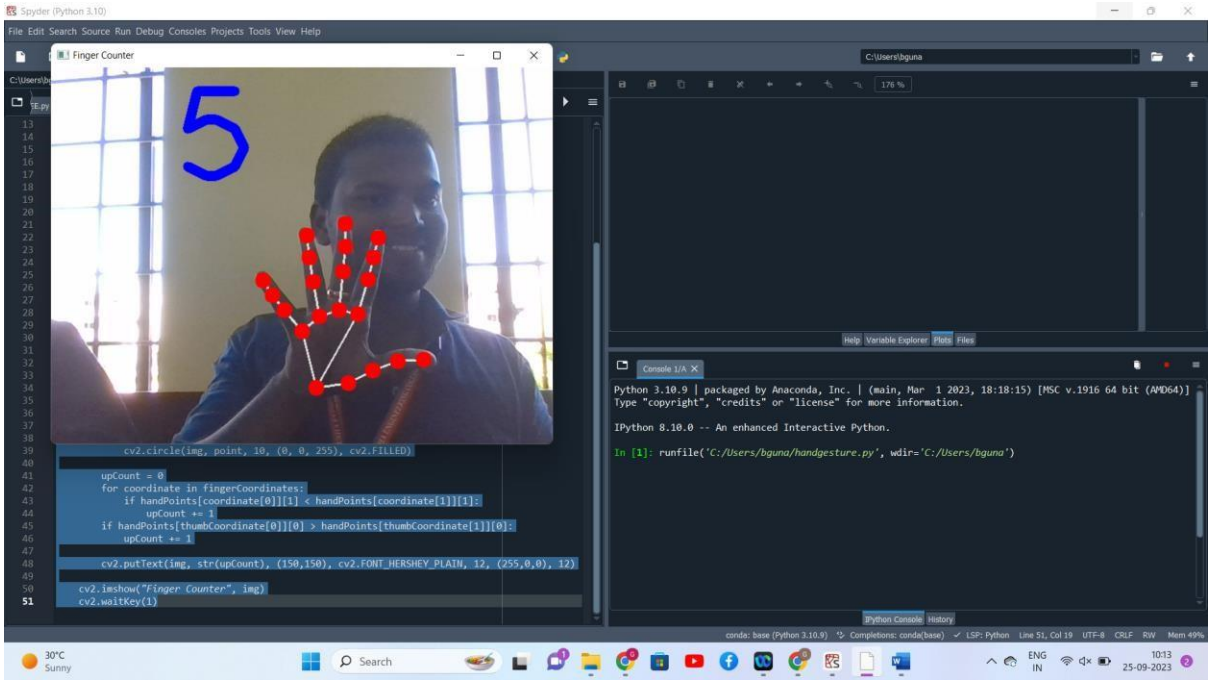
- Import necessary packages.
- Initialize models.
- Read frames from a webcam.
- Detect hand keypoints.
- Recognize hand gestures.

Implementation of a Finger Counting solution using OpenCV.

```
import cv2
import mediapipe as mp

cap =
cv2.VideoCap
ture(0)
mpHands =
mp.solutions.
hands hands
=
mpHands.Ha
nds()
mpDraw =
mp.solutions.
drawing_utils
    fingerCoordinates = [(8, 6), (12, 10), (16, 14), (20, 18)]
    thumbCoordinate = (4,2)


    while True:
        success, img = cap.read()
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


        results = hands.process(imgRGB)
        multiLandMarks = results.multi_hand_landmarks

        if
multiLandMa
rks:
handPoints =
[]      for
handLms in
multiLandMa
rks:
            mpDraw.draw_landmarks(img, handLms,
mpHands.HAND_CONNECTIONS)

            for idx, lm in enumerate(handLms.landmark):

#
print
(idx,
lm)
```

```
h, w,
c =
img.
shap
e
                cx, cy = int(lm.x * w), int(lm.y
* h)             handPoints.append((cx, cy))

          for point in handPoints:
cv2.circle(img, point, 10, (0, 0,
255), cv2.FILLED)

          upCount = 0        for coordinate in
fingerCoordinates:           if
handPoints[coordinate[0]][1] <
handPoints[coordinate[1]][1]:
                upCount += 1                          if
handPoints[thumbCoordinate[0]][0]                     >
handPoints[thumbCoordinate[1]][0]:
             upCount += 1

          cv2.putText(img, str(upCount), (150,150), cv2.FONT_HERSHEY_PLAIN, 12,
(255,0,0), 12)

        cv2.imshow("Finger
Counter",                  img)
cv2.waitKey(1)
```

output:



Part-A

Questions:

1. What is face detection?

Face detection, also called facial detection, is an artificial intelligence (AI)-based computer technology used to find and identify human faces in digital images and video.

2. What is face recognition?

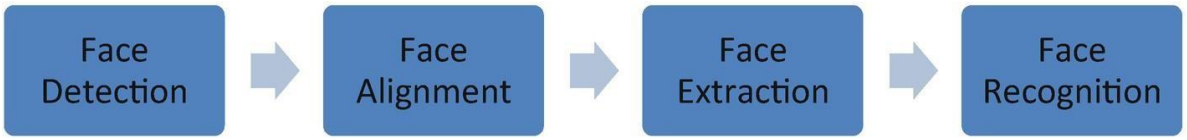
Facial recognition is a way of identifying or confirming an individual's identity using their face. Facial recognition systems can be used to identify people in photos, videos, or in real-time.

3. List the deep learning algorithms for face recognision. DeepFace, VGGFace

D
e
e
p
I
D

F
a
c
e
N
e
t

4. What are the various processes in face recognition systems?



5. What is the concept of facial alignment?

Face alignment is a computer vision technology for identifying the geometric structure of human faces in digit images. Given the location and size of a face, it automatically determines the shape of the face components suc as eyes and nose. A face alignment program typically operates by iteratively adjusting a deformable models, which encodes the prior knowledge of face shape or appearance, to take into account the low-level image evidences and find the face that is present in the image.

6. What is the concept of triplet loss?

The triplet loss is a distance-based loss function that aims to learn embeddings that are closer for similar input data and farther for dissimilar ones. First, we have to compute triplets of data that consist of the following:

- an anchor input sample
- a positive example that has the same label with
- and a negative example that has different label with (and of course)

Then, the goal of the loss function is to learn embeddings such that the distance between the anchor and the positive example is smaller than the distance between the anchor and the negative example. More formally, this is defined as follows:

$$L_{triplet} = max(0, D(a,p) - D(a,n) + \epsilon)$$

where is a distance metric (usually euclidean distance) and is a hyperparameter that controls the minimum distance. When $D(a,n) - D(a,p) < \epsilon$, then $L_{triplet} = 0$ since the restriction of the margin is not violated.

7. What are the various use cases of gesture recognition?

Gesture recognition can be used to control devices or interfaces, such as a computer or a smartphone, through movements or actions, such as hand or body movements, facial expressions or even voice commands.

Gesture recognition has a variety of uses, including:

- **Human-computer interaction:** Gesture recognition can be used to control computers, smartphones, and other devices through gestures, such as swiping, tapping, and pinching.
- **Gaming:** Gesture recognition can be used to control characters and objects in video games, making the gaming experience more immersive and interactive.
- **Virtual and augmented reality:** Gesture recognition can be used to interact with virtual and augmented reality environments, allowing users to control and manipulate objects in those environments.
- **Robotics:** Gesture recognition can be used to control robots, allowing them to perform tasks based on the user's gestures.
- **Sign language recognition:** Gesture recognition can be used to recognize and translate sign language into spoken or written language, helping people who are deaf or hard of hearing communicate with others.
- **Automotive:** Gesture recognition can be used in cars to control various functions such as radio, AC, and navigation systems.
- **Healthcare:** Gesture recognition can be used in rehabilitation of patients with physical disabilities.

8. What is fiducial points?

Fiducial marker or fiducial is an object placed in the field of view of an imaging system that appears in the image produced, for use as a point of reference or a measure.

Part – B

1. Explain about Face Recognition.
2. Elaborate the applications of Face Recognition
3. Illustrate various Process of Face Recognition.
4. Explain about DeepFace solution by Facebook
5. Elaborate about -FaceNet for Face Recognition 6. Explain the implementation of FaceNet Algorithm.

7. Explain about Gesture Recognition.

UNIT V VIDEO ANALYTICS

Video Processing – use cases of video analytics-Vanishing Gradient and exploding gradient problem - ResNet architecture-ResNet and skip connections-Inception NetworkGoogleNet architecture Improvement in Inception v2-Video analytics-ResNet and Inception v3.

Video Analytics Using Deep Learning

Videos are a really powerful medium. Roughly 500 hours of videos are uploaded to YouTube every minute. And the number of videos created is increasing daily. With the advent of smartphones and improved hardware, the video quality is enhanced. More videos are being created and stored across domains and geographies. We have movies, advertisements, short clips, and personal videos. Most of the videos contain human faces, objects, and some movements of objects. A video might be shot during the daytime or in the night, under different lighting conditions. We have cameras capturing the movement of pedestrians on the road, manufacturing cameras for monitoring goods and products on manufacturing lines, security cameras for surveillance on airports, and number plate detection and reading systems in the parking lots, to name a few of the video analytics solutions.

Video is a sequence of images. ResNet and Inception network architectures are advanced networks and most sought after for a lot of cutting-edge Deep Learning solutions.

5.1.Video processing

Videos are not new to us. We record videos using our phone, laptops, hand cameras, and so on. YouTube is one of the biggest sources of videos. Advertisements, movies, sports, social media uploads, TikTok videos, and so on are getting created every second. And by analyzing them, we can uncover a lot of insights about the behavior, interactions, timings, and sequence of things. A very powerful medium indeed!

There can be multiple approaches to design a video analytics solution. We can consider video as a collection of frames and then perform analytics by treating the frames as individual images. Or we can also add an additional dimension of sound to it.

5.2.Use cases of video analytics

Videos are a rich source of knowledge and information. We can utilize Deep Learning– based capabilities across domains and business functions. Some of them are listed as follows:

1. Real-time face detection can be done using video analytics, allowing us to detect and recognize the faces. It has huge benefits and applications across multiple domains. We have discussed the application in detail in the last chapter.
2. In disaster management, video analytics can play a significant role. Consider this. In a flood-like situation, by analyzing videos of the actual area, the rescue team can identify the zones they should concentrate on. It will help reduce the time to action which directly leads to more lives saved.
3. For crowd management, video analytics plays an important role. We can identify the concentration of the population and the eminent dangers in that situation. The respective team can analyze the videos or a real-time streaming of video using cameras. And a suitable action can be taken to prevent any mishappening.
4. By analyzing the social media videos, the marketing teams can improve the contents. The marketing teams can even analyze the contents of the competitors and tweak their business plan accordingly as per the business needs.
5. For object detection and object tracking, video analytics can quickly come up with a decision if an object is present in the video or not. This can save manual efforts. For example, if we have a collection of videos of different cars and we wish to classify them in different brands, the manual process will be to open each and every video and then take a decision – which is both time-consuming and error-prone. Using Deep Learning–based video classification, this entire process can be automated.
6. Video analytics can help in the inspection and quality assurance. Instead of manual inspection of each part present in a machine, a video can be taken for the entire process. And then using Deep Learning, the quality inspection can be conducted. These are not the only use cases. There are a number of applications across domains and sectors. With Deep Learning–based solutions, video analytics is really making an impact into the business world.

5.3. Vanishing gradient and exploding gradient problem

The Neural Networks are trained using backpropagation and gradient-based learning methods. During training, we want to reach the most optimum value of weights resulting in minimum loss. Now, each of the weights constantly gets updated during the training of the algorithm. The update is proportional to the partial derivative of the error function with respect to the current weight in each training iteration.

In Figure 7-1, we are showing that in the sigmoid function(activation function), we can face the problem of vanishing gradient, while in the case of a ReLU (Rectified Linear Unit) or Leaky ReLU, we will not have vanishing gradient as an issue.

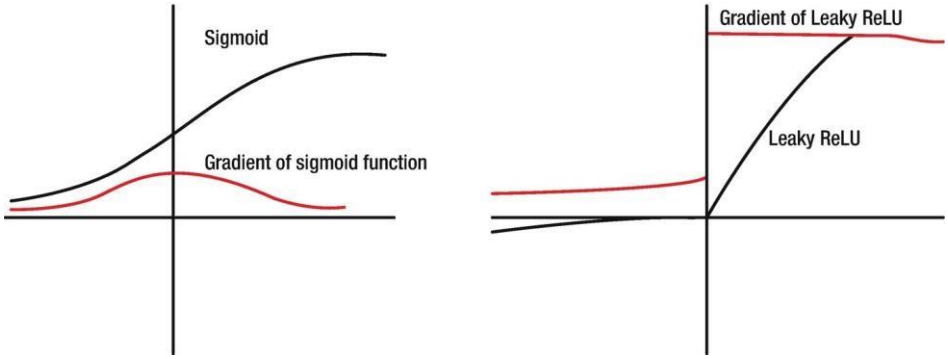


Figure 7-1 Vanishing gradient is a challenge we face with deep Neural Networks. The figure on the left shows that for the sigmoid activation function, we do face a big problem which gets sorted for Leaky ReLU

The challenge can be sometimes this update becomes too small, and hence the weight does not get updated. It results in very less or practically no training of the network. This is referred to as the vanishing gradient problem.

Let's understand the problem in depth now. We are again looking at the basic network architecture in Figure 7-2.

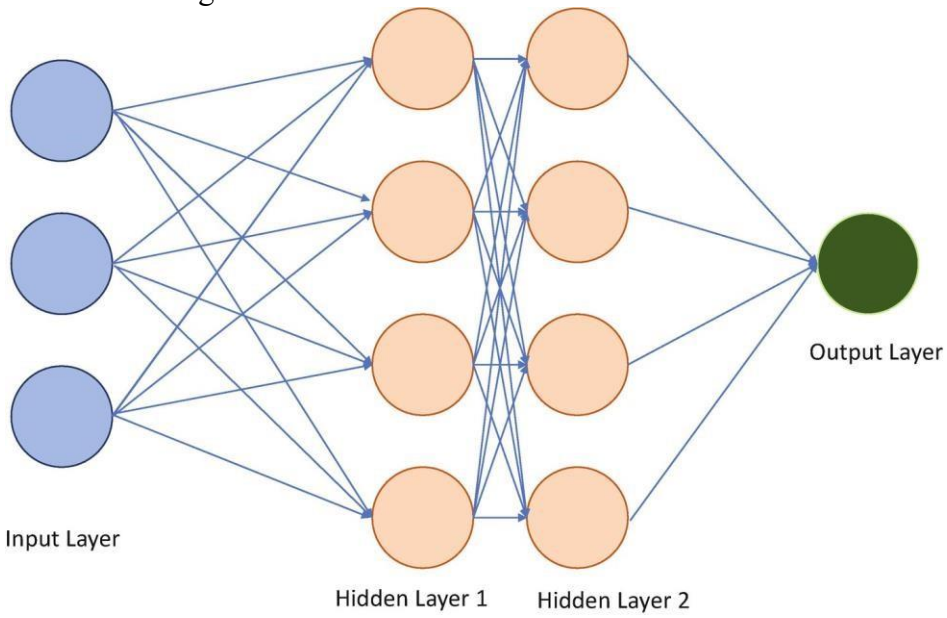


Figure 7-2 Basic neural architecture having an input layer, hidden layers, and an output layer

We know that each of the neurons in the network has an activation function and a bias term. It accepts a finite number of input weight products, adds a bias term to it, and then the activation function is applied on it. The output is then passed to the next neuron.

We also know that in a network, the difference between the expected output and the predicted value is calculated which is nothing but the error term. We would want the error term to be minimized. The error will be minimized when we have achieved the best combination of weights and biases across the layers and neurons which minimizes the error.

When the error is calculated, a gradient descent is applied on the graph of the error function. This gradient descent is the differentiation of the error function with respect to each of the independent variables (weights and biases) present in it. This is the job of the backpropagation algorithm – which takes care of manipulating these weights and biases by a constant term called the learning rate. This is done from the last layer to the first layer in the backward direction or from the right to the left. In each successive iteration, gradient descent is calculated and the direction of change is determined. And hence, the weights and biases are updated till the network minimizes the error – or, in other words, till the error reaches a *global minima* as shown in Figure 7-3. The error gradient hence is the direction and magnitude calculated during the training of the network. It is used to update the weights in the right direction and right magnitude.

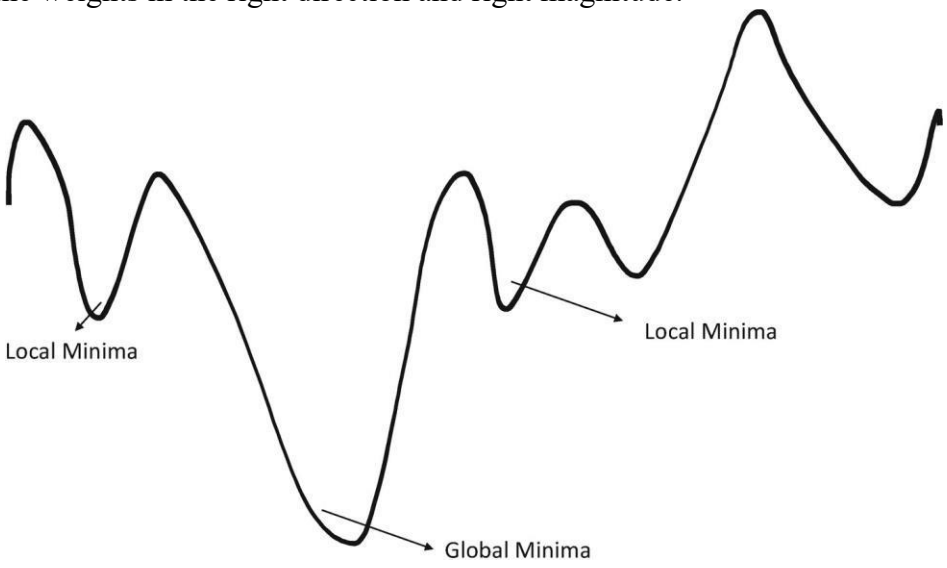


Figure 7-3 To minimize the loss, we would want to reach the global minima of a function. Sometimes, we might not be able to minimize the loss and can be stuck at the local minima

Now a situation arises wherein if we have a very deep network, the initial layers have a very less impact on the final output as compared to the final layers of the network. Or in other words, the initial layers undergo very less training, and their values undergo very less amount of change. This is due to the fact that the backpropagation computes the gradients using a chain rule from the final layers to the initial layers.

Hence, in an n-layered network, the gradient decreases exponentially with the value of n, and hence the initial layers will train very slowly. Or, in the worst-case scenario, they will stop to train.

There can be multiple signs to check for the vanishing gradient problem:

1. The easiest way to detect vanishing gradient is through kernel weight distribution. If the weights are dying to zero or very very close to zero, we might be encountering a vanishing gradient problem.
2. The model's weight close to the final layers will have more change as compared to the initial layers.

3. The model will not improve or will improve very slowly during the training phase.
4. Sometimes, the training stops early. It means that any further training does not improve the model.

(https://www.youtube.com/watch?v=FbxTVRfQFuI , https://www.youtube.com/watch?v=qowp6SQ9_Oo)

There are a few suggested solutions for the vanishing gradient problem:

1. Generally, reducing the number of layers in the network might help in resolving gradient problems. But at the same time, if the number of layers is reduced, the network's complexity goes down, and it can also impact the performance of the network.
2. The ReLU activation function resolves the vanishing gradient problem. ReLU suffers less from vanishing gradients as compared to tanh or sigmoid activation functions.
3. Residual networks or ResNets are also one of the solutions for this problem. They do not resolve the problem by saving the gradient flow; instead, they use a combination or ensemble of multiple smaller networks. And hence, ResNets despite being deep networks are able to achieve lesser loss as compared to shallow networks.

On one hand, we have a vanishing gradient problem, while on the other hand, we have an exploding gradient problem.

In deep networks, error gradients sometimes become very large as they get accumulated. Hence, the updates in the networks will be very large which make the network unstable. There are a few signs of exploding gradients which can help us in detecting exploding gradient:

1. The model is suffering from poor loss during the training phase.
2. During the training of the algorithm, we might encounter NaN for the loss or for the weights.
3. The model is generally unstable, or in other words the updates to loss in subsequent iterations are huge indicating an unstable state.
4. The error gradients are constantly above 1 for each of the layers and neurons in the network.

Exploding gradient can be resolved using

1. We can reduce the number of layers in the network or can try reducing the batch size during training.
2. L1 and L2 weight regularization can be added which will act as a penalty to the network loss functions.
3. *Gradient clipping* is one of the methods which can be used. We can limit the size of the gradients during the process of training. We set a threshold for the error gradients, and the error gradients are set to that limit or *clipped* if the error gradient exceeds the threshold.
4. We can use LSTM (long short-term memory) if we are working with Recurrent Neural Networks. This concept is beyond the scope of this book.

Both vanishing and exploding gradients are a nuisance which will impact the performance of the network. They can make the network unstable and require correction using a few of the options mentioned earlier.

5.4. ResNet architecture

Lot of architectures are used them for image classification, object detection, face recognition, and so on. They are deep Neural Networks and generating good results for us. But in very deep networks, we encounter a problem of vanishing gradients. Residual networks or *ResNets* solve this problem by using skip connections. ResNets were invented by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, and the paper was presented in Dec 2015. More details can be found at https://arxiv.org/pdf/1512.03385.pdf.

Skip connections take the activation from one layer to a much deeper layer in the network which allows us to train even more deep networks, which may be beyond 100 layers. Now, we will discuss ResNet and skip connection in detail in the next section.

5.4.1. ResNet and skip connection

When we talk about Neural Networks and the fantastic performance shown by them, immediately it is attributed to the *depth* of the network. It is assumed that the deeper the network is, the better is the accuracy. The initial layers will learn the basic features, and deeper layers will learn more advanced features.

But it was found, by adding a greater number of layers, we are increasing the complexity of the network. In fact for a deeper network (like 56 layers deep), the loss was greater than a network with less (20) layers.

Note Generally, models using convolutional and fully connected layers between 16 and 30 give the best results for CNN.

This loss can be attributed to the problem of vanishing gradients. To resolve the problem of vanishing gradient, residual blocks are introduced as shown in Figure 7-4. Residual blocks implement *skip connection* or identity mapping.

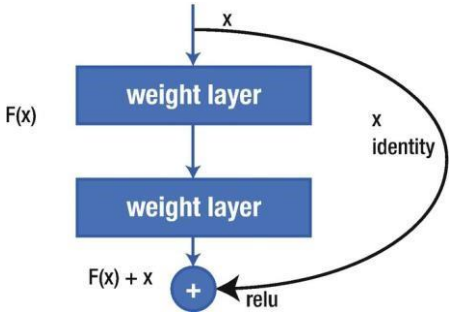


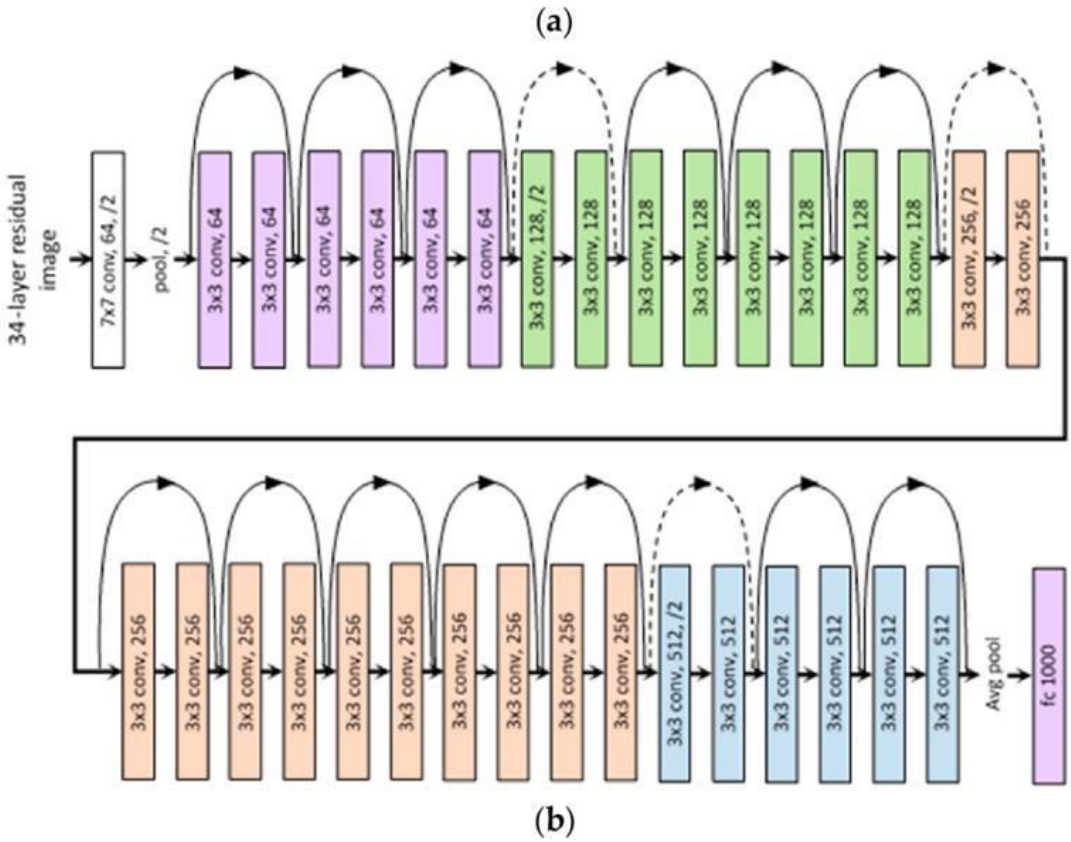
Figure 7-4 Skip connection is the heart of residual networks or ResNets. Note how the output from the previous layer is passed to the next layer, thereby skipping

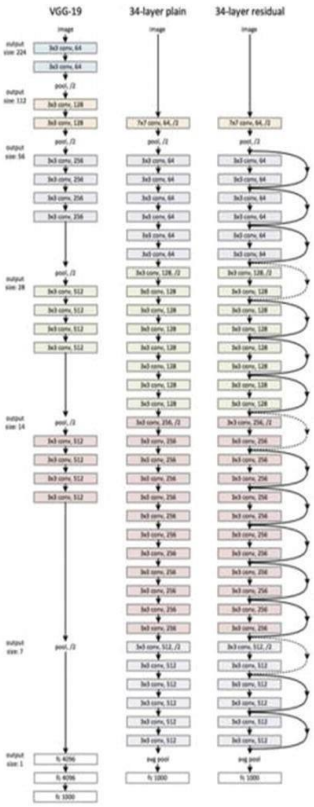
a layer in between. It allows training deeper networks without the problem of vanishing gradients

The intuition behind a network with residual blocks is that each layer is fed to the next layer of the network and also directly to the next layers skipping between a few layers in between. Residual blocks allow you to train much deeper neural networks. The connection(gray arrow) is called **skip connection or shortcut connection** as it is bypassing one or more layers in between. It is also called **identity connection** as we can learn an identity function from it.

This identity mapping has no input parameters of itself; rather, it takes the output from the previous layer and adds to the next layer. In other words, it acts as a shortcut connection before the second activation. Because of this shortcut, it is possible to train even deeper networks without diluting the performance of the network. This is the heart of the solution and the reason for its resounding success.

We are now examining the ResNet-34 architecture in detail in Figure 7-5. The original architecture is taken from the link of the paper: https://arxiv.org/pdf/1512.03385.pdf.





- This is the architecture of ResNet 34
- No fully connected layers and no dropouts were used
- All the convolutions are 3x3
- The dotted lines represent where the dimensions are different.
- To solve the difference in dimensions, the input is down-sample by 2 and then zero padding to match the two dimensions.

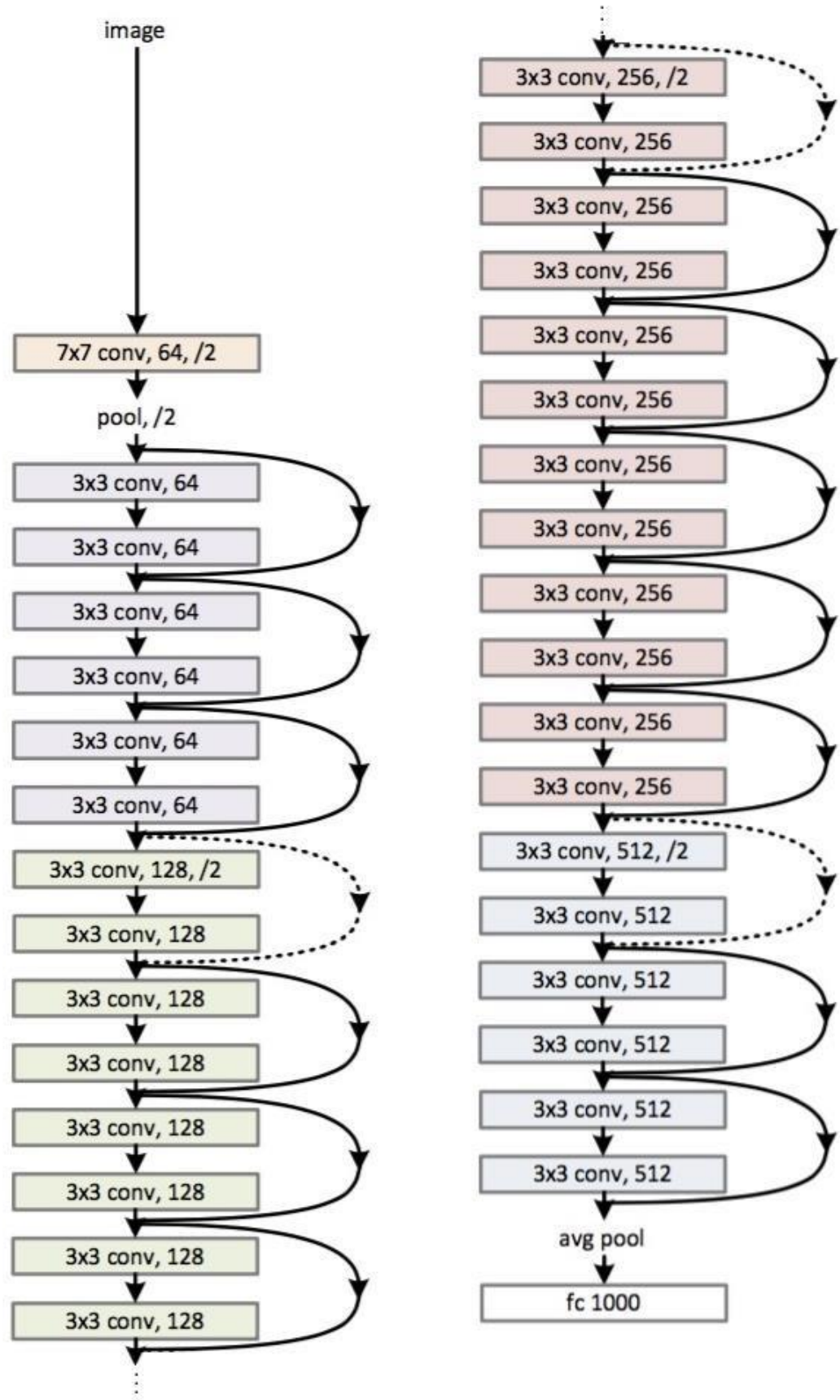

34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000

Figure 7-5 ResNet-34 complete architecture – in the middle is a plain network without skip connections, while on the right a network with residual connections is shown. The architecture has been taken from the original paper at https://arxiv.org/pdf/1512.03385.pdf

Let's go a bit deeper into the network. Observe the four residual blocks in the architecture as shown in Figure 7-6.

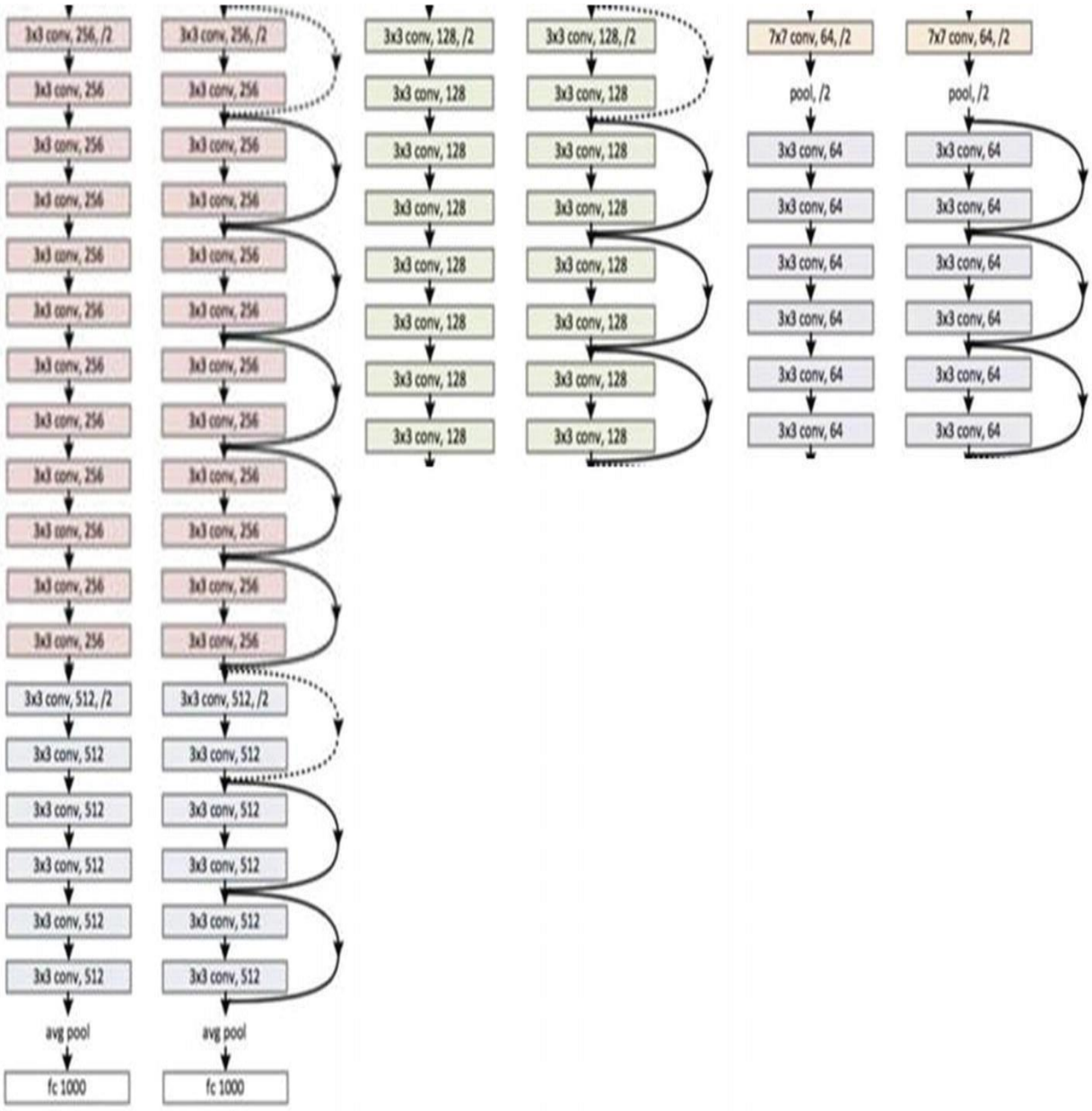


Figure 7-6 Four residual blocks are shown in the figure. Note how for each of the plain networks on the left, we have a corresponding block using skip

connection. Skip connections are allowed to train deeper networks without an adverse impact on the accuracy of the network. For example, for the very first block, we have a plain network having a 7x7 conv layer followed by a 3x3 conv layer. Note the corresponding block using the skip connection

We can analyze that in each residual architecture, skip connection takes the output from the previous layer and shares it two blocks away. This is the core difference with the plain architecture on the left, which boosts the performance for ResNet.

Skip connections extend the capabilities of deep networks in a very interesting manner. The inventors tested the network with 100 and 1000 layers on the CIFAR dataset. The inventors found that using an ensemble of residual networks was able to achieve a 3.57% error rate on ImageNet and hence secured first place in the ILSVRC2015 competition.

There are other variants of ResNet gaining popularity like ResNetXt, DenseNet, and so on. These variants explore the changes which can be made to the original ResNet architecture. For example, ResNetXt introduced *cardinality* as one of hyperparameters for the model. We have listed research papers at the end of the chapter for interested audiences.

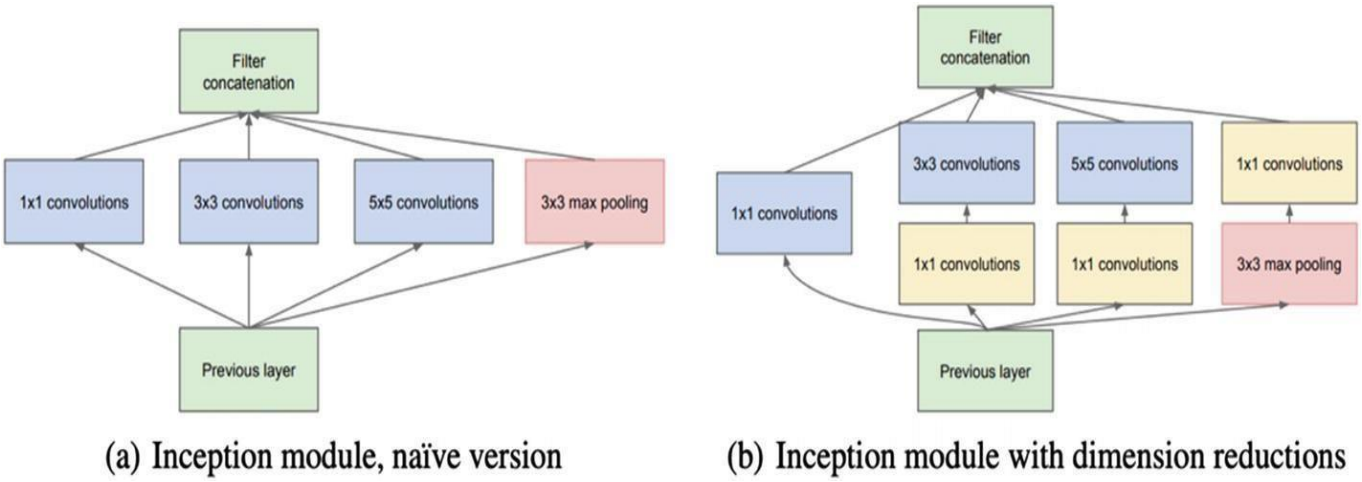
5.4.2. Inception network

Deep Learning is fantastic when it comes to complex tasks. And we have observed that using stacked convolutional layers, we are able to train deep networks. But there are a few challenges with it:

1. Networks become overcomplicated and demand huge computation power.
2. Vanishing and exploding gradient problems are encountered while training the network.
3. Many times, while observing the training and test accuracy, networks overfit and hence are not useful for unseen datasets.
4. Moreover, choosing the best kernel size is a tough decision. A poorly chosen kernel size will lead to ill- fitting results.

To resolve the challenges faced, the researchers thought that why can't we go *wide* rather than going *deep*. More technically, have filters with multiple sizes operate at the same level. And hence Szegedy et al. proposed the *Inception module*. The complete paper can be accessed here: https://arxiv.org/pdf/1409.4842v1.pdf.

Figure 7-7 represents two versions of the Inception module presented in the same paper.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 7-7 On the left, we have the naïve version of the Inception module. In the naïve version, we have 1x1, 3x3, and 5x5 convolutions. To reduce the computation, the researchers added a 1x1 conv layer for dimensionality reduction. The image has been taken from https://arxiv.org/pdf/1409. 4842v1.pdf

In the first version, a naïve version of Inception, three different sizes of convolution were done – 1x1, 3x3, and 5x5. Additionally, a max pooling of 3x3 was also proposed. All the respective outputs are then stacked and fed to the next Inception module.

But as the computation cost increases, the researchers added an additional 1x1 convolutional layer for dimensionality reduction. This limits the number of input channels, and 1x1 is less computationally expensive than 3x3 or 5x5. A salient feature is the 1x1 convolution is after the max pooling layer.

Using this second version of dimensionality reduction, a full network was created which is known as *GoogLeNet*. The researchers chose the name as an homage to Yann LeCuns pioneering the LeNet-5 architecture.

Before we go deep into studying the GoogLeNet architecture, it is imperative to discuss the uniqueness of 1x1 convolutions.

5.4.2.1. 1x1 convolutions

In deep networks, the number of feature maps increases with the depth of the network. So, if an input image has three channels and a 5x5 filter has to be applied, then a 5x5 filter will be applied in blocks of 5x5x3.

Moreover, if the input is a block of feature maps from another convolution layer having a depth of 64, then a 5x5 filter will be applied in 5x5x64 blocks. It becomes a computationally challenging task. 1x1 filters help in resolving this challenge.

1x1 convolutions are also called *network-in-network*. It is very simple to understand and implement. It has a single feature or weight of each channel in the input. Similar to any other filter, the output is also a single number. It can be used anywhere in the network, does not require any padding, and generates feature maps with exactly the same width and height as the input.

If the number of channels in the 1x1 convolution is the same as the number of channels in the input image, then invariably the output will also contain the same number of 1x1 filters. And there, 1x1 acts as a nonlinearity function. It is shown in Figure 7-8.

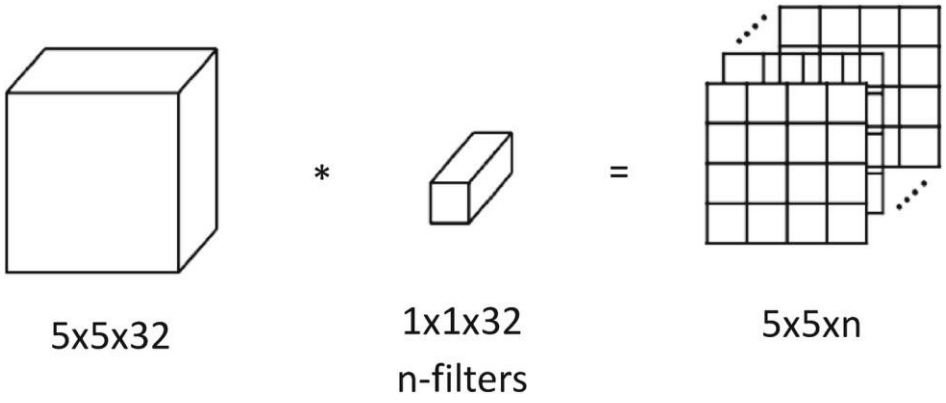


Figure 7-8 1x1 convolutional layer is used to shrink the number of channels. Here, the number of channels in the input and the number of channels in the 1x1 block are the same. Hence, the output has the same number of channels as the number of 1x1 filters

Hence, 1x1 convolution is useful when we want to shrink the number of channels or perform any feature transformation. This results in reducing the computational cost. 1x1 is used in a number of Deep Learning architectures like ResNet and Inception. We will now continue our discussion with the Inception network.

5.4.3. GoogLeNet architecture

We discussed the motivation behind creating the GoogLeNet in the last section. The complete GoogLeNet architecture is shown in Figure 7-9. The blocks in blue represent convolution, red are pooling, yellow are softmax, and green are others.

Figure 7-9 The complete GoogLeNet architecture. Here, blue represents convolution, red blocks are the pooling blocks, while yellow are the softmax ones. We are zooming in on one of the sections later. The image has been taken from https://arxiv.org/pdf/1409.4842v1.pdf

There are a few important properties about the network:

1. The inception network consists of concatenated blocks of the Inception module.

2. There are nine Inception modules which have been stacked linearly.
3. There are three softmax branches (in yellow in Figure 7-9) at different positions. Out of these three, two are in the middle part of the network acting as auxiliary classifiers. They ensure that the intermediate features are good for the network to learn and give regularization effects.
4. The two softmax compute the auxiliary loss. The net loss is the weighted loss of the auxiliary loss and the real loss. The auxiliary loss is useful during the training and not considered for the final classification.
5. It has 27 layers (22 layers + 5 pooling layers).
6. There are close to 5 million parameters in the network.

We are now zooming in on one of the cropped versions from the network to examine the network better (Figure 7-10). Note how the softmax classifier (shown in the yellow block) has been added – to address the problem of vanishing gradients and overfitting. The final loss is the weighted loss of the auxiliary loss and the real loss of the network.

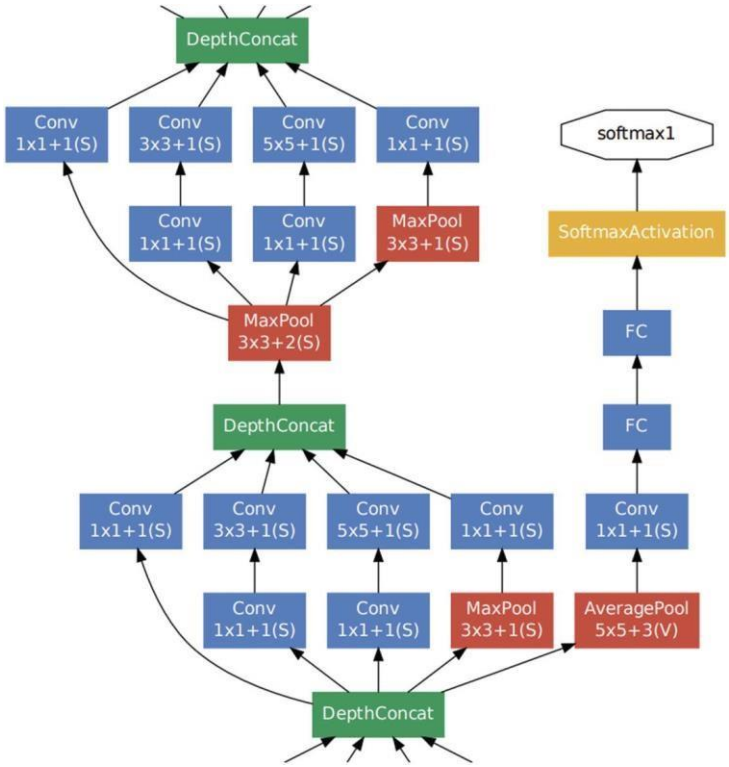


Figure 7-10 A zoomed-in version of a section from the inception network. Note how the softmax classifier has been added (shown in yellow)

Inception v1 proved to be a great solution by getting the first place in ILSVRC2014 and having a 6.67% top-5 error rate.

But the researchers did not stop here. They further improved the solution by proposing Inception v2 and Inception v3 which we are discussing next.

5.4.4. Improvements in Inception v2

Inception versions 2 and 3 were discussed in the following paper: https://arxiv.org/pdf/1512.00567v3.pdf. The motivation was to improve the accuracy and reduce the complexity of the model and hence the computation cost. In Inception v2, there were the following improvements:

1. 5x5 convolutions were factored to two 3x3 convolutions. It was done to improve the computation speed and led to enhanced performance too. It is shown in Figure 7-11. In the figure on the left, we have the original Inception module, and the one on the right is the revised Inception module.

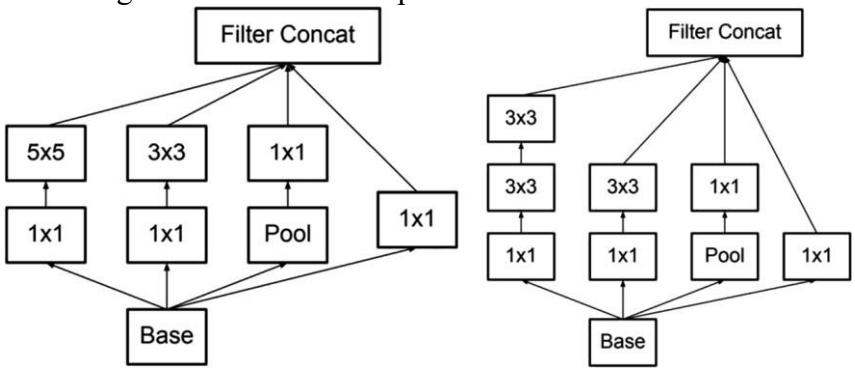


Figure 7-11 Factorization of 5x5 convolutions to two blocks of 3x3 led to the improvement in the computation speed and the overall accuracy of the solution. The image has been taken from https://arxiv.org/pdf/1512.00567v3.pdf

2. The second improvement was the convolutions were factorized such that a filter of nxn size is changed to a combination of 1xn and nx1 as shown in Figure 7-12. For example, 5x5 is changed to performing 1x5 first and then 5x1. This further improved the computation efficiency.

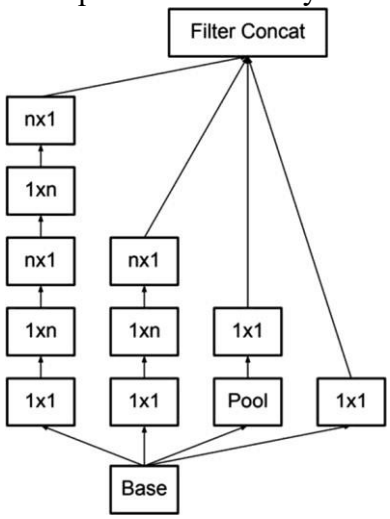


Figure 7-12 Note how nxn conv can be represented as 1xn and nx1. For example, if we put n=5, then 5x[1] is changed as 1x5 and 5x1. The image has been taken from https://arxiv.org/pdf/1512.00567v3.pdf

3
.

With an increase in the depth, the dimensions reduce and hence there can be a loss of information. Hence, an improvement was suggested that the filter banks were made wider instead of going deeper as shown in Figure 7-13.

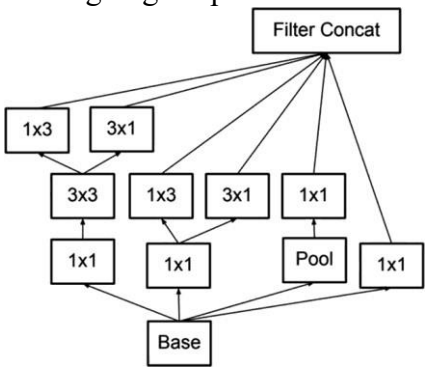


Figure 7-13 The models are made wider instead of deeper. With an increase in depth, the dimensions are reduced drastically, which is an information loss. The image has been taken from https://arxiv.org/pdf/1512.00567v3.pdf

The researchers quoted:

Although our network is 42 layers deep, our computation cost is only about 2.5 higher than that of GoogLeNet and it is still much more efficient than VGGNet.

Moving ahead, in Inception v3, in addition to the preceding improvements, the significant addition was the use of label smoothing which is a regularizing technique to tackle overfitting. The mathematical proof is beyond the scope of the book. In addition, RMSProp was used as an optimizer, and the auxiliary classifier's fully connected layer is batch normalized. It achieved 3.58% top-5 error on an ensemble of four models which is nearly half of the original GoogLeNet model.

There were further improvements in the form of Inception v4 and Inception-ResNet. It outperformed the previous versions, and an ensemble of 3xInception-ResNet(v2) and 1xInceptionv4 resulted in 3.08% top-5 error.

With this, we have completed the discussion on Inception networks.

---

[1] **.5. Video analytics**

Video analytics start with processing the videos. As we can see through our eyes and process the contents of a video using our memory and brain, computers can also see – through a camera. And to understand the contents of that video, Deep Learning is

Both Inception and ResNet are one of the most widely used networks when it comes to really deep Neural Networks. Using Transfer Learning, they can be used for generating fantastic results and are proving to be a real boon to the computer vision problems.

We will now continue studying the video analytics problems we started at the start of the chapter.

providing the necessary support.

Videos are a rich source of information but at the same time are equally complex. In image classification, we take an input image, process it to extract features using CNN, and then classify the image based on the features. In the case of video classification, we first extract the frames from the video and then classify the frames. So, video processing is not one task; rather, it is a collection of subtasks. OpenCV is one of the most popular libraries for video analytics. We are going to use Deep Learning–based solutions for video analytics.

The steps in video classification using Deep Learning are

1. We first get the frames from the video and divide them into training and validation sets.
2. We then train the network on the training data and optimize the accuracy.
3. We will validate on the validation dataset to get the final model.
4. For the unseen new video, we will first grab the frame from the video and then classify the same.

As we can see, the steps are pretty much the same like any image classification solution. The additional step is for the new video – where we first grab a frame and then classify it.

5.6. Python solution using ResNet and Inception v3

Now we will create a Python solution for video analytics. For this, we are going to train a network on a Sports dataset and use it to make predictions for a video file.

You can download the dataset from https://github.com/jurjsorinliviu/Sports-Type- Classifier. The dataset has images of multiple types of sports. We are going to build a classifier for cricket, hockey, and chess. The dataset and the code is uploaded to the GitHub repo at https://github.com/Apress/computer-vision-using-deeplearning/tree/main/Chapter7.

Some examples of images of cricket, hockey, and chess are shown as follows.

Step 1: Load all the required libraries.

```
import matplotlib
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import SGD from sklearn.preprocessing import LabelBinarizer from tensorflow.keras import optimizers from sklearn.model_selection import train_test_split from sklearn.metrics import classification_report from tensorflow.keras.layers import AveragePooling2D from tensorflow.keras.applications import InceptionV3 from tensorflow.keras.layers import Dropout from tensorflow.keras.layers import Flatten from tensorflow.keras.layers import Dense from tensorflow.keras.layers import Input from tensorflow.keras.models import Model from imutils import paths import matplotlib.pyplot as plt import
numpy
as np
import cv2
import os
```

Step 2: Set the labels for the sports we are interested in.

```
game_labels = set(["cricket", "hockey", "chess"])
```

Step 3: Set the value for other variables like location, path, and so on. We will also initiate two lists – complete_data and complete_label – which will be used for holding the values at a later stage.

```
location =
"/Users/vaibhavverdhan/BackupOfOfficeMac/Book/Restart/Apress/Chapter7/Sports- TypeClassifier-master/data"
data_path = list(paths.list_images(location)) complete_data = []
complete_labels = []
```

Step 4: Load the Sports dataset now and read their corresponding labels. The input size is 299x299 because we are training an Inception v3 first. For ResNet, the size is 224x224.

```
for data in data_path:
```

```
        # extract the class label
    from the filename  class_label
    =       data.split("/")[-2]       if
    class_label not in game_labels:
            #print("Not used class
      lable",class_label)  continue
        #print("Used class lable",class_label)
        image = cv2.imread(data)
        image = cv2.cvtColor(image,
        cv2.COLOR_BGR2RGB)   image   =
    cv2.resize(image, (299, 299))

        complete_data.append(image)
    complete_labels.append(class_label)
```

Step 5: Convert the labels to numpy arrays.

```
  complete_data   =   np.array(complete_data)
complete_labels = np.array(complete_labels)
```

Step 6: One-hot encoding is done for the labels now.

```
  label_binarizer = LabelBinarizer() complete_labels
= label_binarizer.fit_transform(complete_labels)
```

Step 7: Divide the data into 80% training data and 20% testing data.

```
(x_train, x_test, y_train, y_test) = train_test_split(complete_data, complete_labels,
    test_size=0.20, stratify=complete_labels, random_state=5)
```

Step 8: We will now initialize the data augmentation object for the training data.

```
training_augumentation =
    ImageDataGenerator(
    rotation_range=25,
    zoom_range=0.12,
    width_shift_range=0.4,
    height_shift_range=0.4,
    shear_range=0.10,
    horizontal_flip=True,
    fill_mode="nearest")
```

Step 9: We are now initializing the testing data augmentation object. Next, we are defining the ImageNet mean subtraction value for each of the objects.

```
  validation_augumentation = ImageDataGenerator()

  mean = np.array([122.6, 115.5, 105.9], dtype="float32")
```

training_augumentation.mean = mean validation_augumentation.mean = mean

Step 10: Load the Inception network now. This model will serve as the base model.

```
inceptionModel = InceptionV3(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(299, 299, 3)))
```

Step 11: We will now make the head of the model which will be placed on the top of the base model.

```
outModel = inceptionModel.output outModel =
AveragePooling2D(pool_size=(5, 5))(outModel)  outModel =
Flatten(name="flatten")(outModel) outModel = Dense(512,
activation="relu")(outModel)  outModel = Dropout(0.6)(outModel)
outModel = Dense(len(label_binarizer.classes_), activation="softmax")
(outModel)
```

Step 12: We get the final model and make the base model layers as nontrainable.

```
final_model = Model(inputs=inceptionModel.input, outputs=outModel)
for layer in inceptionModel.layers:
        layer.trainable = False
```

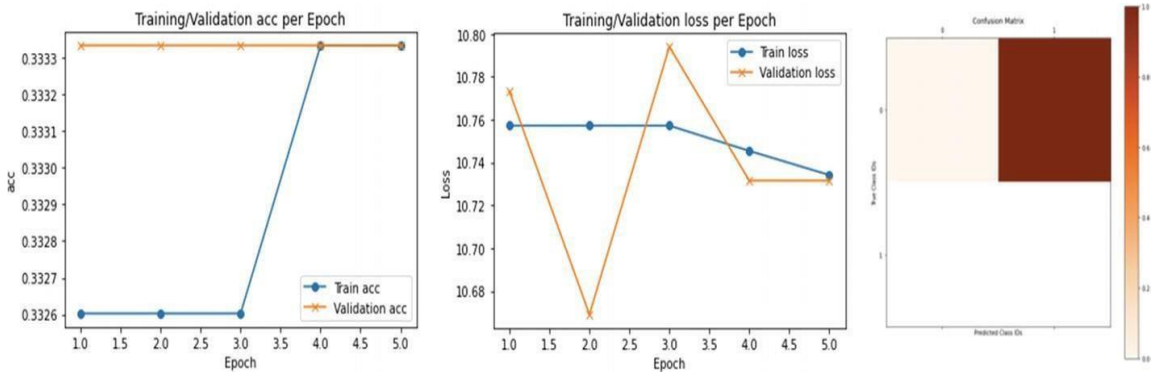
Step 13: We have studied the remaining steps in detail in the last chapters, which are about setting the hyperparameters and fitting the model.

```
num_epochs = 5
 learning_rate = 0.1
 learning_decay = 1e-6
learning_drop = 20
batch_size = 32
sgd = optimizers.SGD(lr=learning_rate, decay=learning_decay,
momentum=0.9, nesterov=True)
final_model.compile(loss='categorical_crossentropy', optimizer=sgd,metrics=
['accuracy'])


model_fit = final_model.fit( x=training_augumentation.flow(x_train, y_train,
    batch_size=batch_size), steps_per_epoch=len(x_train) // batch_size,
        validation_data=validation_augumentation.flow(x
    _test, y_test), validation_steps=len(x_test) //
    batch_size, epochs=num_epochs)
```

```
Epoch 1/5
11/11 [==============================] - 52s 5s/step - loss: 10.7731 - acc: 0.3333
43/43 [==============================] - 217s 5s/step - loss: 10.7571 - acc: 0.3326 - val_loss: 10.7731 - val_acc: 0.
3333
Epoch 2/5
11/11 [==============================] - 37s 3s/step - loss: 10.6691 - acc: 0.3333
43/43 [==============================] - 208s 5s/step - loss: 10.7571 - acc: 0.3326 - val_loss: 10.6691 - val_acc: 0.
3333
Epoch 3/5
11/11 [==============================] - 37s 3s/step - loss: 10.7940 - acc: 0.3333
43/43 [==============================] - 195s 5s/step - loss: 10.7457 - acc: 0.3326 - val_loss: 10.7940 - val_acc: 0.
3333
Epoch 4/5
11/11 [==============================] - 38s 3s/step - loss: 10.7315 - acc: 0.3333
43/43 [==============================] - 195s 5s/step - loss: 10.7187 - acc: 0.3333 - val_loss: 10.7315 - val_acc: 0.
3333
Epoch 5/5
11/11 [==============================] - 39s 4s/step - loss: 10.7315 - acc: 0.3333
43/43 [==============================] - 201s 5s/step - loss: 10.7189 - acc: 0.3333 - val_loss: 10.7315 - val_acc: 0.
3333
```

Step 14: We get the training/testing accuracy and loss.

```
import matplotlib.pyplot as plt f, ax
= plt.subplots() ax.plot([None] +
model_fit.history['acc'], 'o-') ax.plot([None] +
model_fit.history['val_acc'], 'x-') ax.legend(['Train acc',
'Validation acc'], loc = 0)
ax.set_title('Training/Validation acc per Epoch')
ax.set_xlabel('Epoch') ax.set_ylabel('acc')

import matplotlib.pyplot as plt f, ax
= plt.subplots() ax.plot([None] +
model_fit.history['loss'], 'o-') ax.plot([None] +
model_fit.history['val_loss'], 'x-') ax.legend(['Train loss',
'Validation loss'], loc = 0) ax.set_title('Training/Validation
loss per Epoch') ax.set_xlabel('Epoch')
ax.set_ylabel('Loss')

predictions =
model_fit.model.predict(testX) from
sklearn.metrics import
confusion_matrix import numpy as np
rounded_labels=np.argmax(testY,
axis=1) rounded_labels[1]
cm = confusion_matrix(rounded_labels, np.argmax(predictions,axis=1)) def
plot_confusion_matrix(cm):
  cm = [row/sum(row) for row in
    cm] fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111)
    cax = ax.matshow(cm, cmap=plt.cm.Oranges)
  fig.colorbar(cax)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Class IDs')
plt.ylabel('True Class IDs')
plt.show()
plot_confusion_matrix(cm)
```



We can analyze that the network is not good enough for predictions.

Step 15: We will now implement ResNet. The input size changes to 224x224, and everything remains the same. We are also changing the sports classes.

```
game_labels = set(["cricket", "swimming", "wrestling"])
```

Step 16: The complete code is at the GitHub link. We are providing the output here.

```
Epoch 1/5
19/19 [==============================] - 48s 3s/step - loss: 7.8011 - acc: 0.3077
86/86 [==============================] - 619s 7s/step - loss: 1.1110 - acc: 0.6122 - val_loss: 7.8011 - val_acc: 0.30
77
Epoch 2/5
19/19 [==============================] - 49s 3s/step - loss: 0.8106 - acc: 0.7316
86/86 [==============================] - 914s 11s/step - loss: 0.6576 - acc: 0.7295 - val_loss: 0.8106 - val_acc: 0.7
316
Epoch 3/5
19/19 [==============================] - 44s 2s/step - loss: 0.7306 - acc: 0.7692
86/86 [==============================] - 601s 7s/step - loss: 0.5137 - acc: 0.7999 - val_loss: 0.7306 - val_acc: 0.76
92
Epoch 4/5
19/19 [==============================] - 44s 2s/step - loss: 0.6629 - acc: 0.7248
86/86 [==============================] - 736s 9s/step - loss: 0.4913 - acc: 0.8087 - val_loss: 0.6629 - val_acc: 0.72
48
Epoch 5/5
19/19 [==============================] - 45s 2s/step - loss: 0.4514 - acc: 0.8581
86/86 [==============================] - 624s 7s/step - loss: 0.4151 - acc: 0.8482 - val_loss: 0.4514 - val_acc: 0.85
81
```

The algorithm is generating a good validation accuracy of 85.81%.

The model is saved, and then we use it for making predictions for a sample image to check if it is able to predict.

```
model_fit.model.save("sport_classification_model.h5")
```

Step 17: We have covered these steps already in the previous chapters.

```
    file = open("sport_classification",
"wb")
file.write(pickle.dumps(label_binarizer))
file.close()
    modelToBeUsed                                    =
load_model("sport_classification_model.h5")  labels =
pickle.loads(open("sport_classification",   "rb").read())
import numpy as np
    from keras.preprocessing import image
    an_image
=image.load_img('/Users/vaibhavverdhan/BackupOfOfficeMac/Book/Restart/Apress/
    Type-Classifier-master/data/cricket/00000000.jpg',target_size
=(224,224)) # Lo # The image is now getting converted to array of
numbers


    an_image =image.img_to_array(an_image)
    #Let us now expand it's dimensions. It will improve the prediction power
    an_image =np.expand_dims(an_image, axis =0)
    # call the predict method here
    verdict = modelToBeUsed.predict(an_image)
    i                =
np.argmax(verdic
t)       label       =
labels.classes_[i]
```

Step 18: We will now use this model to predict the class from a video of a sport. We took a video of cricket recording. The same video is available at GitHub too.

Step 19: Capture the video in an object.

```
video = cv2.VideoCapture(path_video)
```

Step 20: We are going to iterate over all the frames of the video. For this, we are going to set an indicator isVideoGrabbed as 1 initially. When the end of the video is reached, isVideoGrabbed will become zero, and then we can break from the loop.

We are looping in a while loop. When a frame is grabbed, it is an image and hence is converted to the necessary size and fed to the model for prediction.

```
isVideoGrabbed = 1 while isVideoGrabbed:
      (isVideoGrabbed, video_frame) = video.read()

      if not isVideoGrabbed:
            print("done") break video_frame =
        cv2.cvtColor(video_frame,
        cv2.COLOR_BGR2RGB) video_frame =
        cv2.resize(video_frame, (224,
```

```
        224)).astype("float32") video_frame -=
        mean
      prediction_game                                                    =
  modelToBeUsed.predict(np.expand_dims(video_frame,  axis=0))[0]  i  =
  np.argmax(verdict) game = labels.classes_[i]
      #print(game)
```

Step 21: We can hence generate the predictions for the entire video frame by frame. This way, we can use Neural Networks to have a look at a video and predict the sports being played in it.

This concludes our Python solution using ResNet and Inception v3 network. As we can observe, using transfer learning, it is not a big challenge to harness the powers of these very deep Neural Networks. But creating a tuned solution is still a tough job. In the preceding example, we can analyze the difference between the respective accuracies of ResNet and Inception v3 network. It depends on the dataset and the number of images available.

Part-A

1. What is the purpose of skip connections and how are they useful?
2. What is the problem of vanishing gradients and how can we rectify it?
3. What is the improvement between Inception v1 and Inception v3 networks?
4. What are the use cases of video analytics?
5. Discuss about video analytics.
6. List the uses of skip connection.
7. Discuss about inception v1.
8. What is inception v2.
9. Discuss about inception v3.
10. What is the use of GoogleNet.
11. What is the use of ResNet.

Part-B

1. Explain about Video Processing.
2. Discuss about various use cases of video analytics
3. Elaborate about Vanishing Gradient and exploding gradient problem
4. Explain about ResNet architecture.
5. Illustrate ResNet and skip connections.

6. Discuss about Inception Network.
7. Elaborate about GoogleNet architecture
8. Explain about Improvement in Inception v2
9. Develop a python code for Implementation of ResNet
10. Develop a python code for Inception v3.