**Analytics foundation**

****

**Report: Image Processing using R**

**Submitted To: Submitted By**

**Dr. Ravi Kulkarni Ayush Trivedi(18020343008)**

**B. Sai Teja (18020343009)**

**B. Pranjali(18020343010)**

**Rahul Prajapati(18020343032)**

**Introduction**

# **What's an image?**

# An image refers to a 2D light intensity function f(x,y), where (x,y) denote spatial coordinates and the value of f at any point (x,y) is proportional to the brightness or gray levels of the image at that point. A digital image is an image f(x,y) that has been discretized both in spatial coordinates and brightness. The elements of such a digital array are called image elements or pixels.

**Resolution**

Image resolution is typically described in PPI, which refers to how many pixels are displayed per inch of an image. Higher resolutions mean that there more pixels per inch (PPI), resulting in more pixel information and creating a high-quality, crisp image. Images with lower resolutions have fewer pixels, and if those few pixels are too large (usually when an image is stretched).

**Pixels**

Screens display images through pixels. A pixel is a dot that can be manipulated. They are like building blocks; many are needed to build an image. If you colored in the squares on a piece of graph paper to create an image, each of those squares would be like a pixel. Pixels are not always the same size from device to device.

**What is Image Processing?**

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them. It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

**Image processing basically includes the following steps.**

1. Image acquisition: to acquire a digital image

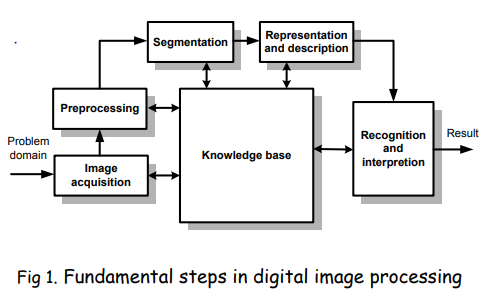
2. Image pre-processing: to improve the image in ways that increase the chances for success of the other processes.

3. Image segmentation: to partitions an input image into its constituent parts or objects.

4. Image representation: to convert the input data to a form suitable for computer processing.

5. Image description: to extract features that result in some quantitative information of interest or features that are basic for differentiating one class of objects from another.

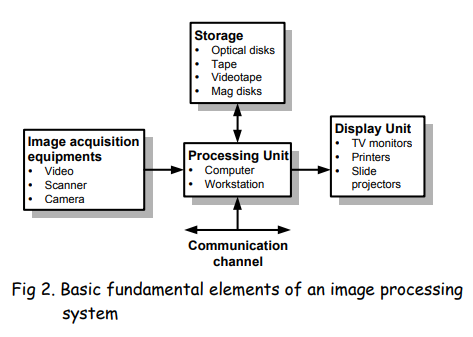
6. Image recognition: to assign a label to an object based on the information provided by its descriptors.

7. Image interpretation: to assign meaning to an ensemble of recognized objects.

# • Knowledge about a problem domain is coded into an image processing system in the form of a knowledge database.

# **Elements of digital image processing systems:**

# • The basic operations performed in a digital image processing systems include (1) acquisition, (2) storage, (3) processing, (4) communication and (5) display



**Applications of image processing:**

As discussed earlier, the technology of image processing can be used in many fields and the process depends on what kind of results are expected from the image but here are some of the most widely used applications of image processing:

1. **Computerised photo editing**: This is the most common and the most widely used feature of image processing and used to edit small images on a personal level and huge images at professional level too. Most of the times, software like Photoshop are used to give final touches to the images so that they can be uploaded to the website or whatever they want to do from the image.
2. **Space imaging**: This is the kind of application of image processing used in making the images captured by the huge telescopes like the Hubble telescope, more readable and so that more amount of information can be extracted from that image because mostly the images captured by telescopes lack in contrast and hence the difference between the two objects cannot be seen, so this might require some enhancement to get a better grasp of the image.
3. **Biomedical Image processing**: This application of image processing is one which is making some really fast progress and is mostly used to make the scanning from any medical device more readable and bring them closer to the accuracy. For instance, let's take an example, a person with a broken bone might get an x-ray done to scan the same and confirm if there is really a fracture or not. Now if the image is filtered accordingly, more accurate results can be seen and a firm confirmation can be made whether that person really has a broken bone or the pain is due to some other reason.
4. **Industrial application**: This application itself is huge and covers aspects like facial recognition, computer vision, and robotic vision. In this article we are not going that deep into these topics hence we are not going to discuss them and a brief introduction is suffice for applications such complex.

**Image manipulations using R**

**What is Magick ?**

**ImageMagick** is one of the famous open source libraries available for editing and manipulating Images of different types (Raster & Vector Images). Magick is an R-package binding to ‘ImageMagick’ for Advanced Image-Processing in R, authored by Jeroen Ooms.

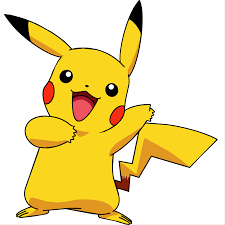
**Magick** supports many common image formats like png, jpeg, tiff and manipulations like rotate, scale, crop, trim, blur, flip, annotate and much more.

**R Code for doing image manipulations using Magick package**

library(magick)

i2=image\_read('pikapi.png')

print(i2)

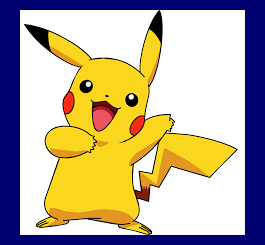
**Output**

blur\_i2=image\_blur(blur\_i2,10,5)

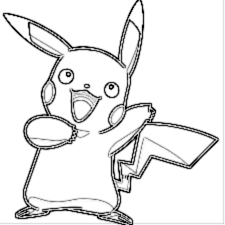
print(blur\_i2)

**Output** 

image\_border(image\_background(i2, "yellow"), "#000070", "20x10")

**Output** 

image\_charcoal(i2)

**Output** 

i3=image\_read('bulba.png')

print(i3)

image\_crop(i3, "100x150+50")

**Output**



image\_scale(i2, "300")

**Output** 

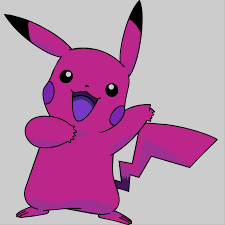
image\_rotate(i2, 45)

**Output**



image\_modulate(i2, brightness = 80, saturation = 70, hue = 50)

**Output**



**Reading Data from Image& PDF files**

**What is Tesseract?**

A tesseract is a cube in 4 dimensional space. It is a 4D space where every shape has a cube.

**What is tesseract package in R?**

A powerful optical character recognition (OCR) engine that supports over 100 languages. The engine is highly configurable in order to tune the detection algorithms and obtain the best possible results.

**Improve the image quality:**

### **Rescaling:** Tesseract works best on images which have a DPI of at least 300 dpi, so it may be beneficial to resize images.

### **Binarisation:** This is converting an image to black and white. Tesseract does this internally, but the result can be suboptimal, particularly if the page background is of uneven darkness.

### **Noise Removal:** Noise is random variation of brightness or colour in an image that can make the text of the image more difficult to read. Certain types of noise cannot be removed by Tesseract in the binarisation step, which can cause accuracy rates to drop.

### **Rotation / Deskewing:** A skewed image is when a page has been scanned when not straight. The quality of Tesseract's line segmentation reduces significantly if a page is too skewed, which severely impacts the quality of the OCR. To address this rotating the page image so that the text lines are horizontal.

### **Border Removal:** Scanned pages often have dark borders around them. These can be erroneously picked up as extra characters, especially if they vary in shape and gradation

**Preprocessing with Magick**

The accuracy of the OCR process depends on the quality of the input image. You can often improve results by properly scaling the image, removing noise and artifacts or cropping the area where the text exists. The awesome magick R package has many useful functions that can be use for enhancing the quality of the image. Some things to try:

* If your image is skewed, use image\_deskew() and image\_rotate() make the text horizontal.
* image\_trim() crops out whitespace in the margins. Increase the fuzz parameter to make it work for noisy whitespace.
* Use image\_convert() to turn the image into greyscale, which can reduce artifacts and enhance actual text.
* If your image is very large or small resizing with image\_resize() can help tesseract determine text size.
* Use image\_modulate() or image\_contrast() or image\_contrast() to tweak brightness / contrast if this is an issue.
* Try image\_reducenoise() for automated noise removal. Your mileage may vary.
* With image\_quantize() you can reduce the number of colors in the image. This can sometimes help with increasing contrast and reducing artifacts.
* True imaging ninjas can use image\_convolve() to use custom convolution methods.

**Need to preprocessing**

Tesseract does various image processing operations internally (using the Leptonica library) before doing the actual OCR. It generally does a very good job of this, but there will inevitably be cases where it isn't good enough, which can result in a significant reduction in accuracy.

**R working code for tesseract:**

**# Low Quality Image**

text= tesseract::ocr("dog.png", engine = eng)

cat(text)

**#Improving the image quality using magick package**

library(magick)

input <- image\_read("dog.png")

text <- input %>%

image\_resize("2000x") %>%

image\_convert(type = 'Grayscale') %>%

image\_trim(fuzz = 40) %>%

image\_write( density = '300x300') %>%

tesseract::ocr()

cat(text)

result= tesseract::ocr\_data("dog.png",engine=eng)

print(result, n = 20)

**# read from pdf**

**# convert the pdf to png file and then perform tesseract**

pngfile <- pdftools::pdf\_convert('OUTFIT Declaration form.pdf', dpi = 600)

text <- tesseract::ocr(pngfile)

cat(text)

**# List all parameters with \*colour\* in name or description**

tesseract\_params('colour')

**#tessedit\_char\_whitelist which restricts the output to a limited set of characters.**

**#This may be useful for reading for example numbers such as a bank account,**

**#zip code, or gas meter.**

numbers <- tesseract(options = list(tessedit\_char\_whitelist = ".0123456789"))

cat(ocr("dog.png", engine = numbers))

numbers <- tesseract(options = list(tessedit\_char\_whitelist = "$.0123456789"))

cat(ocr("receipt.png", engine = numbers))

**# only numbers without the $**

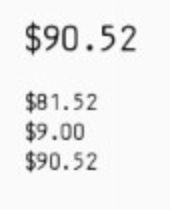
numbers <- tesseract(options = list(tessedit\_char\_whitelist = ".0123456789"))

cat(ocr("receipt.png", engine = numbers))

**# how this forces tesseract to detect a number (3 or 8 or 5)**

**# if we rule out the dollar sign.**

**Images used in R code:**

** **



**Reading Exif Data from Images**

**EXIF stands for Exchangeable Image File Format**

**What is Exif?**

Every time a picture is taken with digital camera or phone, a file (typically a JPEG) is written to device’s storage. In addition to all the bits dedicated to the actual picture, it records a considerable amount of supplemental metadata as well. This can include date, time, camera settings, and possible copyright information. Further metadata can be added to EXIF, such as through photo processing software. EXIF is somewhat limited in that it is only supported by two image formats - JPEG and TIFF. Usually this isn't a problem, since most people will tend to shoot in JPEG, but it does mean that if you set your camera to RAW mode you won't be able to capture EXIF data. Luckily camera manufacturers have thought of this and come up with their own equivalents which do support RAW, and when you later save the file as a JPEG that data will be converted to EXIF and stored with it

* Exif data is stored in the "headers" of the relevant photograph, so not in a separate file but in the same file as the photograph.

**Need of Exif Data**

1. To understand the basics: Understanding the properties of the images can give you a better view of how big the resolution is or how much the exposure was originally and so on and so forth. The Exif data can be accessed easily from the camera, from the laptop or computer. The Exif data is easy to locate and take note of.

2. To take note of the information: The Exif data contains the basic properties of an image. Some of these basic properties that can be easily viewed in a camera are the dimensions, device make, device model, color space, profile name, focal length, exposure time and the date, The image settings will now include properties such as orientation, x resolution, y resolution, resolution unit, date and time, f-stop, exposure program, ISO speed ratings, version, components configuration, shutter speed and aperture value

**Required Package to read Exif Data from Image: exifr**

**Title:** Exif Image data in R

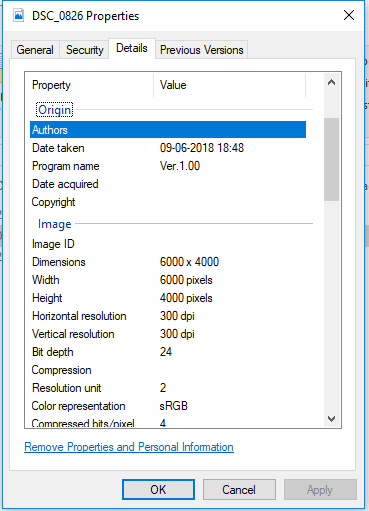
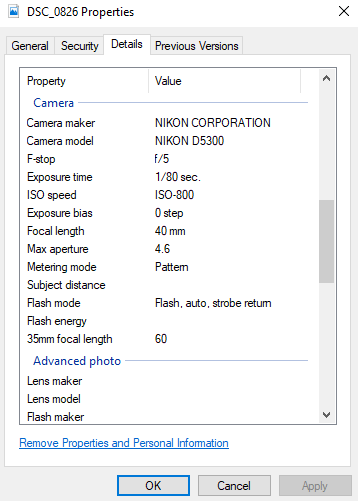
Exif Package has dependencies on the

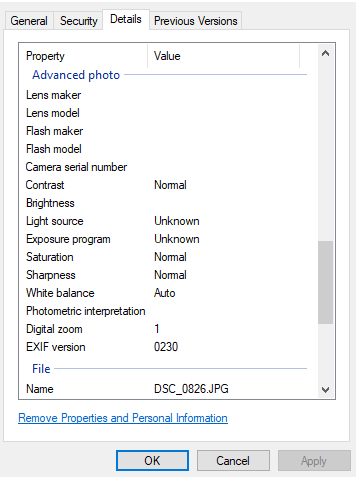
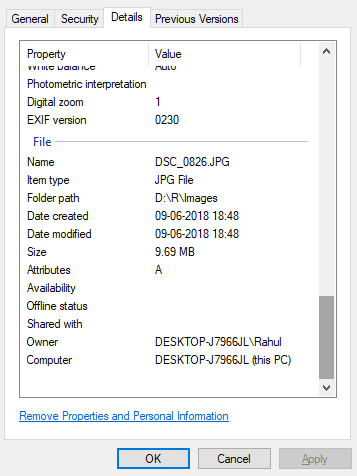
**Description:** Reads EXIF data using ExifTool and returns results as a data frame. ExifTool is a platform-independent Perl library plus a command-line application for reading, writing and editing meta information in a wide variety of files. ExifTool supports many different metadata formats including EXIF, GPS, IPTC, XMP, JFIF, GeoTIFF, ICC Profile, Photoshop IRB, FlashPix, AFCP and ID3, as well as the maker notes of many digital cameras by Canon, Casio, FLIR, FujiFilm, GE, HP, JVC/Victor, Kodak, Leaf, Minolta/Konica-Minolta, Motorola, Nikon, Nintendo, Olympus/Epson, Panasonic/Leica, Pentax/Asahi, Phase One, Reconyx, Ricoh, Samsung, Sanyo, Sigma/Foveon and Sony

**Sample Image:**



**Exif Screen shot from Window:**

**R Code for extracting the Exif Data:**

**## Calling Library for loading image and Reading data from Image**

library(tools)

library(exifr)

library(dplyr)

library(leaflet)

**## Reading Image from Directory**

image<-list.files("D:/R/Images")

View(image)

print(image)

length(image) ## no. file contained by the image

class(image) ## "character"

str(image) ## structure of the image variable it holds all the file in the image

**## To read the exif data of image:**

ed=read\_exif(image)

length(ed) ## No of all the attributes 2017

names(ed) ## List of all the attributes

View(ed) ## View all exif detail in tabular form

**## to read selected exif data of image**

ed1= read\_exif(image, tags = c("SourceFile","FileName", "imagesize","Software",

"ShootingMode", "Model","ExposureTime","FNumber",

"ISO","Flash","FocalLength","LightSource","WhiteBalance",

"FocusMode","Aperture"))

View(ed1)

ed1

print(as.data.frame(ed1))

**## Result**

## **SourceFile FileName ImageSize Software**

##1 DSC\_0826.JPG DSC\_0826.JPG 6000x4000 Ver.1.00

## **ShootingMode Model ExposureTime FNumber ISO Flash**

##1 0 NIKON D5300 0.0125 5 800 31

## **FocalLength LightSource WhiteBalance FocusMode Aperture**

##1 40 0 AUTO AF-A 5

**## Accessing individal Attributes of Image:**

ed$SourceFile ##"DSC\_0826.JPG"

ed$FileName ##"DSC\_0826.JPG"

ed$FileSize ##10166260

ed$FileModifyDate ##"2018:06:09 18:48:08+05:30"

ed$FileCreateDate ##"2018:06:09 18:48:08+05:30"

ed$FileType ##JPEG

ed$Make ##NIKON CORPORATION

ed$ExposureTime ##0.0125

ed$ISO ##800

ed$FNumber ##5

ed$MeteringMode ##5

ed$Flash ##31

ed$LightSource ##0

ed$FocalLength ##40

ed$WhiteBalance ## Auto

ed$FocusMode ##AF-A

ed$`ActiveD-Lighting` ##65535 NA

ed$VibrationReduction ##1

ed$Lens ##"18 55 3.5 5.6" lens configuaration

ed$LensType ## 14

ed$ShootingMode ##0

ed$FirmwareVersion ##1.00

ed$AFAreaMode ## 8

ed$AFPointsUsed ##"7f 6b 00 00 00"

ed$ExifImageWidth ## 6000

ed$ExifImageHeight ## 4000

ed$SceneType ## 1

ed$Contrast ## 0

ed$Sharpness ## 0

ed$Saturation ## 0

ed$Aperture ## 5

ed$ImageSize ## "6000x4000"

ed$Megapixels ## 24

ed$AutoFocus ## 1

ed$ImageWidth ## 6000

ed$ImageHeight ## 4000

ed$NumberOfImages ## 3

**## Writing/storing exif data of image to csv file:**

csv\_data=write.csv(ed, 'Exifdata\_images.csv', row.names = F)

Note: Exif\_Images\_or.csv is original file created through R studio.

**References:**

* https://www.r-bloggers.com/extracting-exif-data-from-photos-using-r/
* https://cran.r-project.org/web/packages/exifr/exifr.pdf
* https://cran.r-project.org/web/packages/magick/vignettes/intro.html
* https://datascienceplus.com/image-processing-and-manipulation-with-magick-in-r/
* https://cran.r-project.org/web/packages/tesseract/vignettes/intro.html
* https://ropensci.org/technotes/2017/08/17/tesseract-16/